C. L. Chen
R. A. Rutledge

Fault-Tolerant Memory Simulator

Memory systems in modern computers employ a variety of methods to achieve fault tolerance, such as single- or double-error correction, page deallocation, or the use of spare chips or cells. Such methods ensure that the failure rate of the system is considerably less than the sum of the failure rates of the components. However, these methods also complicate the task of evaluating system reliability. The memory reliability function is too intractable to handle analytically, and we must turn to Monte Carlo methods. This paper describes the Fault-Tolerant Memory Simulator (FTMS), an interactive APL program which uses Monte Carlo simulation to evaluate the reliability of fault-tolerant memory systems.

1. Introduction

The problem

The designer of a computer memory system can choose from a wide variety of techniques for achieving fault-tolerance. Error-correcting codes have been standard in large memories for years [1], and page deallocation is commonly used to tolerate uncorrectable errors until a repair can be made. The discovery of a new physical mechanism for soft errors in dynamic memories [2] led to new error-correction techniques to handle the new error types [3]. More recently, fault-alignment exclusion [4] and dynamic spare switching [5] have been proposed as ways to quickly repair a memory without bringing the system down for component replacement.

These methods offer significant opportunities to improve the reliability of memory systems. In order to choose among these alternatives, however, the designer must be able to evaluate the impact of each technique, or combination of techniques, on reliability. This is not an easy task. In a system without fault-tolerance, each component failure causes a system failure; therefore, the system-failure rate is simply the sum of the component failure rates. In a fault-tolerant memory, on the other hand, the effect of any single component failure will, in general, depend on which other components have already failed. Therefore, the system failure rate, which in a memory is the uncorrectable error (UE) rate, is a very complicated function of the component failure rates, failure modes, system design, etc.

Background

The calculation of $R(t)$, the probability that no UE occurs before time t, for a memory with single-error correction is difficult but not impossible. A number of similar equations have previously been given for $R(t)$ [6–11]. The main difference among these is the particular failure modes which are assumed for the array chip (i.e., single cell, row or column of cells, whole chip). If the memory is replaced with a "good-as-new" memory each time a UE occurs, then the sequence of UE times forms a renewal process [12]; and $UE(t)$, the expected number of UEs in the first t hours of system life, is the unique solution of the renewal equation,

$$UE(t) = \int_0^t [1 + UE(t - s)]f(s)ds,$$

where $f(t) = -dR(t)/dt$ is the probability density function of the time-to-first-UE. For a small memory contained on a single array card this would be a reasonable assumption; the only replacement possible is the entire memory. If soft errors as well as hard failures are considered, it is still possible to derive an exact analytic expression for the UE rate [13], as long as the entire memory is replaced after each hard UE.

For larger memories which are contained on multiple array cards, the required calculations are far more difficult; the state of the memory after repair is in general very different from the state of a brand-new memory. It may contain cards of

© **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.

different ages (and therefore different failure rates) and may also contain some correctable hard failures which were left in at the time of repair. The future of such a system is by no means independent of its past, and renewal theory does not apply. The situation becomes even more complex when we consider double-error correction, page deallocation, address permutation, spare switching, etc.

It is clear therefore that, except for one-card memories, analytic calculation of memory reliability is impractical, if not impossible: Some form of Monte Carlo simulation program is required [14–16]. Memory designers need a flexible program which can evaluate all current memory design options and which can be quickly adapted to changing circumstances such as new failure modes [2] or new techniques of fault-tolerance [3–5].

FTMS

This paper describes the Fault-Tolerant Memory Simulator (FTMS), an interactive APL program written to fill that need. FTMS is designed to evaluate several reliability parameters for memories which employ single- or double-error correction, page deallocation, address permutation, and spare switching; and which are subject to hard failures and soft errors.

Method of simulation

FTMS estimates memory reliability parameters by simulating the life history of many systems, counting UEs (and other parameters) on each simulated system, and averaging the results. For each individual system this involves simulating random failure times for each component, checking to see whether UEs occur, and simulating the appropriate maintenance action (e.g., card replacement).

The first obstacle in writing such a simulator is to account for the enormous number of possible failures which can occur (e.g., 18 million different cell failures for a 2-megabyte system). This problem was originally solved for a precursor to FTMS when an algorithm was derived which generates random failure times in order of occurrence [17, 18]. The next choice is how to represent the state of the memory, and what information to keep. The representation scheme described here is compact while preserving all needed information, and has proven to be readily adaptable to new designs and fault-tolerant techniques. One of the fundamental principles of Monte Carlo simulation is to replace estimates with exact values wherever possible [19]. This principle was used to good advantage when soft-error capability was added to FTMS. FTMS estimates the soft-error rate without simulating any soft errors, which simultaneously reduces the cost of simulation and improves the accuracy of the estimate.

The law of large numbers guarantees that the estimates from a Monte Carlo program will be close to the true values for sufficiently large sample sizes; the user wants to know how close and how large. FTMS provides confidence limits on all of its estimates, but these are not available until after the program has been run. Before running the simulation the user has no way of knowing how many samples will be needed to produce the accuracy he requires. Too few samples mean he must rerun the job; too many waste computer resources. Therefore FTMS incorporates a sequential stopping rule [20] which continues to sample until each parameter has been estimated with the accuracy and confidence level specified by the user.

In the following section we describe the model used by FTMS to represent memory systems. Section 3 describes some of the internal structure of the FTMS programs and the algorithms used to generate random failure times, estimate the soft UE rate, and decide on an optimal stopping time. Finally, in Section 4, we show by example how FTMS can be used to evaluate the impact of various memory design and maintenance strategies on reliability.

2. Memory model

The memory model consists of four parts: architecture, failure modes, maintenance strategy, and reliability parameters. The architecture defines the logical and physical configuration of the memory. The failure modes and rates define the types and frequencies of component failures which can cause the memory to malfunction. The maintenance strategy defines the actions which are taken to repair the memory when it has failed. Finally, reliability parameters are defined to quantify the effects of architecture, failure rates, and maintenance strategy on the frequency of UEs, the amount of degradation, and the cost of service.

• *Architecture*

The memory architecture consists of the logical structure of the memory and the logical-to-physical mapping of the bits in the memory. The logical structure includes the number and size of data words and pages, the error-correcting capability of the ECC, and any built-in fault-tolerance features. The physical structure includes the number and arrangement of cells per array chip, and chips per array card.

The memory consists of F identical array cards, each logically subdivided into C bit-planes. Each bit-plane is an $A \times B$ matrix of array chips, and each chip is an $X \times Y$ matrix of cells. A data word consists of FC bits, one from the same address in each bit-plane. The memory has either single-error correction (SEC) or double-error correction (DEC) capability for each data word. The number of errors which are correctable is denoted NEC. **Figure 1** shows the structure of a 2-megabyte memory containing 16K-bit chips on 18 array cards.

Fault Alignment Exclusion

The use of Fault Alignment Exclusion (FAE) to remove UEs by address permutation is discussed more fully elsewhere in

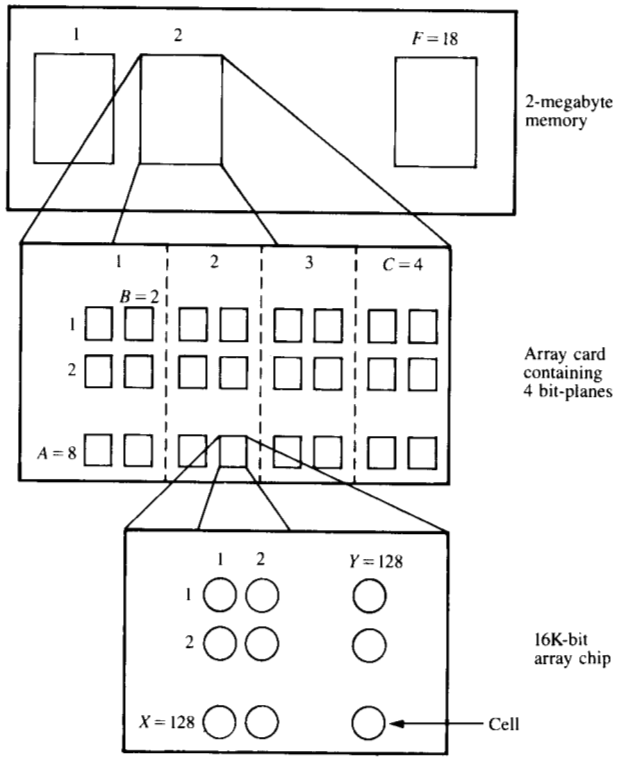

Figure 1 A 2-megabyte memory using 16K-bit chips.

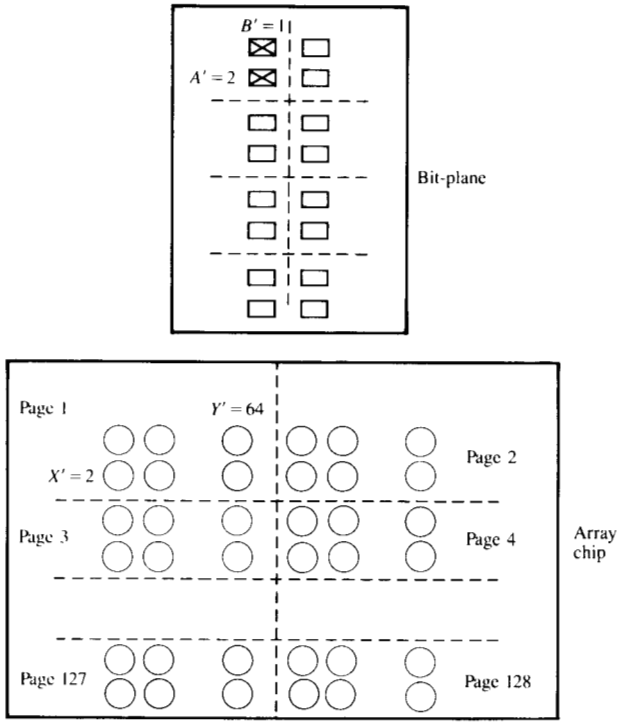

Figure 2 Page locations for pages 1–128: indicated bit positions on each chip marked X on every bit-plane.

this issue [4]. We briefly describe how FAE is incorporated into the FTMS model.

A word address consists of a chip address (row and column within a bit-plane) and a cell address (row and column within a chip). The cell and chip-column portions of the address are fixed physical locations, but the chip-row address is logically determined by the contents of a control register. That is, the logical chip-row address is some permutation of the physical chip-row address. This permutation capability is included to allow us to misalign failures in the memory so that all resultant data words contain only correctable errors.

The address permutation capability is defined by the permutations which can be represented, the algorithm used to choose a permutation, and the information available to that algorithm. The logical chip-row address consists of $N = (\log A)/(\log 2)$ bits used to select one of A chips in a column. The physical address is obtained by forming the exclusive-or of R of these N bits with the bits in a control register, and leaving the remaining $N - R$ bits unchanged. Each column of chips has its own control register and is independently permutable. Thus there are a total of 2^R permutations possible for each column, and $2^{(RFCB)}$ possibilities for the entire memory. Various algorithms have been written to choose a set of permutations which misalign errors so that no data words contain uncorrectable errors. The algorithm used is part of the memory architecture, and the name of the algorithm is denoted ALGO (i.e., if the algorithm named ALGJC is to be used, ALGO = "ALGJC"). The input to the algorithm is a fault-map which assigns each chip to a category which depends on the worst hard failures on the chip. A five-category map tells whether failures on a chip affect single cells only, rows only, columns only, rows and columns, or the entire chip. In a three-category map the middle three categories are combined. The number of categories in the fault-map is denoted M.

FTMS has several built-in FAE algorithms, and also accepts any user-designed algorithm. All the user has to do is copy his algorithm into the APL workspace containing FTMS and then set ALGO equal to the name of his algorithm.

Page deallocation

The $ABXY$ words in the memory are subdivided into P pages, each containing $(ABXY)/P$ data words. Each page consists of an $A' \times B'$ matrix of sub-chips from each bit-plane, and each sub-chip is an $X' \times Y'$ matrix of cells on a chip. If a data word contains an uncorrectable error, the page containing that word may be deallocated. This allows the memory to recover quickly, but at a somewhat reduced capacity, from the effects of a failure. **Figure 2** shows the bit locations of pages 1–128 for the memory in Fig. 1, assuming a 2048-byte page. The page structure is identical to the memory structure with F, C, A, B, X, Y replaced by F, C, A', B', X', Y'.

Spare switching

Each array card contains S additional chips which can be logically switched in to replace one of the other CAB chips on the card. Any spare can replace any other chip, but no more than one spare can be used in any spare domain. There are D spare domains on a card, each consisting of A/D consecutive rows of chips in each bit-plane. **Figure 3** shows the spare domains from one card of the memory in Fig. 1, assuming two spares and four spare domains.

• *Failure modes and rates*

The components which make up the memory are subject to hard failures and soft errors.

Hard failures

A hard failure is a permanent inability of a component to reliably store data in one or more cells. An array chip which consists of a rectangular array of cells may fail in a number of different ways, including single cell, row or column of cells, and entire chip failure [7, 10, 21, 22]. In addition, we consider failures in address lines, data buses or registers, or decoding logic, which can disable groups of chips. The failure is described logically in terms of the cells which cannot be reliably written to or read from. In these terms, there are nine different hard-failure modes:

1. Single cell.
2. Row of Y cells on a chip.
3. Column of X cells on a chip.
4. Entire chip (called a chip-kill).
5. Row of B chips in a bit-plane.
6. Column of A chips in a bit-plane.
7. Entire bit-plane.
8. Entire card.
9. Logic support (entire memory disabled).

Each failure mode has a corresponding failure rate function which is defined as a step function. **Figure 4** shows a typical failure rate curve for a 16K-bit array chip. The rates for the first four modes are expressed as a fixed percentage of the total chip failure rate, and the remaining rates are defined by separate curves similar to Fig. 4. Failures occur at random times in accordance with these rates and at random locations in the memory. When failures affect more than NEC bits in the same word, an uncorrectable error (UE) occurs immediately.

The hard-failure rates are specified in a matrix, denoted **HARD**, which contains in column one the end points of the time intervals and in columns two through ten the failure rates for the nine failure modes over the corresponding time intervals. The failure rates are in units of percent per 1000 hours per component.

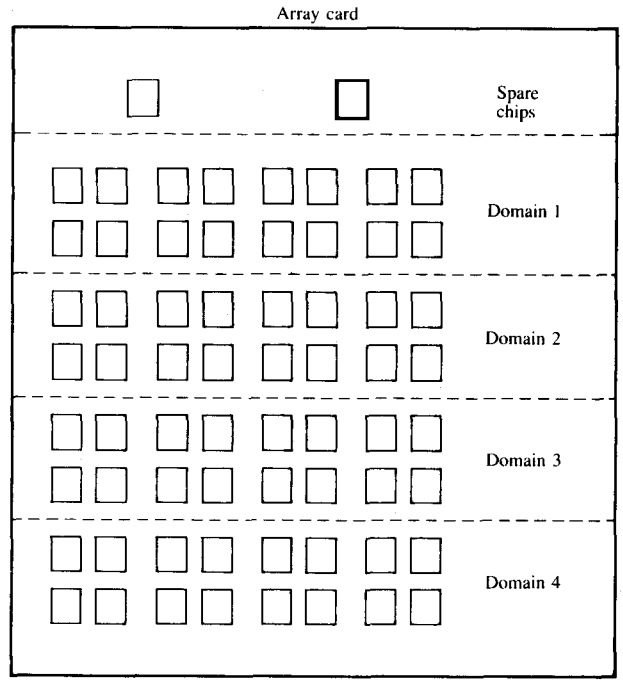

Figure 3 Spare domains on one card of the memory of Fig. 1.

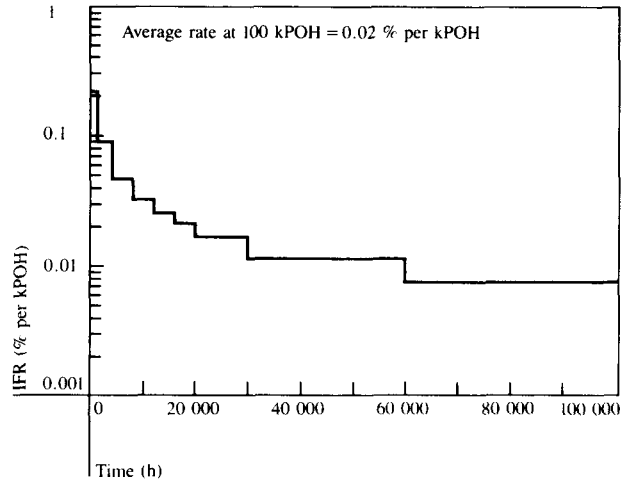

Figure 4 Failure rate curve for 16K-bit chip.

Soft errors

In 1978 May and Woods [2] discovered a new physical mechanism for soft errors in dynamic memories. Alpha particles produced by the radioactive decay of minute quantities of uranium and thorium in the packaging materials can cause bits to flip in dynamic RAMs. These bit flips are soft failures in the sense that they have no permanent effect on the RAM; the cell which suffers the failure is completely recovered when

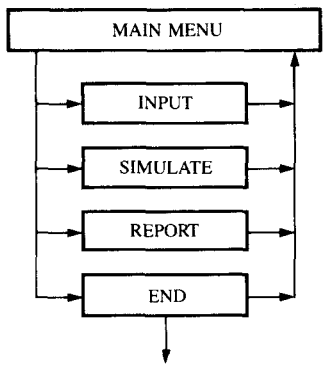

Figure 5 Main menu selections in FTMS.

a new bit is written into it and thereafter has no greater chance of failure than any other cell. We assume that a soft error affects a single cell. If a soft error occurs in a word which already contains *NEC* errors, a soft UE (SUE) occurs, and is immediately repaired by rewriting with good data. If a soft error occurs in a word with fewer than *NEC* errors, it is immediately corrected by rewriting with good data. The soft error rate is assumed to be a constant, denoted *SOFT*, and is expressed in units of percent per 1000 hours per chip.

This model of soft errors neglects some realities which should be considered when applying the model. In the first place, some array chips, especially charge-coupled devices, may be subject to multi-bit soft errors. Secondly, there are algorithms available to correct a combination of hard and soft errors even though the total number of errors exceeds *NEC* [1, 3], so a SUE may not really be uncorrectable. Finally, if good data is not written frequently enough, soft errors may accumulate to line up with other soft errors.

• *Maintenance strategy*

The maintenance strategy is a set of rules which prescribe what action will be taken when various events occur.

Events

There are three types of events which can be used to trigger a maintenance action. If a UE occurs, some action must be taken because a memory containing a UE is considered to be nonfunctioning. If a hard failure occurs which does not cause a UE, maintenance may be specified to reduce the risk of future UEs. Finally scheduled maintenance may be performed at a fixed time independent of failures occurring in the memory.

Actions

The actions which can be taken are card replacement, page deallocation, address permutation, and spare switching. The "card replacement" action causes removal of the minimum number of cards to achieve one of the following: no UEs, at most x bad bits per card, or at most x bad bits in the memory. Page deallocation deallocates all pages containing UEs but no more than x pages in total may be deallocated. Address permutation attempts to produce a configuration with at most $x(NEC)$ bad bits per word. Spare switching attempts to replace a chip-kill with a spare chip. In each case x is a user-specified constant.

A complete strategy is a set of event-action pairs which may be applied sequentially. For example,

1. At 200 hours, replace any card with more than two bad bits.
2. At a chip-kill, switch in a spare.
3. At a UE, deallocate up to 32 pages.
4. At a UE, permute addresses to remove UEs.
5. At a UE, replace cards to remove all UEs.

Action 4 would be taken only if action 3 were unsuccessful, and action 5 only if action 4 were unsuccessful. Action 5 is included as the last resort in any maintenance strategy: it cannot fail to be successful and UEs cannot be left in the memory.

• *Reliability parameters*

The reliability of a fault-tolerant memory can be measured in terms of service cost, frequency of interruptions, and amount of degradation. Service cost is measured in terms of card replacements (CRs) and repair actions (RAs). An RA is defined to be an unscheduled action which requires the intervention of a service person. Specifically, an RA is the replacement of one or more cards due to a hard failure (scheduled maintenance is not included). Frequency of interruption is measured by the rate of hard and soft UEs. Soft UEs are measured separately because they have a much smaller impact on the system (component replacement is not required), and because they can in some cases be corrected [1, 3]. Degradation is measured by the average number of pages deallocated and the average number of words containing one or (if DEC is used) two bad bits. A large number of bad bits may cause performance degradation even if all errors are correctable, because of time taken to perform the corrections. The following specific parameters are defined:

$UE(t)$ = Expected number of UEs in $(0,t)$.
$SUE(t)$ = Expected number of SUEs in $(0,t)$.
$CR(t)$ = Expected number of cards replaced in $(0,t)$.
$RA(t)$ = Expected number of RAs in $(0,t)$.
$B_1(t)$ = Expected average number of words with one bad bit during $(0,t)$.
$B_2(t)$ = Expected average number of words with two bad bits during $(0,t)$.
$DP(t)$ = Expected average number of pages deallocated during $(0,t)$.

The first four parameters are cumulative counts, while the last three are time averages of states of the memory.

3. The FTMS program

FTMS is an interactive menu-driven APL program. The main menu has four selections: INPUT, SIMULATE, REPORT, and END (**Figure 5**). Each selection leads to a sub-menu. The INPUT menu allows the user to enter or change model parameters. The SIMULATE menu sets the conditions of simulation and starts the simulation. The REPORT menu allows the user to print inputs and outputs in various formats. The END menu can be used to save a compacted set of information for future use and to exit from the program.

• *INPUT*

The input menu has six selections (**Figure 6**). Each selection invokes a program for inputting model parameters. ARCHITECT, FAILRATES, and STRATEGY prompt the user to provide the parameters described above. TIMES asks for the total lifetime of systems to be simulated, and the intermediate time points for which the user requires outputs. NAME asks for a job name and other descriptive information to be printed on reports. All inputs are checked for validity as they are entered. CHECK does an overall consistency check on the data and either returns to the main menu or informs the user of inconsistencies.

• *SIMULATE*

The SIMULATE menu is shown in **Figure 7**. The first three selections allow the user to control the stopping rules for the simulation. These are discussed later in detail. SEED allows the user to control the random seed used for starting the simulation. GO begins the actual simulation of memory systems and returns control to SIMULATE when the stopping criteria are satisfied. The results of each simulation run (each invocation of GO) are automatically combined with all previous results since the last model change made using the INPUT menu. If this is not desired, the selection NEW will wipe out all previous results.

Stopping rules

The objective of FTMS is to estimate the reliability parameters defined above with reasonable accuracy at reasonable cost. The accuracy of the results is quantified in terms of confidence limits on the proportional error in any estimate. Let X represent any of the random variables defined above for some specific time point [e.g. $UE(t)$ at $t = 10\,000$ hours], and let $X_1, X_2, \cdots, X_n$ be sample values of X obtained by simulating n memory systems. The mean and variance of X are denoted m and v^2 respectively, and the sample mean and variance are given by

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{i=n} X_i,$$

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{i=n} (X_i - \bar{X})^2.$$

We wish to find a confidence interval of prescribed proportional accuracy p and prescribed coverage probability c for the unknown mean m. The variance v^2 is clearly finite and therefore, from the central limit theorem (e.g. [23], p. 213),

$$\lim_{n\to\infty} P\left[\left|\frac{\bar{X} - m}{m}\right| \le \frac{zv}{m\sqrt{n}}\right] = c,$$

where

$$c = \frac{1}{\sqrt{2\pi}}\int_{-z}^{z} \exp\left(\frac{-u^2}{2}\right)du.$$

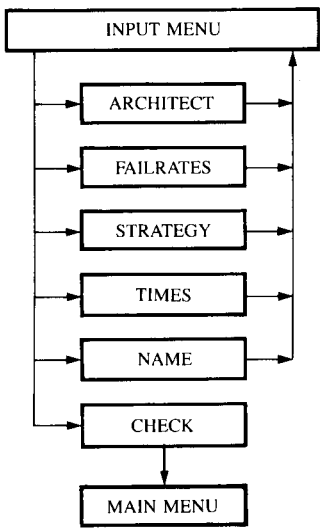

Figure 6 INPUT menu selections in FTMS.

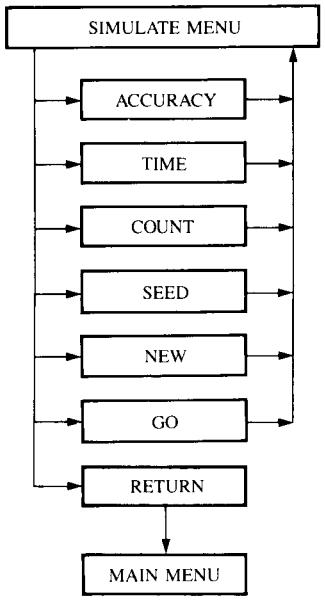

Figure 7 SIMULATE menu selections in FTMS.

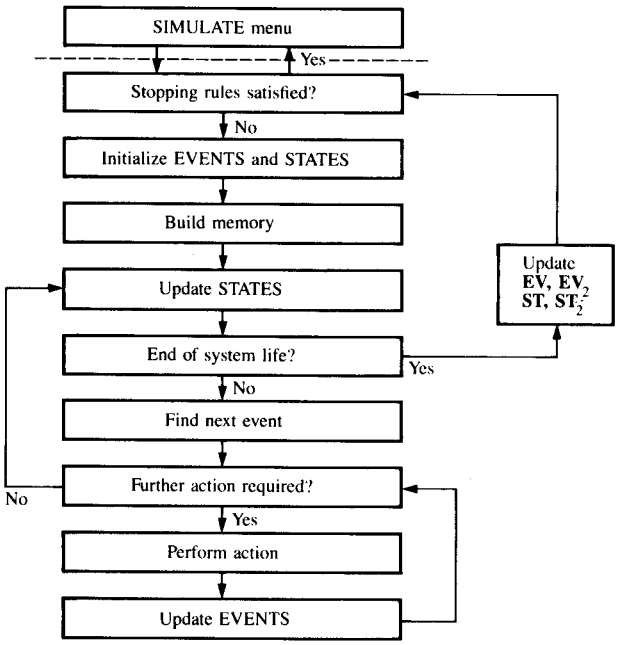

Figure 8 Flow of simulation program.

Moreover, it can be shown ([23], p. 357) that

$$\lim_{n\to\infty} \frac{S}{\bar{X}} = \frac{v}{m},$$

and therefore

$$\lim_{n\to\infty} P\left[\left|\frac{\bar{X} - m}{m}\right| \le \frac{zS}{\bar{X}\sqrt{n}}\right] = c.$$

Thus, for n sufficiently large, the proportional error in $\bar{X}$ (as an estimator of m) is less than $p = zS/\bar{X}\sqrt{n}$ with probability c.

Conversely, in order to achieve proportional accuracy p, one should choose a sample size equal to the smallest n which is greater than or equal to $(vz/mp)^2$.

Before running the simulation, however, the user does not generally know v/m and cannot solve for n. Therefore FTMS contains a sequential stopping rule which continues to simulate until each parameter has been estimated with the proportional accuracy and confidence level specified by the user. The rule is simply to stop sampling as soon as n is greater than or equal to $(Sz/\bar{X}p)^2$. This stopping rule has been studied by Nádas [20], who proved that it is asymptotically consistent, i.e.,

$$\lim_{p\to 0} P\left[\left|\frac{\bar{X} - m}{m}\right| \le p\right] = c,$$

and asymptotically efficient in the sense that the expected sample size approaches the fixed sample size which would be used if v/m were known in advance.

If the required accuracy is too tight, the program may run for an excessive time. The TIME selection on the SIMULATE menu can be used to put upper bounds on the CPU time and elapsed time of any run of GO. Finally, the COUNT selection in SIMULATE allows the user to set a maximum and minimum number of systems to simulate.

If some simulations have already been run for a given model, it is possible to estimate the relationship of time and accuracy to sample size. In this case FTMS will provide the user with an estimate of the running time, proportional accuracy, and number of simulations for the stopping rules he has chosen.

Simulation runs

The flow of simulation is shown in **Figure 8**. A memory is "built" at random, and then the life history of that memory is simulated by computing the effects of each failure and the results of each maintenance action. At the end of system life, the results of the current system are added to the cumulative results from previous systems simulated. Then the stopping rules are checked and the program either begins another system simulation or returns control to the main menu.

Build memory The physical state of the memory at any time is determined by the type and location of the hard failures on each card. This information is stored in a matrix called **FAIL** which contains one row corresponding to each hard failure. Each row is of the form (m, t, f, c, a, b, x, y) where m = failure mode, t = failure time, and (f, c, a, b, x, y) defines the location of the failure. The convention is that f is the location of the card containing the failure if the failure is confined to one card, or $f = 0$ if the failure spans all cards. The remaining parameters are defined similarly. For example (4, 2500, 9, 3, 6, 1, 0, 0) represents a chip-kill at 2500 hours on the chip in row six and column one of the third bit-plane on the ninth card in the memory. Failures on spare chips are stored in a separate matrix of the same form called **SPARE**.

The matrix **FAIL** could be generated sequentially by selecting a random time-to-next-failure after dealing with each simulated event. It is more convenient, however, to choose all random failure times, up to the specified end of system life, T_{MAX} hours, at the beginning. Therefore, at time zero **FAIL** contains all failures which will occur before T_{MAX} (card replacement is discussed later), and the actual state of the system at time t is that part of **FAIL** with times less than or equal to t.

The straightforward way to generate **FAIL** would be to simulate a random failure time for each component in the system, and then select only those times less than T_{MAX}. The problem with this approach is the astronomical number of components which would have to be simulated. For the example shown in Fig. 1 it would be necessary to select 18 million random failure times for cell failures alone.

Fortunately there is another way. FTMS uses a simple algorithm to generate random failure times in order of occurrence up to T_{MAX}. For a given failure mode with cumulative distribution function of time to failure, $F(t) = \text{Prob}[T \le t]$, we require the first r-order statistics $T_1 \le T_2 \le \cdots \le T_r$ from a sample of size n from $F(t)$, where n is the number of components which are subject to that failure mode, and r is the largest integer for which $T_r \le T_{MAX}$. If V denotes a random variable uniformly distributed on (0,1), it is well known that $F^{-1}(V)$ is distributed like T, and $F(T)$ is distributed like V. Therefore, if we can find uniform order statistics $\{V_i\}$, we can find the required $\{T_i\}$ from $T_i = F^{-1}(V_i)$. Furthermore, if we can find order statistics from any other distribution X, with distribution function $G(x)$, we can obtain the $\{V_i\}$ with the required distribution by setting $V_i = G(X_i)$. If $G(x)$ is chosen to be the exponential distribution with mean a, $G(x;a) = 1 - \exp(-x/a)$, the $\{X_i\}$ are easy to get. It can be shown (e.g., [12], p. 18) that the successive differences $D_i = X_i - X_{i-1}$ are independently distributed with distributions $G[x;a/(n+1-i)]$, respectively. Now the problem is reduced to finding r independent samples from r different exponential distributions. This is done by setting $D_i = -\ln(U_i)/(n+1-i)$, where $\{U_i\}$ are r independent uniform random variables. Substituting back through these equations, all of this reduces to

$$T_i = F^{-1}\left[1 - \prod_{j=1}^{j=i} U_j^{(1/n+1-j)}\right], \qquad i = 1, 2, \cdots, r.$$

FTMS uses this expression and the built-in APL random number generator to sequentially generate all failure times up to T_{MAX}.

Time of next event The time of the next event is the time of the next scheduled maintenance or the time of the next hard failure, whichever comes first. If the event is a hard failure, it is necessary to determine whether a UE has occurred. Some failure modes cause a UE independently of other failures, e.g. an entire card failure if a card contributes more than *NEC* bits per data word. For other failure modes, a UE occurs if any bit affected by the new failure lines up with a different bit in the same word which was affected by a previous failure. The convention for representing failure locations is quite convenient for determining whether two failures line up to cause a double-bit error in any data word. If we define the logic function

$$L(u,v) = (u = v) \cup (u = 0) \cup (v = 0),$$

it is easy to see that two failures represented by (f, c, a, b, x, y) and (f', c', a', b', x', y') line up to cause a double-bit error in at least one word if and only if the logical expression

$$L(x,x') \cap L(y,y') \cap L(a,a') \cap L(b,b') \cap \sim[L(f,f') \cap L(c,c')]$$

is true. When all double-bit errors have been located, it is even simpler to find the triple-bit errors: Three failures line up to cause a triple-bit error if and only if each pair of two causes a double-bit error.

Next action Maintenance actions are effected by making the appropriate changes in **FAIL** and **SPARE.** Card replacement is done by removing from **FAIL** and **SPARE** the rows corresponding to the cards to be removed, and adding new rows for the failures in the new cards. Address permutation is accomplished by simply applying a permutation to the numbers located in column 5 (the chip-row location) in **FAIL.** Spare switching involves switching the appropriate rows between **FAIL** and **SPARE.** Pages are deallocated by recording the page addresses in a separate matrix called **DEAL.** If the total number of pages exceeds the specified threshold, deallocation fails and another action must be taken to remove the UE. When a new UE occurs, it is compared to the pages in **DEAL** to verify that it is not in a word already deallocated.

Record keeping Two types of records are kept for each simulated lifetime. The matrix **EVENTS** contains one record for each event-action pair (e.g. UE-card replacement), including the type of event and action, the time of occurrence, and other pertinent information such as the number of cards replaced, the number of pages deallocated, etc. The matrix **STATES** contains one record for each change of state, including the time of change and the new state. The state is defined to be the number of words containing one or (if $NEC = 2$) two bad bits, and the number of pages deallocated. An event may cause several actions at the same time, but the memory state does not change between events. Therefore **EVENTS** is updated after each action, while **STATES** is updated only once for each event (Fig. 8).

After each system is simulated, the results are added to the cumulative results from previous systems. The records updated consist of *NS*, the cumulative number of systems simulated, and four matrices: **EV**, $\mathbf{EV}_2$, **ST**, and $\mathbf{ST}_2$. **EV** and $\mathbf{EV}_2$ contain one row for each time point for which output is required, and one column for each event-count type of parameter, i.e. UEs, CRs, and RAs. The (i,j)th elements of **EV** and $\mathbf{EV}_2$ contain

$$\sum_{k=1}^{NS} X_{ijk}, \qquad \sum_{k=1}^{NS} X_{ijk}^2,$$

respectively, where X_{ijk} equals the total count for the jth parameter, summed over the time interval from zero to the ith time point, for the kth system simulated. **ST** and $\mathbf{ST}_2$ contain similar information for the state type parameters B_1, B_2, and DP. This is the minimum information required to compute the estimates and confidence intervals for each parameter.

REPORT

In the REPORT menu the user can choose to print tables containing the estimated values of the defined parameters as

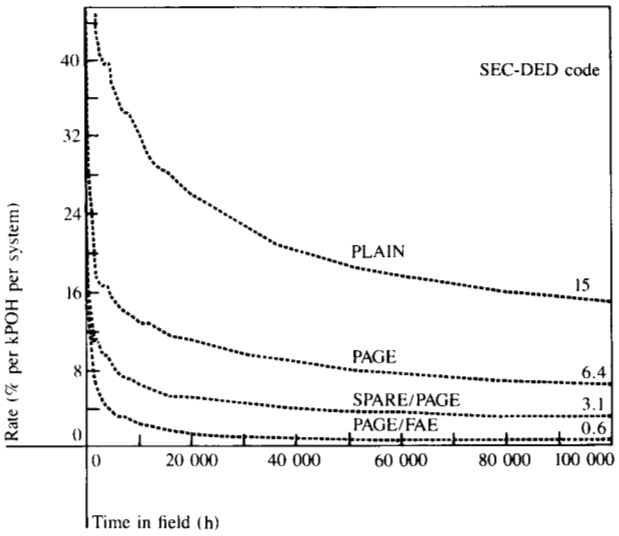

Figure 9 Average RA rates for 4-megabyte memory with SEC.

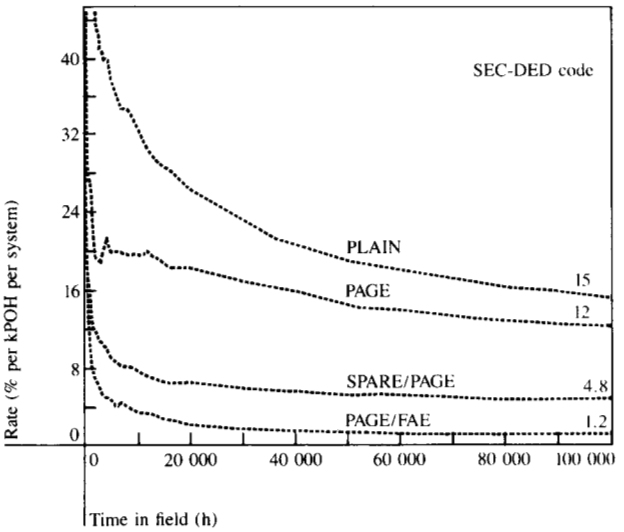

Figure 10 Average CR rates for 4-megabyte memory with SEC.

a function of time-in-field. The values can be calculated for each interval (T_i,T_{i+1}), or for the cumulative intervals $(0,T_i)$. For cumulative intervals the proportional accuracy confidence limits described above may be printed.

Soft errors The outputs include estimates and confidence limits for *SUE*, the soft UE rate, even though soft errors are not simulated. An *SUE* will occur whenever a soft error occurs in a word which contains *NEC* bad bits. Thus the *SUE* rate at any time is simply the soft error rate times the probability that a randomly chosen bit is a good bit in a word containing *NEC* bad bits. This is equal to

SOFT × (number of chips in the memory)
× (fraction of words containing *NEC* bad bits)
× (fraction of good bits in words with *NEC* bad bits).

The number of chips is *FCAB*, the total number of words is *ABXY*, and the fraction of bits good is $(FC - NEC)/FC$. Therefore we can estimate $SUE(t)$ as a constant $[SOFT \times (FC - NEC)/XY]$ times the estimated value of $B_i(t)$, where $i = NEC$. This approach gives a considerable savings in running time because soft error rates can be much higher than hard failure rates. Marston [22] reports a four-to-one ratio of soft to hard failures for a 16K dynamic RAM, based on a soft-error rate of 0.1 percent per 1000 hours, and May and Woods [2] reported orders of magnitude higher soft-error rates.

More importantly, the accuracy of the estimate is actually better because we are computing the exact (conditional on the simulated hard-failure state) *SUE* rate at each point in time instead of estimating that rate by throwing soft errors at the memory. One of the basic principles of Monte Carlo analysis is [19], "If, at any point of a Monte Carlo calculation, we can replace an estimate by an exact value, we shall reduce the sampling error in the final result." Let X denote the estimate of $SUE(t)$ for some t based on the above approach, and denote the mean and variance of X by m and v^2 respectively. [X is unbiased and therefore $m = SUE(t)$.] Let Y denote the estimate for $SUE(t)$ which would result from actually simulating soft errors. It is clear that the conditional distribution of Y given X is Poisson with mean X, and it follows that the mean and variance of Y are m and $(v^2 + m)$, respectively. Both X and Y are unbiased estimates of $SUE(t)$, but the variance of Y is greater by a factor of $1 + m/v^2$.

• *END*

The END selection on the main menu allows the user to exit from FTMS. At this point the APL workspace contains the inputs as well as the simulation results for the model most recently run. The user can save these results and return later to add more runs simply by invoking FTMS again. In order to allow the user to save more than one model without wasting too much disk space, END provides the option of expunging everything from the workspace except the model inputs and simulation results. The user can save this reduced workspace and later copy it into a full FTMS workspace to continue running the same model.

4. Design tradeoffs

FTMS can be used to estimate, to any desired degree of accuracy, the defined reliability parameters for any memory system which fits the model given in Section 2. This gives us the ability to predict the effect of design options on system reliability. The following example illustrates the use of FTMS to compare the use of page deallocation, chip sparing, and FAE as well as single- or double-error correction for a particular system.

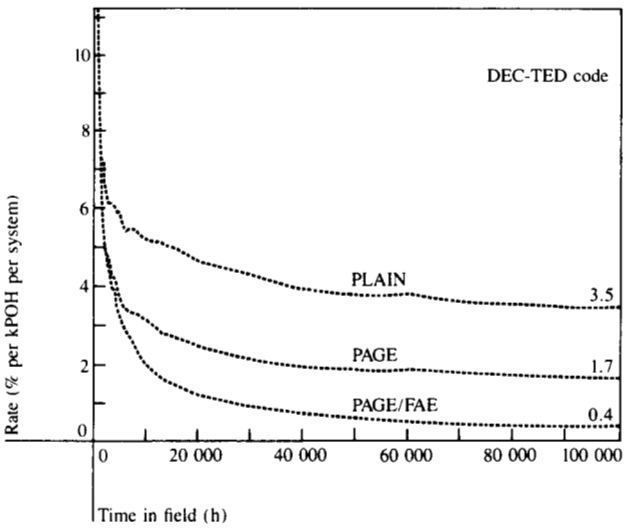

Figure 11 Average RA rates for 4-megabyte memory with DEC.

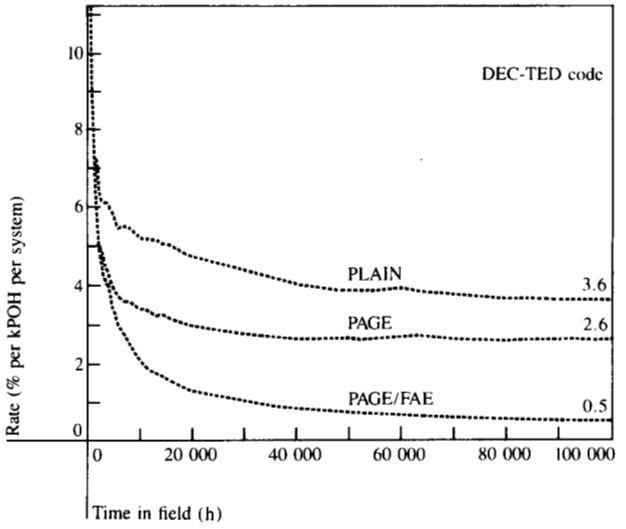

Figure 12 Average CR rates for 4-megabyte memory with DEC.

We consider a sample memory system of 4 megabytes. The system consists of a number of cards, each containing a 32 × 4 array of 16K-bit chips. There are 128 bit-lines and 128 word-lines in a chip. The memory is organized as 1-bit-per-chip with respect to the ECC. At the card level, 4-bits-per-card is assumed. We consider both a (72,64) SEC-DED code and an (80,64) DEC-TED code. Thus, the system consists of 18 cards for the SEC-DED code, and 20 cards for the DEC-TED code.

The failure rate of the memory chip is assumed to follow the step function shown in Fig. 4. The average failure rate over 100 kPOH is 0.02 percent per kPOH. The piece part failure distribution within the chip is 35 percent for cells, 12 percent for word-lines, 18 percent for bit-lines, and 35 percent for chip-kills (same as [11, 21]). In addition, the support logic of a card is assumed to fail at the same rate as that of a chip.

We assume that a service maintenance is scheduled at 200 power-on hours. The maintenance is to clean up the cards so that each card contains no more than two cell fails at the scheduled time. If a card has to be replaced in order to fix a UE, the rule is to replace the card that participates in the UE and has the largest number of defective cells.

A memory page is assumed to contain 2 kilobytes. Consider the memory as a chip array of 32 rows. A page of data resides in a single chip-row. It occupies 2 word-lines and 128 bit-lines within a chip. If page deallocation is used to fix a UE, the threshold of pages that can be deallocated is 32.

Two other features for the system considered are FAE and chip spare. For FAE, 5 bits of address permutation is assumed. The 5-category fault map is also assumed [4]. For chip spare, it is assumed that each card has one spare chip. Whenever there is a chip-kill, the faulty chip is replaced by the spare chip on the card that contains the faulty chip. Unless specifically stated, the strategies described henceforth do not involve chip spare.

For the system using a (72,64) SEC-DED code, we have simulated the following strategies in fixing UEs:

PLAIN: Simply replace a card to fix a UE.

PAGE: To fix a UE, the memory page that contains the UE is deallocated. If the number of pages deallocated exceeds the threshold (32 pages), a card is replaced.

SPARE/PAGE: The spare chip is used to fix a chip-kill on the card, and page deallocation is used to fix a UE.

PAGE/FAE: To fix a UE, the memory page that contains the UE is deallocated. If the number of pages deallocated exceeds the threshold, FAE is performed. If FAE fails to fix the UE, a card is replaced.

The results of the simulations are shown in **Figures 9** and **10** in terms of the rates of repair action and card replacement. The results clearly indicate that page deallocation, FAE, and chip spare can be used to reduce the frequency of repair as well as the number of cards replaced.

For the system using an (80,64) DEC-TED code, we have simulated the strategies of PLAIN, PAGE, and PAGE/FAE. The rates of repair action and card replacement obtained from the simulations are shown in **Figures 11** and **12.** Again, the results show that page deallocation and FAE can be used to increase reliability and decrease maintenance cost.

To show the effectiveness of double-error correction over single-error correction, the repair action rates for the SEC-DED code and the DEC-TED code with PLAIN strategy are plotted in **Figure 13.** At 40 kPOH, there is a slightly greater

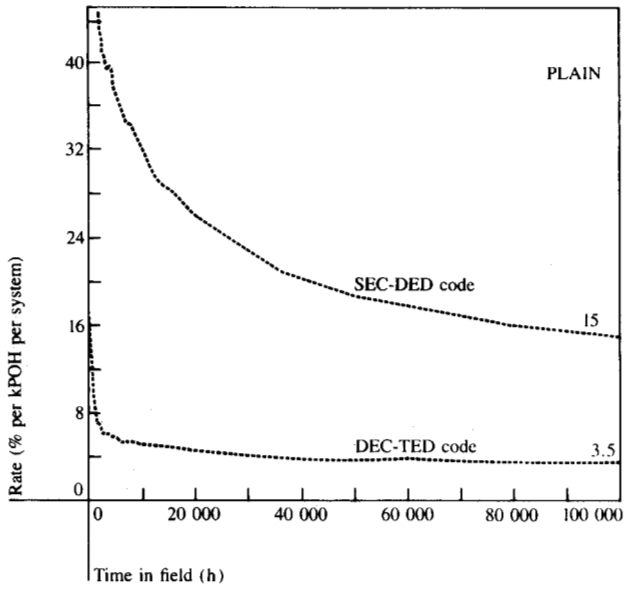

Figure 13 Average RA rates for 4-megabyte memory with SEC and DEC.

than 5 times improvement of double-error correction over single-error correction in the rate of repair action. The improvement factor is even higher at the early life of the memory system. Similar conclusion can also be made on the rate of card replacement.

5. Conclusion

FTMS was written to provide memory designers with a flexible tool with which to evaluate the various techniques for fault-tolerance which can be built into computer memory systems. A wide variety of design options, including options for system architecture, failure modes and rates, and maintenance strategy, can be evaluated simultaneously. The output of FTMS gives the frequency of uncorrectable errors of both types (hard and soft), and also the amount of degradation due to page deallocation and the need to correct bad bits, and the service cost parameters of repair actions and cards replaced. An optimal sequential stopping rule is used to estimate all of these parameters with prescribed accuracy and confidence level, without any prior knowledge of the variance of the estimates.

This program has been successfully applied to evaluate alternative design proposals over the past several years. During that time it has evolved by adding to the list of options which can be evaluated. As future options are proposed, FTMS will be modified to give a quick and accurate prediction of the impact of such proposals on memory reliability.

6. References

1. C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Computer Memory Applications: A State-of-the-Art Review," *IBM J. Res. Develop.* **28,** 124–134 (1984, this issue).
2. T. C. May and M. H. Woods, "A New Physical Mechanism for Soft Errors in Dynamic Memories," *Proceedings, 1978 International Reliability Physics Symposium*, San Diego, April 18–20, 1978, pp. 33–40.
3. D. C. Bossen and M. Y. Hsiao, "A System Solution to the Memory Soft Error Problem," *IBM J. Res. Develop.* **24,** 390–397 (May 1980).
4. 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).
5. D. T. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.
6. L. Levine and W. Meyers, "Semiconductor Memory Reliability with Error Detecting and Correcting Codes," *Computer* **9,** 43–50 (October 1976).
7. S. Q. Wang and K. Lovelace, "Improvement of Memory Reliability by Single-Bit-Error Correction," *Proceedings, 14th IEEE Computer Society International Conference*, February–March 1977, pp. 175–178.
8. G. W. Cox and B. D. Carroll, "Reliability Modeling and Analysis of Fault-Tolerant Memories," *IEEE Trans. Reliability* **R-27,** 49–54 (April 1978).
9. A. V. Ferris-Prabhu, "Improving Memory Reliability through Error Correction," *Computer Design* **18,** 137–144 (July 1979).
10. S. Elkind and D. Siewiorek, "Reliability and Performance of Error-Correcting Memory and Register Arrays," *IEEE Trans. Computers* **C-29,** 920–927 (October 1980).
11. W. F. Mikhail, R. W. Bartoldus, and R. A. Rutledge, "The Reliability of Memory with Single-Error Correction," *IEEE Trans. Computers* **C-31,** 560–564 (June 1982).
12. W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. II, John Wiley & Sons, Inc., New York, 1966, Ch. XI.
13. R. A. Rutledge, "The Reliability of Memory Subject to Hard and Soft Failures," *Digest FTCS-10, 10th Annual International Conference on Fault-Tolerant Computing*, Kyoto, Japan, October 1980, pp. 193–198.
14. S. K. Kwon, "Reliability Simulation of Memory Systems with SECC," *Proceedings, Annual Reliability and Maintenance Symposium*, San Francisco, January 1972, pp. 116–126.
15. S. K. Kwon and H. E. Harvey, "A Simulation Approach to the Reliability Analysis of Main Storage Systems," *12th Annual Simulation Symposium*, March 1979, pp. 257–272.
16. M. R. Libson and H. E. Harvey, "A General-Purpose Memory Reliability Simulator," *IBM J. Res. Develop.* **28,** 196–206 (1984, this issue).
17. R. A. Rutledge, "Monte Carlo Generation of Order Statistics," *Technical Report TR22.1279*, IBM Poughkeepsie, June 1971.
18. D. Lurie and H. O. Hartley, "Machine-Generation of Order Statistics for Monte Carlo Computations," *The American Statistician* **26,** 26–27 (February 1972).
19. J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen & Co. Ltd., London, 1964, Ch. 5.
20. A. Nádas, "An Extension of a Theorem of Chow and Robbins on Sequential Confidence Intervals for the Mean," *Ann. Math. Stat.* **40,** 667–671 (April 1969).
21. B. Pascoe, "2107A/2107B N-Channel Silicon Gate MOS 4K RAMs," *Reliability Report RR-7*, Intel Corporation, Santa Clara, CA, September 1975.
22. D. Marston, "Memory System Reliability with ECC," *Application Note AP-73*, Intel Corporation, Santa Clara, CA, August 1980.
23. H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, 1946.

Received July 1, 1983; revised September 30, 1983

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.

Robert A. Rutledge *IBM Data Systems Division, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Rutledge is a senior engineer in the laboratory Engineering Analysis Department in Poughkeepsie. His current technical interests include reliability evaluation of fault-tolerant systems and statistical analysis of component and system reliability data. He joined IBM in 1968 in the Components Division in Poughkeepsie. From 1968 to 1973, he worked in product assurance on statistical techniques for the prediction and measurement of component reliability. Dr. Rutledge received the B.S. in mathematics from Iona College, New Rochelle, New York, in 1963 and the Ph.D. in mathematical statistics from Columbia University, New York, in 1970. Dr. Rutledge has received an IBM Outstanding Innovation Award for work on error-correcting codes and an IBM Outstanding Technical Achievement Award for work on reliability data analysis. He is a member of the American Statistical Association, the Institute of Electrical and Electronics Engineers, and the Institute of Mathematical Statistics.