Φ_1^f to reach cell h and simplifying we obtain the following.

$$(f-s)\nu+(h-1)=(i-1)\ \delta\ \cdots\ (1)$$

Now Ψ_1^g is stored in cell $1 + (g - 1)\delta$, and hence $h = 1 + (g - 1)\delta$. Substituting $1 + (g - 1)\delta$ for h in (1) we obtain $(f - s)\nu = (i - g)\delta$. But by design, $(f - s)\nu \neq (i - g)\delta$ unless $f = s$ and $i = g$. Hence, $\Phi_1^f = \Phi_1^s$ and $\Psi_1^g = \Psi_1^i$ and $h = 1 + (i - 1)\delta$.

Now Ψ_1^i is stored in location $(i - 1) \bmod s$ in the control RAM of cell $1 + (i - 1)\delta$. Recall that along with a_{is} we insert the address $(i - 1) \bmod s$. Hence when a_{is} reaches cell $1 + (i - 1)\delta$ the address on its address line is $(i - 1) \bmod s$ and so a_{is} is activated.

We can similarly show that a_{is} meets a Φ_2 control signal in cell $n + (i - 1)\delta$. A Ψ_2 control signal is stored in location $(i - 1) \bmod s$ of this cell and hence a_{is} is deactivated here. So a_{is} is active in cells whose indices range from $1 + (i - 1)\delta$ to $n + (i - 1)\delta$. □

Having identified the cells in which a_{is} is active, we will now establish that the product term $a_{is}b_{sj}$ is added to c_{ij}.

Lemma B.3: $\forall s$, a_{is} and b_{sj} meet in cell $\delta(i - 1) + j$.

Proof: a_{is} is inserted at time $t_0 + (s - 1)\nu + (i - 1)\delta$ and b_{sj} is inserted at time $t_0 + (s - 1)\nu - (j - 1)$. a_{is} reaches cell $j + \delta(i - 1)$ at time $t_0 + (s - 1)\nu + 2(i - 1)\delta + (j - 1)$ and b_{sj} reaches the same cell at $t_0 + (s - 1)\nu - (j - 1) + 2[\delta(i - 1) + j - 1)]$ which simplifies to $t_0 + (s - 1)\nu + 2(i - 1)\delta + (j - 1)$.

From Lemma B.2, a_{is} is active in cell $\delta(i - 1) + j$. Now c_{ij} is stored in location $(i - 1) \bmod s$ of this cell which is the address on its address line when a_{is} and b_{sj} meet. □

From Lemma B.3, we can assert that c_{ij} has accumulated at least all the innerproduct terms $a_{is}b_{sj}$, $s = 1, \cdots, n$. To assert that $c_{ij} = \sum_{s=1}^{s=n} a_{is}b_{sj}$ we must ensure that c_{ij} is only updated with the correct product terms. We next show that this indeed is the case in the following Lemma.

Lemma B.4: Let $p \neq i$. When a_{pq} visits cell $j + (i - 1)\delta$ it does not contribute any product term to c_{ij}.

Proof: c_{ij} is stored in location $(i - 1) \bmod s$ of cell $j + (i - 1)\delta$. The address propagating in the array along with a_{pq} is $(p - 1) \bmod s$. If $(p - 1) \bmod s \neq (i - 1) \bmod s$ then c_{ij} cannot be updated. Therefore, let $(p - 1) \bmod s = (i - 1) \bmod s$ and this in turn implies $p - i = \gamma s$. Now $0 < |p - i| < n$ and hence $0 < |\gamma s| < n$. Let $h = j + (i - 1)\delta$. Substituting $p - \gamma s$ for i in h we obtain the following.

$$h=(p-1)\delta-\gamma\delta s+j\ \cdots\ (1).$$

Case 1: $\gamma > 0$. This implies $\gamma\delta s \geq n$ as $s = \lceil n/\delta \rceil$. Therefore, $j - 1 - \gamma\delta s < 0$ and hence, $h < 1 + (p - 1)\delta$.

Case 2: $\gamma < 0$. This implies $-\gamma\delta s \geq n$ and hence $j - \gamma\delta s > n$. So $h > (p - 1)\delta + n$.

From the above two cases we therefore conclude that a_{pq} is not active when it visits cell $j + (i - 1)\delta$. Therefore, it does not contribute any product term to c_{ij}. □

Theorem A.2: $c_{ij} = \sum_{s=1}^{s=n} a_{is}b_{sjS3}$.

Proof: From Lemma B.3 and Lemma B.4. □

References

[1] J. Bentley and T. Ottmann, "The power of a one-dimensional vector of processors," Universitat Karlsruhe, Bericht 89, Apr. 1980.

[2] J. W. Greene and A. Gamal, "Configuration of VLSI arrays in the presence of defects," *J. Ass. Comput. Mach.*, vol. 31, pp. 694–717, Oct. 1984.

[3] G. H. Hardy and E. M. Wright, *Introduction to the Theory of Numbers.* Fifth ed. New York: Oxford University Press, 1978.

[4] A. V. Kulkarni and D. W. L. Yen, "Systolic processing and an implementation for signal and image processing," *IEEE Trans. Comput.*, vol. C-31, pp. 1000–1009, Oct. 1982.

[5] H. T. Kung, "Why systolic architectures," *IEEE Computer,* vol. 15, pp. 37–46, Jan. 1982.

[6] ——, "Systolic algorithms for the CMU WARP Processor," in *Proc. Seventh Int. Conf. Pattern Recognition,* July 1984, pp. 570–577.

[7] F. T. Leighton and C. E. Leiserson, "Wafer-scale integration of systolic arrays," in *Proc. 23rd Annu. Symp. Foundations Comput. Sci.,* Nov. 1982, pp. 297–311.

[8] J. Raffel, "On the use of nonvolatile programmable links for restructurable VLSI," in *Proc. Caltech Conf. Very Large Scale Integration,* Jan. 1979, pp. 95–104.

[9] I. V. Ramakrishnan, D. S. Fussell, and A. Silberschatz, "Systolic matrix multiplication on a linear array," *Twentieth Annu. Allerton Conf. Comput. Contr. Commun.,* Oct. 1982.

[10] I. V. Ramakrishnan and P. J. Varman, "Modular matrix multiplication on a linear array," *IEEE Trans. Comput.,* vol. C-33, pp. 952–958, Nov. 1984.

[11] ——, "Synthesis of an optimal family of matrix multiplication algorithms on linear arrays," Dep. Elec. Comput. Eng., Rice Univ., Houston, TX, Tech. Rep. 8409, Dec. 1984.

Row/Column Replacement for the Control of Hard Defects in Semiconductor RAM's

TOM FUJA AND CHRIS HEEGARD

Abstract—**We describe and analyze row/column replacement, the technique currently used to control hard cell defects in semiconductor RAM's during manufacture. This strategy is shown to be asymptotically ineffective; it is demonstrated that this ineffectiveness may become a limiting issue for very large memory arrays.**

Index Terms—**Error-control coding, hard defects, RAM's, redundancy, reliability, row/column replacement, yield improvement.**

I. Introduction

A semiconductor random access memory (RAM) is a storage device consisting of many binary memory cells, each capable of storing one bit of information. These cells are generally laid out as one or more arrays; every bit in the memory is addressed as a row/column intersection or as a row/column/array intersection.

A *hard defect* in a RAM is one or more consistently unreliable cells. Such defects are usually caused by an imperfection in the semiconductor material and take the form of "stuck-at" cells; that is, the same value is read from the afflicted cell(s) regardless of what had been written.

Hard defects usually occur in one of three ways; row failures, column failures, and individual cell failures. Row and column failures occur when every cell in an entire row or column becomes unreliable; this is usually caused by a defect in a row/column enable line or in a row/column sensor. Individual cell failures are isolated defects which do not affect more than one bit of memory.

The technique widely used to control hard defects during manufacture is row/column replacement. With this method, extra rows and/or columns of memory cells are placed in each array and are switched into use to replace rows or columns which contain hard defects [1]–[6]. Previous analyses of row/column replacement have demonstrated impressive yield improvements when this strategy is applied to 4K [2], 64K [2], [4], and 256K [1] RAM's.

Our approach differs from published results in that we examine the

Manuscript received January 20, 1985; revised April 9, 1985. This work was supported in part by the National Science Foundation under Grant ECS-8352220, NATO under Grant 215/84, and by AT&T Bell Laboratories Ph.D. Scholarship Program.

The authors are with the School of Electrical Engineering, Cornell University, Ithaca, NY 14853.

IEEE Log Number 8610081.

asymptotic effectiveness of the row/column replacement strategy (i.e., its effectiveness as the array size grows unbounded). In doing so, we demonstrate that this strategy is effective for the current generation of RAM's only because of the very small expected number of defects in any memory array. However, our results indicate that for very large memories, where the expected number of defects is not trivially small, the asymptotic ineffectiveness of the row/column replacement strategy may become a dominant factor.

The figure-of-merit in evaluating this strategy will be bounds on the yield γ_0, the fraction of chips containing a potentially correctable pattern of hard defects. This figure is in fact optimistic, since there is no evidence that the algorithms currently used to switch in redundant rows and columns are optimal. (By an "optimal" algorithm we mean one which produces a defect-free chip every time it is possible to do so.) However, we will see that even if we assume that the algorithms *are* optimal, the row/column replacement strategy can be a most ineffective one.

Finally, in our analysis we will consider row/column replacement for controlling only individual cell defects. There are two reasons for this; first, individual cell defects are the dominant defect type. Second, row/column replacement seems a reasonably efficient means of controlling defects which corrupt an entire row or an entire column. On the other hand, it is heuristically unsatisfying to replace an entire row or column because of a single bad cell. It is this heuristic argument that we formalize and prove in our analysis.

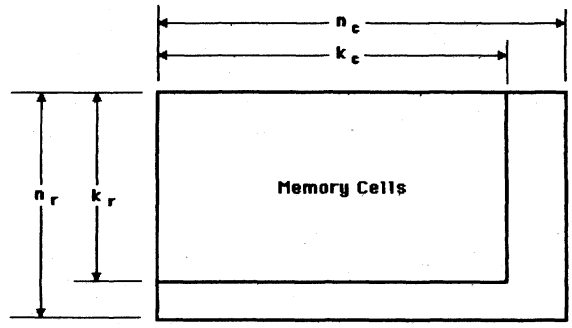
II. Analysis of Row/Column Replacement

In this section we will examine the effectiveness of the row/column replacement strategy as a means of controlling individual cell defects. We will begin by giving a precise model of the strategy. Then, the asymptotic effectiveness of row/column replacement as the memory array size grows will be examined. In addition, we will show where the asymptotic effects become pronounced. Finally, we will give bounds on the strategy's yield and will demonstrate with examples the implications of the analysis.

A. Strategy Model

Consider an $n_r \times n_c$ array of memory cells from which we want to obtain a $k_r \times k_c$ subarray containing a controlled number of defects ($k_r \leqslant n_r$, $k_c \leqslant n_c$). We will call such an array an (n_r, k_r, n_c, k_c) *code word* (Fig. 1). Every memory cell to be used is associated with an L_r-bit row address and an L_c-bit column address; thus, $k_r = 2^{L_r}$ and $k_c = 2^{L_c}$. Let $N = n_r n_c$ be the total number of cells in the array and $L = L_r + L_c$. Thus, $2^L/N$ is the fraction of "useable" cells, i.e., the *rate* of the code.

Finally, we will assume that each cell in the array is defective with independent probability p. This assumption is consistent with the Poisson distribution on hard defects which is usually assumed [1], [2].



$n_r \geqslant k_r = 2^{L_r}$

$n_c \geqslant k_c = 2^{L_c}$

Address Ratio: $\alpha = \dfrac{L_r}{L_r + L_c}$

Code Rate: $R = \dfrac{k_r k_c}{n_r n_c}$

Fig. 1. An (n_r, k_r, n_c, k_c) code word.

B. Asymptotic Behavior

Consider an (n_r, k_r, n_c, k_c) code as described above. Let γ_q be the fraction of codewords which contain a $k_r \times k_c$ subarray with no more than qk_rk_c defects; that is,

$$\gamma_q = \Pr\{\text{There exists a } k_r \times k_c \text{ subarray with } \leqslant qk_rk_c \text{ defects}\}.$$

Theorem: If

$$\lim_{L\to\infty} \frac{L_r}{L} = \alpha, \quad 0<\alpha<1 \quad \text{and} \quad \lim_{L\to\infty} \frac{2^L}{N} = R, \quad R>0$$

then

$$\lim_{L\to\infty} \gamma_q = 0 \quad \text{for all } q<p.$$

Proof:

$$\gamma_q = \Pr\left\{\bigcup_{g\in G}\{g \text{ contains } \leqslant qk_rk_c \text{ defects}\}\right\}$$

where $G = \{\text{All } k_r \times k_c \text{ subarrays}\}$. Clearly,

$$|G| = \binom{n_r}{k_r}\binom{n_c}{k_c}.$$

Furthermore, for all $g \in G$,

$$\Pr\{g \text{ contains } \leqslant qk_rk_c \text{ defects}\} = \sum_{j=0}^{qk_rk_c} \binom{k_rk_c}{j} p^j(1-p)^{k_rk_c-j}$$

$$\leqslant (1+qk_rk_c)\binom{k_rk_c}{qk_rk_c} p^{qk_rk_c}(1-p)^{k_rk_c(1-q)}$$

where we have made use of the fact that for $q < p$ the maximum term inside the sum is the last, $j = qk_rk_c$.

Using the union of events bound

$$\gamma_q \leqslant \binom{n_r}{k_r}\binom{n_c}{k_c}(1+qk_rk_c)\binom{k_rk_c}{qk_rk_c} p^{qk_rk_c}(1-p)^{k_rk_c(1-q)}$$

$$= 2^{\log_2\binom{n_r}{k_r} + \log_2\binom{n_c}{k_c} + \log_2\binom{k_rk_c}{qk_rk_c} + \log_2(1+qk_rk_c) + k_rk_c[q\log_2 p + (1-q)\log_2(1-p)]}$$

we can now make use of the inequality [7, p. 310]

$$\log_2\binom{n}{k} \leqslant nh\left(\frac{k}{n}\right)$$

where $h(x) = -x\log_2 x - (1-x)\log_2(1-x)$, the binary entropy function. This yields

$$\gamma_q \leqslant 2^{n_rh(k_r/n_r) + n_ch(k_c/n_c) + \log_2(1+qk_rk_c) + k_rk_c[h(q) + q\log_2 p + (1-q)\log_2(1-p)]}$$

$$= 2^{N[(1/n_c)h(k_r/n_r) + (1/n_r)h(k_c/n_c) + (1/N)\log_2(1+qk_rk_c) - (2^L/N)D(q\|p)]}$$

where $D(q\|p = q\log_2(1|p) + (1-q)\log_2(1-q/1-p)$ is the

divergence between q and p [8]. ($D(q\|p)$, a quantity often used in statistics and information theory, is related to the two-hypotheses testing problem.)

Now, $\log_2 n_r \geqslant \log_2 k_r \rightarrow \alpha L$. Thus, $n_r \rightarrow \infty$ as $L \rightarrow \infty$. Similarly, $n_c \rightarrow \infty$. In addition, $h(x) \leqslant 1$ for all x, $0 \leqslant x \leqslant 1$. Therefore,

$$\lim_{L\to\infty} \left[\frac{1}{n_c} h\left(\frac{k_r}{n_r}\right) + \frac{1}{n_r} h\left(\frac{k_c}{n_c}\right) + \frac{1}{N} \log_2 (1+qk_rk_c) - \frac{2^L}{N} D(q\|p) \right] = -RD(q\|p).$$

Finally, $D(q\|p) \geqslant 0$ for all p and q, $0 \leqslant q, p \leqslant 1$, with equality iff $q = p$; thus, for $R > 0$ the exponent $\lim_{L\to\infty} 1/N \log_2 \gamma_q$ is negative and so

$$\lim_{L\to\infty} \gamma_q = 0. \qquad \text{Q.E.D.}$$

This result shows that, asymptotically, row/column replacement is ineffective. As we let the array size grow unbounded in each dimension, the probability of decreasing the fraction of defects on a chip goes to zero for any positive rate R (the ratio of memory capacity to size); row/column replacement cannot produce defect-free chips, it cannot even produce marginally better ones. The question arises: Why is row/column replacement used in practice?

To see why row/column replacement has been used with success to control hard defects, we must answer the following question: How big does a RAM have to grow before the asymptotic failure demonstrated above begins to dominate?

To investigate this issue, we will use a slightly simpler model of a RAM. In this model, we will set $n_r = n_c = n$ and $k_r = k_c = k$ (i.e., a square array, with $\alpha = 1/2$). Thus, $k^2 = 2^L$ and $R = k^2/n^2$. Also, consider the yield for $q = 0$; thus, we insist that the strategy yields a *defect-free* chip.

In our derivation above, we found that

$$\gamma_0 \leqslant 2^{n^2 E(L,R)}$$

where

$$E(L, R) = \frac{k^2}{n^2} \log_2 (1-p) + \frac{2}{n} h\left(\frac{k}{n}\right)$$

$$= R \log_2(1-p) + \frac{2\sqrt{R}h(\sqrt{R})}{\sqrt{2^L}}.$$

This bound reaches a critical point when $E(L, R)$ becomes negative; once that occurs, the upper bound on γ_0 becomes very small very quickly. Thus, it is natural to ask for what values of L and R does $E(L, R) = 0$, or

$$\sqrt{R2^L} \log_2 (1-p) + 2h(\sqrt{R}) = 0$$

or, equivalently,

$$L = -2 \log_2 \left(\log_2 \left[\frac{1}{1-p} \right] \right) + 2 \log_2 \frac{2h(\sqrt{R})}{\sqrt{R}}.$$

For small p we can make the approximation $\log_2 [1/1-p] \approx p \log_2 e$. This means the critical point occurs when

$$L \approx -2 \log_2 p + 2 \log_2 \frac{2h(\sqrt{R})}{\sqrt{R} \log_2 e}$$

$$= -6.64 \log_{10} p + 2 \log_2 \frac{2h(\sqrt{R})}{\sqrt{R} \log_2 e}.$$

The second term in the above expression is a slowly decreasing function of R; as R increases from 0.7 to 0.95, the second term decreases from 0.18 bits to -4.1 bits, a loss of only 4.3 bits in the useable address space. The first term in the expression, however, removes over six and one half bits from the address space for every decade increase in p. This indicates that the probability of a defective cell is critical in determining the limits of row/column replacement.

TABLE I
CRITICAL VALUES OF L FOR VARIOUS VALUES OF R AND p

R \ p	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
.95	9.2	15.8	22.5	29.1	35.8
.80	12.5	19.1	25.7	32.4	39.0
.70	13.5	20.1	26.7	33.4	40.0

Shown in Table I are the critical values of L for rates of 0.70, 0.80, and 0.95 and for values of p between 10^{-2} and 10^{-6}. This shows that for RAM's similar to those used today (i.e., RAM's with rates close to 1.0 and $5 \times 10^{-6} \leqslant p \leqslant 5 \times 10^{-5}$) the limit to row/column replacement effectiveness lies in the four to sixteen Mbit range.

The largest RAM's currently available are 256K; thus, it is not surprising that the asymptotic ineffectiveness of row/column replacement has not been a factor. However, as RAM's increase in size, or if the probability of individual cell defects increases as more bits are squeezed onto a chip, then this asymptotic failure of the row/column replacement strategy will become a limiting issue.

Of course, RAM's are rarely built as a single array; usually, several such arrays are included on a chip. However, when n arrays are placed on a chip, the fraction of defect-free chips is $(\gamma_0)^n$ where γ_0 is the probability of producing a single defect-free array. Suppose, for instance, we would like to produce a 16M chip at an effective yield of 0.90. To construct such a chip from 256K arrays, we would have to fabricate 64 arrays each with a yield in excess of 0.998. Similarly, if we were to use 16 1M arrays, they would each have to have a yield in excess of 0.993. This reinforces the notion that, as chip size increases, so must array size, and the fundamental ineffectiveness of row/column replacement must still be considered.

C. A Lower Bound on the Yield

The row/column replacement strategy yields a good chip whenever every defect in an array can be eliminated by disabling $r_r = n_r - k_r$ rows and $r_c = n_c - k_c$ columns. Thus,

$$\gamma_0 = \Pr\{\text{correctable defect}\}$$

$$= \Pr\{\text{defect-free } k_r \times k_c \text{ subarray}\}$$

$$= \sum_{j=0}^{n_rn_c} \Pr\{\text{defect-free } k_r \times k_c \text{ subarray} | j \text{ defects}\} \Pr\{j \text{ defects}\}$$

$$\geqslant \sum_{j=0}^{r_r+r_c} \Pr\{\text{defect-free } k_r \times k_c \text{ subarray} | j \text{ defects}\} \Pr\{j \text{ defects}\}.$$

But

$$\Pr\{j \text{ defects}\} = \binom{n_rn_c}{j} p^j (1-p)^{n_rn_c-j}$$

and

$$\Pr\{\text{defect-free } k_r \times k_c \text{ subarray} | j \text{ defects}\} = 1$$

for $0 \leqslant j \leqslant r_r + r_c$.

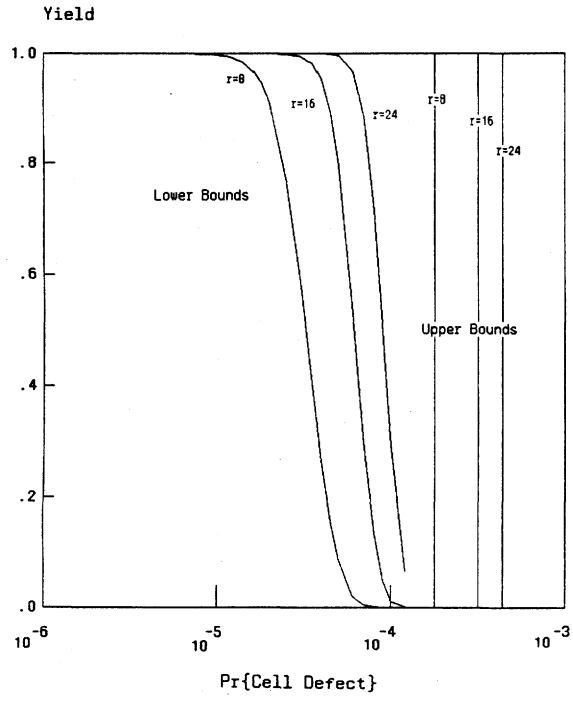


Fig. 2. Yields for a 256K array.

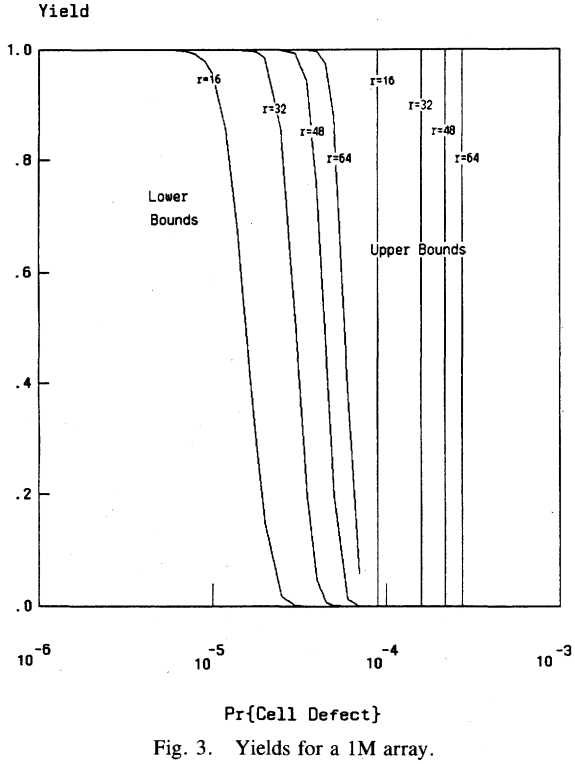


Fig. 3. Yields for a 1M array.

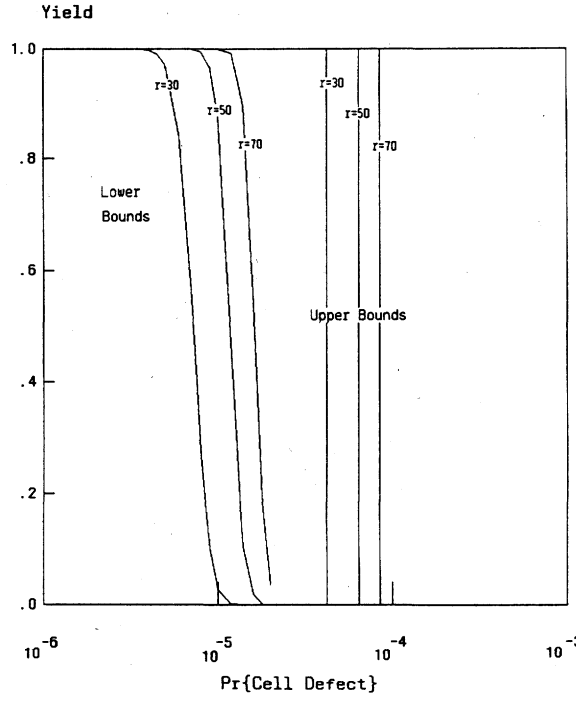


Fig. 4. Yields for a 4M array.

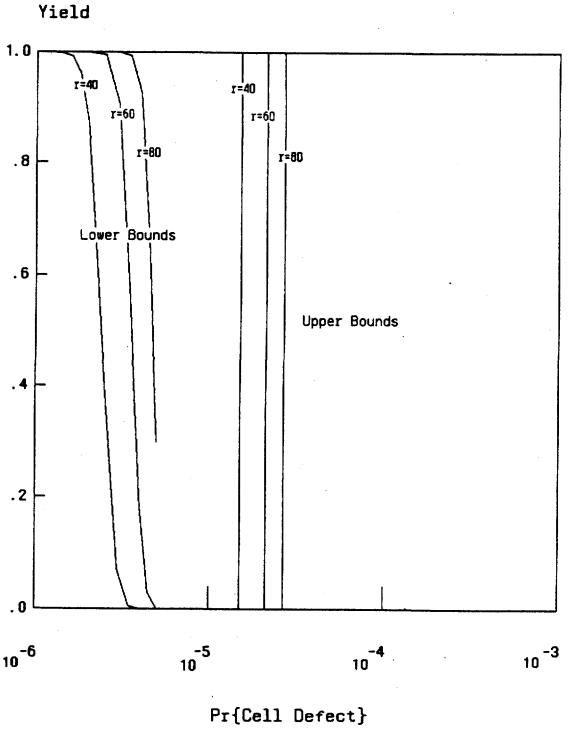


Fig. 5. Yields for a 16M array.

Thus, this provides the lower bound on γ_0

$$\gamma_0 \geq \underline{\gamma_0} = \sum_{j=0}^{r_r+r_c} \binom{n_r n_c}{j} p^j (1-p)^{n_r n_c - j}.$$

D. Some Examples

Figs. 2–5 show graphs of upper and lower bounds on γ_0 for 256K, 1M, 4M, and 16M memory arrays with varying amounts of redundancy. (Note: in each case, the results for a square ($\alpha = 1/2$) array are plotted. Thus, $r_r = r_c = r/2$ where r is the parameter on each curve.) The lower bound is $\underline{\gamma_0}$. The upper bound is in fact an approximation to the upper bound developed in Section II-B; it consists of a "brick wall" function at the critical probability where $E(L, R)$ becomes negative, i.e.,

$$\overline{\gamma_0} \approx \begin{cases} 0 & \text{for } p > p_{crit} \\ 1 & \text{otherwise} \end{cases}$$

where $p_{crit} \equiv 1 - 2^{-[2nh(k/n)]/k^2}$. (The probability of a correctable defect is negligible for $p > p_{crit} + \epsilon$ for a very small ϵ.)

The graphs support the earlier claims about row/column replacement. For a 256K array, the addition of 12 redundant rows and 12 redundant columns provide a yield in excess of 99 percent for $p \leqslant 10^{-4}$. However, for the 16M array, the addition of 40 extra rows and 40 extra columns produces a negligible yield for $p > 2.7 \times 10^{-5}$.

III. Conclusion

In this correspondence we show how hard defects can corrupt a random access memory and describe the current technique for controlling these defects during manufacture: row/column replacement.

Row/column replacement is asymptotically ineffective as a means of controlling hard defects. As we let the memory array grow unbounded in each direction, the probability of reducing the fraction of defects in the array goes to zero regardless of the fraction of extra rows and columns available for spare switching.

Finally, the asymptotic failure described above may become a significant limitation for very large memory arrays.

IV. Future Considerations

As random access memories get larger, a more effective means of controlling hard defects must be incorporated into their design. One obvious method is the inclusion of on-chip error correction. From Shannon's theory we know that for any $R < C(p)$ there exists a (n_c, k_c) code with $k_c/n_c \geqslant R$ such that the probability of a decoding error can be made arbitrarily small. (Here, $C(p) = 1 - h(p)$ where p is the probability of a defective cell; if the location of the defects are made available to the encoder or decoder, then $C(p) = 1 - p$ [9].) Such a code could be implemented on the rows of a RAM and provide a high degree of protection. This contrasts vividly with the "zero yield" which row/column replacement offers for large RAM's.

Currently, on-chip error correction is being increasingly considered as a means of providing protection from so-called "soft errors" [10]–[13]. These errors are transient in that they can be "scrubbed" from the system by rewriting the contents of the affected memory cells. The advisability of using on-chip ECC's to control *both* hard and soft errors is something which should be considered.

Acknowledgment

The authors thank T. Berger for his helpful suggestions on the asymptotic results.

References

[1] T. Mano *et al.*, "A redundancy circuit for a fault-tolerant 256K MOS RAM," *IEEE J. Solid State Circuits,* vol. SC-17, pp. 726–730, Aug. 1982.

[2] S. E. Schuster, "Multiple word/bit line redundancy for semiconductor memories," *IEEE J. Solid State Circuits,* vol. SC-13, pp. 698–703, Oct. 1978.

[3] R. P. Cenker *et al.*, "A fault tolerant 64K dynamic random access memory," *IEEE Trans. Electron. Dev.*, vol. ED-26, pp. 853–860, June 1979.

[4] R. T. Smith, "Laser programmable redundancy and yield improvement in a 64K DRAM," *IEEE J. Solid State Circuits,* vol. SC-16, pp. 506–513, Oct. 1981.

[5] T. E. Mangir, "Sources of failures and yield improvement for VLSI and restructurable interconnects for RVLSI and WSI: Part I—Sources of failures and yield improvement for VLSI," *Proc. IEEE,* vol. 72, pp. 690–708, June 1984.

[6] C. H. Stapper *et al.*, "Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product," *IBM J. Res. Develop.*, vol. 24, pp. 398–409, May 1980.

[7] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes.* Amsterdam, The Netherlands: North Holland, 1977.

[8] I. Csiszar and J. Korner, *Information Theory: Coding Theorems for Discrete Memoryless Systems.* New York: Academic, 1981.

[9] C. Heegard and A. A. El Gamal, "On the capacity of computer memory with defects," *IEEE Trans. Inform. Theory,* vol. IT-29, pp. 731–739, Sept. 1983.

[10] T. C. May and M. H. Woods, "Alpha particle induced soft errors in dynamic memories," *IEEE Trans. Electron. Dev.*, vol. ED-26, pp. 2–9, Jan. 1979.

[11] F. I. Osman, "Error-correction technique for random access memories," *IEEE J. Solid State Circuits,* vol. SC-17, pp. 877–881, Oct. 1982.

[12] T. Mano *et al.*, "Circuit techniques for a VLSI memory," *IEEE J. Solid State Circuits,* vol. SC-18, pp. 463–469, Oct. 1983.

[13] J. Yamada *et al.*, "A submicron 1 Mbit dynamic RAM with a 4-Bit-at-a-time built-in ECC circuit," *IEEE J. Solid State Circuits,* vol. SC-19, pp. 627–633, Oct. 1984.

A Parallel Algorithm to Compute the Shortest Paths and Diameter of a Graph and Its VLSI Implementation

BHABANI P. SINHA, BHARGAB B. BHATTACHARYA, SURANJAN GHOSE, AND PRADIP K. SRIMANI

Abstract—**In this correspondence we develop a parallel algorithm to compute the all-pairs shortest paths and the diameter of a given graph. Next, this algorithm is mapped into a suitable VLSI systolic architecture and the performance of this proposed VLSI implementation is evaluated.**

Index Terms—**Diameter, parallel algorithms, pipelining, shortest paths, VLSI architecture.**

I. Introduction

Enumeration of shortest paths between all pairs of vertices and finding the diameter of a graph constitute an important problem in graph theory and have many practical applications involving some commodity flow, e.g., in a computer communication network. In a communication network, the diameter of the network graph is a deciding factor in choosing the system topology which defines the interprocessor communication architecture. Further, a knowledge of the shortest paths between every two processing nodes in a network is essential to determine dynamically the optimal feasible route from one processor to the other in order to minimize the communication delay.

Various algorithms [2]–[5] exist for this shortest path problem; they are sequential in nature, and the time complexity of the best known algorithm of this class to compute all-pairs shortest distances is $O(n^{5/2})$ [5] while that of all-pairs shortest paths is $O(n^3)$ where n is the number of vertices.

The availability of low-cost, high-speed processor arrays during the last decade gave an impetus for parallelization of programs [12]. With the steep decrease in hardware cost due to the recent VLSI technology, there is a growing trend toward parallelization of different existing algorithms and their VLSI implementation [7]–[10], [13]–[16] to improve upon the execution time at the cost of providing a larger number of processors. Guibas, Kung, and Thompson [9] have given algorithms for dynamic programming and transitive closure problems suitable for VLSI implementation and their ideas can be readily extended to solve the shortest path problem as well. In this correspondence we follow a different approach to design a

Manuscript received September 17, 1984; revised August 9, 1985.

B. P. Sinha and S. Ghose are with the Electronics Unit, Indian Statistical Institute, Calcutta, 700 035, India.

B. B. Bhattacharya is with the Department of Computer Science, University of Nebraska, Lincoln, NE 68588.

P. K. Srimani is with the Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, on leave from the Indian Institute of Management, Calcutta, India.

IEEE Log Number 8610487.