

Dynamic Test Set Selection using Implication-Based On-Chip Diagnosis

Nicholas Imbriglia, Nuno Alves, and Jennifer Dworak
Brown University

Abstract

As circuits continue to scale to smaller feature sizes, wearout and defects that could not be detected during manufacturing test are expected to cause an increasing number of errors in the field. Online error detection techniques are capable of detecting at least some of these errors as they occur. However, recovery may be expensive, and depending on the source of the error, the underlying problem may lead to increasingly frequent failures over time. Furthermore, in the case of a multi-core architecture, additional cores may suffer from the same or similar issues—either because of a fundamental susceptibility in the design or because of the applications that the user is running. In this paper, we will investigate the diagnostic capabilities of logic implications to pinpoint possible failure locations when an error is detected online. We will then utilize this information to select highly efficient test sets that can be used to effectively test the identified suspect locations.

1. Introduction

Continued circuit scaling is contributing to great increases in logic density in pursuit of increasing processor performance. As a result, although processor chips today generally contain cores numbering in the single digits, in the future, such chips are predicted to contain a hundred or even a thousand cores [1]. However, circuit scaling also comes with disadvantages. Among these is the fact that future devices are expected to become more susceptible to soft errors, wearout, and latent or complex defects [2,3]. Reducing the impact of the resulting errors is a significant challenge.

Multiple techniques have previously been proposed to detect errors that occur during normal circuit operation. Some of these techniques include logic duplication and triple modular redundancy, parity prediction, and Berger and Bose Lin codes. Others have used high level assertions instantiated in the hardware to determine that an error is present.

In addition to these methods, logic implications have been proposed as an effective online error detection technique. Logic implications encapsulate

expected relationships between the values at different circuit sites. These relationships should always hold if the circuit is operating correctly. Thus, if the expected relationship is violated during circuit execution, it indicates the presence of an error. Numerous logic implications occur naturally in logic circuits, and additional hardware may be added to the circuit to check for implication violations and thus to flag the appearance of errors.

Using logic implications for online error detection provides several advantages. Specifically, no high-level information regarding design intent is required for implication identification and selection. Furthermore, it is possible to easily tradeoff additional error coverage against additional area overhead.

However, once an error has been flagged, multiple courses of action can be followed in response. For example, when a particle strike is the source of a problem, returning to a previous checkpoint state and re-running the corrupted instructions will likely be sufficient to correct the problem. Particle strikes are relatively rare, the same problem is unlikely to happen again, and the re-execution is likely to proceed correctly.

However, it is possible for other problems to have caused the error. The error could be due to a latent or complex manufacturing defect that was not detected directly after manufacturing. Similarly, the error could have arisen due to circuit wearout. Issues such as hot carrier injection, NBTI, and electromigration can all cause circuits to begin to fail over time.

Depending on the nature of the defect and its susceptibility to noise and environmental conditions, re-execution may or may not lead to successful and correct completion of the desired operation. Furthermore, even if re-execution is successful, subsequent failures associated with the same defective locations are increasingly likely. In fact, in a multi-core system, other cores in the system may also suffer from similar problems—especially if the underlying source of the error is related to an underlying susceptibility in the design or in the way the device is being operated by the end user.

In such cases, identifying that a particular core is likely to cause future failures is valuable. Thus, in

addition to simply re-executing the offending sequence on the same core or a different core, it would also be very useful to eventually apply a test pattern sequence to each of the cores in the design to identify whether or any of the other cores is also beginning to suffer from this problem. If so, those cores could be taken offline and (if possible) replaced with working spares.

However, to maximize the effectiveness and minimize the cost of the applied tests, the test set should satisfy three significant characteristics. Specifically, it should:

- Focus on those areas of the circuit that could have caused the original error.
- Provide multiple detections of faults within the area of interest to thoroughly test for defects associated with those faults.
- Be very short to reduce the amount of time, power, and congestion involved in sending the test set to each core and applying it.

In this work, we will show that using logic implications for online error detection and test set selection satisfies these requirements, and thus they are particularly useful for this application. Specifically, because any given logic implication only covers a subset of the circuit, knowledge of which implication has failed provides automatic diagnostic information that can be used to generate or select an appropriate test set to apply to each core that will be tested.

Thus, this paper will investigate the ability of implications to provide the diagnostic information required for appropriate fault targeting. It will then explore the ability to generate very short and effective test sets targeted to those portions of a circuit that have been identified as the likely source of an online error.

Section 2 describes some related work in online error detection and test. Section 3 introduces background work on logic implications and shows how they can be used for online error detection. Section 4 explores the capability of using these implications for automatic on-chip diagnosis of observed errors. Sections 5 and 6 describe our test set selection algorithm. Section 7 presents some experimental results, and Section 8 provides some conclusions.

2. Related Work

Many researchers have investigated techniques for online error detection. In general, these techniques generally introduce some type of redundancy in information to the circuit. One of the most effective

and well-known techniques is logic duplication or triple modular redundancy (TMR). In the case of TMR, three copies of the circuit are run in tandem and their results compared [4]. Of course, while highly effective, this scheme is very expensive in area and power overhead.

As an alternative, different coding techniques such as parity, Berger, and Bose Lin codes have also been studied for many years [5-11]. Yet, these methods often have relatively high overhead as well. As a result, some researchers have developed techniques for identifying which parts of the design are most likely to cause errors as well as those that are most likely to cause serious (as opposed to inconsequential) errors. Then, only the critical portion is protected so that the overall cost can be reduced [12-14].

Other online error detection schemes check a portion of the input space. For example, in Built-In Concurrent Self Test (BICST), hardware is added to predict the appropriate response to a set of pre-computed test vectors. The Reduced Observation Width Replication (ROWR) scheme is similar to BICST and improves on that approach, reducing the hardware overhead by only checking the minimum required output values.

Others have taken high level assertions originally found for design verification and have implemented them in hardware—allowing them to be used for online error detection as well [15,16]. In addition, logic implications identified at the gate level have been proposed as a means for detecting errors online [17-19]. As already stated, these implications require no knowledge of designer intent and allow for easy tradeoffs of coverage and overhead. In this paper, we will propose a new use for these implications—as a source of online diagnostic data for on-chip test set selection.

Test patterns applied on chip have been studied for many years with respect to built-in self test (BIST) methodologies. These methods use on-chip hardware to generate pseudo-random or weighted random patterns on chip (e.g. [20,21]). Others have studied approaches that combine BIST and deterministically generated ATPG patterns to obtain better coverage more efficiently than can be achieved with pseudo-random patterns alone (e.g. [22-25]).

Online test and sensing has also been proposed for circuit failure detection and prediction due to wearout and aging. One of the most recent entries into this arena is CASP (Concurrent Autonomous Chip Self-Test Using Stored Test Patterns) [26]. Here, the authors propose storing structural scan patterns on a hard disk or in flash memory and then applying the stored patterns to one or more cores in a system concurrently with the normal operation of that system

so that no downtime is experienced. This work was extended in [27], which proposed the use of virtualization and hardware-software co-design to enable the concurrent testing of cores.

In this paper, we will take advantage of an important characteristic of logic implications—their ability to automatically identify a restricted set of faults as the potential cause of an observed error. This information will then be used to choose an appropriate test set to apply to each core to detect latent defects or wearout both in the original core and other identical cores.

3. Logic Implications for Online Error Detection

Logic implications are relationships that are expected to always hold between different sites in a digital logic circuit. Obviously, such relationships naturally exist across logic gates. For example, consider the NOR gate shown in Figure 1a. A logic 1 on input x implies that output z is equal to 0. Thus, we can say that: $x = 1 \Rightarrow z = 0$. Similarly, if z is equal to 1, this implies that x must be equal to zero. If these relationships don't hold for some input combinations, the gate is not acting as a NOR, and thus an error must be present.

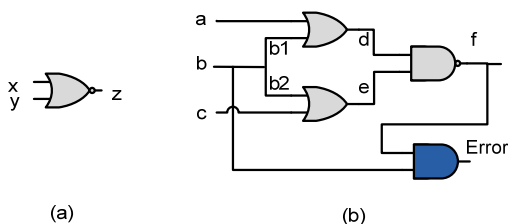


Figure 1: Implication example

However, checking such an implication online would only allow us to check for faults associated with this single gate. Fortunately, implications with a much larger distance between implication sites also naturally occur in circuits. For example, consider the circuit fragment in Figure 1b. In this figure, an implication exists due to the presence of reconvergent fanout. Specifically, if b is equal to 1, this implies that f (the output of the NAND gate) must equal 0. Similarly, if f is equal to a logic 1, then this implies that site b must be equal to 0.

In the simplest case, we can check that this relationship holds at runtime by inserting a single additional gate—in this case an AND gate into the circuit. The output of the AND gate equals 1 when both b and f are equal to 1, a situation that should never occur if the circuit is operating correctly.

If we insert checking hardware to verify that logic implications hold at runtime, it is possible to detect multiple errors. For example, in this case, faults $b1$ $sa0$, $b2$ $sa0$, d $sa0$, e $sa0$, and f $sa1$ (before the fanout branch) may all be detected for at least some input combinations.

Because including all possible implications in online checker logic would lead to unacceptable overhead, only the best implications should be selected for inclusion in the checker logic. In [17-19], a series of procedures were described that enable the identification and selection of an appropriate implication subset to maximize online error detection subject to a given hardware overhead constraint. In that work, it was shown that over 50% error coverage could be achieved with only 10% hardware overhead for some circuits.

4. Implications for Automatic Online Diagnosis

From the discussion regarding Figure 1, it should be obvious that a given implication will cover only a subset of all faults in a circuit. In fact, the faults that may be covered are directly related to the spatial locations of the implication sites and the underlying source of the site value relationship.

Specifically, there are three primary circuit structures that lead to logic implications:

1. A logic implication may arise from reconvergent fanout. In this case, some faults along paths from the fanout stem to the point of reconvergence may be detectable by the implication.
2. A logic implication may arise from divergent fanout where the implication sites are downstream from the fanout branch. In this case some faults along paths from the implication points to their common ancestor may be detectable by the implication.
3. A logic implication may arise along a direct path, where a particular logic value at the upstream site implies controlling values at each gate on the path to the second implication site. In this case, some faults along the path may be detectable by the implication.

In the previous work described in [17,18], the error signals from each individual implication were OR'ed together to generate a single error signal. Unfortunately, this single error signal does not provide

much information regarding which faults are good candidates for having caused the failure. However, if we can identify *which* implication was violated, we can obtain much better diagnostic resolution.

Thus, we propose a modification to the checker logic design. Specifically, we propose to have each implication check gate feed into a flip-flop. On a failure, the values in the flip-flops can be read, allowing us to determine which implication had failed. This will automatically pinpoint a subset of the circuit that could have caused the error.

To see the overall impact of this, several experiments were run on ISCAS '85 benchmark circuits. First, a set of valid implications in the circuit were identified using logic simulation and a SAT solver as described in [19]. However, as already mentioned, a full implication set is much too large to instantiate in checker hardware. Thus, only the implications that do the best job of covering faults should be included. Thus, as described in [19], we selected a subset of valuable implications where the number of implications included were limited to a particular area overhead—in this case 10%, 20% or 30% of the overall circuit area. (Note: these overheads do not include the additional flip-flops we will insert to determine *which* implication failed. Thus, the implication set is identical to that in [19].) To determine which faults were detected by each implication, we fault-simulated each circuit with a 20-detect ATPG test set created with Mentor Graphics FastScan. Any time a fault was detected by an implication, it was permanently associated with that implication. It is also possible for a fault to be associated with more than one implication. (Note that a more exact method would perform an exhaustive analysis. Thus, in the future, we also intend to investigate using a SAT solver to create our fault/implication association list.)

Table 1: Number of Faults Detected by Each Implication for 10% Area Overhead

Circuit	Average	Max	Min
c432	80.6	317	3
c499	6.7	16	4
c880	5.1	17	2
c1355	4.0	17	2
c1908	45.3	234	4
c2670	103.9	1440	3
c5315	5.5	38	1

Table 2: Number of Faults Detected by Each Implication for 20% Area Overhead

Circuit	Average	Max	Min
c432	50.0	317	3
c499	7.1	16	3
c880	5.8	28	2
c1355	4.8	17	2
c1908	34.8	234	3
c2670	71.0	1440	2
c5315	4.3	38	1

Table 3: Number of Faults Detected by Each Implication for 30% Area Overhead

Circuit	Average	Max	Min
c432	42.0	317	2
c499	7.7	16	3
c880	5.7	28	2
c1355	5.2	17	2
c1908	32.9	234	3
c2670	64.8	1444	2
c5315	3.9	38	1

The results of these experiments can be found in Tables 1-3. As one can see, the percentage of all faults covered by an implication varies significantly from implication to implication and from circuit to circuit. However, in all cases, the portion of the circuit covered by an implication is much smaller than the circuit as a whole.

5. Test Set Selection

To take advantage of the results obtained in Section 4, we have developed a test set selection procedure that can be easily implemented on-chip. This final test set should meet several requirements:

- The test set to be applied should thoroughly test the area of interest. As a result, we want multiple detections of each fault that could have caused the initial implication violation.
- The test set applied should be very short. This reduces the amount of data that must be transferred across the chip to each core—reducing power and (in a network-on-chip environment) network congestion. It also reduces the overall time required to apply the test and the amount of time that the core under test must be offline.

- The test selection procedure should be efficient and require minimal additional data or computation.

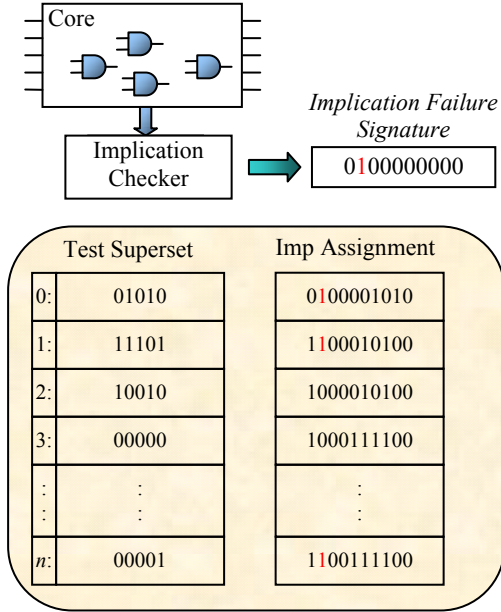


Figure 2: On-Chip Test Selection Procedure

Our overall on-chip test selection procedure is shown in Figure 2. Specifically, a test superset is stored in memory. This could be a flash memory or some other form of non-volatile memory. Also stored in memory would be an array containing information regarding which patterns are associated with each implication. Intuitively, if a pattern is selected as part of the test set to run when a particular implication fails, that test pattern should be able to do a good job of detecting at least some of the faults that were associated with the implication. Thus, when an error is detected during normal operation, the failing implication will be identified, and the corresponding implication assignment list will be used to fetch the test patterns to be applied as a part of each test set.

For example, in Figure 2, the implication signature shows that the second implication has failed. The patterns associated with that implication in the implication assignment list are 0, 1, and n . Thus, patterns 0, 1, and n will be included in the test set to be applied now and whenever the 2nd implication registers a failure.

6. Creating the Implication Assignment Table

The implication assignment table must be created so that each test set satisfies the principles discussed earlier to the largest extent possible. We start by

creating a large superset of test patterns to select from. In this paper, we begin by creating a 15-detect test set with Mentor Graphics FastScan. This will become the Test Superset that will be stored in memory.

Next, we choose an implication i in the list and determine which faults it can potentially detect. Each fault in the original fault list is given a weight:

$$\begin{aligned} Weight_{i,j} &= 0, \text{ if fault } j \text{ cannot be detected by imp } i \\ &= 1, \text{ if fault } j \text{ can be detected by imp } i \end{aligned}$$

To save subsequent processing time, we then remove all faults with a weight of 0 from the fault list considered for implication i . Then, each pattern p is assigned a score based upon the faults it detects:

$$Score_p = \sum_{j=1}^{num\ faults} Weight_{i,j} \cdot Detected_j$$

where, $Detected_j$ can be obtained from a fault dictionary and is equal to 0 if pattern p does not detect fault j and 1 otherwise. Once the score for every pattern is obtained, the pattern with the highest score is added to the pattern set for implication i .

However, while multiple detections of a fault are valuable, the first detection of a fault tends to be much more valuable than the second for defect detection. Similarly, the second detection is more valuable than the third. In the past, this decrease has been shown to follow a decreasing exponential function with a time constant τ [28]. Thus, once a pattern is chosen, the weights of all the faults detected by that pattern are updated with the following equation:

$$Weight_{i,j}(p+1) = Weight_{i,j}(p) \cdot e^{-\#det_j/\tau}$$

In this equation, $Weight_{i,j}(p)$ represents the weight that was used for fault j when pattern p was chosen for the test subset. In addition $\#det_j$ represents the number of times that fault j has been detected by patterns that have been added to the test subset. Once the new weights are obtained, the score for each pattern remaining in the superset is re-calculated. The unselected pattern with the highest score is then added to this implication's test pattern assignment list. The process repeats until the desired number of patterns are included in the list. Then the same procedure is followed for each of the other implications.

7. Experimental Results

To evaluate the ability of our chosen procedure to create good test sets for the faults identified by an implication, we created test sets for c432 and c499. In

each case, a 15-detect superset was initially created using Mentor Graphics FastScan. The value of τ was chosen to be 0.5, and the number of patterns to be applied on each implication failure (i.e. the number of patterns in the subset) was set to 20. (Note that this is less than the absolute minimum required for a 100% fault coverage test set for c432 [29]. The results can be seen in Figures 3 through 8.

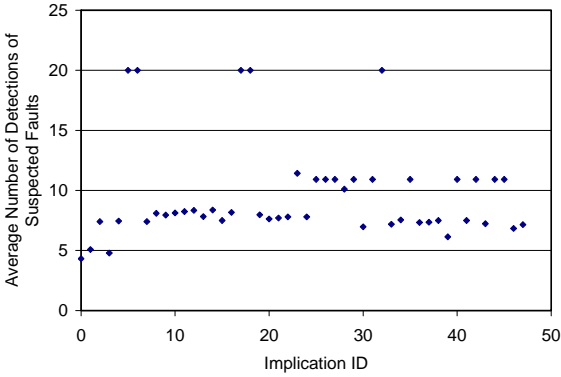


Figure 3: Average Number of Detections for Suspect Faults in c432.

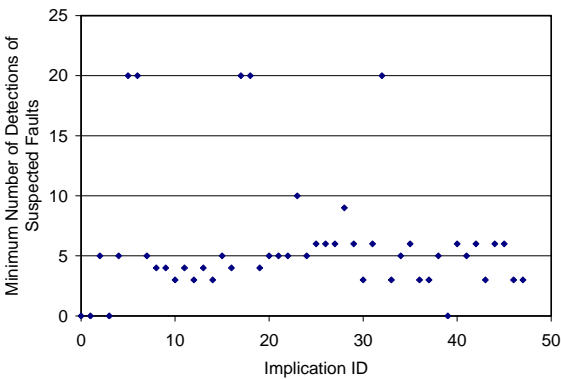


Figure 4: Minimum Number of Detections for Suspect Faults in c432

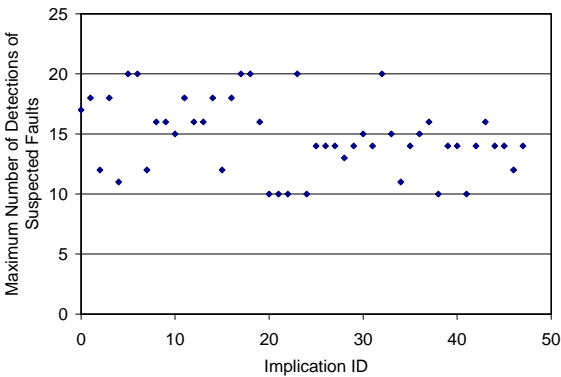


Figure 5: Maximum Number of Detections for Suspect Faults in c432.

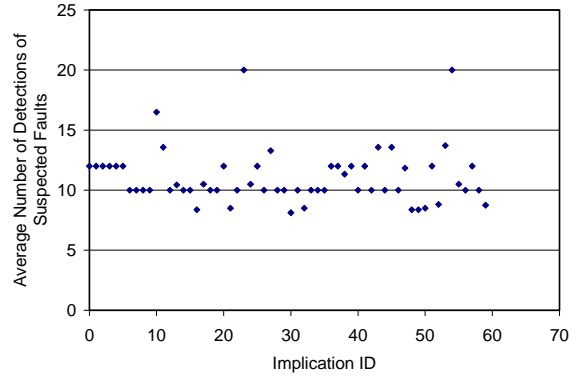


Figure 6: Average Number of Detections for Suspect Faults in c499

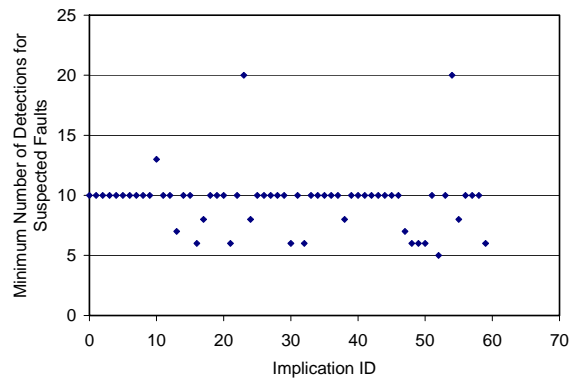


Figure 7: Minimum Number of Detections for Suspect Faults in c499

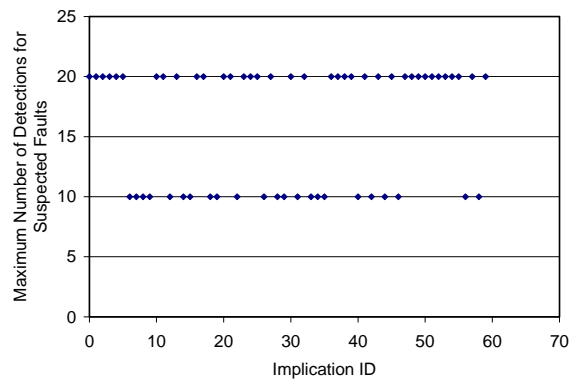


Figure 8: Maximum Number of Detections for Suspect Faults in c499.

Each figure shows the results from implications up to 30% overhead where the implications are sorted in the order in which they are added to the implication set. Thus, the first third of each chart represents the data from the implications corresponding to an overhead of roughly 10%. The first two-thirds

correspond to implications corresponding to an overhead of roughly 20%, etc.

It is clear from the graphs that numerous detections can be achieved of many faults for all implications. In general, the average number of fault detections varies between 4.3 and 20 depending on the implication and circuit. For a few cases in c432, however, the chosen test set misses a few of the faults that could have caused the implication to fail (i.e. faults in the suspect fault list) and the minimum number of fault detections is equal to 0. We believe that this is due to the fact that some of the implications in c432 can potentially detect a very large number of faults. Thus, while those implications are good for online error detection, they provide low diagnostic resolution, and it is difficult to detect so many faults with such a small test set. Adding a few “top-off” patterns would enable these faults to be detected at least once. This problem does not occur in c499 because each implication generally detects faults in a much smaller area of the circuit. Furthermore, there is a significant amount of symmetry in c499 related to the implications that exist and the faults that are detected. Thus, many of the implications exhibit similar behavior.

8. Conclusions

In this paper, we have described a method for identifying the probable locations of errors in circuits protected by logic implications. We have shown that because logic implications can only cover a limited area of the circuit, they can often provide good diagnostic information when a failure occurs.

Furthermore, once the possible fault locations have been identified, those locations can be explicitly targeted for subsequent online testing. We have presented a method for choosing a subset of test patterns from a superset stored in memory based merely upon which implication caused the failure. Such test sets can be very short and yet can generally detect every targeted fault multiple times—thoroughly testing the areas of interest. In those cases where less diagnostic resolution is possible because an implication can detect many faults, a slight increase in the overall size of the test set may be necessary to obtain good coverage.

Future work will investigate combining information across implications to obtain additional diagnostic information, the excitation balance attained by the chosen test sets, and additional fault models.

9. References

- [1] S. Borkar, “Thousand Core Chips - A Technology Perspective,” *44th ACM/IEEE Design Automation Conference, 2007. DAC '07.*, 2007, pp. 746-749.
- [2] M. Agarwal, B. Paul, Ming Zhang, and S. Mitra, “Circuit Failure Prediction and Its Application to Transistor Aging,” *25th IEEE VLSI Test Symposium, 2007.*, 2007, pp. 277-286.
- [3] S. Borkar, “Designing reliable systems from unreliable components: the challenges of transistor variability and degradation,” *IEEE Micro*, vol. 25, 2005, pp. 10-16.
- [4] J. von Neumann, “Probabilistic logics and synthesis of reliable organisms from unreliable components,” *Automata Studies*, 1956, pp. 43-98.
- [5] S. Mitra and E. McCluskey, “Which concurrent error detection scheme to choose ?,” *International Test Conference, 2000*, pp. 985-994.
- [6] B. Bose and Der Jei Lin, “Systematic Unidirectional Error-Detecting Codes,” *IEEE Transactions on Computers*, vol. C-34, 1985, pp. 1026-1032.
- [7] S. Ghosh, N. Touba, and S. Basu, “Synthesis of low power CED circuits based on parity codes,” *Proceedings of the 23rd IEEE VLSI Test Symposium*, 2005, pp. 315-320.
- [8] D. Das and N. Touba, “Synthesis of circuits with low-cost concurrent error detection based on Bose-Lin codes,” *Proceedings of the 16th IEEE VLSI Test Symposium*, 1998, pp. 309-315.
- [9] K. De, C. Natarajan, D. Nair, and P. Banerjee, “RSYN: a system for automated synthesis of reliable multilevel circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, 1994, pp. 186-195.
- [10] N. Touba and E. McCluskey, “Logic synthesis of multilevel circuits with concurrent error detection,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, 1997, pp. 783-789.
- [11] S. Almkhaizim, P. Drineas, and Y. Makris, “Cost-driven selection of parity trees,” *22nd IEEE VLSI Test Symposium, 2004. Proceedings.*, 2004, pp. 319-324.
- [12] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba, “Synthesis of low-cost parity-based partially self-checking circuits,” *9th IEEE On-Line Testing Symposium, IOLTS 2003*, 2003, pp. 35-40.
- [13] P. Samudrala, J. Ramos, and S. Katkooari, “Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant

- synthesis for FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 51, 2004, pp. 2957-2969.
- [14] C.F. Webb and J.S. Liptay, “A high-frequency custom CMOS S/390 microprocessor,” *IBM J. Res. Dev.*, vol. 41, 1997, pp. 463-473.
- [15] R. Drechsler, “Synthesizing checkers for on-line verification of System-on-Chip designs,” *Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS '03*, 2003, pp. IV-748-IV-751 vol.4.
- [16] M. Boule, J. Chenard, and Z. Zilic, “Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis,” *8th International Symposium on Quality Electronic Design, 2007. ISQED '07*, 2007, pp. 613-620.
- [17] N. Alves, K. Nepal, J. Dworak, and R.I. Bahar, “Detecting errors using multi-cycle invariance information,” *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09*, 2009, pp. 791-796.
- [18] K. Nepal, N. Alves, J. Dworak, and R. Bahar, “Using Implications for Online Error Detection,” *IEEE International Test Conference, ITC 2008*, 2008, pp. 1-10.
- [19] N. Alves, A. Buben, K. Nepal, J. Dworak, and R.I. Bahar, “A Cost Effective Approach for Online Error Detection Using Invariant Relationships,” *accepted for publication in IEEE Transaction on CAD*, 2010.
- [20] I. Pomeranz and S. Reddy, “3-weight pseudo-random test generation based on a deterministic test set for combinational and sequential circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, 1993, pp. 1050-1058.
- [21] R. Kapur, S. Patil, T. Snethen, and T. Williams, “Design of an efficient weighted random pattern generation system,” *International Test Conference*, 1994, pp. 491-500.
- [22] Z. He, G. Jervan, Z. Peng, and P. Eles, “Power-constrained hybrid BIST test scheduling in an abort-on-first-fail test environment,” *Proceedings of the 8th Euromicro Conference on Digital System Design*, 2005, pp. 83-86.
- [23] R. Ubar, M. Jenihhin, G. Jervan, and Zebo Peng, “Hybrid BIST optimization for core-based systems with test pattern broadcasting,” *Second IEEE International Workshop on Electronic Design, Test and Applications, DELTA 2004*, 2004, pp. 3-8.
- [24] G. Jervan, T. Shchenova, and R. Ubar, “Hybrid BIST Scheduling for NoC-Based SoCs,” *24th Norchip Conference*, 2006, pp. 141-144.
- [25] S. Hellebrand, H. Wunderlich, and A. Hertwig, “Mixed-mode BIST using embedded processors,” *Proceedings of the International Test Conference*, 1996, pp. 195-204.
- [26] Y. Li, S. Makar, and S. Mitra, “CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns,” *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 885-890.
- [27] H. Inoue, Yanjing Li, and S. Mitra, “VAST: Virtualization-Assisted Concurrent Autonomous Self-Test,” *IEEE International Test Conference, ITC 2008*, 2008, pp. 1-10.
- [28] J. Dworak, M. Grimaila, Sooryong Lee, L. Wang, and M. Mercer, “Enhanced DO-RE-ME based defect level prediction using defect site aggregation-MPG-D,” *Proceedings of the 2000 International Test Conference*, 2000, pp. 930-939.
- [29] I. Hamzaoglu and J. Patel, “Test set compaction algorithms for combinational circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, 2000, pp. 957-963.