

Towards a Strongly Fault Tolerant VLSI Processor Array

Sudarshan P. B.

Srivatsan P.

Department of Electrical and Electronics Engineering
Sri Venkateswara College Of Engineering, Chennai, India

e-mail: sudarshanpb@acm.org

Tel: (91)-44-28282084

ABSTRACT

In this paper a novel methodology to achieve fault tolerance in VLSI Array Processors is proposed. A “Fence” based approach is adopted in which the logic array is partitioned and spares are distributed along the boundary of the active array. The emulator as in conventional fault tolerance techniques takes care of fault mapping and re-configuration. The latency, reconfiguration interconnect length and fault coverage issues, which are critical areas in VLSI Arrays, are discussed in detail.

Index Terms—VLSI Array Processors, Emulator, Fence Approach, Reconfiguration, Processing Elements.

I. INTRODUCTION AND PREVIOUS WORK

Fault tolerance techniques for VLSI array processors has been a critical area of research [3][10][2][5]. It is now possible to build an integrated system on a chip or wafer by interconnecting a large number of identical elements such as memory cells and processors. Yield degradation is a problem of important significance as device geometries shrink and the density of VLSI systems increase. One important method of increasing yield is through Reconfigurable VLSI which employs redundancy that can be used to replace faulty modules [9]. This technique has already found practical application in large random access memories where spare rows and columns of memory cells together with their decoders are programmed to repair faulty memories. In all reconfigurable designs for fault tolerance, implementation of the actual reconfiguration algorithm is of crucial importance. Since there are a limited number of spares the goal is to effectively utilize these spares to repair the system whenever it is repairable with minimal computations. This necessitates the implementation of an effective repair algorithm which takes the current system configuration and fault patterns as inputs and generates a repair solution.

An early method to achieve fault tolerance was by gracefully degrading the working area of the array by eliminating the faulty Processing Elements (PEs) [4]. But

this also meant reduced functionality because of the incapability of these techniques to support the same problem size during the occurrence of faults. Another popular approach was Triple Modular Redundancy. This technique though simple involves the addition of significant overhead through simple triple replication of the PE. This led to the evolution of techniques which incorporated structural redundancy [6]. Even though this involved the addition of considerable overhead this has proved to be the most efficient and common technique. The tradeoff between yield and number of spares is an important consideration even today.

There are generally two approaches adopted for Fault Tolerance (FT)- Algorithmic and Architectural. Algorithmic approaches towards FT are very complex and increase with the complexity of the problem. Architectural approaches in recent times are gaining popularity because, it does not need to take into account the complexity of the algorithm and also due to its ease of implementation. The interstitial redundancy technique for Processor Arrays which proposes adding redundancy for every pair of row and column. This involves adding redundancy with poor utilization.

Kung and Lam[3] proposed an index mapping architectural approach for purely VLSI Arrays. In this approach the logical indices were mapped onto the fault free PEs of the target array. This approach gave less importance to locality as a factor to be considered in Fault Tolerance issues. This algorithm involves use of modified interconnection paths along with certain delay paths incorporated in them.

The above mentioned papers all had their own drawbacks. The method proposed in [1][6] does not consider that faults may occur locally and hence has very poor spare utilization and also leads to considerable wastage of fault free PEs. A major limitation of [3] is the design constraints which require the design of highly complicated switch networks to support complex interconnections which arise to the index mapping approach.

The approach of adding redundancy is currently the

most popular approach. This redundancy technique is employed in most papers(fig.1). Our paper tries to overcome

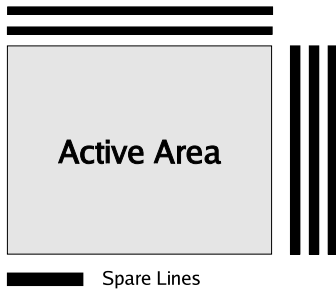


Fig. 1. Conventional Redundancy Technique

the drawbacks in existing approaches and aims to achieve the following- reduced interconnect length during reconfiguration, reduced fault-latency, maximum use of spares, reduced design complexity. All the above mentioned factors are highly critical with respect to VLSI Arrays.

Our paper is organized in the following manner. Section two gives an insight into our proposed methodology, its advantages and limitations and the working of the algorithm. Section three explains the proposed algorithm by means of an example. The final two sections deal with the experimental results and conclusions.

II. OUR APPROACH

A. The Emulator

The reconfiguration during the occurrence of faults, we propose, is controlled by a module external to the processing array. This module we call the Emulator must have memory to store configurations which must be updated during the course of operation. The action is similar to that of an embedded microcontroller. This Emulator stores the state of the PEs in the logic array and designates special notation for fault free and faulty PEs, say 0 and 1 respectively. The fault list is updated at regular intervals of time. The Emulator also acts as a Reconfiguration Controller and initiates fault reconfiguration and spare allocation processes. This unit determines configuration changes that must be made to the working area for the faulty PEs to be bypassed and reconfigured to the fault free ones. Due to the need of the Emulator to perform these operations, they also have the ability to suspend the processor clock for certain short intervals of time for safe fault reconfiguration. The Emulator design must be such that it is strongly fault tolerant, but a fault tolerant design of this unit is well beyond the scope of discussion.

B. The Fence-Emulator Approach

As in conventional approaches we will make the following assumptions in our approach- the PEs are self-

testing and the interconnect networks are fault free-that is the fault of the interconnect network is assumed to be the fault of the associated PE. The difference between existing methods and our methods is that, we do not assume the spare PEs to be fault free. In case of a fault occurring in any of the spare PEs, then the emulator designates the spare PE to be unsuitable for reconfiguration.

Our fault tolerant approach is designed for rectangular arrays of size $M \times N$. Here M is the number of rows and N the number of columns. The spares in our approach are placed symmetrically around the active array. These spares surround the active area like a fence. This is why our approach is otherwise known as the ‘‘Fence-Emulator Approach’’.

The logic array of size $M \times N$ is partitioned into four logic sub-arrays as per the given algorithm. The underlying assumption in our approach is that the number of rows is lesser than the number of columns. If that is not that case then the N is assigned to be the number of rows and M the number of columns. By logic partitions we mean they are not physical partitions. These partitions are for the purpose of designing the Algorithm. These partitions are recognized by the Emulator.

C. Test methodology- Advantages and Limitations

Our approach can also be compared and contrasted with a fault tolerant approach for Field Programmable Gate Arrays (FPGAs) presented in [8]. In the approach the spares are arranged vertically and horizontally. The spares are made to rove along the active area of the array. The scan process is completed by N movements. The algorithmic complexity of the methodology proposed is considerably significant. The latency also increases in such an approach due to the fact that an occurrence of a fault cannot be detected within the next rove. The latency thereby increases linearly with increase in size of the array.

In our approach the logic array is partitioned into four quads. By partitioning the logic array, the latency, which is the time between fault occurrence and reconfiguration, is reduced drastically. In the normal case of an $N \times N$ logic array the reconfiguration occurs after N row scans or column scans. In our approach this reduces to $N/4$ row (or) column scans. Further the emulator will pre-compute faulty processors reconfiguration, as it is updated at regular intervals. This way our methodology achieves speed up over other approaches.

The switching process for reconfiguration, in our methodology is also straight forward. Only horizontal and vertical routers are employed in contrast with [3][5] where complex switching topologies are employed. This

by itself could also be a limitation, because certain pattern of faults, such a fault clusters may not be efficiently reconfigured. But extensive simulations performed show that even presence of clustered faults in our algorithm the fault reconfiguration efficiency is comparable to existing approaches.

The Spare utilization factor (SUF), which is the ratio between the number of spares utilized and the spares present, is a highly critical parameter in reconfigurable systems. In our approach, the spares are utilized efficiently, due to the arrangement of the spare PEs and the nature of the algorithm which searches for spares in adjacent quads before classifying the fault as irreparable.

The interconnect reconfiguration length which governs various factors including delay is also reduced in our approach. The fence approach by arranging the spares in the manner described and partitioning the array into four sub-systems, tries to reconfigure the fault to the spares present in the sub-system/ quad initially. Only when there is no spare present in the quad, the search process for spares in the adjacent quads is performed. Also within the quad preference is given to either the row (or) column spare, depending on their proximity to the fault. This unique feature of the proposed methodology is that the need for highly complex designs for interconnects as those presented in [7] is not necessary.

D. Working of the Algorithm

Algorithm: Fence-Emulator

- 1) **Input fault list:** The locations of the faulty elements are input to the emulator.
- 2) **Partition Algorithm:** The partition algorithm is performed by the emulator to get four logic sub-arrays (For $M \times N$ array where N is even and M is odd).
 Row Partitions \rightarrow 1 to $(N/2)$, $((N/2) + 1)$ to N
 Columns \rightarrow 1 to $((M + 1)/2)$, $((M + 1)/2 + 1)$ to M
- 3) **Fault Scanning and Reconfiguration:** The partitioned sub-arrays are scanned simultaneously.
Scanning: Fault scanning is carried out from the row nearest to the spare row in each sub-array.
Reconfiguration: Faults are reconfigured to the nearest spare (row/column). If no spare is located in the sub-array, search adjacent sub-arrays for availability of spares in the same row/column. If spares are present in both the adjacent sub-arrays, the fault is reconfigured to the nearest spare. If no spares are present the fault is classified as *irreparable*.
- 4) **Emulator Update:** The emulator is refreshed with the current status of the logic array.

- 5) **Iterate:** Repeat the above steps until all the faults are covered.

In our approach the reconfiguration process begins simultaneously in each of the four quads. The scan begins from the spare row in each quad. The emulator maps the faulty PEs at regular intervals of time. When a fault is detected in any of the first row PEs in each quad the emulator directs reconfiguration. The reconfiguration is carried out by following the following heuristic. The emulator on the location of a fault pre-computes the logic distance between the faulty PE and the row spare and column spare present in that quad. The reconfiguration on the direction of the emulator is carried out to the nearest spare.

This process is repeated continuously till each one the rows in the four quads are scanned for reconfiguration. Whenever the Emulator finds that there are no faults in each of the four corresponding rows, it skips to the next row for reconfiguration. The most unique feature to our algorithm is that whenever the Emulator encounters a fault in a particular row that cannot be reconfigured, it also searches the corresponding row and column in the adjacent quads for a spare. If a spare is present just in anyone of the corresponding quads then its reconfigured to that spare. Else the Emulator computes the shortest distance between the faulty PE and the two spares and reconfigures to the nearest spare. If there are no spares in both of these adjacent quads then the fault is classified as irreparable.

III. EXAMPLE

We consider a 6×8 active area, surrounded by a single “fenced” spare row and column to explain our algorithm. The fault list was generated in random and 18 faults were generated (fig 2). The logic array is partitioned by the Emulator into four logic sub-arrays. According to the algorithm, the emulator initially scans in each sub-array the row closest to the spare row. The reconfiguration process takes place, wherein the fault is reconfigured to the spare closest to it. In this case the fault at location (1, 4) is reconfigured to the row spare as it is closer. In this fashion all 4 sub-rows in all the four sub-arrays are reconfigured for faults. The Emulator then moves on to the second row in each sub-array, and a similar process takes place. In this case the fault at location (5, 5) is unable to find a spare PE in its sub-array. The emulator then searches the adjacent sub-arrays for spare PEs. The Emulator then locates a spare in the second quad. Since there is no spare in the fourth quad, the emulator reconfigures to the spare in the second quad. If the Emulator had located a spare in both the second and fourth quad, it would have reconfigured to the spare in the quad closest to it. The same process is

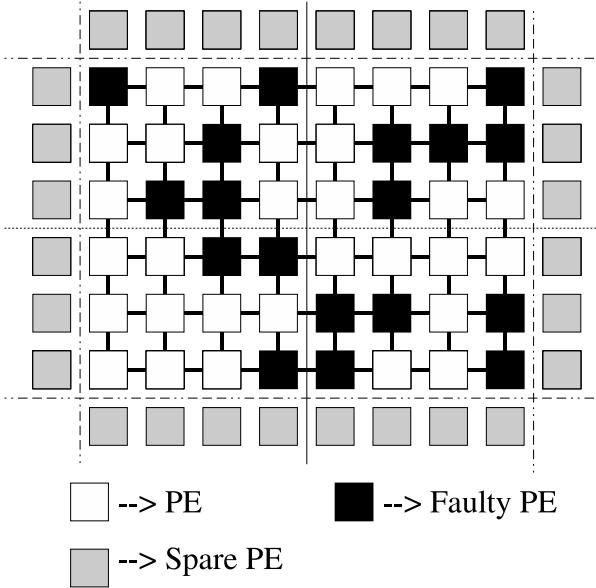


Fig. 2. Initial Fault Locations (6×8 active area, with 18 faults)

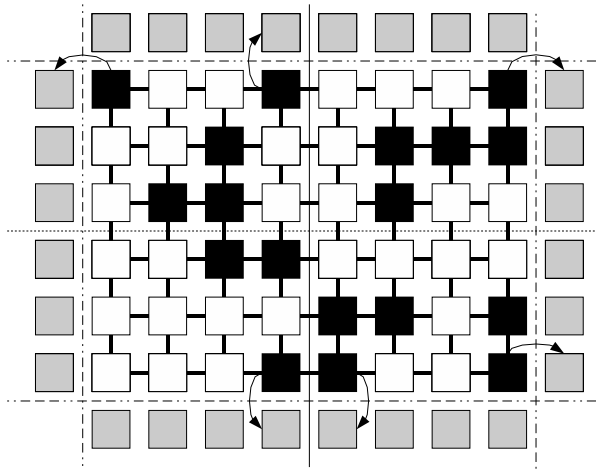


Fig. 3. Reconfigurations after Scanning Row 1 of each quad

continued in the third row and until all faults are reconfigured. The figures given below show the initial fault location and reconfigurations that are taking place after each row scan. Each figure retains the reconfiguration (if any) done in the previous step so as to make them clear.

IV. EXPERIMENTAL RESULTS

The simulations were carried out using the C++ programming language. The program written was extensive and was made to support any size of logic array with a single “fence” of spares. The number of faults and the number of spares was also made defined. Faults were generated using the in-built pseudo-random number generator function in the standard C++ library. The simula-

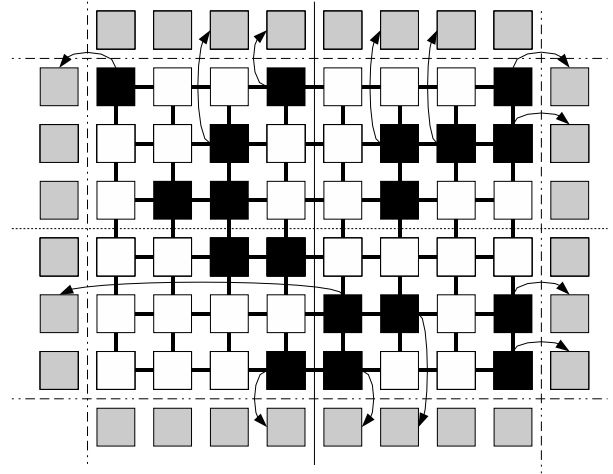


Fig. 4. Reconfigurations after Scanning Row 2 of each quad

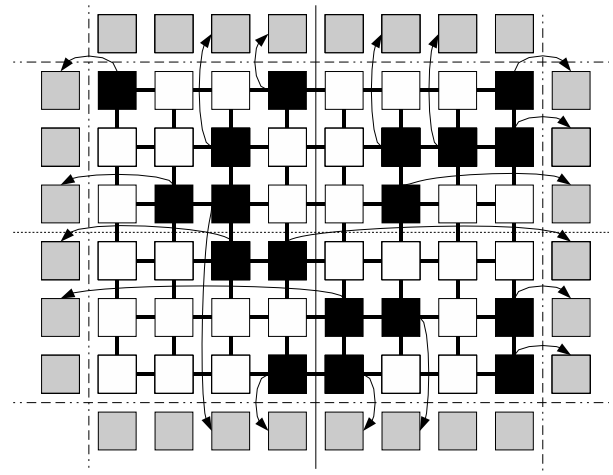


Fig. 5. Reconfigurations after Scanning Row 3 of each quad

tions were made to run repeatedly for generation of various fault patterns and also for different sizes of logic array. The following modes of faults were simulated: fixed size of array with variable and fixed number of faults. Since the faults were generated randomly, we feel that even for fixed array size with fixed number of faults, our simulation covered various possible fault locations (including multiple faults in a single row/column). The results are shown in Table I. The figures given in brackets indicate the number of spares for the array size shown according to the “fence” approach for positioning the spares. The results shown were obtained by running the program for 50 times for each combination of array size and faults and averaging the results. As can be seen from the results, the percentage of spares utilized i.e. the spare utilization factor is high. This gains significance in the light of current constraints in maintaining the chip area as well a trying to

TABLE I
SIMULATION RESULTS

Array Size	Faults at start	Faults at end
512 × 1024 (3076)	1000	30
	2000	249
	3076	851
1024 × 1024 (4100)	1000	9
	2000	153
	3000	482
1024 × 1024 (6148)	4100	1057
	2000	36
	3000	171
	4000	501
	5000	937
	6148	1642

improve its fault tolerance.

V. CONCLUSIONS

The algorithm was found to be successful for all sizes of logic array, various fault patterns and different number of faults. VLSI processor array structures become more complex by the advancement of design and production technologies. Continual improvement of the algorithm and development of even more efficient algorithms for repair analysis will be a part of our future work. Our future work will also involve developing an improved algorithm for the currently popular approach of row/column elimination. This technique involves considerable wastage of fault free PEs. We are also currently carrying out analysis with respect to fault pattern generation which reflects the actual fault map and develop the algorithm for the fault patterns generated.

ACKNOWLEDGEMENTS

The authors would like to thank Bhaskaran P.P, Department of Electrical and Electronics- Sri Venkateswara College of Engineering, for taking his time out to help the authors simulate the algorithm extensively. Sudarshan would like to thank Prof. N. Venkateswaran for his guidance and valuable suggestions.

REFERENCES

- [1] Fan R.K. Chung, Leighton F.T., and Rosenberg A.L. Diogenes: A methodology for designing fault-tolerant VLSI processing arrays. In *Proceedings of the Fault Tolerant Computing Symposium*, pages 26–32, 1983.
- [2] Fan R.K. Chung, Leighton F.T., and Rosenberg A.L. Efficient spare allocation in reconfigurable arrays. In *Proceedings of the 23rd Design Automation Conference*, pages 385–390, 1986.
- [3] Kung H.T. and Lam M.S. Fault tolerant VLSI systolic arrays and two level pipelining. In *Journal of Parallel and Distributed Processing*, pages 32–63, 1984.
- [4] Fortes J.A.B. and Raghavendra C.S. Gracefully degradable processor arrays. In *IEEE Transactions on Computers*, number C-34, pages 1,033–1,044, 1985.
- [5] Greene J.W. and Gamal A.E. Configuration of VLSI arrays in the presence of defects. In *Journal of the Association for Computing Machinery*, number 31 vol.4, pages 694–717, 1984.
- [6] Belkhale K.P. and Banerjee P. Reconfiguration strategies for VLSI processor arrays and trees using a modified diogenes approach. In *IEEE Transactions on Computers*, number 41,1, pages 83–96, 1984.
- [7] Kurian L., Brewer D., and John E. Design of a highly reconfigurable interconnect for array processors. In *Proceedings of the 8th International Conference on VLSI Design*, pages 321–325, 1995.
- [8] Abramovici M., Stroud C., Wijesuriya S., Hamilton C., and Verma V. Using roving stars for on-line and diagnosis of FPGAs in fault-tolerant applications. In *Proceedings of the 1999 International Test Conference*, pages 973–982, 1999.
- [9] Chean M. and Fortes J.A.B. A taxonomy of reconfiguration techniques in fault-tolerant processor arrays. In *IEEE Survey and Tutorial Series*, pages 55–69, 1990.
- [10] Negrini R., Sami M.G., and Stefanelli R. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*. The MIT Press, 1989.