

On-chip Fault-tolerance Utilizing BIST Resources

Sumit Dharampal Mediratta, Jeffrey Draper

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

Abstract— Recent and projected advances in VLSI fabrication technology will allow for integration of billions of transistors and advanced architectures on a single chip. According to the International Technology Roadmap for Semiconductors (ITRS), widespread reliability challenges are expected for these VLSI fabrication technologies (65nm and below). Effective and efficient on-chip fault-tolerance solutions are needed. A new approach of achieving on-chip fault-tolerance using built-in-self-test (BIST) resources is proposed in this paper. 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.

I. INTRODUCTION

With the introduction of multiple changes and shrinking feature sizes in VLSI technologies, it has become possible to fabricate billion-transistor chips. Emerging paradigms utilizing these vast on-chip resources are system-on-chip and multi-core architectures [1][2]. One of the major problems being faced by architects of these systems, is to achieve effective on-chip fault-tolerance without sacrificing too much area, energy and performance.

To further elaborate on the problem, according to 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][4] and revolutionary changes associated with new materials and devices. The introduction of multiple materials, processes and structural changes in a short period will increase the difficulty of understanding and controlling failure modes. Therefore, fault-tolerance should be considered a necessity, rather than as a feature.

Traditional redundant logic (like Triple Modular Redundancy [5]), arithmetic coding and algorithm-based fault tolerance (ABFT) approaches are limited in the type and number of faults [6][7] they address, in addition to introducing hardwired performance and very high implementation overhead in designs (section IV). 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.

Our philosophy is to take unknown faults, less fault diagnosis and characterization time before production and unknown failure rates as a *specification* for providing a fault-tolerance solution for future VLSI architectures. The above requirements translate into a programmable flexible fault-tolerance solution. A post-fabrication programmable fault-coverage (FC) and fault-type fault-tolerance method is provided by the proposed approach of utilizing on-chip built-in self-test (BIST) resources. These BIST resources are expected to be already present on VLSI chips because they are becoming standard in the production environment. The proposed approach, in addition to its various other advantages mentioned in section 4, reduces production cost while leaving the trade-off analysis of maximizing FC and minimizing performance overhead to be programmable until post-fabrication.

Section II describes the proposed approach of BIST resource utilization, section III explores the design space by giving analysis dimensions to an interested designer, section IV elaborates on the advantages of proposed approach and finally section V concludes this paper.

II. PROPOSED 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, i.e., BIST components gain more significance as dynamic resources in this new fault-tolerance paradigm. Our approach uses rollback and recovery mechanisms as an integral instrument to provide correct results in case of failures, and thus achieves fault tolerance. Rollback and recovery has been used before for achieving fault-tolerance at the software level of abstraction. 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. The following paragraphs delineate this proposed new fault-tolerance paradigm along with several of the variants that one may consider for effective realization.

A. 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.

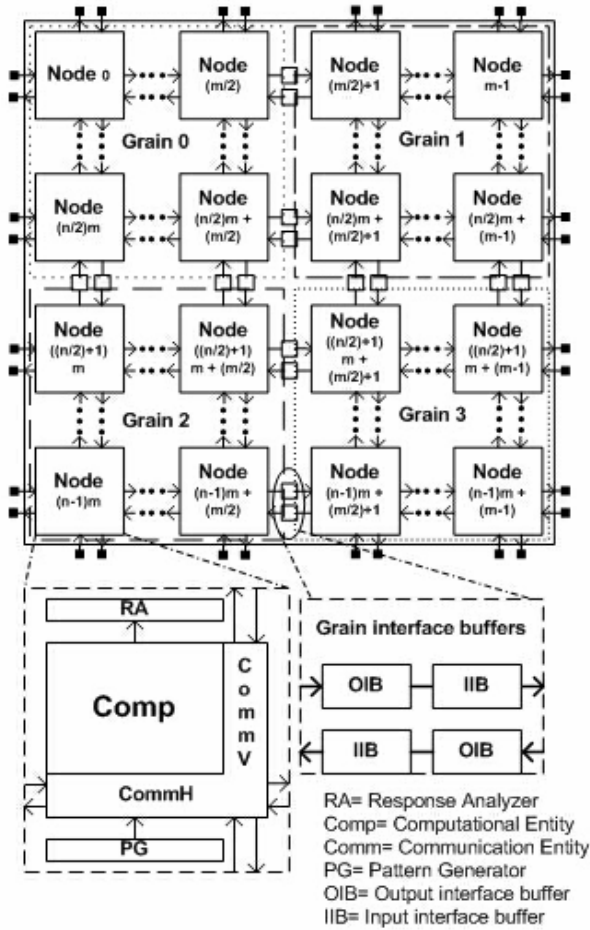


Figure 1. Concept of grain in a multi-core architecture

B. 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.

C. Intergrain and Intragrain Communication

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 [8] 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 this small amount of logic.

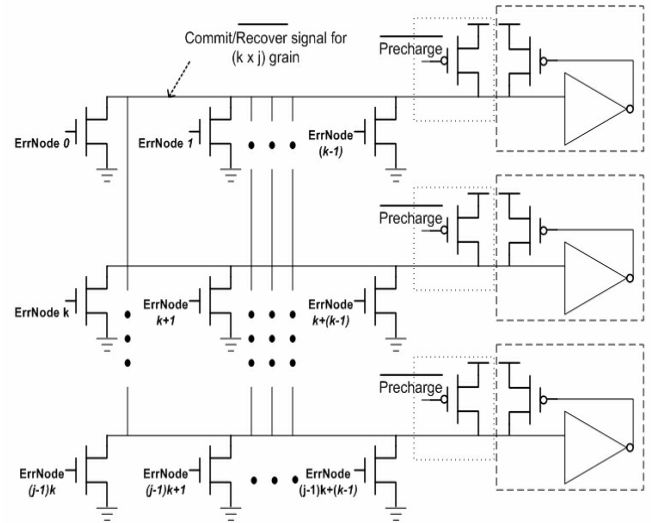


Figure 2. Distributed NOR approach for commit or recover signal.

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, e.g., by using domino logic distributed NOR gate.

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.

D. Saving Checkpoints and Performing 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 two ways: -

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.

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.

E. Scheduling of Test and Recover Phase

Hardware enters *test and recover* phase periodically, which may be scheduled in one of the following ways: -

1) *As a separate phase between normal execution phases.*

Here, *test and recover* phase is executed mutually exclusively with the normal execution phase. Advantages of this approach are less intergrain synchronization complexity and ease of testing for intergrain communication links. Also, this approach doesn't affect and interfere with the dynamics of intergrain communication because all messages are stalled unless system goes into normal execution again. Its affect is predictable and static in fault-free case, and depends on time spent in *test and recover* phase. This approach is suitable for applications that display bursty communication characteristics.

2) *Overlapping of test phase with normal execution.*

Overlapping test and recover phase for certain grains with normal execution of other grains can be done to achieve one of the two purposes discussed below. This approach requires special consideration for the testing of inter-grain links because nodes at the other end of those links may be in normal execution phase. Thus, it requires some test functionality from some nodes in normal execution phase.

a) *For lower buffer requirements and more sustained performance*

The objective here is to flatten out (or temporarily decompose) the burst of output data, and hence to reduce the buffering requirement at the outputs and achieve high sustained system performance. Here, sustained performance increases because some smaller numbers of entities give results at a shorter interval of time as compared to non-overlapped case where all entities give results at the same time. Both worst-case latency and throughput are low initially as compared to non-overlapping case, but they reach their counterpart values in steady state at the end of the first run of testing of all grains (Fig. 3). Notice that, for some

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.

IV. ADVANTAGES COMPARED TO OTHER APPROACHES

A. Programmability and suitability for unpredictable post fabrication results.

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.

B. Area, power and performance compared to traditional approaches.

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 [6], 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.

C. 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 (and hence designer salary), high yield and long lifetime of chips.

D. High productivity because of small learning curve.

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 [6], 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.

E. Reusability and scalability

The basic framework remains unaltered and highly reusable. Only various dimensions discussed in section IV 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.

V. CONCLUSION

According to ITRS, widespread reliability challenges are expected for recent and future VLSI fabrication technologies, a situation that calls for effective and efficient on-chip fault-tolerance solutions. A new approach of achieving on-chip fault-tolerance using BIST resources is proposed in this paper. These BIST resources are expected to be already present on VLSI chip because they are becoming standard in the production components. 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 approach will result in obvious advantages of yield enhancement and prolonged lifetime of VLSI chips as well.

VI. REFERENCES

- [1] A. S. Leon, J. L. Shin, K. W. Tam, W. Bryg, F. Schumacher, P. Kongetira, 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, July 2005
- [3] <http://public.itrs.net>
- [4] <http://www.sematech.org>
- [5] R. E. Lyons, W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability", IBM Journal of Research and Development, 1962
- [6] C. N. Hadjicostis, Coding Approaches to Fault Tolerance in Combinational and Dynamic Systems, Springer, 2001.
- [7] S. Almukhaizim, Y. Makris, "Fault tolerant design of combinational and sequential logic based on a parity check code", Defect and Fault Tolerance in VLSI Systems, 2003.
- [8] C. Mead, L. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.