Combined Defect and Fault Tolerance for Reconfigurable Nanofabrics

David de Andrés, Juan-Carlos Ruiz, Daniel Gil, Pedro Gil
Fault Tolerant Systems Research Group (GSTF), Universidad Politécnica de Valencia (UPV)
DISCA – ETS Informática Aplicada, Campus de Vera s/n, E-46022, Valencia, Spain
Phone: +34 96 3877007 Ext {75752, 85703, 75777, 79707}, Fax: +34 96 3877579
{ddandres, jcruizg, dgil, pgil}@disca.upv.es

Abstract

Reconfigurable architectures represent a promising option for tolerating the extremely high defect and failure rates of emerging nanodevices. Different approaches have been devised throughout the years for coping with the occurrence of defects and faults in Field-Programmable Gate Arrays (FPGAs). However, due to the expected defect and fault rates, many of these methodologies cannot be directly applied to reconfigurable nanocircuits or are useless if applied in isolation. Computer Aided Deisgn (CAD) tools and new design flows for reconfigurable applications should be aware of the existing defects in nanofabrics and automatically include all the mechanisms required to tolerate the occurrence of faults. This work explores the existing methodologies that are eligible to tolerate defects and operational faults, and how they should be interwoven to increase the confidence the user can put on the correct operation of nanocircuits.

1. Introduction

The 2007 International Technology Roadmap for Semiconductors (ITRS) [1] expects to scale CMOS to 11 nm by 2022. Consequently, other physical mechanisms should be considered to extend scaling beyond the ITRS expectations.

Figure 1 shows a bottom-up representation of a system's hierarchy that summarizes different approaches which, combined in some yet to be defined way, may improve the current information processing paradigm. Emerging technologies such as 1D structures (nanowires (NW) and carbon nanotubes (CNT)), Single Electron Transistors (SET), Molecular Devices, Ferromagnetic Devices, and Spin Transistors, are expected to achieve densities of 10^{12} devices/cm^2 [2] in comparison with the 10^{10} transistors/cm^2 CMOS limit.

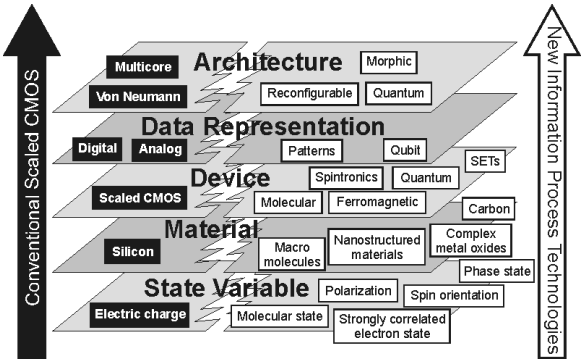

Figure 1. CMOS vs. emerging nanodevices [1].

However, the manufacturing process followed by many of these devices, which rely on self-assembly and self-alignment, largely increases the defect rate of the chip. Expected defect densities are around 10^{-1} per device while they are only 10^{-7}–10^{-6} per device (transistor) for present-day scaled CMOS [3]. So, testing the chip and rejecting it in the case of finding a small number of defects will no longer be an option. It seems that once again the old problem of how to develop reliable chips from unreliable devices [4] is in the limelight. It will be necessary to develop new solutions, at an architectural level, that can tackle the extremely large amount of defects expected.

Furthermore, new devices are likely to be fragile and very sensitive to their physical environment due to cosmic radiation, electromagnetic noise, thermal effects, and background charges. Hence, in order to exert the potential of projected densities, new architectural solutions must also be able to tolerate the occurrence of transient and permanent faults.

At present, one of the most promising architectures for nanocircuits is based on reconfigurable fabrics such as Programmable Logic Arrays (PLAs) [5] and Field-Programmable Gate Arrays (FPGAs) [6]. On the one hand, the stochastic nature of the manufacturing

process mainly allows building regular structures, such as 2D crossbars. Reconfigurable fabrics consist of a set of programmable logic resources that are interconnected by means of routing elements. These resources can be programmed or configured to change the functionality provided by the circuit. All of them may be implemented using regular structures which may act as PLA (for logic), interconnection (for routing), and memory (for configuration) cores. On the other hand, the reconfigurable capabilities of these structures make them very interesting to tolerate the high rate of manufacturing defects or the occurrence of operational faults in nanodevice-based circuits.

Although different approaches have already been proposed so far for defect and fault tolerance in FPGAs, the common design flow for reconfigurable application shown in Figure 2 should be modified to deploy those approaches on nanodevice-based circuits. This design flow first extracts, via a *synthesis* process, the interconnected set of logic elements the designer has specified by means of a model. The resources required to implement that model are obtained after *mapping* those logic elements to the selected *technology*. After that, these resources are distributed (*placed*) among the free resources of the FPGA and their interconnection is performed (*routed*). The result of this process is a configuration file that can be used to (re-)program the reconfigurable device.

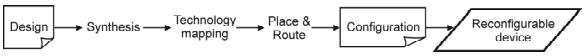

Figure 2. Reconfigurable computing design flow.

This paper focuses on how this design must evolve to obtain reliable applications out of unreliable nanodevice-based circuits. Section 2 introduces some work related to the applicability of fault tolerance techniques to nanocircuits. Section 3 and Section 4 review different FPGA-based defect and fault tolerance techniques that could be applied to these new architectures. Finally, Section 5 summarises the proposed defect and fault tolerance driven design flow.

2. Related work

Different redundancy-based techniques to tolerate high defect/failure rates in chips with densities of 10^{12} devices were analysed in [7]. The R-fold redundancy (RMR) and the NAND Multiplexing are generic techniques that were already proposed by von Neumann fifty years ago [4], and Reconfiguration is a technique specifically developed for FPGA systems.

Figure 3 compares the three methodologies in terms of the level of redundancy required for a given defect/failure rate to ensure that a chip with 10^{12} devices will work with a 90% of probability. Classical techniques do not behave very well since, for instance, a defect rate between 10^{-7}–10^{-6} (close to nowadays values) requires a redundancy level of 1000 for the RMR technique and only 10^9 devices can be used for logic implementation. NAND multiplexing can tolerate even higher defect/failure rates. Reconfiguration seems more promising, since it can tolerate defect/failure rates up to 10^{-1} which are close to what is expected for nanodevices. However, this is achieved at the expense of huge redundancy levels (10000).

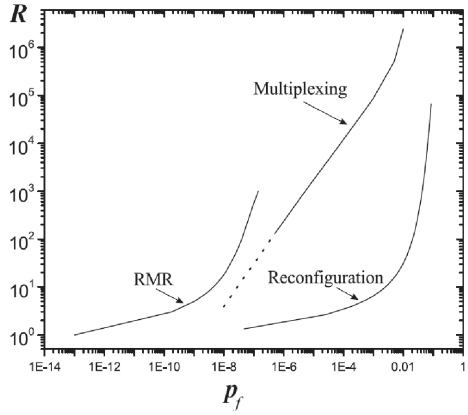

Figure 3. Required redundancy level (R) as a function of the failure rate per device (p_f) [7].

This clearly shows that unless defects/failures can be located and the system reconfigured to avoid them, the future use of nanodevices could be compromised.

3. Defect tolerance

The deployment of efficient fault tolerance techniques usually relies on a defect-free circuit. Hence, defects must be first located for the mapping, placement and routing processes to implement the desired defect-free functionality.

3.1. Finding defects

The offline detection and location of defects in reconfigurable fabrics [8] may be based on the programmability of the circuit or the design for testability (DFT). Since it is not feasible to implement a specific testing circuit out of nanodevices due to their enormous defect rate, it may be more interesting to take benefit of the intrinsic programmability of the circuit to locate its defects. Built-in self test/diagnosis (BIST/D) seems the most promising approach among the different proposed techniques [8].

The FPGA-based BIST/D approach [9] divides the reconfigurable logic blocks into nodes that are configured as Test Pattern Generators (TPG), Blocks Under Test (BUT), or Output Response Analysers (ORA). The TPG, which is usually a linear feedback shift register (LFSR) or a counter, feeds the inputs of the BUT. The ORA collects and analyses the outputs of the BUT by means of a comparator-based or signature analyser. All the blocks are tested (across and down the FPGA) to determine whether they are defect-free or not and the location of existing defects. A similar approach can be used to locate defects in routing resources.

This methodology should be modified to take into account the large defect rates of nanodevices. As only defect-free resources may be used to test adjacent neighbours, it could be impossible to determine whether some resources are defect-free or not. This leads to the definition of a measure known as *recovery* [3], which represents the percentage of defect-free resources that are successfully identified as such. Furthermore, blocks should be small enough to allow for a reasonable probability of being defect-free but not so small that no required logic could be implemented.

This approach was adapted in [10] to a particular nanodevice-based structure (nanofabric). In that case, each block (TPG and ORA) tests the three adjacent blocks (BUT), which in turn test their neighbours, and so on, in a wave-like manner until the whole circuit has been tested. The result of each test is stored to build the defect map of the circuit (see Figure 4). This process can be repeated from more than one corner of the circuit to increase the attainable recovery. It is to note that the initial corner must be tested from outside the chip with a defect-free (probably CMOS) tester. If all 4 corners are defective, any other block may be used at the cost of leaving some whole rows/columns untested.

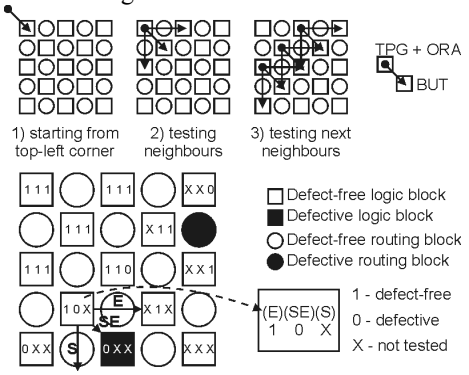

Figure 4. Wave-like BIST/D for nanocircuits [10].

Another approach, which aims at reducing the complexity of the TPG and the ORA, was presented in [11]. The BIST/D procedure is performed in parallel, so three stages are required to change the role played by each block (see Figure 5). Due to the particular architecture of the analysed nanofabric, three different TPG must be used. Thus, the final defect map must be derived from three partial ones. It is to note that, along the BIST/D procedure, a defect-free (probably CMOS) circuit will be required as a either TPG or ORA.

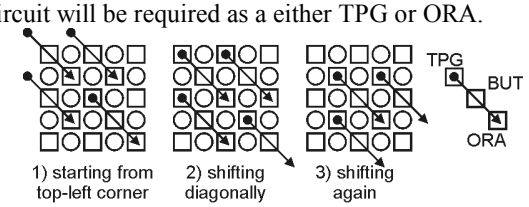

Figure 5. Parallel BIST/D for nanocircuits [11].

We can conclude that BIST/D-like approaches are suitable to be used in nanocircuits to locate defective nanofabric resources. Future research should focus on increasing the attainable recovery, decreasing the execution time and the dependence on external testers, and reducing the size of the obtained defect map.

3.2. Mapping around defects

Mapping, placement and routing processes, which are in charge of implementing a system function, should be aware of the obtained defect map in order to avoid defective resources. Traditionally, CAD tools select those resources that will implement the system from a pool containing all the available resources of the FPGA. So, it is just a matter of removing the defective resources from that pool to ensure that the final implementation will be defect-free.

In fact, a similar approach has been presented in [12] to handle multicore architectures in which up to 20-30% of the cores are defective. However, more aggressive approaches are required to handle the enormous defect rate of nanotechnologies: 20-30% of all devices (not cores but CMOS-like transistors) will be defective. So, most resources are likely to contain defects and it is not feasible to just throw them away.

The mapping and routing processes should determine whether a defective resource is still usable for implementing a specific system function. For instance, a defective routing element, which permanently connects two endpoints, may still be used if the system has to connect these precise endpoints. The placement process should know all this information to select those resources that will be finally used for implementing the system functionality.

This kind of strategy is considered in [13] to use defective logic arrays for implementing sets of OR terms. As shown in Figure 6, this is reduced to a matching problem.

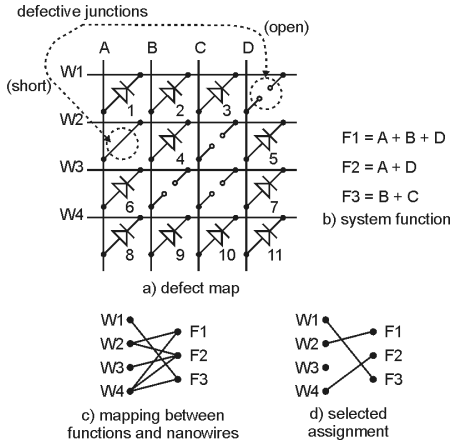

Figure 6. Mapping functions to a defective resource [13].

In this way it is possible to increase the actual recovery obtained at the defect location phase by effectively using defective resources that can accommodate the required logic. The main problem of this approach resides in the time required to compute whether the defective resource may implement the given system function. With 10^{12} devices per chip and defect densities of 10^{-1}, the obtained defect map will easily contain about 10^{11} elements. So, reducing the size of the defect map and devising new algorithms to accelerate the mapping process is a must.

4. Fault tolerance

Previously studied techniques enable the defect-free implementation of a system function on reconfigurable nanofabrics. However, the obtained defect map is static, i.e. (permanent) faults occurring during system operation will lead to new defects that might not be tolerated. This is also the case of transient faults, which will probably affect the behaviour of the system.

This section explores some techniques that should be included in a fault-tolerance aware design flow.

4.1. Redundancy

Although the expected failure rate remains unknown, it is possible to assume that it will be lower than the defect rate and higher than the current failure rate for CMOS devices. As defects have been now successfully tolerated, classical redundancy-based fault tolerance techniques may be accommodated by the large number of resources embedded in the nanofabric.

In that way, every system should be designed having fault tolerance in mind. However, adding fault tolerance capabilities is not always easy for designers who are not dependability experts. The development of tools supporting the (semi-)automatic deployment of these mechanisms before synthesising the design constitutes an issue for further research.

It is also necessary to take into account the particular characteristics of the reconfigurable circuits before implementing (placing and routing) the selected technique. It may happen that a single fault (a junction is now "on") may affect the behaviour of more than one replica (it may cause a short between two inputs common to all replicas) and, therefore, cause a failure in the system. This problem must be handled by the CAD tool, which should distribute the system function among the available resources to minimise the probability for a fault to cause a failure.

A reliability-oriented place and route algorithm was presented in [14] to improve the implementation of TMR and duplication with comparison with concurrent error detection, techniques on FPGAs. Although FPGA manufacturers have been usually more interested in defect tolerance, Xilinx has lately developed a tool to transform plain designs for FPGAs into TMR ones [15]. Similar approaches should be developed to adapt redundancy-based techniques to the particularities of each reconfigurable nanofabric.

Although this approach will effectively tolerate the occurrence of transient faults in the system, in the long run, it will only tolerate the occurrence of a certain number of permanent faults. As a result, the nanocircuit could stop working after a relatively short amount of time. Due to the cost of locating new defects and mapping the system function each time the nanocircuit fails, new approaches must be developed to tolerate the occurrence of those permanent faults on the fly.

4.2. Reconfiguration

Probably, the reconfigurability is the most interesting capability of FPGAs. Different approaches were developed throughout the years to change, on the fly, the implemented system function, either to provide new functionalities or tolerate the occurrence of faults.

Changing the functionality provided by the reconfigurable circuit involves re-implementing the new function system (map, place and route). This procedure requires very time consuming computations, so it is not sensible to run the whole process to perform a small change in the design, which is usually the case for fault tolerance. Techniques known as *incremental synthesis* [16] are aimed at reducing this time by re-implementing only the part of the design that has really changed, keeping as it is the rest of the implementation. This should be taken into account when deploying new fault tolerance mechanism at run-time.

A related problem is that the FPGA has to be completely reprogrammed to change its functionality. That process could introduce a huge timing overhead. Manufacturers have tackled that issue by developing new families that allow for their *partial configuration* [17], i.e. only that part of the FPGA implementing the functionality that changes is reprogrammed.

In this context, different FPGA-based fault tolerance methodologies were developed [18] to minimise the reconfiguration required to avoid the faulty resource upon detection. Tiling [19] seems a very good option for being applied to reconfigurable nanofabrics, since it does not reserve redundant resources for fault tolerance. Instead, different configurations are computed for each resource to map a given function to different elements within the resource. If a fault occurs, it is just a matter of changing the current configuration of the resource by an alternative one that tolerates such fault. Figure 7 shows a possible application of the Tiling approach to the logic array of Figure 6.

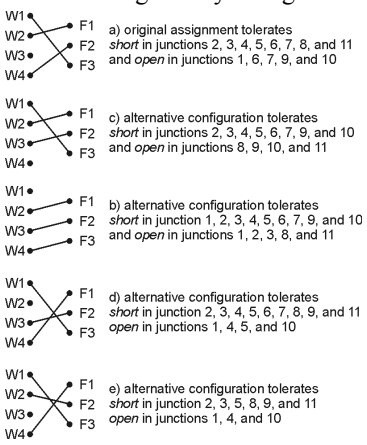

Figure 7. Example of fault tolerance with Tiling.

The occurrence of permanent faults may prevent the system function to be mapped to the faulty resource (a short in junctions 1 and 8 in Figure 7, for instance). In that case, any other reconfiguration technique [18], usually involving placing and routing again the desired function, should be used to reallocate the system function to a suitable resource.

The main drawback of reconfiguration approaches is that an external processor is usually required in order to compute the new partial fault-free configuration. This computation certainly induces some timing overhead in the system execution. In the case of nanofabrics, this time can be quite significant, since a much slower (reliable) CMOS processor should be in charge of generating the new configuration. The Tiling methodology, which minimises the intervention of the external processor is, then, very interesting.

Furthermore, reconfiguration techniques assume that it is possible to detect the occurrence of new faults and precisely locate them. Hence, it will be necessary to perform new defect finding processes upon fault detection and, after locating the fault, apply some reconfiguration technique to tolerate it. Again, an external reliable processor could be an (slow) option to perform this process. On-line approaches, like the one presented in [20], may employ unused resources of the reconfigurable fabric to continuously scan the rest of the circuit for new defects. This could eliminate the dependency on an external processor and decrease the timing overhead associated to that scanning process. It must be noted that the nanofabric should be continuously reconfigured to test all the available resource, those dedicated to implement the system function and those implementing the scanning process.

5. Conclusion

As it has been shown, adapting FPGA-based defect and fault tolerance techniques to reconfigurable nanofabrics will require a profound modification of CAD tools and their supported design flow. Figure 8 depicts our proposed design flow and identifies open issues for further research that are next commented.

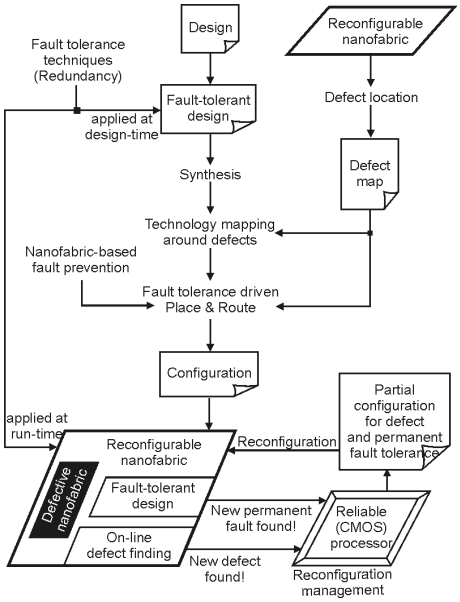

Figure 8. Proposed defect and fault-tolerance aware design flow.

The most notable change is that designs can not be implemented just once anymore. Since the defect map is unique for each single nanofabric, the design must be implemented for every considered circuit. So, reducing the implementation time is of prime importance.

Due to the expected high rates of faults occurrence, any design should include basic techniques for (transient) fault tolerance. Therefore, the possibility of (semi-)automatically inserting these mechanisms into the nanofabric, either at design- or run-time, is a must to assist designers who are not dependability experts.

The extremely high defects rate involves generating a defect map for the selected nanofabric. This defect map will be used after synthesising the design to map the extracted logic to the underlying reconfigurable fabric. In this way, it is possible to use those defective devices that do not affect the behaviour of the system.

The next step consists in performing a fault-tolerance driven place and route process. It will distribute the logic among the available resource of the nanofabric taking into account its architecture. This will help preventing, as much as possible, that future faults may affect the normal operation of the system.

Now, the nanofabric can be programmed with the computed configuration. It is to note that on-line defect finding mechanisms should have been included, either at design time or by the placer and router.

Finally, coupling a reliable processor will probably be needed to tolerate the occurrence of permanent faults via partial reconfiguration. The dependence on this processor may be reduced by implementing the reconfiguration management process on the nanofabric.

6. Acknowledgements

This work was sponsored by the Spanish projects MCYT TEC 2005-05119/MIC, MEC TIN2006-08234.

7. References

[1] ITRS 2007 Edition, *International Technology Roadmap for Semiconductors*, 2007 [Online]. Available: http://www.itrs.net/reports.html

[2] M. Butts, A. DeHon, and S. C. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips", *IEEE/ACM Int. Conference on Computer-Aided Design*, San Jose, CA, USA, pp. 433–440, 2002.

[3] M. Mishra and S. C. Goldstein, "Defect Tolerance at the End of the Roadmap", *International Test Conference*, Charlotte, NC, USA, pp. 1201–1210, 2003.

[4] J. von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", *Automata Studies*, C. E. Shannon and J. McKarthy, Eds., Princeton University Press, 1956, pp. 43–98.

[5] A. DeHon "Nanowire-Based Programmable Architectures" *ACM Journal on Emerging Technologies in Computing Systems*, Vol. 1, No. 2, 2005, pp. 109–162.

[6] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A Reconfigurable Architecture for Hybrid Digital Circuits with Two-terminal Nanodevices", *Nanotechnology*, Institute of Physics Publishing, Vol. 16, 2005, pp. 888–900.

[7] K. Nikolić, A. Sadek, and M. Forshaw, "Fault-tolerant techniques for nanocomputers", *Nanotechnology*, Institute of Physics Publishing, Vol. 13, 2002, pp. 357–362.

[8] A. Doumar and H. Ito, "Detecting, Diagnosing, and Tolerating Faults in SRAM-Based Field Programmable Gate Arrays: A Survey", *IEEE Trans. on Very Large Scale Integration Systems*, Vol. 11, No.3, 2003, pp. 386–405.

[9] S.–J. Wang and T.–M. Tsa, "Test and diagnosis of faulty logic blocks in FPGAs", *IEEE/ACM Int. Conf. on Computer-Aided Design*, San Jose, USA, pp. 722–727, 1999.

[10] J. G. Brown and R. D. Blanton, "CAEN-BIST: Testing the NanoFabric", *International Test Conference*, Charlotte, NC, USA, pp. 462–471, 2004.

[11] Z. Wang and K. Chakrabarty, "Built-In Self-Test and Defect Tolerance of Molecular Electronics-Based Nanofabrics", *Journal of Electronic Testing*, Vol. 23, No. 2–3, 2007, pp. 145–161.

[12] P. Zając, et al., "Resilience through Self-Configuration in the Future Massively Defective Nanochips", *Supplemental Volume of the 37th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks - Workshop on Dependable and Secure Nanocomputing*,, Edinburgh, UK, 2007 [Online] Available: http://www.laas.fr/WDSN07/

[13] D. DeHon and H. Naeimi, "Seven Strategies for Tolerating Highly Defective Fabrication", *IEEE Design and Test of Computers*, Vol. 22, No. 4, 2005, pp. 306–315.

[14] L. Sterpone, et al., "Evaluating Different Solutions to Design Fault Tolerant Systems with SRAM-based FPGAs", *Journal of Electronic Testing*, Vol. 23, 2007, pp. 47–54.

[15] TMRTool, Xilinx Inc., San Jose, USA, 2006 [Online] Available: http://www.xilinx.com/ise/optional_prod/tmrtool.htm

[16] Mentor Graphics Corporation, "Incremental FPGA Synthesis", San Jose, CA, USA, 2007.

[17] Xilinx Inc., "Virtex 2.5V FPGA Complete Data Sheet", San Jose, CA, USA, 2002 [Online] Available: http://www.xilinx.com/support/documentation/data_sheets/ds003.pdf

[18] J. A. Cheatham and J. M. Emmert, "A Survey of Fault Tolerant Methodologies for FPGAs", *ACM Trans. on Design Automation of Electronic Systems*, Vol. 11, No. 2, 2006, pp. 501–533.

[19] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low Overhead Fault-Tolerant FPGA Systems", *IEEE Trans. on VLSI Systems*, Vol. 6, No. 2, 1998, pp. 212–221.

[20] N. R. Shnidman, et al., "On-Line Fault Detection for Bus-Based Field Programmable Gate Arrays", *IEEE Trans. on Very Large Scale Syst.*, Vol. 6, No. 4, 1998, pp. 656–666.