Simulation of Reconfigurable Memory Core Yield*

Marco Ottavi
Dept. Electronic Eng.
University of Rome "Tor Vergata"
Rome, Italy
ottavi@ing.uniroma2.it

Xiaopeng Wang
Dept. Elec. & Comp. Eng.
Northeastern University
Boston, MA 02115
xiawang@ece.neu.edu

Fred J. Meyer
Dept. Elec. & Comp. Eng.
Wichita State University
Wichita, KS 67260
fred.meyer@wichita.edu

Fabrizio Lombardi
Dept. Elec. & Comp. Eng.
Northeastern University
Boston, MA 02115
lombardi@ece.neu.edu

ABSTRACT

We give a Markov chain model of the yield of an embedded memory core. The model allows easy inclusion of the effect of possible defects elsewhere on the chip that includes the embedded memory. We propose a reconfiguration algorithm for the case of both spare rows and columns that is simple enough that it could serve as built-in self-repair on the chip. Compared to an optimal configuration algorithm, there is no visible difference in the yield. We use parameters from an IBM embedded SRAM process to illustrate the yield calculation. We study the effect of different spare allocations. We conclude that as long as there is at least one spare of each type, the spares do not need to be balanced, once the yield impact of being part of a system-on-a-chip has been taken into account.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Miscellaneous; G.3 [**Probability and Statistics**]: Miscellaneous

General Terms

Reliability

Keywords

yield, manufacturability, defect tolerance, Markov chain

1. INTRODUCTION

*Research reported supported in part by the International Test Conference endowment at Northeastern University.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

The occurrence of a defect on a critical spot of a chip layout can cause the failure of the whole chip if no redundancy has been introduced in the design or if the defect's effects can't be repaired with the provided redundancy. The manufacture of defect-tolerant VLSI circuits requires an accurate estimation of the yield because the redundancy introduced in the design depends on it.

Memories are laid out as two-dimensional arrays with logic to address the rows and columns. The addressing logic can be modified to allow additional (spare) rows and/or columns to be addressed. The area overhead for this is not high, because the number of rows/columns grows as the square root of the number of cells. As a result, memories often have redundancy; however, it is inconvenient to introduce spare columns, so using spare rows only is desired if feasible.

We evaluate the yield of redundant RAM arrays. A specific embedded SRAM is used for illustration. In particular, in Section 2 the global yield model based on a negative binomial distribution is espoused. To evaluate the yield, we need to consider a particular repair algorithm; we propose a new one in Section 3 for memory arrays with both spare rows and spare columns. In Section 4 we report the critical area and defect density models that provide λ_0, i.e., the expected number of defects per chip, used for the yield results reported in Section 5.

2. GLOBAL YIELD MODEL

The evaluation of the effect of defect density, D, on a chip is related to the evaluation of the defect distribution and the critical area, A, in which they occur. The product $AD = \lambda_0$ provides the mean number of defects expected on a chip. The value of λ_0 represents the mean value of the probability distribution of the defects present on the chip.

However, different wafers have different qualities, so each wafer will have its own value, λ, instead of λ_0. Usually, in what is called a large-area clustering model, if the value of λ is known for a wafer, then the chips on it are considered to be subject to defects that are Poisson distributed [3, 8]

$$\Pr\{X = k\} = \frac{\lambda^k}{k!} e^{-\lambda} \tag{1}$$

Here, k is the number of defects on the chip. The mean and variance of Equation 1 are both equal to λ. If the chip has no redundancy, the yield is obviously given by Equation 1 computed at $k = 0$

$$Y = \Pr\{X = 0\} = e^{-\lambda} \tag{2}$$

Namely, yield is defined as the probability of having 0 defects on a chip.

To determine λ for a wafer, most common is to draw it from a Gamma distribution. The yield can then be determined by taking the yield given λ and integrating out λ. Numerical integration is the simplest. This operation is

$$\begin{aligned} Y(\lambda_0, \alpha) &= \int_0^\infty Y(\lambda) \cdot g(\lambda|\lambda_0, \alpha) d\lambda \\ &\approx \sum_{n=0}^{\text{large}} Y(n\Delta\lambda) \cdot g(n\Delta\lambda|\lambda_0, \alpha)\Delta\lambda \end{aligned} \tag{3}$$

where $g(\lambda|\lambda_0, \alpha)$ is the Gamma probability distribution function. The Gamma probability distribution function is defined as [5]

$$g(\lambda|\lambda_0, \alpha) = \frac{\alpha^\alpha}{\lambda_0^\alpha \cdot \Gamma(\alpha)} \lambda^{\alpha-1} e^{-\alpha\frac{\lambda}{\lambda_0}} \tag{4}$$

Alternatively, the probability distribution function for the number of defects on a chip can be taken directly from the mixture of the Gamma distribution function and the Poisson distribution function, yielding the negative binomial distribution function

$$\Pr\{X = k\} = \frac{1}{k \cdot \text{Beta}(k, \alpha)} \cdot \frac{\left(\frac{\lambda_0}{\alpha}\right)^k}{\left(1 + \frac{\lambda_0}{\alpha}\right)^{\alpha+k}} \tag{5}$$

The mean and the variance of this distribution are $E(X) = \lambda_0$ and $\text{Var}(X) = \lambda_0\left(1 + \frac{\lambda_0}{\alpha}\right)$, respectively. Therefore, yield for an irredundant chip is, from Equation 5 calculated at $k = 0$,

$$Y = \Pr\{X = 0\} = \left(1 + \frac{\lambda_0}{\alpha}\right)^{-\alpha} \tag{6}$$

called the negative binomial yield model. In Equation 6, α represents the clustering effect of the mean defect density λ_0 due to having a common value for λ on a wafer. A usual value of α adopted in industry is around 2.

3. PROPOSED REPAIR METHOD

We will analyze yield for memory arrays with both spare rows and spare columns. The problem of determining whether such a memory can be repaired is NP-complete. However an exhaustive solution is computationally feasible when the number of spares is small. We compared the repair success rates of our proposed algorithm against an optimal repair algorithm and found no visible difference in plotted yield.

Our algorithm is simple enough to be implemented in hardware, so it could be used for built-in self-repair. The algorithm is based on the premise that as long as a line (row or column) in the memory array has at most one unrepaired defective memory cell present, we defer the decision on how to repair it. As soon as a line has multiple defects, we mark it for repair. For this reason, we call the algorithm Single Deferral.

To describe the state of the algorithm after 0 or more defects have been processed, we track 3 variables:

- i number of already repaired rows,
- j number of already repaired columns,
- k number of not already repaired isolated/independent defective cells (or flexibility factor).

The defective cells that are independent of all other defective cells/lines (have no line in common with other unrepaired defects) can be repaired flexibly, using either a spare row or a spare column, as available.

When a defect is reported at a single cell, if $k = 0$, then k is merely set to 1 and no repair decision is made yet, but the location is remembered in a list of deferred defects. When another single cell defect is reported, if it is independent of the defects on the deferred list, then it is added to the list and k is incremented. If it is on the same line as a previous saved value, a spare line is allocated to repair them, and the prior defect is removed from the deferred list (k decreases).

Similarly, if a line defect is reported in common with a cell defect on the deferred list, then the cell defect is removed from the list. Each time a line defect is repaired, or each time a spare line is allocated to repair multiple cell defects, i or j is incremented.

Some defects occur affecting pairs of memory cells. If a memory array uses grouping (spare lines are grouped and only an entire group can substitute for a defective group), then these paired defects will have less impact, they will tend to behave more often like single cell defects. For our illustrative memory, we let groups be of size 1.

When a paired cell defect (horizontal, HP, or vertical, VP) is reported, a repair is made according to the following:

- If the fault is isolated (no overlap with previous ones) the row (for HP) or the column (for VP) is allocated and k is decremented if the row/column already contained a single unrepaired cell defect.
- If the fault is (horizontally/vertically) overlapping a previous single cell defect the repair is made as though it were a single cell defect; if there is also overlap in the other direction (vertical/horizontal), the choice of spare row/column usage is random.

When a spare is needed and there are none left unallocated, then the memory reconfiguration fails. If there are s_r spare rows and s_c spare columns, then the state (i, j, k) must satisfy $i \le s_r$, $j \le s_c$, and $i + j + k \le s_r + s_c$. Otherwise, we should have transitioned to the failed state instead. These constraints define the boundary conditions for the state diagram. However, a typical state not near the boundary has the outgoing edges depicted in Figure 1.

The intensities of the state transitions in Figure 1 are given in Table 1, where n_r and n_c are the number of rows and columns in the memory array. The intensities do not add up to λ_0, because edges from a state to itself do not need to be included in the CTMC.

4. CRITICAL AREA

For a wafer with quality dictated by λ, drawn from a Gamma distribution, the Poisson number of defects must still be distributed across the wafer. This is done by randomly assigning defects to locations according to the susceptibility of the various devices on the wafer. The concept of critical area is used as a metric for susceptibility. Another

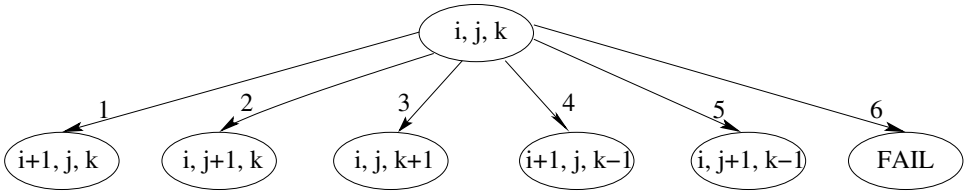

Figure 1: Generic node transitions for the Single Deferral algorithm.

Table 1: Failure Intensities for Single Deferral Algorithm

Defect Type	Edge(s)	Value(s)	Comment
Chip Kill	6	λ_{ck}	In every state
Row	1	$(n_r - i - k)\lambda_{row}$	No other sc on the row
	4	$k\lambda_{row}$	Unrepaired sc on the row
	6	$(n_r - s_r)\lambda_{row}$	if $i = s_r$ boundary
Column	2	$(n_c - j - k)\lambda_{col}$	No other sc on the column
	5	$k\lambda_{col}$	Unrepaired sc on the column
	6	$(n_c - s_c)\lambda_{col}$	if $j = s_c$ boundary
Single Cell	3	$(n_r - i - k)(n_c - j - k)\lambda_{sc}$	Isolated sc
	4	$[k(n_r - i - 1) - k(k-1)/2]\lambda_{sc}$	Unrepaired sc on the row
	5	$[k(n_c - j - 1) - k(k-1)/2]\lambda_{sc}$	Unrepaired sc on the column
	2	$[(n_r - i)(n_c - j) - k]\lambda_{sc}$	if $i = s_r$ boundary
	1	$[(n_r - i)(n_c - j) - k]\lambda_{sc}$	if $j = s_c$ boundary
	6	$(n_r - i - k)(n_c - j - k)\lambda_{sc}$	if $i + j + k = s_r + s_c$ boundary
Horizontal Pair	1	$(n_r - i - 2k)(n_c - 2j - 2k)\lambda_{hp}$	Totally isolated
	2	$2k(n_r - i - 2k)\lambda_{hp}$	Unrepaired sc on the column
	3	$2j(n_r - k - i)\lambda_{hp}$	Overlap with repaired column like an isolated sc
	4	$(2k + k(n_c - 2j - 2k) + k2j)\lambda_{hp}$	$2k$=one overlapping $k(n_c - 2j - 2k)$=no overlap $k2j$=overlap with repaired col
Vertical Pair	1	$2k(n_c - j - 2k)\lambda_{vp}$	Unrepaired sc on the row
	2	$(n_c - j - 2k)(n_r - 2i - 2k)\lambda_{vp}$	Totally isolated
	3	$2i(n_c - k - j)\lambda_{vp}$	Overlap with repaired row like an isolated sc fault
	5	$(2k + k(n_r - 2i - 2k) + k2i)\lambda_{vp}$	$2k$=one overlapping $k(n_r - 2i - 2k)$=no overlap $k2i$=overlap with repaired row

view of critical area is that it is (or is proportional to) the expected number of defects. When the Poisson process is applied with the location dictated by the critical area distribution, we have a space-varying Poisson process. To account for this space-varying process, the only thing we need is to fill in the details about values for the intensities: λ_{ck}, λ_{row}, etc. Since a memory array is quite regular, we assume each row loss defect (etc.) is equally likely to be any row (etc.).

We used defect rates from an IBM embedded SRAM process. To not be too revealing, we only give approximations of these defect rates and we scale them. Table 2 breaks down the critical area, λ_0, of a half megabit memory. As SRAM size increases, the expected number of defects of each type increases in proportion, except for the chip kill defect, which increases only gradually. Three other rare types of defects found to exist for this embedded SRAM are not included.

The λ^* are the observed defects. Generally, there are actually more defects of each type due to masking by other defects. For instance, a chip kill defect might involve a short between power and ground. Due to the short, whatever cell and other defects were present are not detected.

Table 2: Defect Type Distribution

Defect	Variable	Relative Frequency
Chip Kill	λ^*_{ck}	$0.05 \cdot \lambda_0$
Single Cell	λ^*_{sc}	$0.45 \cdot \lambda_0$
Horizontal Pair	λ^*_{hp}	$0.1 \cdot \lambda_0$
Vertical Pair	λ^*_{vp}	$0.1 \cdot \lambda_0$
Single Row	λ^*_{row}	$0.15 \cdot \lambda_0$
Single Column	λ^*_{col}	$0.15 \cdot \lambda_0$

When the grid is large, the extent of masking is not much, except for the impact of chip kill defects. Even so, we used the approximate equations below to obtain the genuine defect intensities from the observed ones. We use λ_{defect} for the genuine intensity and λ^*_{defect} for the observed rate.

In our case, values of λ and λ^* are close, and the values in Table 2 are approximations anyway, so we use the same approximations for λ as are given in Table 2 in our computations.

$$\lambda^*_{ck} \approx 1 - e^{-\lambda_{ck}} \quad (7)$$

$$\lambda^*_{row} \approx n_r(1-\lambda^*_{ck})[1-e^{-\lambda_{row}}] \quad (8)$$

$$\lambda^*_{col} \approx n_r(1-\lambda^*_{ck})[1-e^{-\lambda_{col}}] \quad (9)$$

$$\lambda^*_{hp} \approx n_r n_c(1-\lambda^*_{ck})e^{-\lambda_{row}}e^{-2\lambda_{col}}[1-e^{-\lambda_{hp}}] \quad (10)$$

$$\lambda^*_{vp} \approx n_r n_c(1-\lambda^*_{ck})e^{-2\lambda_{row}}e^{-\lambda_{col}}[1-e^{-\lambda_{vp}}] \quad (11)$$

$$\lambda^*_{sc} \approx n_r n_c(1-\lambda^*_{ck})e^{-\lambda_{row}}e^{-\lambda_{col}} \cdot e^{-2\lambda_{hp}}e^{-2\lambda_{vp}}[1-e^{-\lambda_{sc}}] \quad (12)$$

5. YIELD RESULTS

Whether the memory array is successfully repaired depends on the final state, (i, j, k), reached. As long as it is not a failed state, reconfiguration is successful. We can introduce

$P(t)$: the probability state vector at time t

which tells us the probability of being in each state at time t. $P(0)$ is a unit vector with probability 1 of being in state $(0, 0, 0)$. The amount of time to allow to pass has to be gauged so that the expected number of defects in the space-varying Poisson process is the desired amount. In our case, we want $P(\lambda)$. Moreover, we want this for all values of λ, because λ will be drawn from a Gamma distribution with mean λ_0 and clustering parameter α. However, accurate results do not require us to consider large values for λ.

This technique of determining $P(\lambda)$, or at least determining it for a set of λ values, and integrating out λ with the Gamma distribution relies on using Markov chains such as the one depicted in Figure 1. These Markov chains assume the defects "arrive" in a random order, so it is most relevant for repair algorithms that process one defect at a time and get their defects in random order (or are not harmed by ordering). This is the same technique as that used in [4]. Contrast this with the methods of [2, 6], which rely, for example, on fault trees.

Finite state Markov modeling can be made both in discrete time and continuous time; in the first case we deal with Discrete Time Markov Chains (DTMC) in the second case we have Continuous Time Markov Chains (CTMC). Our CTMC models have an embedded Markov chain [5].

For these yield results we estimated the probability state vector via numerical integration, as so

$$P(k\Delta) \approx P(0)(\mathbf{I} - \mathbf{A}\Delta)^k \quad (13)$$

where $\mathbf{A}$ is the state transition matrix [5] corresponding to the transition rates (intensities) in the Markov model. This is the numerical approximation to

$$P(t) = P(0)e^{-\mathbf{A}t} \quad (14)$$

Many reliability evaluation tools have been developed to deal with the numerical solution of this equation, such as [1]. Our results were obtained using Matlab.

Our simulated memory array has 1024 rows, 512 columns, and 4 spares. We vary how the spares are allocated. The clustering parameter, α, is set to 2. The rates of individual defect types is as in Table 2.

We are interested in what happens when spare columns are introduced, because it is architecturally inconvenient to have spare columns in a memory. This is why memories that use error control codes [7] only use spare rows, even though they would have slightly superior reconfigurability if they used spare columns instead.

Figure 2 shows the probability of successfully reconfiguring a memory array when exposed to Poisson defects. The horizontal axis gives the expected number of defects, λ. The plot lines show reconfigurability with different numbers of spares: "2+2" means 2 spare rows and 2 spare columns, "0+4" means 4 spare rows and no spare columns. We can see important increases in yield by balancing the spares equally between rows and columns.

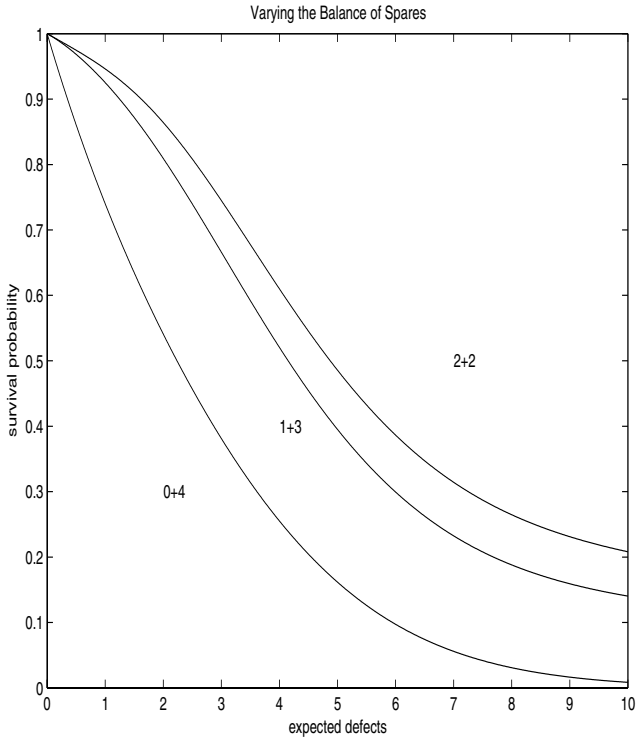

Figure 2: Reconfigurability with different spare arrangements.

A memory is likely to be embedded, so its effect on yield is only valid in the context of the entire chip within which it resides. We need to assign a critical area to the remainder of the chip. That critical area is likely to be substantially more than that of a SRAM memory, although a large DRAM embedded memory might have comparable critical area to the rest of a system-on-a-chip. We set the critical area of the remainder of the chip to be $1.00 * \lambda_0$ and note the results. This is adequate to illustrate our point, and the result would be stronger if the critical area were higher. It is easy to add this new critical area to our model by adding it to λ_{ck}.

Figure 3 shows the probability, as a function of λ, of whole chip survival—i.e., the memory array is successfully repaired and no defects in the remainder of the chip. We note now that the introduction of the first spare column (in exchange for a spare row) has significant benefit, but the introduction of subsequent columns is harder to justify.

While there is a clear difference between Figures 2 and 3, we cannot be sure that this difference will translate to differences in yield. We need to integrate out λ. The result is in Figure 4. This time the horizontal axis variable is λ_0.

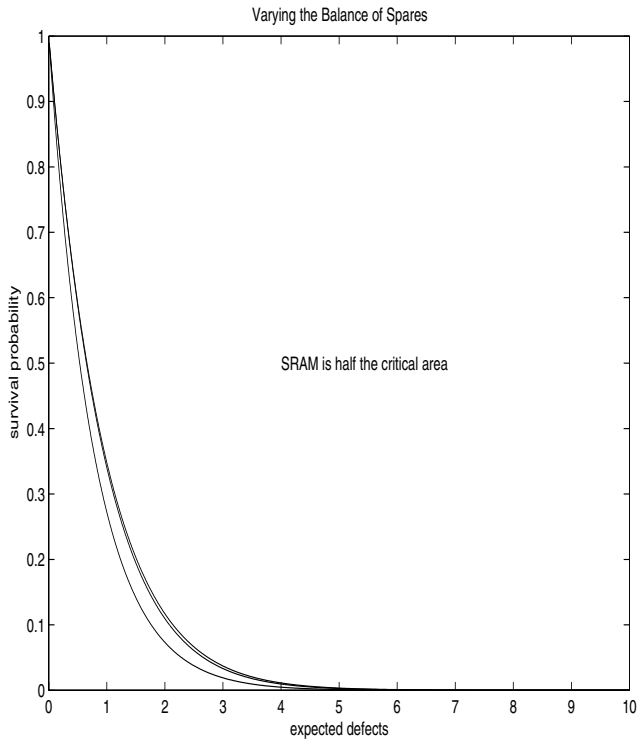

Figure 3: Reconfigurability and survival of entire chip.

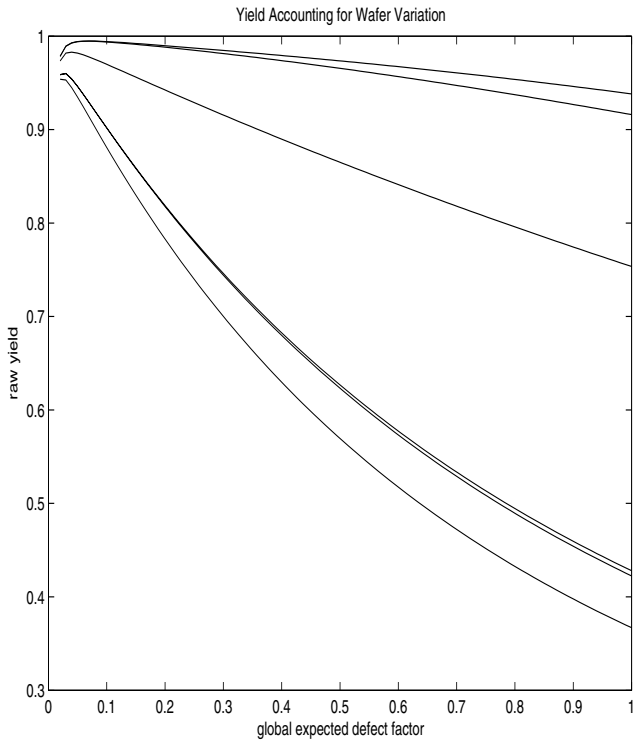

Figure 4: Yield after integration with Gamma distribution.

The plot has an anomaly with low λ_0; this is due to the granularity of the numerical integration. Additional calculation would need to be done if we were interested in the part of the curves where λ_0 is very low.

The lower set of three plot lines is for the different spare arrangements for the entire chip; the upper set is for the embedded memory only. We note that the observation still holds that, after taking the entire chip into account, we want at least 1 spare column, but additional ones are of dubious merit.

6. CONCLUSION

The manufacturing yield of an embedded memory can be predicted by modeling defects as arriving over time. As defects arrive, the repair algorithm processes them. The decision diagram for the repair algorithm can be converted to a continuous time Markov chain. The solution of this Markov chain is the reconfigurability for a given defect density.

We gave a particular spare allocation algorithm, Single Deferral, that defers decisions on how to repair single isolated cell defects until a second defect aligned with it occurs. This algorithm is near optimal and is simple enough to be implemented in hardware.

The Markov models presume that defects arrive in random order. If they do not, it could be possible to take advantage of this in the repair algorithm. For example, if built-in self-test sought to detect the line defects prior to the cell defects, then the line defects would all appear to "arrive" earlier than the cell defects. This slightly enhances reconfigurability, because when the less severe defects are processed, it is already known how the more severe defects needed to be repaired.

7. REFERENCES

[1] R. W. Butler. The SURE approach to reliability analysis. *IEEE Transactions on Reliability,* 41(2):210–218, June 1992.

[2] Coudert and Madre. Fault tree analysis: 10^{20} prime implicants and beyond. *IEEE Reliability and Maintainability Symposium (RAMS),* pp. 240–245, 1993.

[3] I. Koren and Z. Koren. Defect tolerant VLSI circuits: Techniques and yield analysis. *Proceedings of the IEEE,* 86:1817–1836, September 1998.

[4] F. J. Meyer and D. K. Pradhan. Modeling defect spatial distribution. *IEEE Transactions on Computers,* 38(4):538–546, April 1989.

[5] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes,* McGraw-Hill: New York, 2002.

[6] W. Shi and K. Fuchs. Probabilistic analysis and algorithms for reconfiguration of memory arrays. *IEEE Transactions on Computer-Aided Design,* 11(9):1153–1160, September 1992.

[7] C. H. Stapper and H.-S. Lee. Synergistic fault-tolerance for memory chips. *IEEE Transactions on Computers,* 41(9):1078–1087, September 1992.

[8] 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(3):398–409, 1980.