FAULT TOLERANT METHODS FOR RELIABILITY IN FPGAs

Edward Stott, Pete Sedcole, Peter Y. K. Cheung

Electrical and Electronic Engineering
Imperial College, London, SW7 2BT
{edward.stott07,pete.sedcole,p.cheung}@ic.ac.uk

ABSTRACT

Reliability and process variability are serious issues for FPGAs in the future. Fortunately FPGAs have the ability to reconfigure in the field and at runtime, thus providing opportunities to overcome some of these issues. This paper provides the first comprehensive survey of fault detection methods and fault tolerance schemes specifically for FPGAs, with the goal of laying a strong foundation for future research in this field. All methods and schemes are qualitatively compared and some particularly promising approaches highlighted.

1 INTRODUCTION

As process technology scaling continues, manufacturing large defect-free integrated circuits become increasingly difficult. There is also the added problem of degradations over time *after* a device has been successfully deployed in the field. These reliability issues are particularly acute with FPGAs. Similar to memories, FPGAs have high density of transistors *and* interconnect wires. A recent study [1] suggested that future FPGAs at and beyond the 45nm technology node will have such low yield that defect tolerance scheme will be unavoidable in large FPGAs. This paper provides the first comprehensive survey on the issues of reliability, defect detection and fault tolerance methods in FPGAs over the past 15 years.

FPGAs present interesting opportunities for fault tolerance due to their ability to be reconfigured. The most promising systems that have been proposed exploit this feature very heavily.

A fault tolerant system consists of two main components; these are fault detection and fault repair. Section 3 surveys fault detection methods and Section 4 considers fault repair. Causes of faults and application issues are briefly discussed in Section 2.

2 BACKGROUND

2.1 Causes of degradation

Degradation in VLSI circuits can be attributed to a number of mechanisms [2]. The *hot-carrier effect* leads to a build up of trapped charges in the gate-channel interface region [3]. This causes a gradual reduction in channel mobility and increase in threshold voltage in CMOS transistors. The effect on the circuit is that switching speeds become slower, leading to delay faults. This is effect is also seen as a result of *Negative-Bias Temperature Instability*, which exhibits a similar build up of trapped charges [4].

Electromigration is a mechanism by which metal ions migrate over time leading to voids and deposits in interconnects. Eventually these can cause faults due to the creation of open and short circuits [5].

Time-Dependent Dielectric Breakdown (TDDB) affects the gates of transistors, causing an increase in leakage current and eventually a short circuit. The mechanism here is the creation of charge traps within the gate dielectrics, diminishing the potential barrier it forms [6] [7].

All of these degradation mechanisms have the potential to become more severe with the shrinking of process geometry. This is due to increasing gate field strength, higher current density, smaller feature size, thinner gate dielectrics and increasing variability. In the case of TDDB, the situation is made complicated by the introduction of new processes such as high-K dielectrics and metal gates [8].

2.2 Other types of fault

In addition to degradation, there are two other types of faults which can affect FPGAs. These are highly relevant to this study as some of the techniques which have been developed in response to them can also be applied to faults caused by degradation.

The first of these is *manufacturing defects*. Manufacturing defects can be exhibited as circuit nodes which are stuck-at 0 or 1 or switch too slowly to meet the timing specification. Defects also affect the interconnect network and can cause short or open circuits and stuck open or closed pass transistors [9]. Test and repair of manufacturing defects is well established in VLSI.

The second type of fault which is widely discussed in relation to FPGAs comprises of *Single Event Upsets* (SEUs) and Single Event Transients (SETs) caused by certain types of radiation. This is of particular concern to aviation, nuclear research and space applications where devices are exposed to higher levels of radiation. The most commonly considered failure mode is the flipping of an SRAM cell in the configuration memory. This causes an error in the logic function which persists until the configuration memory is refreshed in a process known as scrubbing. Whilst this recovery method is not applicable to permanent faults caused by degradation, ways of detecting SEU faults are relevant.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

Table 1. Comparison Matrix of Fault Detection Methods

Method	Speed of detection	Resource overhead	Performance overhead	Granularity	Coverage
Modular Redundancy	Fast – As soon as fault is manifest	Very Large – Triplicate plus voting logic	Very Small – Latency of voting logic	Coarse – Limited to size of module	Good - All manifest errors are detected.
Concurrent error detection	Fast – As soon as fault is manifest	Medium – trade-off with coverage	Small – Additional latency of CRC logic	Medium – trade-off with resource	Medium – Not practical for all types of functionality.
Off-line BIST	Slow – only when off-line	Very small	Small – Slight start-up delay	Fine – Possible to detect the exact error	Very Good – All faults including dormant.
Roving (Segmented Interconnect)	Medium – order 1 seconds	Medium – Empty test block plus test controller	Large – Clock must be stopped to swap blocks. Critical paths may lengthen	Fine – Possible to detect the exact error	Very Good – Multiple Manifest and latent faults are detected.

2.3 Applications of fault tolerance

All of the fault detection and repair methods surveyed have individual strengths and weaknesses and which method is most appropriate depends on the application.

In some cases reliability is critical for safety or mission success [10]. Fast detection is crucial here so that erroneous data or state is not acted upon. Another class of reliable system is one where a physical repair is not practical. Fault coverage and repair flexibility are important here.

In the light of variability and reliability concerns associated with future VLSI process nodes, it may become economical to use fault tolerance in general purpose, high volume applications. In this case, it will be important that the detection and repair method has the lowest possible overhead on timing performance and area.

3 METHODS OF FAULT DETECTION

The first step of a fault-tolerant scheme is fault detection. Fault detection has two purposes; firstly, it alerts the supervising process that action needs to be taken for the system to remain operational and, secondly, it identifies which components of the device are defective so that a solution can be determined. These two functions may be covered simultaneously, or it may be a multi-stage process comprising of different strategies.

Fault detection methods can be categorised into three broad types:

1. **Redundant/concurrent error detection** uses additional logic as a means of detecting when a logic function is not generating the correct output.
2. **Off-line test methods** cover any testing which is carried out when the FPGA is not performing its operational function.
3. **Roving test methods** perform a progressive scan of the FPGA structure by swapping blocks of functionality with a block carrying out a test function.

The different approaches to fault detection are evaluated against a set of metrics in Table 1.

3.1 Functional redundancy and Concurrent Error Detection (CED)

Redundancy is widely used as a method of fault detection in FPGAs, particularly in the form of Triple Modular Redundancy (TMR) [11] [12] [13] [14]. The main driver for error detection of this kind is the need to detect and correct errors due to SEUs and SETs. However, these methods will detect any error which occurs whilst the system is operating.

These detection methods work by invoking more than the minimum amount of logic than is necessary to implement the logic function. When an error occurs, there is a disagreement between the multiple parts of the circuit over which a particular calculation is processed and this is flagged by some form of error detection function.

The simplest form is *modular redundancy*. A functional block is replicated, usually two or three times and the outputs compared. Any difference indicates that a fault is present.

Concurrent Error Detection (CED) allows a more space efficient design than modular redundancy. Data flows and stores are widened using error coding algorithms such as parity. Data validation circuitry at the output to functional blocks can then detect faults which arise.

Redundancy provides a very fast means of error detection, as a fault is uncovered as soon as a discrepancy occurs. In addition, this form of error detection has a small impact on timing performance; just the latency of voting or parity logic, or similar.

The chief drawback of error detection using redundancy is the area overhead needed to replicate functionality, which can be over three times in the case of TMR [15]. Furthermore it provides a limited resolution for identification of the faulty component. The fault can only be pinned down to a particular functional block or, in the case of TMR, an instance of a functional block. Fault resolution can be increased to a certain extent by breaking functional areas down and adding additional error detection logic.

Modular redundancy detects all faults which become manifest at the output of a functional block, including transient errors. In CED, coverage comes at the expense of additional area. These methods provide no coverage of dormant faults, which may be relevant if an FPGA is going to be reconfigured in the field, either for fault repair or to alter the functionality.

Redundancy does not have to be restricted to the circuit-area dimension. It is also possible to detect errors in a trade-off with latency/data throughput. [16] proposes a scheme where operations are carried out twice. In the second operation, operands are encoded in such a way that they exercise the logic in a different way. The output is then passed through a suitable decoder and compared to the original.

Although most of the work on redundancy has been aimed at detecting and correcting SEUs, there have been some notable publications which apply the techniques to fault detection. [17] uses Dual Modular Redundancy (DMR) to grade the 'fitness' of competing configurations in an evolutionary approach. Parity checking is used in [18] as part of a fault tolerant scheme which is structured so that detection is applied to small regular networks, rather than being bespoke to the function that is implemented.

Redundant and data-checking detection systems are generally designed into an FPGA configuration, as they fit around the specific data and control functions that are implemented. In [19], a FPGA structure was considered which has redundancy built in, so that it is transparent to the user who is designing the configuration.

3.2 Off-line fault detection / Built-In Self-Test

Off-line fault detection is another widely-used technique; usually as a means of quickly identifying manufacturing defects in FPGAs. Any scheme which does this without any external test equipment is known as *Built-In Self-Test* (BIST), and is a suitable candidate for fault detection in the field.

BIST schemes for FPGAs work by having one or more test configurations which are loaded separately to the operating configuration. Within the test configuration is a test pattern generator, an output response analyser and, between them, the logic and interconnect to be tested, arranged in paths-under-test. To be fully comprehensive, a BIST system will have to test not only the logic and interconnect, but also the configuration network. Many recent FPGAs feature a self-configuration port which can make this possible without the need for a large number of different test configurations, which can take a long time to load. Specialised features such as carry chains, multipliers and PLLs also need to be considered.

Compared to traditional built-in and external test methods for ASICs, FPGAs have the advantage of a regular structure which does not need a new test programme to be developed for each design. Also, the ability to reconfigure an FPGA reduces or removes the need for dedicated test structures to be built into the silicon. However, with the ability to reconfigure comes a vast number of permutations in the way the logic can be expressed, making optimisation of test patterns important.

The advantage of BIST as a fault detection method is that it has no impact on the FPGA during normal operation. The only overhead is the requirement to store test configurations which are typically small. BIST also allows complete coverage of the FPGA fabric, including features which may be hard to test with an on-line test system, such as PLLs and the clock network.

The major drawback of BIST is that it can only detect faults during a dedicated test mode when the FPGA is not otherwise operational. This can be either during system start-up, as part of a maintenance schedule or in response to an error detected by some other means.

Published BIST methods have competed for coverage, test duration and memory overhead. Many focus on testing just one subset of FPGA structures, e.g. interconnect, suggesting a multi-phased approach for testing the whole chip.

Testing of LUTs is a mature field [20] [21] [22] and recent developments have considered timing performance as well as stuck-at faults. [23] and [24] have both proposed methods for analysing the propagation delays of logic chains. [25] and [26] have considered the optimum test patterns for exercising delay faults.

Many publications have focussed on testing interconnect in response to the large amount of configuration logic and silicon area it consumes [27] [28]. In [29], a BIST system for interconnect is given which reduces test time through a large degree of self-configuration. [9] and [30] use a hierarchical approach which locates stuck-at faults, short circuits and open circuits with the highest accuracy. Elements of BIST can be found in roving test systems, where only a small part of the FPGA is taken off-line for testing at any point in time. [23] cites both roving and off-line testing as suitable applications for their delay-test method. [31] proposes an off-line test which uses a roving sequence to remove the need for reconfiguration; instead a small self-test area is always present and is shifted around the array to gain full coverage.

3.3 Roving fault detection

Roving detection exploits run-time reconfiguration to carry out BIST techniques on-line, in the field, with a minimum of area overhead. In roving detection, the FPGA is split into equal-sized regions. One of these is configured to perform self-test, while the remaining areas carry out the design function of the FPGA. Over time, the test region is swapped with functional regions one at a time so that the entire array can be tested while the FPGA remains functional.

Roving test has a lower area overhead than redundancy methods; the overhead comprising of one self-test region and a controller to manage the reconfiguration process. The method also gives excellent fault coverage and granularity, comparable to BIST methods.

The speed of detection, while better than off-line methods that cannot test during normal operation, is not as good as redundancy techniques. The detection speed depends on the period of a complete roving cycle; the best reported implementations of roving test have maximum detection latency in the order of a second [32].

Roving test impacts performance in two ways. Firstly, as the test region is moved through the FPGA, connections between adjacent functional areas are stretched. This results in longer signal delays and may force a reduction in the system clock speed, reported to be in the range of 2.5-15% [32]. Secondly, implementations in current FPGAs require the functional blocks to be halted as they are switched. A 250μs pause for each swapping move has been reported [32].

The dominant work in the field of roving test and repair has been carried out by Emmert, Stroud and Abramovici [32] [33]. Called Roving STARs, this system uses two test areas, one covering entire rows and one covering entire columns. A roving test method was also proposed in [34] that uses FPGAs with bussed, rather than segmented, interconnects. This system had no impact on system clock performance. However, the connectivity constraints of this architecture limit the potential applications.

4 METHODS OF FAULT TOLERANCE AND RECOVERY

Once a fault is detected and located, it must be repaired. Repair can be considered at a number of different levels:

1. **Hardware level repair** performs a correction such that the FPGA remains unchanged for the purposes of the configuration. The device retains its original number and arrangement of useable logic clusters and interconnects.
2. **Configuration level repair** is achieved using resources that are unused by the design. The spare resources can replace faulty ones in the event of a fault.
3. **System level repair** works at a higher level. When a design is highly modular, a fault can be tolerated by the use of a spare functional block or by providing degraded performance [35]. Such methods are not considered in more detail here, as they are not limited in application to FPGAs.

It should be noted that some fault detection methods also provides a level of fault tolerance. The voting system in TMR allows the erroneous output of one module to be ignored. Also, roving test provides fault tolerance by stopping the roving process if a fault is detected. If the fault stays within the test area it will not be used by the operational part of the FPGA. In both these situations, the system operates in a reliability degraded state where another fault would not be tolerated and may not even be detected. But they do allow the system to carry on functioning whilst a permanent repair is carried out.

Table 2 shows the classes of fault repair techniques that have been reported and evaluates them against a range of metrics.

4.1 Hardware Level Repair

The regular structure of FPGAs makes them suitable candidates for hardware level repair, using methods similar to those used for defect tolerance in memory devices. In the event of a fault, a part of the circuit can be mapped to another part with no change in function.

Hardware level repair has the advantage of being transparent to the configuration. This makes repair a simple process, as the repair controller does not need any knowledge of the placement and routing of the design. Another benefit is that the timing performance of the repaired FPGA can be guaranteed, as any faulty element will be replaced by a pre-determined alternative.

Hardware level fault tolerance has a drawback in that it can tolerate just a low number of faults for a given overhead and there are likely to be certain patterns of faults which cannot be tolerated.

The most common basis for hardware level repair is *column/row shifting* [36] [37]. Multiplexers are introduced at the ends of lines of cells, allowing a whole row or column to be bypassed by shifting across to a spare line of cells at the end of the array. If the FPGA is bus-based, the shifted cells can connect to the same lines of interconnect. For segmented interconnect, bypass sections need to be added to skip the faulty row / column. This method can be found today in some commercial devices [39]

Adding more bypass connections and multiplexers allows greater flexibility for tolerating multiple faults and makes more efficient use of spare resources [38] [39]. In [40], faults in the configuration logic where considered and the proposed solution was to split the FPGA up into sub-arrays which can be configured independently.

4.2 Configuration Level Repair

Configuration level repair exploits two key features in FPGAs; reconfiguration and the availability of unused resources. Configuration level repair strategies can be divided into three subclasses:

4.2.1 Alternative configurations

A straightforward way of achieving fault tolerance is to pre-compile alternative configurations. The FPGA is split into tiles, each with its own set of configurations which have a common functionality and interface to the adjacent tiles [41] [42] [43]. Fault tolerance is achieved by replacing a configuration tile with an alternative in which the faulty resource is not used. This method requires little

Table 2. Comparison Matrix of Fault Repair Methods

Method	Fault Pattern Tolerance	Resource Overhead	Performance Overhead / Degradation	Complexity of Repair
Hardware Level	Poor – Limited number and distribution tolerated.	Medium	Low overhead and little degradation	Low – Transparent to configuration
Multiple Configurations	Poor – Limited number and distribution tolerated. Interconnect tolerance causes complexity	Low – Uses naturally spare resources, but requires ROM for configurations	Low – Each configuration can be fully optimised	Medium – Selection and loading of configurations
Chain Shifting	Poor – Limited number and distribution tolerated. Poor for interconnect	Medium – A set of interconnect must be reserved	Low – Alternative routing is pre-determined	Low – Alternative routing already reserved
Pebble Shifting	Medium – Relies on nearby spare PLBs	Low – Uses naturally spare resources	Medium, rerouting causes uncertainty	High – Re-routing necessary
Cluster Reconfig.	Poor – Reliant on spare resource in cluster. Poor tolerance in interconnect	Low – Uses naturally spare resources	Low – Changes only local interconnect, slight uncertainty	Medium – Analysis of logic, no re-routing
Cluster Reconfig.+ Pebble Shifting	Good – Flexible solutions possible.	Low – Uses naturally spare resources	Low – Usually a fast alternative will be found, medium uncertainty	High – Analysis of logic and rerouting
Evolutionary	Good – Implementation is completely flexible	Large – Configuration grading and storage	Variable – Solution is arrived through random mutations.	Massive – May take a long time to repair

run-time computation as the placement and routing is already done.

This strategy performs relatively poorly in terms of area efficiency and fault pattern. It is dependent on there being a configuration available in which any given resource is set aside as a spare. If only a small amount of spare resource is available then a large number of configurations are needed to cover all possible faults.

4.2.2 Incremental mapping, placement and routing

A simple method of tolerating faults in logic clusters exists if the cluster can be reconfigured to work around the fault [44]. *Cluster reconfiguration* is simple to evaluate and does not have a significant effect on design timing, but there may be cases where reconfiguration is not possible, for example if an output register is faulty.

If there are spare clusters, then these can be used to replace faulty ones. To minimise the impact on timing and routing in the area around the fault, *pebble shifting* is used [45] [46]. In [32] and [33], pebble shifting is used in combination with cluster reconfiguration. Cluster reconfiguration is carried out in preference and faulty clusters can be reused by a different function if the fault will not be manifest.

It is also important to consider faults in interconnect. In [47], the design of switch blocks is considered with regard to the ease of finding an alternative path in the event of a fault. An incremental router is developed in [48] which uses both cluster relocation and interconnect faults as drivers for rerouting. [49] provides a novel method that uses a small amount of configuration during power-up to avoid defects in interconnects.

Incremental mapping, placement and routing provide a high degree of flexibility for dealing with random fault patterns, especially when cluster reconfiguration and pebble shifting are used together. However, this comes at the cost of increased computational effort for the repair which must be carried out in the field.

In [50], a repair method known as *chain shifting* is given. Spare clusters and interconnect are allocated at design-time in order to reduce the complexity of a repair in the field and to provide guaranteed timing performance. This technique, however, reduces flexibility and overhead efficiency. In [18] a more coarse-grained approach is taken. Clusters are arranged into networks such that most faults can be repaired by reconfiguring the network and leaving the wider placement unchanged.

4.2.3 Evolutionary algorithms

Reconfiguration makes FPGAs well suited to evolutionary algorithms. [17] proposed a complete system for achieving fault tolerance, based on a pool of competitive configurations. Configurations compete for correctness and those which are faulty are mutated. Synthesis by evolutionary algorithms was tested in [51] and [52].

Although this approach allows a large degree of flexibility with the number and distribution of faults that can be tolerated, the area and computational overhead required is very large. There is also no guarantee of how long a solution will take to evolve or what its timing performance will be.

5 CONCLUSION

The regularity of FPGA architectures together with their run-time configurability provides interesting opportunities for both defect detection and fault tolerance. In particular, the ability to run-time reconfigure offers the possibility of detecting and coping with faults due to degradation over time, something quite unique to the FPGA technology. This may prove to be a characteristic that allows VLSI to continue scaling down to 22nm and beyond.

6 REFERENCES

1. N. Campregher et al, "Analysis of yield loss due to random photolithographic defects in the interconnect structure of FPGAs", *ACM Int. Workshop on FPGAs*, p.138-148, 2005.
2. S. Srinivasan et al, "FLAW: FPGA lifetime awareness", *Design Automation Conference*, p.630-635, 2006.
3. C. Guérin et al, "The Energy-Driven Hot-Carrier Degradation Modes of nMOSFETs", *IEEE Transactions on Device and Materials Reliability*, v7, n2, June 2007.
4. K. Dieter et al, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing", *Journal of Applied Physics*, v94, n1, July 2003.
5. P.J. Clarke et al, "Electromigration - a tutorial introduction", *International Journal of Electronics*, 69:3, p333 – 338, 1990.
6. D. Esseni et al, "On Interface and Oxide Degradation in VLSI MOSFETs—Part I: Deuterium Effect in CHE Stress Regime", *IEEE Transactions on Electron Devices*, v49, n2, February 2002.
7. D. Esseni et al, "On Interface and Oxide Degradation in VLSI MOSFETs—Part II: Fowler–Nordheim Stress Regime", *IEEE Transactions on Electron Devices*, v49, n2, February 2002.
8. K.P. Cheung, "Can TDDB continue to serve as reliability test method for advance gate dielectric?", *Int. Conference on Integrated Circuit Design and Technology*, 2004.
9. I.G. Harris et al, "Testing and diagnosis of interconnect faults in cluster-based FPGA architectures", *IEEE Transactions on CAD of Integrated Circuits and Systems*, v21, n11, p 1337-43 Nov. 2002.
10. A. Steininger et al, "On the Necessity of On-line-BIST in Safety-Critical Applications", Int. Symp. On Fault-Tolerant Computing, p.208-215, 1999.
11. S. D'Angelo et al, "Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems", *Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, p 233-40, 1998.
12. S. D'Angelo et al, "Transient and permanent fault diagnosis for FPGA-based TMR systems", *IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, p 330-8, 1999.
13. G.A. Mojoli et al, "KITE: A behavioural approach to fault-tolerance in FPGA-based systems", *International Workshop on Defect and Fault Tolerance in VLSI Systems*, p 327-334, 1996.
14. C. Carmichael, "Triple Module Redundancy Design Techniques for Virtex FPGAs", *Xilinx Application Note XAPP197*, 2006.
15. M. Berg, "Fault tolerance implementation within SRAM based FPGA designs based upon the increased level of single event upset susceptibility", *Int. On-Line Testing Symposium*, 2006.
16. F. Lima et al, "Designing fault tolerant systems into SRAM-based FPGAs", *Design Automation Conference*, p 650-655, 2003.
17. R.F. DeMara et al, "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration", *NASA/DoD Conference of Evolution Hardware*, 2005.
18. M. Alderighi et al, "A fault-tolerant FPGA-based multi-stage interconnection network for space applications", *IEEE Int. Workshop on Electronic Design, Test and Applications*, p. 302-6, 2002.
19. S. Durand et al, "FPGA with self-repair capabilities", *Int. Workshop on Field Programmable Gate Arrays*, p.1-6, 1994.
20. C. Stroud et al, "Built-In Self-Test of Logic Blocks in FPGAs", *14th VLSI Test Symposium*, 1996.
21. S. Lu et al, "Fault detection and fault diagnosis techniques for lookup table FPGAs", *VLSI Design*, v 15, n 1, p 397-406, 2002.
22. A. Alaghi et al, "An optimum ORA BIST for multiple fault FPGA look-up table testing", *Asian Test Symposium*, p 293-298, 2006.
23. C. Stroud et al, "BIST-Based Delay-Fault Testing in FPGAs", *Journal of Electronic Testing*, 19, p549–558, 2003.
24. J. Wong et al, "Self-characterization of Combinatorial Circuit Delays in FPGAs", *Int. Conf. on Field Programmable Tech.*, p.17-23, 2007.
25. P. Girard et al, "High Quality TPG for Delay Faults in Look-Up Tables of FPGAs", *Int. Workshop on Electronic Design, Test and Applications*, 2004.
26. P. Girard et al, "Defect analysis for delay-fault BIST in FPGAs", *Int. On-Line Testing Symposium*, p 124-8, 2003.
27. J. Liu et al, "BIST-diagnosis of interconnect fault locations in FPGA's", *Canadian Conference on Electrical and Computer Engineering*, p 207-10, 2003.
28. N. Campregher et al, "BIST based Intercconect Fault Location for FPGAs", *FPL'04,* LNCS 3203, p.322-332, 2004.
29. J. Smith et al, "An automated BIST architecture for testing and diagnosing FPGA interconnect faults", *Journal of Electronic Testing: Theory and Applications*, v 22, n 3, p 239-53, 2006.
30. I. Harris et al, "Diagnosis of interconnect faults in cluster-based FPGA architectures", *Int. Conf. on Computer Aided Design*, p472-5, 2000.
31. A. Doumar et al, "Testing Approach within FPGA-based Fault Tolerant Systems", *IEEE Asian Test Symp.*, p.411, 2000.
32. M. Abramovici et al, "Roving STARs: An Integrated Approach to On-Line Testing, Diagnosis, and Fault Tolerance for FPGAs", *NASA/DoD Workshop on Evolvable Hardware,* p.73, 2001.
33. J.M Emmert et al, "Online Fault Tolerance for FPGA Logic Blocks", *IEEE Trans. on VLSI Systems*, v15, n2, February 2007.
34. N.R. Shnidman et al, "On-Line Fault Detection for Bus-Based Field Programmable Gate Arrays", *IEEE Transactions on VLSI Systems*, v6, n4, December 1998.
35. Y. Nakamura et al, "Highly fault-tolerant FPGA processor by degrading strategy", *Pacific Rim International Symposium on Dependable Computing*, p75-8, 2002.
36. J. Kelly et al, "A novel approach to defect tolerant design for SRAM based FPGAs", *Int. Workshop on FPGAs*, 1994.
37. F. Hatori et al, "Introducing redundancy in field programmable gate arrays", *Custom Integrated Circuits Conference*, 1993.
38. J. Kelly et al, "Defect tolerant SRAM based FPGAs", *Int. Conference on Computer Design*, p 479-82, 1994.
39. C. McClintock et al, "Redundancy Circuitry for Logic Circuits", US Patent 6,6166,559, Dec., 2000.
40. N.J. Howard et al, "The Yield Enhancement of Field-Programmable Gate Arrays", *IEEE Trans. on VLSI Systems*, v2, n1, March 1994.
41. J. Lach et al, "Enhanced FPGA Reliability Through Efficient Run-Time Fault Reconfiguration", *Transactions on Reliability*, v49, n3, Spetember 2000.
42. J. Lach et al, "Low overhead fault-tolerant FPGA systems", *IEEE Transactions on VLSI Systems*, v 6, n 2, p 212-21, June 1998.
43. J. Lach et al, "Algorithms for efficient runtime fault recovery on diverse FPGA architectures", *Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, 1999.
44. V. Lakamraju et al, "Tolerating operational faults in cluster-based FPGAs", *ACM International Workshop on FPGAs*, 2000.
45. J.M. Emmert et al, "Partial reconfiguration of FPGA mapped designs with applications for fault tolerance and yield enhancement", *Int. Workshop on Field Programmable Logic and Applications*, LCNS 1304, p.141-150, 1997.
46. J. Narasimhan, "Yield enhancement of programmable ASIC arrays by reconfiguration of circuit placements", *IEEE Trans. on CAD of Integrated Circuit Systems*, p.976-986 1994.
47. J. Huang et al, "Fault tolerance of switch blocks and switch block arrays in FPGA", *Trans. on VLSI Systems*, v13, n7, p794-807, 2005.
48. J.M. Emmert et al, "A fault tolerant technique for FPGAs", *Journal of Electronic Testing*, v16, n6, p591-606, 2000.
49. N. Campregher et al, "Reconfiguration and Fine-Grained Redundancy for Fault Tolerance in FPGAs", *Int. Conf. on Field Programmable Logic (FPL)*, p.455-460, 2006.
50. F. Hanchek et al, "Node-covering based defect and fault tolerance methods for increased yield in FPGAs", *Int. Conf. on VLSI Design*, 1996.
51. A.P. Shanthi et al, "Exploring FPGA structures for evolving fault tolerant hardware", *NASA/DoD Conference on Evolvable Hardware*, p 174-81, 2003.
52. G.V. Larchev et al, "Evolutionary Based Techniques for Fault Tolerant Field Programmable Gate Arrays", *Int. Conference on Space Mission Challenges for Information Technology*, 2006.