Logic Self Repair Based on Regular Building Blocks

Tobias Koal, Heinrich T. Vierhaus
Brandenburg University of Technology Cottbus
Department of Computer Science
tkoal, htv@informatik.tu-cottbus.de

Abstract

The scalability of CMOS technology is apparently approaching physical limits. In particular, technology forecasts expect higher rates of permanent and transient faults, which make fault tolerant design and, eventually, built-in self repair (BISR) capabilities a necessity. While BISR works reasonably in regular structures such as memory blocks, BISR for random logic is by far an unsolved problem. This paper introduces a novel systematic approach towards logic BISR and gives some indications for cost and limitations.

1 Introduction

During the last 3-4 years several authors have predicted problems with further scalability of standard CMOS semiconductor technology, since structure sizes approach quantum-physical limitations [1-3]. Furthermore, ultra-thin wires and insulation layers will probably exhibit a reduced level of long-term stability. For the design of complex microelectronics-based systems, the design methodology therefore has to deal with a reduced reliability of underlying hardware structures. Essentially, dependable systems have to be designed from unreliable hardware and software components [4,5]. On the hardware side, on-line self-test and error correction [6,7] is an established technology, which has found application in many real-life systems of today [8]. The problem with upcoming systems will possibly be that a combination of transient faults and permanent faults is becoming more likely. The road map of semiconductor industries [9] therefore also sees a strong need for built-in self repair (BISR). While BISR is relatively simple for regular structures such as memory blocks [10,11], it is much more difficult to implement for irregular and random logic. Thereby the granularity of blocks that can be identified as faulty and replaced is an essential feature. Methods that discard and replace large-scale blocks such as total CPUs have been used for years. Relatively little research has been reported about self-repair in somewhat regular logic that is based on the replacement of RT-level functional blocks [12]. In previous work we have shown that a very fine-granular replacement of transistor groups within gates and the replacement of gate-level structures is possible, whereby fault isolation rather than the attachment of backup hardware elements is the most critical issue [13, 14]. In most cases dealt with, the overhead needed for backup hardware, administration of redundancy and organisation of repair seems to exceed the overhead for triple modular redundancy, which is seen as the ultimate solution to reliability problems by many engineers, though it does not include capabilities of BISR.

Also solutions that rely on field-programmable gate arrays (FPGAs) have been proposed [15]. The problem with state-of-the-art FPGAs is that their configurable logic blocks (CLBs) have a complexity of up to several thousand transistors. Furthermore, a non-regular replacement scheme that does not discard and replace full columns and rows of CLBs in a regular manner needs "in-systems EDA tools" for computing the re-wiring of the FPGA [16]. Essentially, a BISR scheme has to satisfy a set of partly conflicting aspects:

1. The repair scheme must be fine- granular enough to match the expected fault density. As blocks used for replacement have a chance to be faulty themselves, their own complexity must much smaller than the transistor count where a fault is likely to occur with a 50 % probability.
2. The repair scheme must be able to provide both fault isolation and replacement. Fault isolation is thereby the critical point for on-type, bridge-type and short-type faults.
3. Repair requires a diagnostic test of reasonable solution as a pre-condition. For in-field BISR, the test must be based on very limited resources.
4. Fault administration is a critical bottleneck. If the status of faults and replacements must be monitored, stored and re-implemented after shut-down or a "power off" condition, the total overhead increases considerably.
5. Above all, the necessary overhead must be affordable in terms of gate count, chip area and power dissipation.

So far, no methodology that comes even close to these objectives seems to exist. At the logic gate level, we have proposed the duplication of gates as the simplest possible solution [13,14]. Besides problems of fault-isolation, the need for a comprehensive scheme of fault management seems to be the main obstacle here. More effective schemes were only indicated but hardly explored in detail. In this paper, we will first introduce a repair scheme that

is based on elementary logic blocks containing redundant elements. Then we introduce a new way of logic synthesis and evaluate overhead and cost. The fourth chapter discusses ways of reducing switching elements. Then we present some first results and describe open problems and forthcoming work.

2 Structured Logic for Self Repair

Random logic is a difficult candidate for the removal and replacement of specific elements such as logic gates, since it does not allow for physical insertion of replacement cells. Beyond duplication, only the structuring of logic into blocks that contain specific types of elements (ANDs, ORs, XORs, Flip-Flops, adder-stages, etc.) could reach the necessary level of regularity. There is a relatively simple geometry based on cluster of, for example, 4 gates including one as a backup (figure 1).

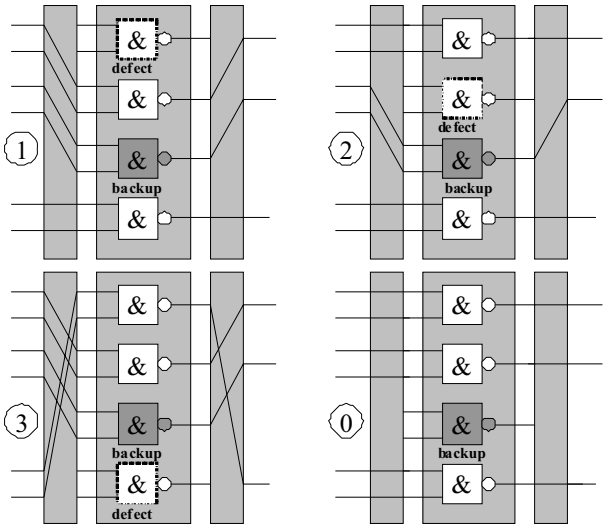

Figure 1 Switching scheme for replacement

With a circular switching scheme implemented with 4 states and 2 control bits, we get along with a single switch at every input which is alternatively into a "straight" or a "shift" position. The scheme is illustrated in figure 1. The backup device may be contacted for testing purposes, which costs 3 additional switching elements. For simplification of the control scheme, several of such 4-element blocks may be driven by the same configuration control bits. At the average, the scheme works as long as not more than one of 4 basic elements in a group is faulty. The switching scheme can provide an effective fault isolation even for building blocks that have gate shorts at input transistors. On the other side, stuck-on faults on switching transistors at inputs do not cause failure conditions. The size of the basic elements that make the blocks is not limited. Depending on the fault density to be handled, the basic building blocks can be logic gates or even RT-level macros. In order to explore the basic overhead associated with such a scheme of re-configurable logic blocks (RLBs), we first consider logic gates as basic elements in the following chapters. Random logic that needs to be managed may contain any type of logic gate available from standard cell libraries. The number and type of RLBs that are needed depends on the logic design. It is essentially not fixed, but can favourably be optimised in a specific process of technology mapping. This process will be critical, since it will strongly influence the quality of the design and the overhead.

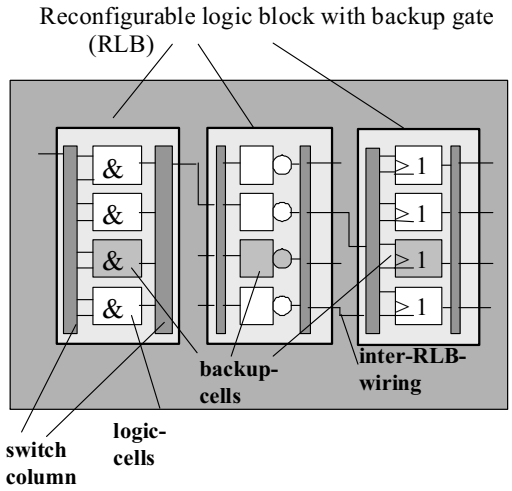

Figure 2 Block structured logic for built-in self repair (BISR)

The total length and area of the wiring between RLBs is a critical parameter. If a large variety of different types of RLBs (e. g. 2-AND, 2-NAND, 2-NOR, 2-OR, 3-AND, 3-NOR, XOR, Inv.) is used, the distance between 2 different types of logic elements that are to be connected according to the net list will become long, while connecting 2 elements of equal type is potentially much shorter. Hence it is desirable to get along with few different types of RLBs as basic elements. However, transformations that can, for example, replace a NAND by an OR with inverted inputs use additional resources. This tendency becomes even stronger, if non-elementary gates such as XORs are used. Except for specific circuitry, e. g. XOR trees, the interconnects between such types of circuits, for which only few RLBs are available in a larger design, may become very long. On the other hand, resolving an XOR into an equivalent circuit consisting of simpler gates almost triples the number of necessary transistor functions, compared with switch-level XORs. Unlike well-known logic designs that can use arbitrary combinations of gates in a certain section of a layout, we need neighbourhoods of equal-type gates for minimum wire length. Furthermore, RLBs have a specific gate count that gets along with a minimum of switching elements. In previous research we identified groups of 4 or 8 elements, including 1 or 2 for backup as an optimum configurations. In total, the technology mapping process has the following steps:

- Optimisation of net lists to get along with a minimum of types of RLBs,
- Mapping for a minimum of circuit elements,
- Mapping to fit fixed cluster sizes in neighbourhoods.

How such mappings can work and how many gates / switches are needed can be shown in some simple examples.

3 Mapping and Optimisation

As a first example, we consider a logic network that consists of primitive gates only (figure 3). In a first attempt of optimisation, a transformation that results in a set of 2-input NANDs, NORs and inverters is applied (figure 3).

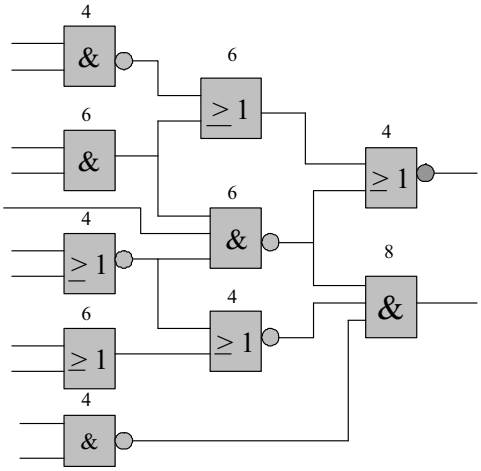

Figure 3 Irregular logic network consisting of basic gates (52 transistors)

The first transformation applied is the mapping of the 3-input NAND into an equivalent network of 2-input gates. This transformation results in a transistor count of 60. The next step of transformation tries to use inverting gates wherever possibly and will save on inverting functions. In the final step of optimisation, we try to save inverters by using NAND / OR and NOR / AND transformations. However, this step needs to consider also other aspects. It is a favourable solution to have neighbourhoods of equal-type gates for mapping.

For example, a 2-input NAND that is connected to NORs on all inputs and outputs is favourably mapped into an OR / NOR to fit into a cluster of identical gates. The last aspect is that logic clusters always have a fixed size. For example, blocks of equal gates that contain either 4 gates with one used for replacement or 8 gates with 2 available for repair are favourable with respect to minimized switching networks and simple re-organisation.

Under these aspects, the final mapping is performed into a set of 6 NANDs, 6 NORs, and a set of 5 inverters (figure 4) The transistor count is 58. The result indicates that the transformations needed to get along with few basic types of gates may cost about 10% in transistor count. The network is then physically mapped into 3 blocks of cells (figure 5). The analysis with respect to the real overhead implied, however, indicates that this is not the best available solution.

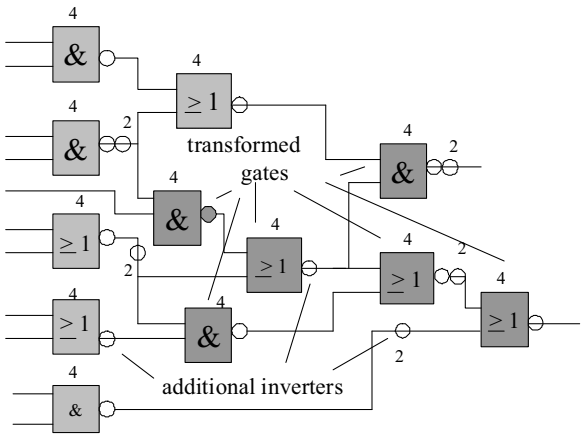

Figure 4 Logic network after final trans-formations (58 transistors)

The arrangement, as shown in figure 5, has two witches at every gate input and at every gate output likewise for re-configuration. Therefore the NAND-block and NOR-block each have 32 transistors within gates (including backups) plus 48 2-way switches. The INV block has 16 transistors and 32 switches at input and outputs. Apparently, the overhead needed for switching dominates the additional overhead in the logic itself by far.

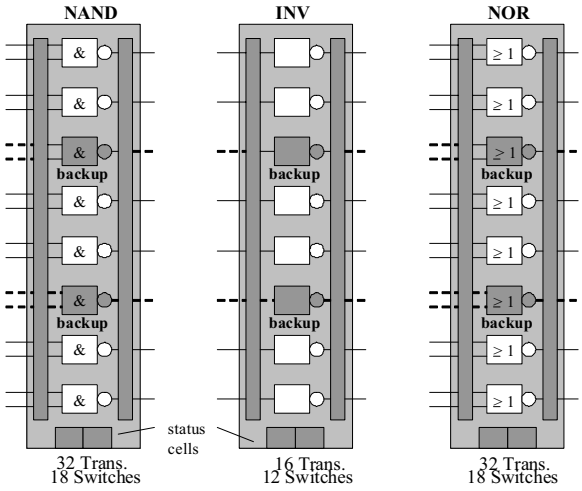

Figure 5 Logic mapping of gates into blocks of 6 gates with additional 2 for backup

The conclusion is two-fold: First, initial logic transformations that facilitate mapping are not very “expensive”, neither are backup elements. Second, the number of switching elements is the essential bottleneck. If we assume only 2 transistors for every 2-way switch, the number of switching transistors is about as high as the count in functional transistors. Under worst case conditions with minimum-sized re-configurable elements,

a RLB with 4 2-NANDs or 4 2-NORs (16 transistors including 4 for backup) needs 9 switches, consisting of 2 transistors each. There is, however, a specific difference that may save the feasibility of the approach. Switching transistors are not dynamically involved into signal flow operations. They can therefore be manufactured with thicker gate oxides and longer channels, which makes them less vulnerable to malfunctions by shifting technology parameters. Under any circumstances, a strategy that gets along with a minimum of switching elements is needed.

4 Reduction of Switching Elements

The structuring according to figures 5 and 6 is strictly based on the replacement of gate-level structures at the minimum level of granularity. If larger structures than basic gates or inverters are taken as basic re-configurable units, the overhead in switching elements is reduced at the expense of manageable fault densities. In a first step we consider the alternative of using gates plus optional wiring and switching efforts. The NAND / AND block shown in figure 6 is the essential compromise.

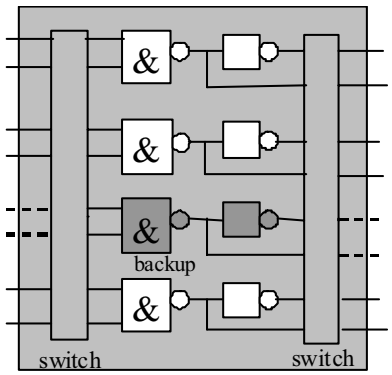

Figure 6 Building block with double-output gates

At the expense of 8 extra output switches, each gate is available as an inverting or non-inverting entity. This modification requires more transistors within building blocks, but reduces the wiring overhead by saving on types building blocks by avoiding many separate inverters. What is essentially saved is the column of input switches from the “Inverter” (INV) column in figure 5.

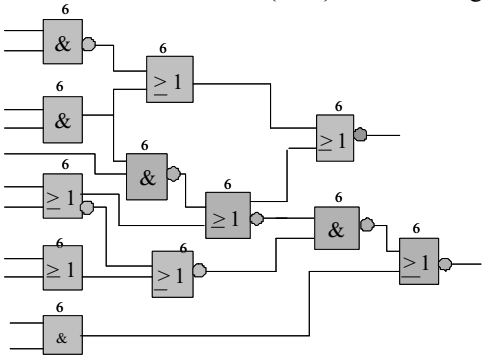

Figure 7 Example network with double-output gates

To our surprise, such an arrangement even shows favourable effects, if double-gate outputs can actually be used in the logic design (figure 7), since every logic output can easily be double-used with inverted and non-inverted output signals. This option gives some additional degrees of freedom in logic transformations where fan-out nodes and inverters are involved. First results indicate that, despite the higher transistor count for basic units, this scheme shows an advantage where more than 50% of gate outputs have to be non-inverted

5 Results

So far, we investigated on a few small logic networks. All examples include a mix of inverting and non-inverting gates, and fan-out nodes. In the first case, the network consist of 12 primitive 2-input logic gates only. In the second case, the network includes 2 out of 10 gates with 3 inputs, one gate is an XOR. This example serves to estimate the essential extra cost of mapping more complex gates (multi-input gates, XORs) into elementary gates. The first and general insight is that the number of switches and the cost of redundancy administration by far dominates the cost for the pure provision of redundancy. For example, the allocation of one spare gate in a unit of four makes sense, since a single column of switches is enough for (re-) configuration. In case of, for example, one out of 8 gates taken for backup, the extra cost for switching units by far outgrows the savings in backup devices. Therefore the “one out of four” scheme or an equivalent 2 out of 8 gates is a favourable compromise, whereby the administration for switching can be allocated to serve several blocks in common.

Table 1 Transistor overhead for repair functions / number of switches needed

Circuit	**Transistors** in CMOS implementation		**Trans. / Switches** in reconfig. blocks	
	Orig. netw.	**Log. transf.**	**Inv. Separ.**	**2-outp. gates**
2-inp. gates only	48	46	64 / 39	72 / 36
3-inp. gates, XOR	48	60	80 / 48	72 / 48

Logic transformations to resolve multiple input gates and to optimise the logic for a high share of inverting gates are not very costly. However, resolving an XOR into either 4 NANDs or 3 NANDs and 2 inverters is relatively expensive. In general, using larger entities for replaceable units, e. g. XOR-macros instead of elementary gates, reduces the cost of switching considerably. We can consider the conditions arising for the second example as close-to worst case, since a single XOR (with 6 transistors in a pass-transistor implementation) was resolved into a 16-transistor gate-level version, whose internal gates can be re-allocated for self repair. Given the need of a minimum of 2 pass-transistors for a switch, the number of transistors in switches is about as high as the number of functional transistors. Essentially, the switching alone makes about a 100 % overhead in transistor count, not

counting an additional overhead for switching administration, which is variable with the size of independently re-configurable logic blocks combined.
Though the overhead looks high at the first glance, we can consider conditions shown here as close to the upper limit or the worst-case condition in truly irregular logic. Any regularity within logic structures that can take, for example, XORs, selectors, decoders or adder stages as basic entities, will most likely result in an overhead which is below 100 %. A comparison concerning core transistors versus switching elements is shown in table 2.

Table 2 Transistor overhead depending on basic unit size

	Transistors			
Block	Functional	backup	norm switch	ext. switch
3 /4- 2-NAND	12	4	18	24
3 / 4 2-AND	18	6	18	24
3/4 2-XOR	18	6	18	24
H- Adder	36	12	24	30
F- Adder	90	30	30	36

A detailed look at the problems of testing such logic blocks including fault diagnosis exhibits the need for an additional test access via the functionally unused inputs / outputs of the backup elements. With few additional switches it even possible to connect each of the functional blocks with this input in one of the modes installed for replacement (figure 8).

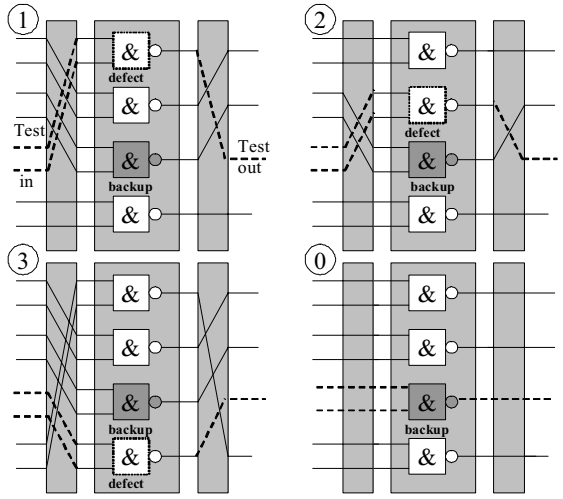

Figure 8 Extended switching for test access

The overhead associated with this extension is shown in the last column of table 2. In general, these overhead figures still compare well with triple modular redundancy (about 300 %) or FPGA- based solutions (up to a factor of 10). A regularized logic repair scheme is also much more efficient than gate duplication schemes or even schemes where groups of transistors from within gates can be replaced [13, 14]. At the level of basic gates, fault densities up to one transistor in 1000 should be manageable. This level, however, is reduced if the size of basic RLB grows. While the vulnerability of switching units is a non-trivial problem, it is not as severe as it may seem at the first glance. A single transistor stuck-on or stuck-off fault in a switching unit will not make the logic block non-functional, but will just reduce its capability to cope with further faults in the same basic unit. On the other hand, gate oxide shorts in pass transistors remain a critical issue.

6 Unsolved Problems and Further Work

A diagnostic test that can identify the faulty logic block with reasonable overhead is still a problem. Most likely, methods that perform a crude diagnosis first and then try and validate repair functions will be needed. Test functions can be simplified, if the outputs of identical logic blocks can be compared for fault detection. If a system that contains repaired faults is switched off and re-started, the actual status of repair must be re-installed. Therefore, either a “fault memory” or a fast and comprehensive test during the start-up phase is necessary, which is the base for an initial configuration.
In the schemes shown so far, we can allocate backup elements only for faulty gates, but there is no remedy for faulty interconnects between logic blocks. As interconnects are known to have reliability problem not much less that active elements [7], this is a serious problem. In earlier work, double-feeding of any logic “consumer” element by double connection with the source was proposed [17]. Unfortunately, such additional wiring is a remedy for “open-“-type defects only, but not for shorts or bridges. Presently, we are working on a scheme that can employ a few extra lines between the basic block, normally used for test access, alternatively also for the replacement of faulty wires.

7 Acknowledgments

The work presented here was sponsored by the German Research Board (DFG) within the SELNA project.

8 References

[1]S. Sirisantana, B. C. Paul, K. Roy: “Enhancing Yield at the End of the Technology Roadmap”, IEEE Design and Test of Computers, Vol. 21, No. 6, Nov.-Dec. 2004, pp. 563-571

[2] M. Mishra, S. C. Goldstein, “ Defect Tolerance at the End of the Roadmap”, Proc. IEEE Int. Test Conf 2003, pp. 1201-1210

[3] M. A. Breuer, S. K. Gupta, T. M. Mak: “Defect and Error Tolerance in the Presence of Massive Numbers of Defects”, IEEE Design and Test of Computers, Vol.21, No. 3, May / June 2004, pp. 216-227

[4] H. Kopetz, “System Failure is the Norm, not the Exception“, Keynote, DATE 2008, Munich, March 13th, 2008

[5] S. Borkar, „Designing Reliable Systems from Unreliable Components: The Challenge of Transistor Variability and Degradation“, IEEE Micro, Vol. 25, No. 6, Nov. / Dec. 2005, pp. 10-16

[6] P. G. Lala, „Self- Checking and Fault Tolerant Digital Design“, Morgan Kaufmann Publishers, San Francisco, 2001

[7] A. De Hon, H. Naemi, “Seven Strategies for Tolerating Highly Defective Fabrication“ , IEEE Design and Test of Computers, Vol. 22, No. 4, July-August 2005, pp. 306-315

[8] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, H. T. Vierhaus, „Evaluating Coverage of Error Detection Logic for Soft Errors using Formal methods”, Proc. IEEE DATE 06, Munich, March 2006

[9] ITRS Roadmap, 2006 Update, Semiconductor IndustriesAssociation, http://www.itrs.net/links/2006update/FinalToPoist/03_Test2006Update.pdf

[10] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. P. Higgins, J. L. Lewandowski, “Built in Self Repair for Embedded High Density SRAM”, Proc. IEEE Int. Test Conf. 1998, pp. 1112-1118

[11] J. F. Li, C. C. Yeh, R. F. Huang, C. W. Wu, “A Built-In Self Repair Scheme for Semiconductor Memories with 2-D Redundancy”, Proc. IEEE Int. Test Conf. 2003, pp. 393-398

[12] A. Benso, S. D. Carlo, G. Di Natale, P. Prinetto, “Online Self-Repair of FIR Filters“, IEEE Design & Test of Computers, Vol. 20, no.3, May-June 2003, pp. 50-57

[13] R. Kothe, H. T. Vierhaus, B. Straube et al., „Embedded Self Repair by Transistor and Gate Level Reconfiguration“, Proc. DDECS06, pp. 210-215, IEEE CS Press 2006

[14] H. T. Vierhaus, „Transistor- and Gate Level Self Repair for Logic Circuits”, Proc. IEEE Signal Processing 2005, Poznan, Ed. A. Dabrowski, ISBN 83-913251-6-4

[15] S. Mitra, W.-J. Huang, N. R. Saxena, S.-Y. Yu, E.J. McCluskey, “Reconfigurable Architecture for Autonomous Self Repair”, IEEE Design and Test of Computers, Vol. 21, No. 3, May-June 2004, pp. 228-240

[16] S. Habermann, R. Kothe, H. T. Vierhaus, “Built-in Self Repair by Reconfiguration of FPGAs”, Proc. IEEE Int. On-line Testing Symposium 2006

[17] P. Panitz, A, Quiring, H. Müller, E. Barke, J. Koehl, “Erhöhung der Ausbeute durch robuste Verdrahtungsnetzwerke”, Proc. 1. GMM/GI/ITG Fachtagung „Zuverlässigkeit und Entwurf“, März 2007, München, GMM-Fachbericht 52, VDE-Verlag Berlin, 2007, pp.117-124