

Effective Realization of On-chip Fault-tolerance Utilizing BIST Resources

SUMIT DHARAMPAL MEDIRATTA, JEFFREY DRAPER

USC Information Sciences Institute
Marina del Rey, California-90292
USA
sumitm@isi.edu, draper@isi.edu

Abstract: - Widespread reliability challenges are expected for 65nm and below VLSI fabrication technologies. On-chip fault-tolerance solutions are required to counter reliability challenges. A new post-fabrication reconfigurable and scalable approach of achieving on-chip fault-tolerance, using built-in-self-test (BIST) resources, has been proposed by the authors. This paper gives more insight into the proposed approach by considering issues pertaining to its efficient and effective realization and giving a methodology of using the proposed approach for desired design objectives. It also provides effective methods to address BIST faults and the split-brain problem. The proposed approach reduces production cost, implementation overhead and time-to-market; increases reusability, post-fabrication reconfigurability and productivity; and is scalable across multiple VLSI processes and feature sizes. This will result in obvious advantages of yield enhancement and prolonged lifetime of VLSI chips as well.

Key words: - On-chip fault-tolerance, reliability, multi-core, system-on-chip, built-in-self-test (BIST)

1 Introduction

One of the major problems being faced by designers of system-on-chip and multi-core architectures (e.g. [1][2]) is to achieve effective on-chip fault-tolerance without sacrificing too much area, energy and performance. According to the International Technology Roadmap for Semiconductors (ITRS), widespread reliability challenges are expected in near-term VLSI fabrication technologies (65nm and below) because of the evolutionary changes in scaling current materials and devices [3] and revolutionary changes associated with new materials and devices. Introduction of multiple materials, processes and structural changes in a short period will increase the difficulty of understanding and controlling failure modes. So, fault-tolerance should be considered a necessity, rather than as a feature.

Considering unknown faults, less fault diagnosis and characterization time before production, and unknown failure rates as an objective, a post-fabrication programmable fault-coverage (FC) and fault-type fault-tolerance method of utilizing on-chip BIST resources was proposed [4]. These BIST resources are expected to be already present on VLSI chips because they are becoming standard in the production environment for manufacturer and boot-up testing. The proposed approach reduces production cost, implementation

overhead and time-to-market while leaving the trade-off analysis of maximizing FC and minimizing performance overhead to be programmable until post-fabrication. This is in addition to increasing reusability, post-fabrication reconfigurability and productivity, as well as enabling scalability across multiple VLSI processes and feature sizes.

This paper considers and provides solutions to problems pertaining to an effective and efficient realization of the proposed technique. Efficient solutions are provided to various prominent problems like faults in BIST resources themselves, BIST-node path failures, split brain problem etc. Limitations of the proposed approach are also discussed. A methodology for achieving desired design objectives is also proposed.

Section 2 gives a background on the proposed approach of BIST resource utilization, section 3 explores the issues to be considered for its effective and efficient realization, section 4 gives the methodology of using the proposed approach, and finally section 5 concludes this paper.

2 Approach of BIST Utilization

The proposed approach is to perform fault-detection and isolation dynamically (during system operation) using available BIST resources. Our approach uses

rollback and recovery mechanisms as an integral instrument to provide correct results in case of failures, and thus achieves fault tolerance. The novelty of our approach lies in giving BIST resources a new meaning of existence and more significance, and thus rollback and recovery are simply facilitators in achieving this purpose. The following paragraphs delineate the basic concepts necessary to understand the remainder of the paper. Refer [4] for further details.

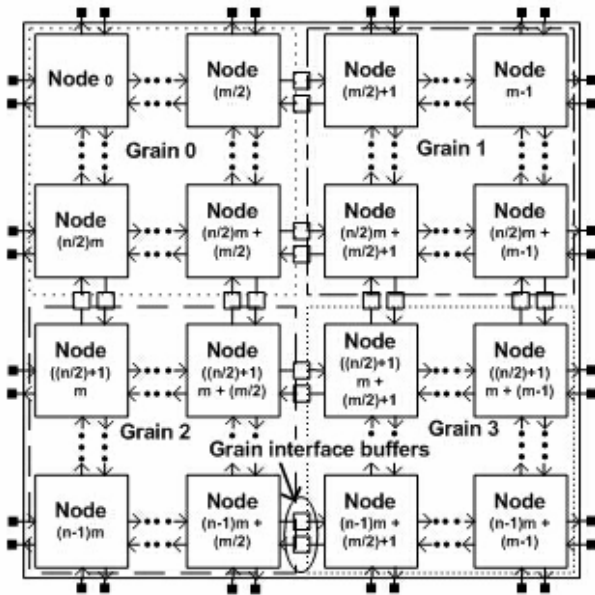


Fig.1 Concept of *grain* in a multi-core architecture

2.1 Basic Concept

The basic idea of the proposed approach is to periodically test system entities for permanent failures using BIST resources, and recover using a checkpoint of previous system intact state in case of any failure. This can be achieved by scheduling test and recover phases during system dynamic execution. After each test and recover phase, each system entity will commit its results and save a checkpoint, unless it fails the test or receives a recover message.

2.2 Concept of Grain

A grain is defined to be a block or collection of individual entities, which are rolled back and undergo recovery together as one unit upon finding any failure in that collection during *test and recover phase* and communicate with each other only after successfully committing their results (Fig. 1). Grain size can be smaller or greater than a whole chip depending on particular system architecture and application. The

concept of a grain is defined to minimize the overhead incurred in the recovery process.

3 Hardware Realization Issues

Issues pertaining to effective and efficient realization of the proposed approach are considered in this section.

3.1 BIST Resources and Node-BIST Paths Fault-tolerance

3.1.1 BIST resources

BIST resources have to be very robust because they detect faults. False negative test failures should be minimized, if not eliminated, to achieve desired functionality. BIST resources can be made fault-tolerant by using arithmetic code, algorithm-based fault tolerance (ABFT) or redundancy techniques. Linear pattern generators [5] are desirable because they can be made fault tolerant using arithmetic codes easily. This is because most arithmetic codes are based on finite group theory that is closed under one particular operation [6]. Although, care should be taken that a node's responses to the potential erroneous test vector (but in error correcting limit of arithmetic code) should be interpreted as fault-free by the BIST resource. Or else, test vector should be checked and corrected for errors before application to the node. Additionally or alternatively, BIST resources should cross verify each other in a periodic or pseudo-random fashion (direct paths from pattern generator (PG) of one node to response analyzer (RA) of another node Fig. 2). The following actions can be taken upon finding a BIST resource faulty: -

- a). Another BIST resource assumes the responsibility of system entities under the responsibility of the faulty BIST (multiplexers after PG and before RA in Fig. 2). This is equivalent to a reconfiguration of the system BIST structure.
- b). All entities under the responsibility of the faulty BIST are also considered faulty and simply discarded. This has a negative impact on the overall system performance and yield, but results in simpler and lesser demands on the reconfigured system BIST architecture.

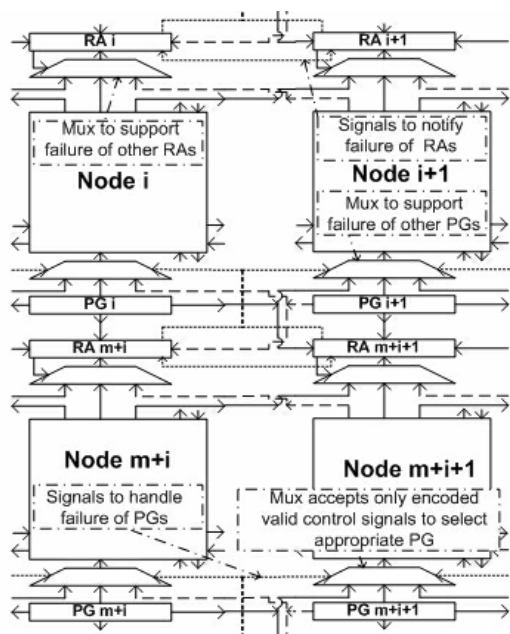


Fig.2 Support for BIST fault detection and failures.

3.1.2 Node-BIST paths

If a BIST encounters a situation where its paths from and to nodes under its responsibility are faulty, then again it can follow one of the two approaches above (section 3.1.1(a) or 3.1.1(b)) in that case. Node-BIST path failures can be detected by running test vectors for this particular case by the BIST after a threshold number of node test cases fail.

3.2 Recovery in case of Node Failures

BIST notifies the grain of the node failure, if any, as soon as it completes testing of the node. It does so by declaring it unreachable. In case of on-chip communication link failures, recovery is easy because previous checkpoint data is available at the node. But, in case of node failures, checkpoint data is lost either if DMA (direct memory access) to node's memory is not provided to its neighbors (assuming node's memory is still intact), computation has not been replicated (and preferably physically distributed) or coherent system state (integrated checkpoint information of all nodes) is not periodically written to a common (preferably external) storage device. To circumvent this problem, one or all of the discussed countermeasures can be provided and then any of the progressive or regressive recovery approaches discussed in [4] can be used.

3.3 Split-brain Problem

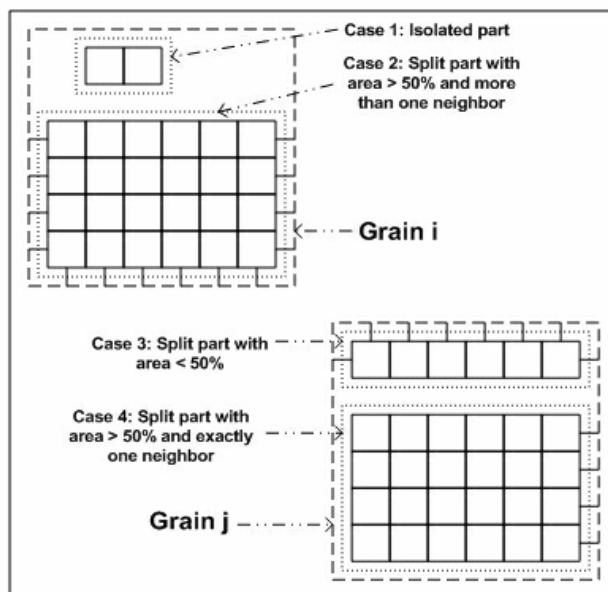


Fig.3 Split-brain problem illustration.

An issue called the *split-brain problem* [7] may be encountered in a grain during system reconfiguration. This problem occurs because different groups of nodes may arise that do not have any knowledge about the existence of other nodes in a grain because of an unreachability problem (Fig. 3). So, each group will replicate the computation assigned to other group and distribute it on group's resources, assuming wrongly that other group has failed totally.

This problem can be taken care of by isolating the split parts of a grain that have no neighbors during reconfiguration (case 1 in Fig. 3). Also, part of grains whose area (proportional to number of nodes) is less than a pre-specified percentage (e.g. 50%) of the previous original grain size and have 1 or more neighboring grains can be merged into neighboring grains (case 3 in Fig. 3). Grain parts with area greater than or equal to a pre-specified percentage and 1 or more neighbors can take the place of the original grain (case 2 in Fig. 3), thus conserving the total number of grains. The special case of a grain with 1 neighbor and area more than a pre-specified percentage (case 4 in Fig. 3) should be dealt with carefully because treating it as a separate grain will cause hot spots on its one connected edge that becomes the bottleneck in its communication.

One way of implementing the above functionality of calculating area in hardware is by requiring every node at the bottom of each split part to broadcast its identification along its column in the grain and then sending a packet from the top left most and right most

nodes to the top right most and left most nodes respectively, along the top grain boundary (Fig. 4(a)).

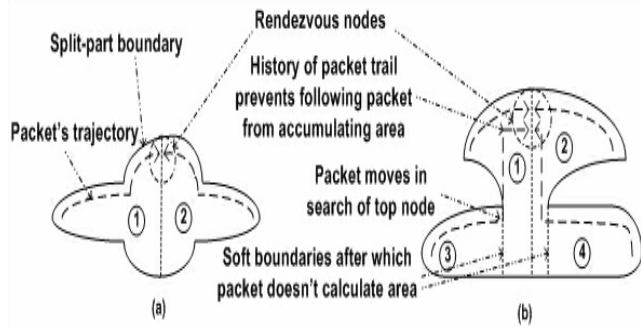
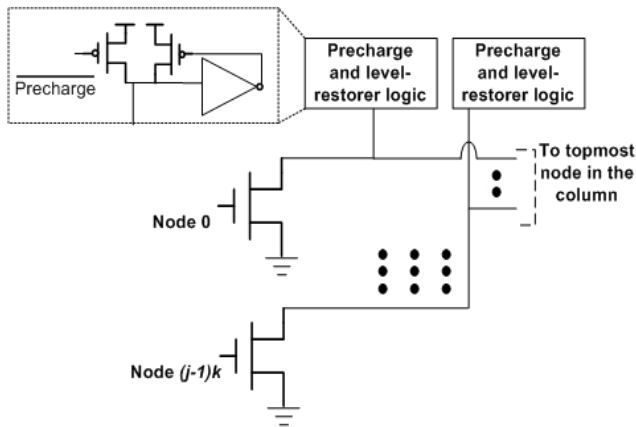


Fig.4 Concept of area and shape determination.



Circuit deployment table				
	PHASE 1	PHASE 2	PHASE 3	PHASE 4
010 (over 3clks)	Bottom node	Leftmost or Rightmost node	Leftmost node	Neighbor node faulty
101 (over 3clks)	Inverse of above	Inverse of above	Rightmost node	Neighbor is in another grain

Fig.5 Support for area and shape determination.

The top grain boundary is defined to consist of nodes whose top neighbor is a node that is either in another grain or faulty. A packet traveling across the top grain boundary can simply sum up the difference of top node and bottom node identification in every column to determine area. For handling the special case depicted in Fig. 4(b), packets (3 & 4) have to travel in some regime without calculating any area and to communicate their calculated area to a rendezvous node by first travelling upwards and then following a trail of packets (1 & 2) responsible for area calculation of that region. To ascertain the shape of the split part (also important for determining resulting action of isolation,

merger or maintaining identity), each node along the bottom, leftmost and rightmost boundaries of the split part additionally send the part's neighbor connectivity status (whether neighbor node is in another grain or faulty node) to the top most node in the corresponding columns (Fig. 5). This approach will add its worst-case overhead only in case of worst-case scenarios because of its dynamic nature.

One approach of implementing connectivity communication and bottom node identification propagation is to use long wires (Fig. 5). Some encoding scheme should be used on these lines (Fig. 5) for tolerance from single event transients and detection of some stuck-at faults. Another approach is to send a packet towards up direction from every bottom node to corresponding topmost node, which carries the bottom node's identification and connectivity status, and additionally logs connectivity status in leftmost and rightmost boundaries.

3.4 Fault-tolerant communication

A fault-tolerant communication mechanism is required to route messages after a declaration of resources as erroneous, if stand-by hardware is not provided to keep resources functional. This means the routing algorithm employed has to remain connected in presence of faults and provide paths when the network topology changes.

For a comparison, the proposed approach, Triple Modular Redundancy (TMR) and arithmetic code based techniques achieve the same functionality of error detection and correction. The proposed approach achieves this functionality in time rather than space, as is the case with TMR and arithmetic code based approaches. The proposed approach reconfigures around the erroneous resource instead of keeping erroneous resources functional (by correcting errors) by other approaches. This reconfiguration is not required, and can be avoided by providing equivalent stand-by hardware resources used as required redundant resources in other approaches (e.g. extra hardware used in TMR). Mathematically, reliability of stand-by systems is, in general (for a not very bad switch failure rate), better than that of parallel systems [7]. Moreover, this reconfiguration step is required in other approaches as well, when their capability reduces to error detection (not error correction). The option of reconfiguration at an early stage in the proposed approach is desirable because of different fault environments.

3.5 Effective and Efficient Application of Testvectors

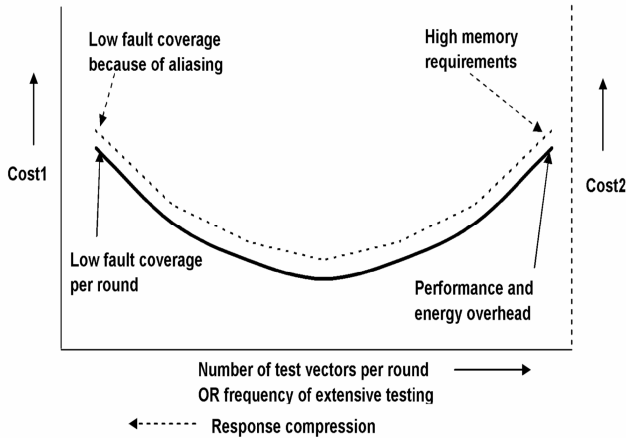


Fig.6 Overhead cost function for test application.

Test vectors can be applied in a parallel manner instead of using serial scan chains to enable *at-speed* testing. Further test vector compaction and response compression techniques can be applied to reduce the test application time and memory requirements for signature storage respectively. Test vector compaction can be achieved using weighted pseudo-random generators [5]. Response compression can be implemented by using either count based techniques (either 1's count or transition count) or a multiple-input signature register (MISR). MISR based approach can be realized very efficiently using modified linear feedback shift registers (LFSR) with some combinational compactors.

3.6 Limitations

Limitations of proposed approach are: -

3.6.1 Faults in comparator.

While even testing the RA comparator during BIST testing, faults in the comparator of the tester coupled with faults of tested entity may mask the fault. Two negatives may make a positive. Even a fault in the tester comparator is enough to ensure wrong operation.

3.6.2 Faults in memory.

If stored response signatures in memory are garbled then good entities may be labeled as bad and vice versa. Anyway, protecting only memories either using TMR or error-correcting codes (ECC) is much less expensive in terms of hardware requirements than protecting the

whole computational and communication logic. Additionally, ECC based approaches are much easier to implement for memories because stored data is expected to be equal to the input data, and not some computation on the input data that has to satisfy property of closure in the group for which used code is defined.

3.6.3 Handling Transient Errors

Proposed scheme does not take care of transient errors because only hardware (not data) is tested for errors during test and recover phase. These transient errors may not be present during hardware test but can corrupt the data. To handle transient errors, we propose the use of standard or desired soft error detection and correction techniques [8].

4 Methodology of Using Proposed Technique

In this section, a methodology for using the proposed approach to achieve various design goals, e.g. performance, fault coverage etc., in an efficient manner is provided. First of all, an appropriate scheduling policy according to throughput requirements (sustained vs. bursty) should be selected and appropriate calculations should be used in the following steps. The methodology is as follows: -

1. Define grain allocation (GA) map, test time interval (t_{ij}) allocation (TA) map, and test application time (a_{ij}) allocation (AA) map in mathematical form. GA, TA and AA are two-dimensional matrices given by: -

$$GA = \begin{matrix} & \begin{matrix} 1 & 2 & * & N \end{matrix} \\ \begin{matrix} g1 \\ g2 \\ * \\ gm \end{matrix} & \begin{bmatrix} n_{11} & n_{12} & * & n_{1N} \\ n_{21} & n_{22} & * & n_{2N} \\ * & * & * & * \\ n_{m1} & n_{m2} & * & n_{mN} \end{bmatrix} \end{matrix}$$

Where, there are m grains and N nodes. Setting corresponding node column bit in grain row to be 1 indicates presence of a node in that particular grain.

$$TA = \begin{matrix} & \begin{matrix} T1 & T2 & * & TR \end{matrix} \\ \begin{matrix} g1 \\ g2 \\ * \\ gm \end{matrix} & \begin{bmatrix} t_{11} & t_{12} & * & t_{1R} \\ t_{21} & t_{22} & * & t_{2R} \\ * & * & * & * \\ t_{m1} & t_{m2} & * & t_{mR} \end{bmatrix} \end{matrix}$$

Where, number of columns represents R number of rounds of normal execution and *test and recover* phases required. Each column entry denotes time taken in normal execution phase by a grain.

$$AA = \begin{matrix} & A1 & A2 & * & AR \\ \begin{matrix} g1 \\ g2 \\ * \\ gm \end{matrix} & \begin{bmatrix} a_{11} & a_{12} & * & a_{1R} \\ a_{21} & a_{22} & * & a_{2R} \\ * & * & * & * \\ a_{m1} & a_{m2} & * & a_{mR} \end{bmatrix} \end{matrix}$$

Where, each column entry denotes test application time during *test and recover* phase by a particular grain (represented by row) in that round.

- Derive an expression for system throughput $Y(GA, TA, AA)$, and execution time $E(GA, TA, AA)$ taking into account all factors affecting throughput and execution time in these expressions e.g. performance impact of given GA and TA on intergrain and intragrain communication, normal case and recovery performance overhead in terms of TA, performance overhead of applying test vectors in terms of AA etc. Elements of $AA(a_{ij})$ and $TA(t_{ij})$, together with $R(GA, TA, AA)$, satisfy the following expression: -

$$\max_{i=1}^{i=m} \left\{ \sum_{j=1}^{j=R(GA, TA, AA)} (t_{ij} + a_{ij}) \right\} = E(GA, TA, AA)$$

Where, $R(GA, TA, AA)$ is number of rounds (same as R defined before) of normal execution and *test and recover* phases.

- Derive an expression for system $FC(AA)$ in terms of AA map. Expected failure distribution and fault types affect AA map. AA map should detect faults, if it can, as soon as they occur according to failure distribution. Let, $M(AA)$ be the total number of distinct faults detected by different test application runs in AA map. Assuming F to be the total number of distinct detectable faults for a given fault distribution, $FC(AA)$ can be defined to be $M(AA)/F$.
- Derive an expression for the non-performance related cost function, $C(GA, TA, AA)$. This cost function takes into account recovery energy consumption overhead, maximum interface buffer size, intragrain communication implementation overhead, energy consumption overhead of grain testing that depends on test frequency and test application time etc. To take into account the dynamic affect of failure distribution, related overhead cost and performance functions can be accounted for with the affect of failure distribution in the form of a function of (F_d, GA, TA, AA) . This will result in skewing of GA, TA and AA towards optimum values.
- Minimize $C(GA, TA, AA) + C(F_d, GA, TA, AA)$ subject to the following constraints: -

$$Y(GA, TA, AA) - Y(F_d, TA, AA) > Y1 \quad (1)$$

$$E(GA, TA, AA) + E(F_d, TA, AA) < E1 \quad (2)$$

$$FC(AA) > FC1 \quad (3)$$

Where, Y1, E1 and FC1 are required throughput, execution time and fault coverage respectively.

Desired combinations of constraints (1), (2) and (3) can be treated as *strict* or *loose* constraints (meet them only when strict constraints are met) depending on objective.

5 Conclusion

This paper considers and provides solutions to problems pertaining to effective and efficient realization of a fault-tolerance technique that utilizes BIST resources. Efficient solutions are provided to various problems like faults in BIST resources themselves, BIST-node path failures, split brain problem etc. The limitations of the proposed technique are also discussed. A methodology for achieving desired design objectives using the proposed approach is also proposed.

References:

- A. S. Leon et al., A power-efficient high-throughput 32-thread SPARC processor, *IEEE Solid State Circuit Conference*, 2006
- G. Koch, Discovering multi-core: extending the benefits of moore's law, *Intel Magazine*, Jul 2005
- <http://public.itrs.net>
- S. Mediratta, J. Draper, On-chip fault-tolerance utilizing BIST resources, *IEEE MWSCAS*, 2006
- N. K. Jha, S. Gupta, *Testing of Digital Systems*, Cambridge University Press, May 2003
- C. N. Hadjicostis, *Coding Approaches to Fault Tolerance in Combinational and Dynamic Systems*, Springer, 2001.
- M. L. Shooman, *Reliability of Computer Systems and Networks*, John Wiley & Sons, 2002.
- R. Naseer, J. Draper, The DF-dice storage element for immunity to soft errors, *MWSCAS*, 2005