JOURNAL OF VLSI AND COMPUTER SYSTEMS
Vol. 2, 1987, 23-35.

Restructuring Hexagonal Arrays of Processors in the Presence of Faults

DAN GORDON,* ISRAEL KOREN,† AND GABRIEL M. SILBERMAN‡

Abstract—**The issue of fault-tolerance in VLSI processing arrays has been the subject of several recent studies where different schemes for achieving fault-tolerance have been proposed. We concentrate here on fault-tolerance in hexagonal arrays, while most previous publications dealt with fault-tolerance in linear and rectangular arrays. Hexagonal arrays have been used for various computational algorithms and were shown to be more flexible when reconfiguring the array to match a given algorithm. It is, therefore, appropriate to develop a fault-tolerance strategy suitable for these processing arrays.**
Keywords: **Fault-tolerance, manufacturing defects, faults, restructuring, hexagonal processor array, VLSI.**

1. INTRODUCTION

In recent years, large arrays of identical processing elements have been recognized as a viable architecture alternative in VLSI. These processing arrays have a very regular structure resulting in simple designs and implementations. A variety of array topologies like linear arrays, rectangular arrays, and hexagonal arrays have been suggested and different computational algorithms for these topologies have been developed [5], [8].

To increase the number of applications for a given topology of a processor array it has been suggested that several logical topologies be mapped on a given physical topology [1, 4, 10, 11]. Mapping a binary tree on rectangular and hexagonal arrays [2, 4, 10, 11] is one example. Clearly, the large number of links per processor in the hexagonal topology can simplify the task of mapping and may result in a better area utilization. For example, it has been shown in [2] that when mapping a binary tree on an hexagonal array, up to 71 percent of the total chip area may be utilized, as compared to 50 percent for the rectangular array.

Because of imperfect integrated circuit implementations, failures occur in the arrays, which gave rise to several studies on the issue of fault-tolerance in regular arrays [1, 3, 4, 6, 10]. There are two distinct types of failures that may occur in

* Department of Computer Studies, University of Haifa, Haifa 31999, Israel.
† Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, on leave from the Department of Electrical Engineering, Technion, Haifa 32000, Israel.
‡ Department of Computer Science, Technion, Haifa 32000, Israel.

VLSI systems, namely, *production defects* and *operational faults* [7]. These two differ in their probability of occurrence and in the costs associated with them.

To deal with these two types of failures, schemes for introducing redundant hardware have been suggested. The hardware added can be in the form of switching elements [1, 11], or redundancy in processors or communication links [3, 4, 11]. Most of these schemes attempt to achieve 100 percent utilization of the processing elements (PEs), which are considered to be the most important system resource. For example, in [1, 10, 11] switching elements are added between processors to assist in achieving this goal. However, the silicon area devoted to switching elements capable of interconnecting four to eight separate parallel busses [11] cannot be ignored. Moreover, these switches necessitate the introduction of additional communication links, which will consume an even larger percentage of the chip area. Therefore, schemes that use switching elements might be beneficial only for processors that are substantially larger than the switches [9, 10].

In addition, one of the underlying assumptions in these schemes is that the switching elements and the communication links are almost failure-free and only processors can fail. However, larger silicon areas devoted to those switches and their associated communication links increase their susceptibility to defects or faults and the above-mentioned assumption might not be valid any more.

Consequently, for processor arrays in which the silicon area occupied by each PE is relatively small, schemes that do not incorporate switching elements might be appropriate [9]. These schemes do not attempt to achieve 100 percent utilization of the fault-free PEs (when the array is restructured to avoid the use of faulty ones). Such schemes have been proposed in [4] for rectangular arrays; they can be attractive especially when dealing with operational faults (which are few in number). Here, the lack of additional hardware (switches or links) allows a larger number of PEs to fit into the same chip area, thereby offsetting the penalty of giving up the use of fault-free PEs when restructuring.

Another objective of a strategy such as [4] is in making the restructured array transparent to the various algorithms that use it. These algorithms fall into two broad categories: Algorithms which make direct use of the regular array for various purposes [5, 8]; and algorithms that map logical topologies, such as binary trees, on the physical topology of the processor array [2, 4]. The only change in the array is a reduction of its size.

It might seem that this simple strategy is inappropriate to handle the large number of production flaws. However, in a recent paper by Koren and Breuer [6], it is shown that such simple strategies might be effective even against manufacturing defects. Employing practical defect distributions and not asymptotic results, it has been shown in [6] that the effective yield might even increase when a fault-tolerant array of PEs with the above restructuring strategy is designed. This is due to the fact that the probability of getting a chip with only a single or two defects is high, and if the chip can successfully handle such a small number of defects, the overall yield is increased.

The generalization of the fault-tolerance scheme in [4] to other physical array topologies is not trivial. We introduce, therefore, in the next section a similar scheme for hexagonal arrays. These arrays were shown to be more flexible and more suitable for various applications compared to simpler topologies like linear and square arrays [2, 5, 8].

2. FAULT-TOLERANCE IN HEXAGONAL ARRAYS

The fault-tolerance scheme for hexagonal arrays to be presented next is based on the following approach: When some PEs or connections become faulty, the other PEs will be restructured into an hexagonal array (of smaller size). The importance of such an objective is that the failure of some of the PEs does not preclude the usage of the same alogrithm for mapping logical structures onto the hexagonal array.

Fault tolerance is achieved in two basic stages, the *testing* stage and the *reconfiguration* stage. In the testing stage, the PEs test their neighbors and themselves, in order to identify faulty PEs or connections. In the reconfiguration stage, the PEs with neighboring faults turn into *connecting elements* (CEs) and initiate messages which turn some other PEs into CEs. These CEs cease to perform processing per se and behave like connectors between pairs of neighboring PEs.

Each remaining PE is not aware of the presence of the CEs and continues to communicate with six neighbors as it did before the reconfiguration occurred, using the same links as it did before (i.e., the neighbor in a given direction is still accessed in that direction). It is possible, however, that some of its neighbors are not physically the same as before, and the PE reaches them through some CEs. This concept first appeared in [4], where it was applied to rectangular arrays.

2.1 A Distributed Testing Procedure

We propose a distributed testing procedure in which every PE tests all its neighbors [4]. In this way, faulty PEs and faulty connections between PEs are detected by the adjacent PEs.

The procedure first partitions all of the PEs into seven disjoint testing groups, $T_0, T_1, \ldots, T_6$. After this partitioning, there are seven phases of testing, where at phase i ($i = 0, 1, \ldots, 6$), the members of T_i test all of their neighbors.

The partition is such that (1) every PE is surrounded by PEs of other groups, and (2) no PE has two neighbors belonging to the same group. These two properties guarantee that for every i, no two members of T_i will test each other, or try simultaneously to test a third PE. It can easily be seen that seven groups are both necessary and sufficient for a partition wth the above properties.

An example of one such partitioning is given in Figure 1. This particular method was chosen because it leads to a very concise algorithm for assigning group numbers.

The testing procedure for this example is initiated externally by assigning the group number 0 to the left-upper corner PE in the whole array. Every other PE, after being assigned a group number i, assigns group numbers to its neighbors in directions 1, 2, and 3 as follows:

Assign $(i+2)$ mod 7 to PE in direction 1.
Assign $(i+3)$ mod 7 to PE in direction 2.
Assign $(i+1)$ mod 7 to PE in direction 3.

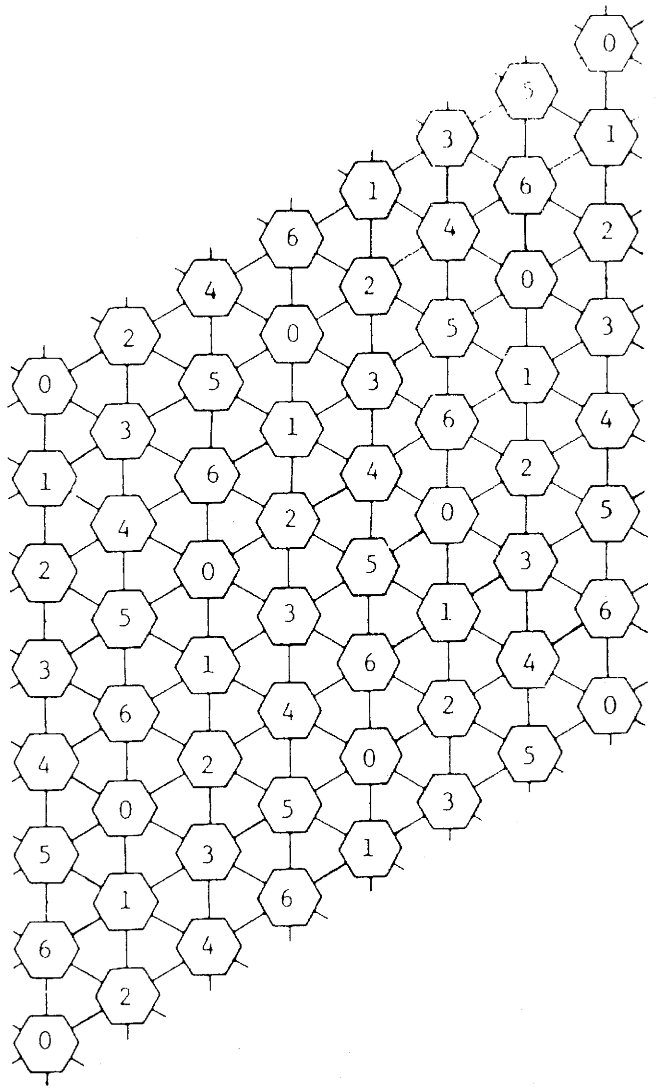

Figure 1 An example of partitioning the PEs into seven testing groups.

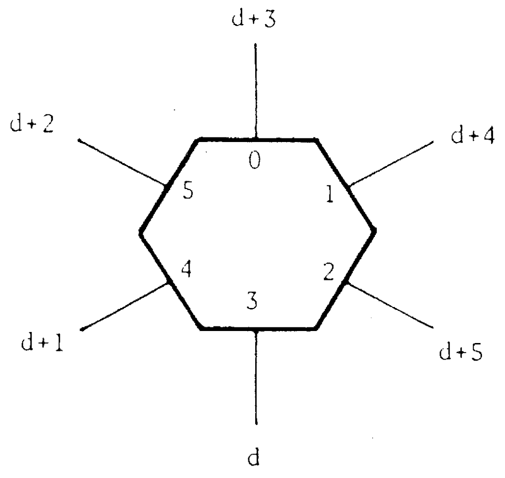

Figure 2 Absolute and relative-to-*d* numbering of directions.

The numbering of directions follows that suggested in [2], and is shown in Figure 2.

Note that if each element in the hexagonal array has been assigned matrix indices (i,j), then their group number is equivalent to $(i+2j)$ mod 7.

After waiting a suitable period of time, phase 0 of the testing is externally triggered, and broadcast to all PEs. All PEs in T_0 then start testing all of their neighbors. The transition from phase to phase is always externally controlled, after a suitable period of time has elapsed.

The grouping stage is necessary every time before the testing stage because of possible reconfigurations resulting from earlier testings.

2.2 Reconfiguration

In the reconfiguration stage, every PE is assumed to know the status (faulty/not-faulty) of its six connecting links or neighboring PEs. There is no difference if the actual fault is in the neighboring PE proper, or in the link leading to it.

The basic unit for reconfiguration purposes is the CE. Out of all possible configurations of a CE, only the three shown in Figure 3 are actually used in the following algorithm. Any one of these is completely defined by the link that connects opposite directions.

For reasons that will become clear later, we find it convenient to label the three types of CEs by adding (modulo 6) the numbers of the opposite directions that characterize them. Thus, CEs of type 1, 3, and 5 link the opposite pairs 2–5, 0–3, and 1–4, respectively. Note that if we disregard absolute directions, there is only one type of CE.

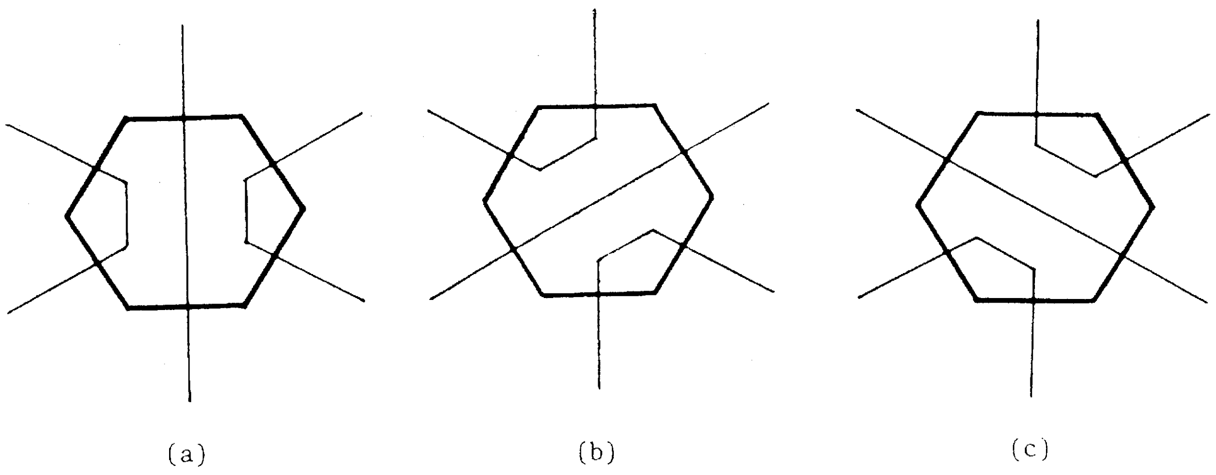

Figure 3 The three types of connecting elements: (a) type 3(0 + 3), (b) type 5 (1 + 4), (c) type 1 (2 + 5).

2.3 Single Faults

A single fault is either a faulty connection or a faulty PE. We begin with the description of the action taken by a PE which discovers that it has a single faulty connection (or faulty neighbor). In Figures 4–6, a CE is represented by a stroke through the processor, the slope of this stroke indicating the link which characterizes the type of CE.

Assume that the PE senses a fault in direction d ($d = 0, 1, \ldots, 5$) from itself. The PE transmits a $\underline{C}$ (for $\underline{C}$onvert) message in the opposite direction to d (i.e., $(d+3) \bmod 6$); furthermore, if d is odd, then the PE also transmits $\underline{C}$ in direction $(d+1) \bmod 6$ (see Figure 2 for relative-to-d numbering of directions). After transmitting its message(s), the PE becomes a CE of type $(2d-(-1)^d) \bmod 6$.

When a PE receives $\underline{C}$ from direction d, it retransmits $\underline{C}$ in the opposite direction and becomes a CE of type $(2d-(-1)^d) \bmod 6$. The $\underline{C}$ message thus travels in a (virtual) straight line from the fault, turning all PEs in its path to CEs of identical type.

An example is given in Figure 4. The link between processors A and B is assumed faulty. A and B discover this and initiate $\underline{C}$ messages in opposite directions. In addition, B sends a $\underline{C}$ message to processor C which in turn retransmits $\underline{C}$ in direction 0. This results in three "rays" of CEs emanating from the area of the fault. Consider now the path of a connection between two PEs around the newly formed CEs; it either goes straight through (as between D and E), or makes two "turns" through two CEs of the same type, and comes out in the same direction it started (as between F and G). This fact is important because a message sent *out* by a PE in direction d has to be received by its (logical) neighbor as if coming *from* the opposite direction.

In Figure 5 we see an example of a single faulty PE. All of its six neighbors initiate $\underline{C}$ messages away from the fault and six rays of CEs are thus formed.

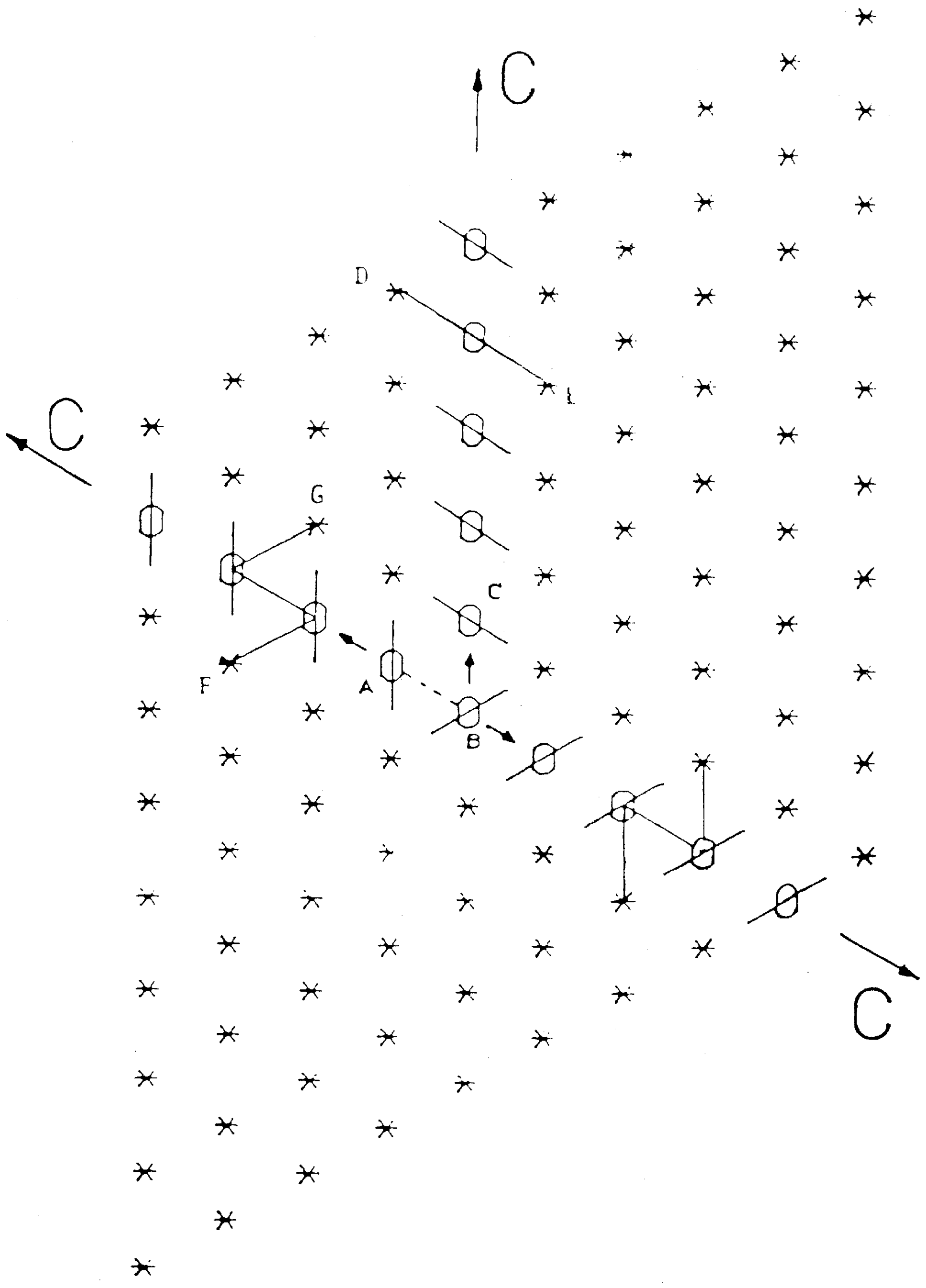

Figure 4 CEs formed by a faulty connection between A and B.

Recall that all testing is initiated externally, and we also assume that the reconfiguration phase is externally triggered. In this way, every PE with a neighboring fault takes the required action (transmits C̲ message(s) and becomes a CE) before handling any incoming messages.

The above sequence of events causes the extra C̲ messages produced by three of the PEs surrounding a faulty PE to be "bounced" towards the fault, thus preventing them from propagating further.

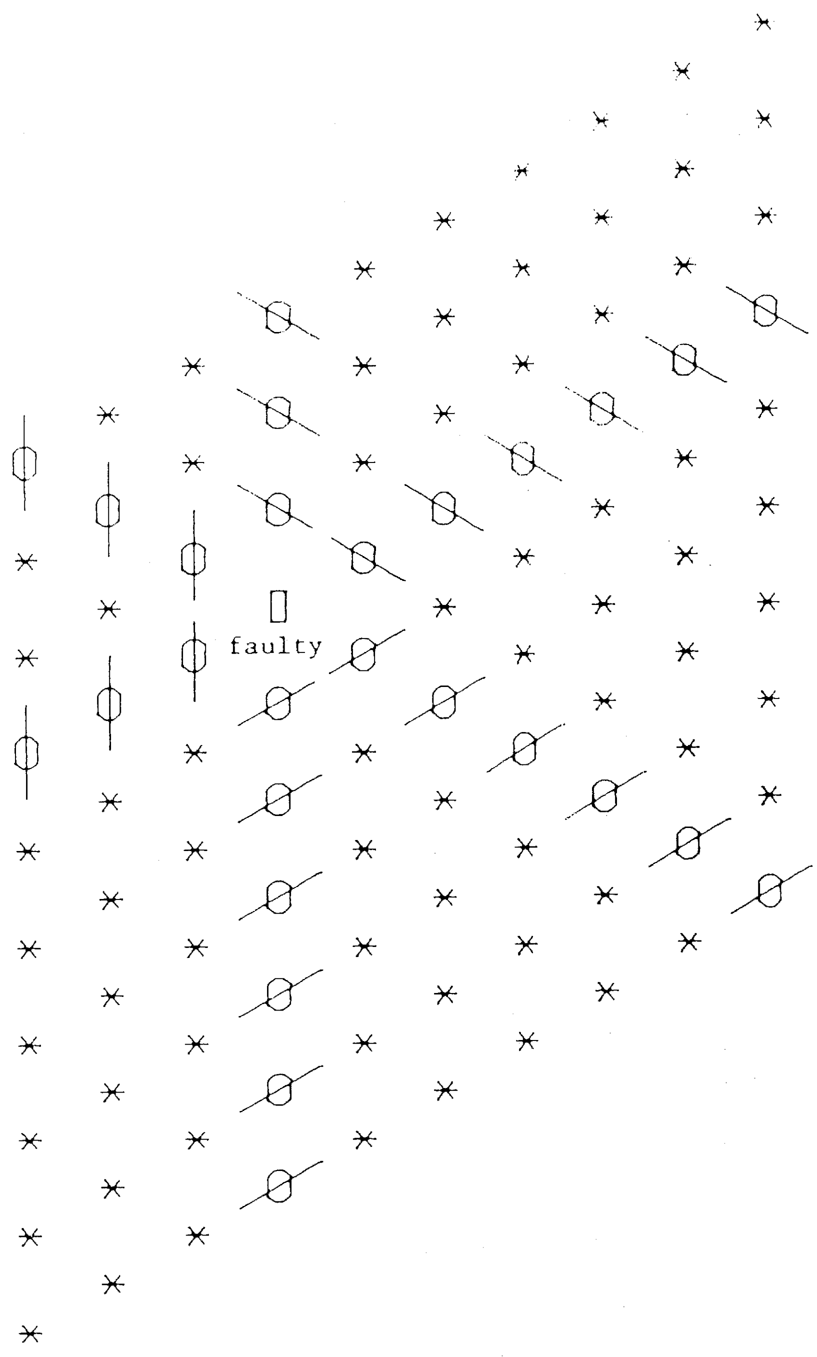

Figure 5 CEs formed by a faulty PE.

2.4 Multiple Faults

When a PE detects more than one fault, it transmits C̲ messages to all of its nonfaulty neighbors. If the faults are only in the connections, this has the same effect as if all the neighboring PEs had detected that particular PE as being faulty.

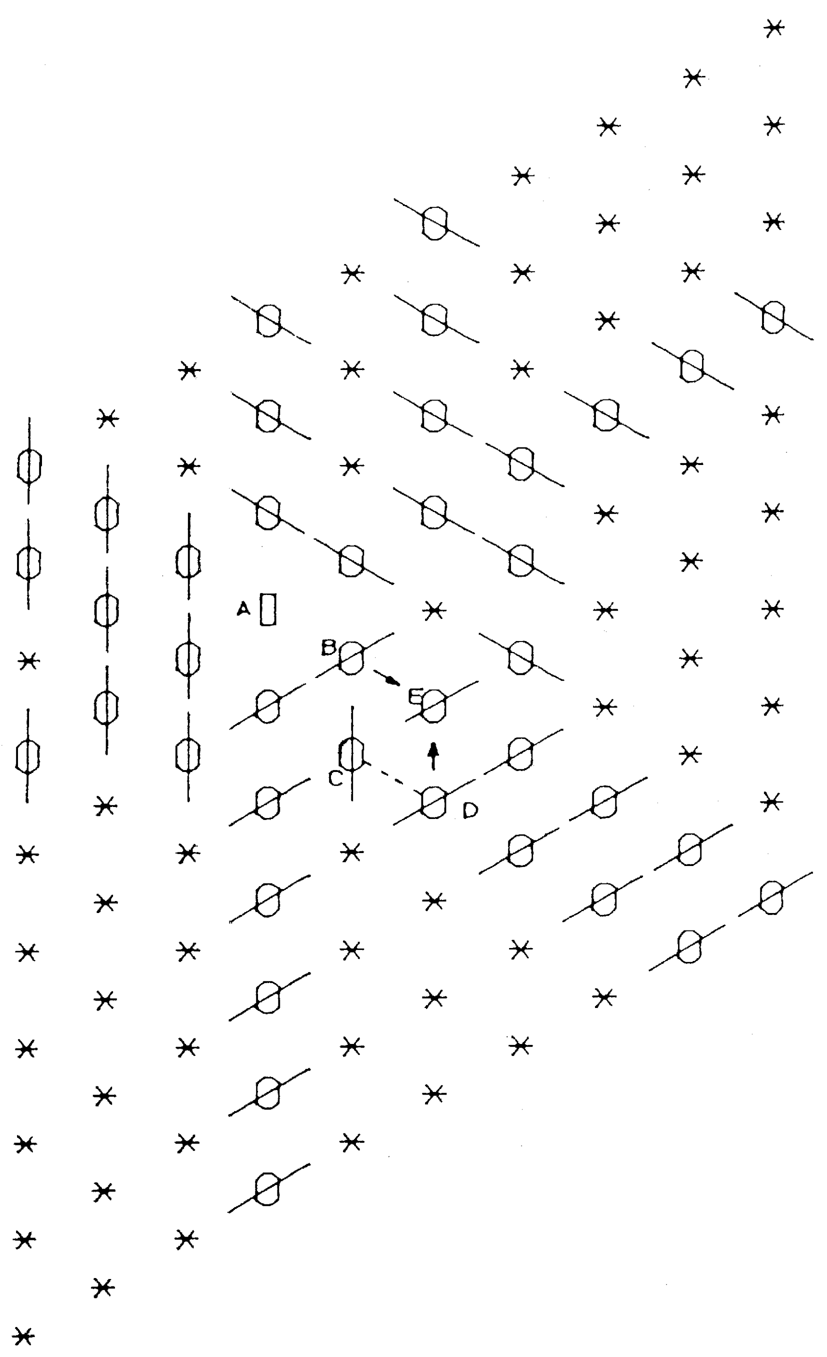

Figure 6 The effect of multiple faults; A and the connection between C and D are faulty.

When one (or more) of the neighboring PEs is faulty, we get a situation similar to the one depicted in Figure 6 (A and the connection between C and D are faulty).

Six rays of CEs emanate from A and the $\underline{C}$ messages from C and D travel in virtual straight lines through these CEs. All of the PEs in these lines are also turned into CEs. The resulting configuration of CEs is shown in Figure 6.

2.5 Correctness of the Scheme

We prove the correctness of the scheme for single faults and for multiple faults that occur sequentially. In order to formalize the concept of an hexagonal array, we define an *$n*n$ hexagonally-connected array* (HCA) as a two-dimensional array of processors, with n PEs to a side, and connected as follows: If $P(i,j)$ denotes the PE in row i and column j, then $P(i,j)$ is linked to $P(i,j+1)$, $P(i+1,j)$ and $P(i+1,j+1)$, for $i,j=1,2, \ldots ,n-1$.

Proposition: Assume the array is restructured after a single fault. Then:

(a) If the fault is in a link, the resulting configuration contains an $(n-1)*(n-)$ HCA.

(b) If the fault is in a PE, the resulting configuration contains an $(n-2)*(n-2)$ HCA.

Proof: See Appendix.

Corollary: Assuming an initial $n*n$ HCA, if p connections and q PEs become faulty in sequence, then the resulting configuration will contain an $(n-p-2q)*(n-p-2q)$ HCA. ■

A program simulating the effect of single and multiple faults has been written. Figures 4–6 are sample outputs from the simulating program. Notice that in Figure 4, the resulting configuration contains a 9×9 HCA, while in Figure 5 with a faulty PE, the resulting configuration contains an 8×8 HCA only.

The results of the simulations for both sequential and *simultaneous* multiple faults are in agreement with the corollary. This leads us to conjecture that the statement of the corollary also holds for simultaneous faults. Note that when a CE becomes faulty, the logical effect is the same as several simultaneous link faults.

3. CONCLUSION

A scheme for fault-tolerance in hexagonal arrays has been suggested. Its main advantage is that it makes the restructured array (following the identification of the faulty PE or communication link) transparent to the various algorithms utilizing the hexagonal array.

4. REFERENCES

[1] D. Fussel and P. Varman. Fault-tolerant wafer-scale architectures for VLSI. *Proc. of the 9th Annual Symp. on Comp. Arch.*, May 1982.

[2] D. Gordon, I. Koren, and G.M. Silberman. Embedding tree structures in VLSI hexagonal arrays. *IEEE Trans. on Computers,* Vol. C–33, pp. 104–107, Jan. 1984.
[3] J.W. Greene and A. El Gamal. Configuration of VLSI arrays in the presence of defects. *Journal of the ACM,* Vol. 31, No. 4, pp. 694–717, Oct. 1984.
[4] I. Koren. A reconfigurable and fault-tolerant VLSI multiprocessor array. *Proc. of the Eighth Annual Symp. on Comp. Arch.*, pp. 425–441, May 1981.
[5] I. Koren and G.M. Silberman. A direct mapping of algorithms onto VLSI processor arrays based on the data flow approach. *Proc. of the 1983 Internl. Conf. on Parallel Processing,* pp. 335–337, August 1983.
[6] I. Koren and M.A. Breuer. On area and yield considerations for fault-tolerant VLSI processor arrays. *IEEE Trans. on Computers,* Vol. C–33, pp. 21–27, Jan. 1984.
[7] I. Koren and D.K. Pradhan. Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems. *Proc. of IEEE, Special Issue on Fault-Tolerance in VLSI,* Vol. 74, No. 5, pp. 699–711, May 1986.
[8] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, MA, 1980.
[9] H. Mizrahi and I. Koren. Evaluating the cost-effectiveness of switches in processor array architectures. *Proc. of the 1985 Internl. Conf. on Parallel Processing,* pp. 480–487, August 1985.
[10] A.L. Rosenberg. The Diogenes approach to testable fault-tolerant arrays of processors. *IEEE Trans. on Computers,* Vol. C–32, pp. 902–910, Oct. 1983.
[11] L. Snyder. Introduction to the configurable highly parallel computer. *Computer,* pp. 47–56, January 1982.

APPENDIX: PROOF OF THE PROPOSITION

The proof is a straightforward examination of a number of typical cases. Part (a) is proved first, and (b) will be shown to follow from (a). The different cases one has to examine for (a) are:

(1) The faulty link is parallel to one of the sides of the HCA.
(2) The faulty link is on the main diagonal.
(3) The faulty link is not on the main diagonal but parallel to it.

One can see that the rays of CEs partition the PEs into at most three groups of PEs. We shall only examine case (3) in detail, the other cases follow along similar lines. Figure 7 is an example of case (3) and will serve as an illustration of the proof. The indices above the PEs are the original ones on an 8*8 HCA. The indices below are the new ones after restructuring. We assume the fault to be below the main diagonal (similar proof for a fault above the diagonal).

We call the three groups of remaining active PEs A, B, and C, where A is the group containing P(1,1), B contains P(n,1), and C contains P(1,n). Let P(i,j) be an active PE in the reconfigured array, where i,j are its indices in the original HCA (shown above the PE in Figure 7). We assign to P(i,j) new indices (i',j') as follows:

$$(i',j') = \begin{cases} (i,j) & \text{if } P(i,j) \in A \\ (i-1,j) & \text{if } P(i,j) \in B \\ (i-1,j-1) & \text{if } P(i,j) \in C \end{cases}$$

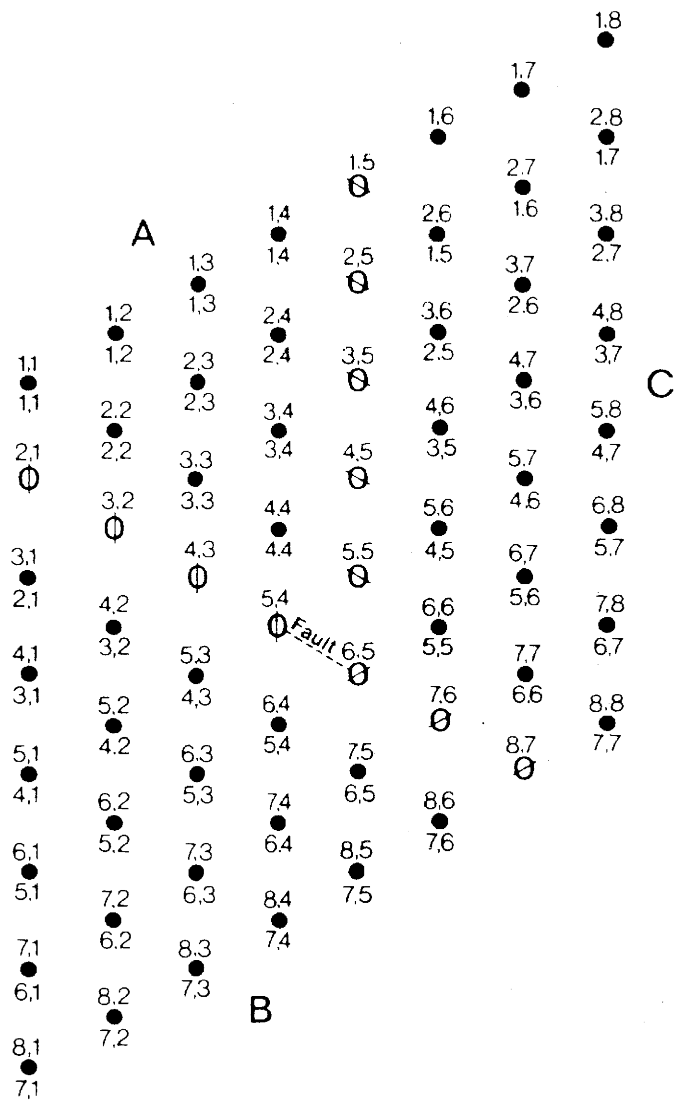

Figure 7 An hexagonally-connected array with its PE indices before and after restructuring.

It is clear that within each of the three groups A, B, and C, all links remain as before and satisfy the requirement for an HCA. In order to prove that the reconfigured array is an HCA, we have to show that the connections between the groups (across the CEs) are as required.

Consider a PE $P(i,j)$ on A's rightmost column. $P(i,j)$'s new indices are $(i',j') = (i,j)$. Going from $P(i,j)$ in direction 1, one crosses two CEs ($P(i,j+1)$ and then $P(i+1,j+1)$) and comes out at $P(i+1,j+2)$ of C's leftmost column. The new indices of this PE are $\{(i+1)', (j+2)'\} = (i,j+1) = (i',j'+1)$, which is exactly as required when going from (i',j') in direction 1. Similar arguments show that all other connections between A, B, and C are also as required for an HCA.

The new indices (i',j') satisfy the following conditions: They start at (1,1); they differ by at most 1 from (i,j) and so they run up to $n-1$; all required connections for an HCA hold. Consequently, the remaining active PEs contain an $(n-1)*(n-1)$ HCA. The proof for cases (1) and (2) is similar.

Part (b) follows immediately from (a) by observing that the pattern of "rays" of CEs from a faulty PE is a successive superposition of two sets of rays similar to the ones resulting from a faulty link. According to (a), each such set of rays reduces the size of the HCA by at most 1, so we remain with an HCA of size $(n-2)*(n-2)$ at least.

4. REFERENCES

[1] D. Fussel and P. Varman. Fault-tolerant wafer-scale architectures for VLSI. *Proc. of the 9th Annual Symp. on Comp. Arch.*, May 1982.

[2] D. Gordon, I. Koren, and G.M. Silberman. Embedding tree structures in VLSI hexagonal arrays. *IEEE Trans. on Computers,* Vol. C–33, pp. 104–107, Jan. 1984.

[3] J.W. Greene and A. El Gamal. Configuration of VLSI arrays in the presence of defects. *Journal of the ACM,* Vol. 31, No. 4, pp. 694–717, Oct. 1984.

[4] I. Koren. A reconfigurable and fault-tolerant VLSI multiprocessor array. *Proc. of the Eighth Annual Symp. on Comp. Arch.,* pp. 425–441, May 1981.

[5] I. Koren and G.M. Silberman. A direct mapping of algorithms onto VLSI processor arrays based on the data flow approach. *Proc. of the 1983 Internl. Conf. on Parallel Processing,* pp. 335–337, August 1983.

[6] I. Koren and M.A. Breuer. On area and yield considerations for fault-tolerant VLSI processor arrays. *IEEE Trans. on Computers,* Vol. C–33, pp. 21–27, Jan. 1984.

[7] I. Koren and D.K. Pradhan. Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems. *Proc. of IEEE, Special Issue on Fault-Tolerance in VLSI,* Vol. 74, No. 5, pp. 699–711, May 1986.

[8] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA, 1980.

[9] H. Mizrahi and I. Koren. Evaluating the cost-effectiveness of switches in processor array architectures. *Proc. of the 1985 Internl. Conf. on Parallel Processing,* pp. 480–487, August 1985.

[10] A.L. Rosenberg. The Diogenes approach to testable fault-tolerant arrays of processors. *IEEE Trans. on Computers,* Vol. C–32, pp. 902–910, Oct. 1983.

[11] L. Snyder. Introduction to the configurable highly parallel computer. *Computer,* pp. 47–56, January 1982.