

Achieving 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. Effective and efficient 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. This paper describes the approach and issues pertaining to its efficient and effective realization. A methodology of using the proposed approach for desired design objectives is also provided. 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.

Traditional redundant logic (like Triple Modular Redundancy [4]), arithmetic coding and algorithm-based fault tolerance (ABFT) approaches are limited in the type and number of faults [5] they address, in addition to introducing hardwired performance and very high implementation overhead in designs (section 5). Traditional approaches will require another fabrication run for more and different types of faults, adding to the production time and hence time to market, not to mention total cost of the product. Thus, there is clearly a need for new fault tolerance techniques.

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 is proposed [7]. 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.

Section 2 describes the proposed approach of BIST resource utilization, section 3 explores the aspects to be considered for its effective and efficient realization, section 4 gives the methodology of using the proposed approach, section 5 compares proposed approach with traditional approaches and finally section 6 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 turns already present BIST resources into useful resources during a system's normal execution. 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 is not merely using rollback and recovery at a hardware level, but giving BIST resources a new meaning of existence and more significance. Thus, rollback and recovery are just a facilitator in achieving this purpose.

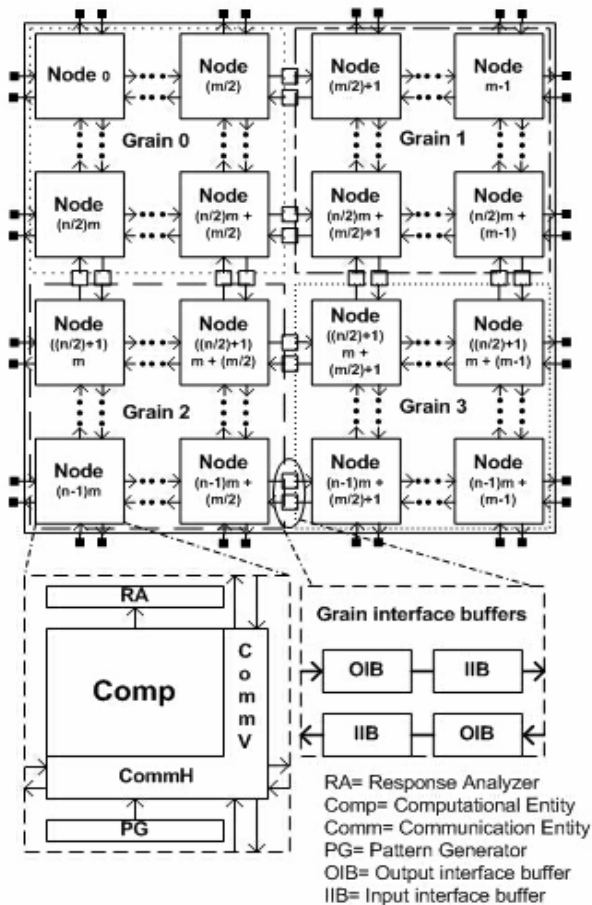


Fig.1 Concept of *grain* in a multi-core architecture

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 a *test and recover phase*. Grains 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 a particular system architecture and application. The determination of grain size and grain boundaries is crucial 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 Intergrain and Intragrain Communication

3.1.1 Intragrain Communication

Communication of *rollback and recover* signal in an effective, efficient and error-free manner is an important intragrain communication issue, besides normal communication that can be done using any desired fault-tolerant routing algorithm that provides a path in reconfigured network. One way of communicating a *rollback and recover* signal is sending messages over a dedicated interconnection network, which itself is made fault-tolerant using BIST resources. Efficient broadcasting of rollback and recovery related messages can be done by aggregating messages from different nodes and sending them collectively. Another more area-efficient, but not highly scalable and non-standard-cell based approach is a circuit-level approach that makes use of a distributed NOR gate used in fast searches and pattern matches (Fig. 2). Hierarchical configuration [6] to reduce wiring delay (due to long wires and large number of driver transistors) to logarithmic proportionality to number of transistors can be used to reduce the affect of a large number of drivers. This circuit itself should be tested for permanent faults, if redundancy is not provided for given small logic.

3.1.2 Intergrain Communication

By definition of grain, no intergrain communication is allowed until a particular grain commits its computation and communication. There are two types of interface buffers that have to be used for intergrain

communication. Output interface buffers (Fig. 1) are required to save the intergrain communication for the current execution phase, and forward their contents to input interface buffers (Fig. 1) of neighbor grains upon successful committing of current normal execution phase. These interface buffers can be implemented either as separate buffers or some part of local memory of system entities. In the latter case, with DMA support and more memory per node becoming standard, a host processor will not need to be interrupted for this task. Intergrain communication must be maintained in input interface buffers for the previous checkpoint of each associated grain. This information is required during recovery of a grain. Another intergrain communication issue is propagation of *continue* or *stall* signal. This can be achieved by using a multiphase extension of the circuit in Fig. 2.

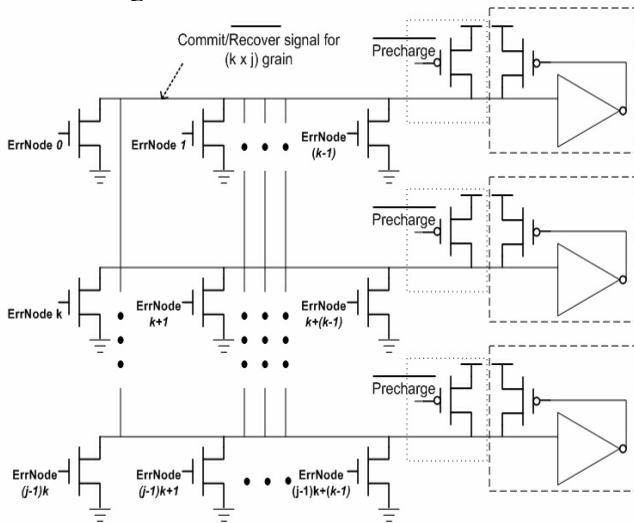


Fig.2 Distributed NOR approach for commit signal.

Interface buffer size and injection of large interface buffer traffic into the network in normal execution phase are two issues here. Both can be handled by minimizing intergrain communication, which can be achieved by judiciously defining the grain size and mapping application to grains of different shapes and sizes using chip partitioning, floorplanning and placement techniques used in traditional VLSI CAD.

3.2 Checkpoints and Recovery

After each successful test and recover phase, a checkpoint of system state is kept in an ECC and redundancy protected memory and circuitry. The system *computation state* information includes the value of all register files, value of program counter, processor status and mode protected registers, processor special purpose registers etc. System *communication state* involves saving all the packets in transit at the routers. In

addition, *external inputs* received during the grain's normal execution before commit should be stored. This is important because during a recovery operation external inputs are difficult to re-create from outside the chip or system in general. This state saving approach is practical because this capability is inherent in computing elements to handle context switching. Although such capability has slightly higher memory requirements, processing nodes in future multicore architectures are projected to have abundant local memory anyway to alleviate the memory wall problem, which would become prohibitive otherwise for large number of on-chip cores [2]. So, proposed approach is not imposing any new restriction on the hardware architecture but it is making use of the resources that will already be available in next generations of processor architecture.

Recovery can be achieved in following ways: -

3.2.1 Progressive Recovery

In progressive recovery, only a grain containing faulty node(s) is recovered to the current time stamp, while other grains wait for the recovery as well as preserve their communication state. Other grains need not be rolled back because they have not received faulty communication yet. Grain level recovery should maintain the identity of a faulty node and assign its computation to any spare node, and guarantee the redirection of its traffic to the appropriate newly assigned node by updating information in the routers. An alias table can be maintained at the neighbor routers and destination address field of the packet can be updated upon receiving packet for the faulty node.

3.2.2 Regressive Recovery

For regressive recovery, all the grains belonging to a particular application are rolled back, while discarding all the communication in transit, to a common checkpoint. This case may be required, if the integrated common checkpoint is not node or grain specific, but contains state information for a particular application. Another scenario making this case of application-level recovery is dissolution of faulty node during recovery and pending packets destined for faulty node in communication state or interface buffers. The disadvantage of this approach, as compared to progressive recovery, is more recovery overhead in terms of performance and energy consumption. This is due to recovery of additional grains without faults in them and re-structuring of computation of nodes pertaining to an application as compared to recovery of only grains containing faults and re-assignment of only faulty node's computation in progressive recovery.

3.3 BIST Resources Fault-tolerance

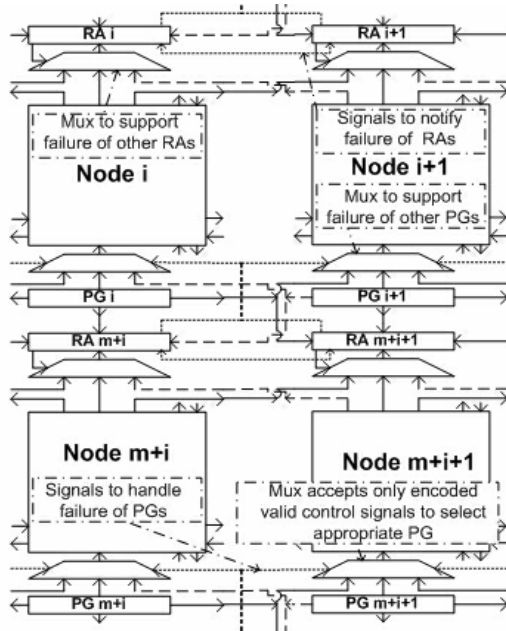


Fig.3 Support for BIST fault detection and failures.

3.3.1 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 [8] 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 [5]. 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. 3). 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. 3). 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.

3.3.2 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 3.3.1(a) or 3.3.1(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.

3.4 Split-brain Problem

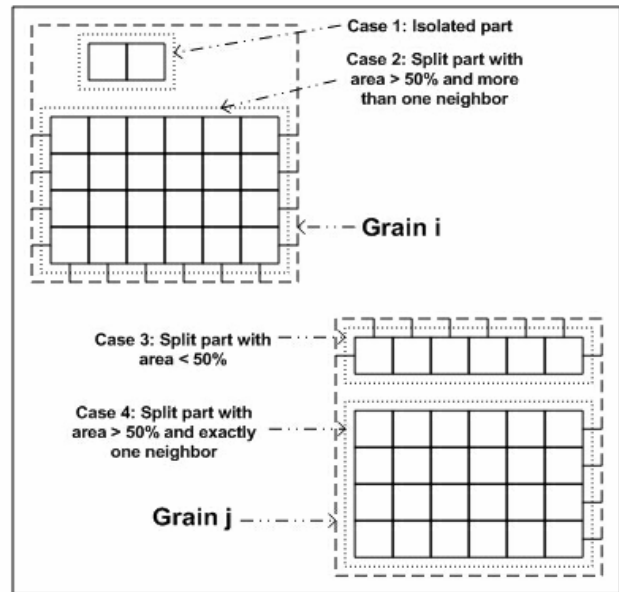


Fig.4 Split-brain problem illustration.

An issue called the *split-brain problem* [9] 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. 4). 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. 4). 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. 4). 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. 4), 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. 4) 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. 5(a)). 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. 5(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. 6). This approach will add its worst-case overhead only in case of worst-case scenarios because of its dynamic nature.

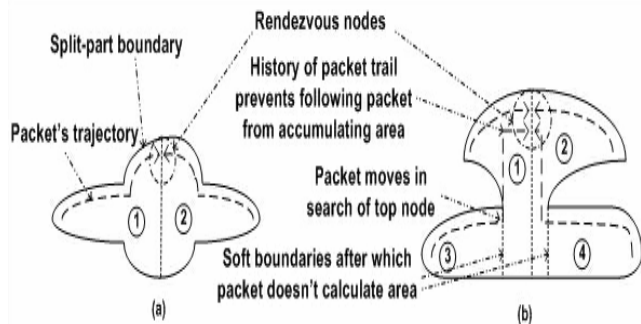
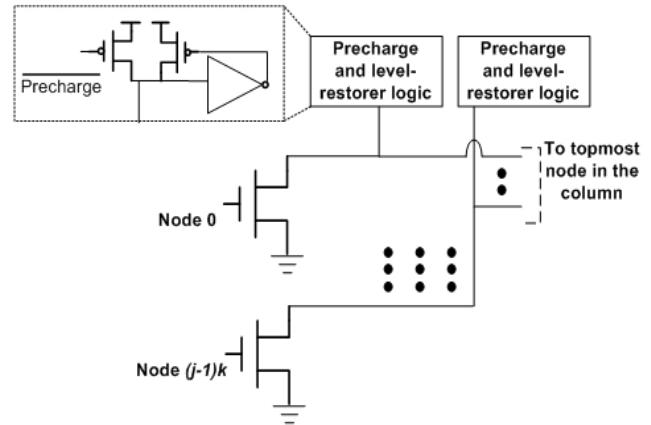


Fig.5 Concept of area and shape determination.

One approach of implementing connectivity communication and bottom node identification propagation is to use long wires (Fig. 6). Some encoding scheme should be used on these lines (Fig. 6) 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.



	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.6 Support for area and shape determination.

3.5 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 [9]. 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.

3.6 Analysis Dimensions

3.6.1 Grain Size Determination

Grain size should be carefully chosen for a particular application (Fig. 7). If grain size is big, then much energy has to be spent on recovery, in addition to its impact on intragrain communication. If grain size is small then a significant overhead is incurred in intergrain communication due to large number of interface buffers.

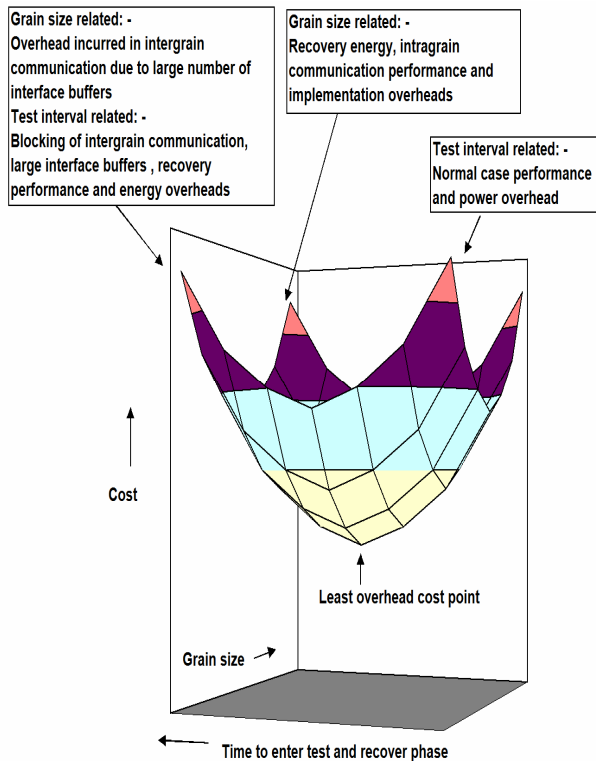


Fig.7 Overhead cost function with respect to grain size and time to entering *test and recover* phase.

3.6.2 Scheduling of Test and Recover Phase

The time interval or frequency for initiation of *test and recover* phases should be chosen carefully after analyzing its impact on system level performance (Fig. 7). If the interval is too big then system latency and recovery time will be very large, which means increased recovery overhead and decrease in system performance. In addition, blocking of intergrain communication for a long time will also degrade system performance. If the time interval is very short, then also performance and

energy overhead increases because much time and energy is spent in *test and recover* phase increasing the overall program execution time.

3.6.3 Efficient Application of Testvectors

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 [8] can be applied to reduce the test application time/energy and memory requirements for signature storage respectively. Trade-off to be considered is possibly lower fault coverage because of reduced test vectors and aliasing respectively.

3.7 Limitations

Limitations of proposed approach are: -

3.7.1 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.

3.7.2 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.

3.7.3 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 [10].

4 Proposed Methodology

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: -

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

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.

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

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.

2. 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.

3. 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 $M(AA)/F$.

4. 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.

5. 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 Comparison with other approaches

5.1 Programmability

This solution is programmable for different fault coverage and fault types. This is important because faults may not be characterized for a particular process technology until post fabrication.

5.2 Less Overhead

The proposed approach imposes very low, if any, performance overhead in the maximum normal execution case because the design itself is unaltered. With other approaches like arithmetic code, ABFT and

redundancy, performance overhead is implied and hardwired, resulting in much larger performance overhead. Also, the proposed approach offers more performance gain by making much more efficient use of available hardware for normal computation and not wasting potential processing power for doing redundant computation or computationally unuseful function (like ECC). Also, there is some scope for post fabrication performance optimization of the final product depending on fabrication results.

Regarding area overhead, arithmetic code and ABFT techniques can only be applied to protect computations that possess certain algebraic structure [5], and thus limiting the use of such techniques. So, complexity and overhead of building a generic system component e.g. CPU, router etc. can be very high and very unfriendly to generic hardware implementation by limiting resource sharing. TMR results in large area overhead for obvious reasons. Area overhead in our approach is minimal because BIST is becoming standard for manufacturer and boot-up tests, at least in production environments, and only small additional control circuitry is required.

Regarding power consumption, redundancy based approaches require exact replication of hardware, as indicated above. Assuming roughly uniform switching activity factors, power comparisons track area comparisons. Thus, the proposed approach is expected to be less power-hungry as well.

5.3 Low Time to Market and Production Cost

Low time to market is evident because of less number of chip respins, if any, required with the proposed approach. BIST resources can be configured to tune a wide range of potentially bad fabrication results to acceptable. Also, design time is low for the proposed approach because no design and fault environment specific mechanisms need to be hardwired like arithmetic codes. Low production cost is achieved because of less number of chip respins, low design time (salary), high yield and long lifetime of chips.

5.4 High Productivity

BIST techniques are well developed and have already made their way into chip design for mass production. So, utilizing that knowledge for achieving fault tolerance, without learning many new concepts, will translate into higher designer productivity. For a comparison, productivity is inherently low with arithmetic code and ABFT techniques because of more complex design [5], and redundancy based approaches will still require understanding of some extra concepts for efficient utilization of redundancy that were not part of the design flow before.

5.5 Reusability and Scalability

The basic framework remains unaltered and highly reusable. Only various dimensions discussed in section 3.6 need to be analyzed and a performance point needs to be tuned for a particular design and application. The proposed approach is also scalable across multiple process technologies and feature sizes, given the underlying framework of using BIST resources.

6 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:

- [1] A. S. Leon et al., A power-efficient high-throughput 32-thread SPARC processor, *IEEE Solid State Circuit Conference*, 2006
- [2] G. Koch, Discovering multi-core: extending the benefits of moore's law, *Intel Magazine*, Jul 2005
- [3] <http://public.itrs.net>
- [4] R. E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, *IBM Journal of R&D*, 1962
- [5] C. N. Hadjicostis, *Coding Approaches to Fault Tolerance in Combinational and Dynamic Systems*, Springer, 2001
- [6] C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
- [7] S. Mediratta, J. Draper, On-chip fault-tolerance utilizing BIST resources, *IEEE MWSCAS*, 2006
- [8] N. K. Jha, S. Gupta, *Testing of Digital Systems*, Cambridge University Press, May 2003
- [9] M. L. Shooman, *Reliability of Computer Systems and Networks*, John Wiley & Sons, 2002.
- [10] R. Naseer, J. Draper, The DF-dice storage element for immunity to soft errors, *MWSCAS*, 2005