

The NanoBox: A Self-Correcting Logic Block for Emerging Process Technologies with High Defect Rates

AJ KleinOsowski
ajko@mail.ece.umn.edu

Priyadarshini Ranganath
priya@ece.umn.edu

Mahesh Subramony
mahesh@ece.umn.edu

Vijay Rangarajan
rvijay@ece.umn.edu

Kevin KleinOsowski
kevinko@mail.ece.umn.edu

David J. Lilja
lilja@ece.umn.edu

Department of Electrical and Computer Engineering
Minnesota Supercomputing Institute
University of Minnesota, Minneapolis, MN 55455

Abstract

Semiconductor fabrication trends indicate that emerging process technologies will experience an increase in the number of noise induced errors and device defects. While past work in fault tolerance has focused on static testing, modular redundancy in combinational logic, and error correcting codes in memories, the prevalence of transient errors may make these techniques less effective. In this project, we introduce the NanoBox, a field-programmable gate array (FPGA) style lookup table with fault tolerance coding applied to the static random access memory (SRAM) array within the lookup table. In this way, we contain and self-correct errors within the lookup table, thereby presenting a robust logic block to higher levels of logic design. Our results show that triplicating the SRAM array within the lookup table gives excellent error coverage, has a 1.9x area overhead, 2.2x power overhead, and a 1.3x delay overhead, as compared to a design with no fault tolerance coding. We also investigate NanoBox implementations using information coding, but we find these implementations have significantly higher overhead, and worse error coverage, than the triplicated SRAM bit implementation.

1 Introduction

Recent work in physics, chemistry, and materials science has produced nanometer-scale structures out of exotic materials using sophisticated fabrication techniques [4, 11]. These new devices have the potential to be the “killer device” for the next generation of computers. However, there is widespread agreement among device researchers that nanodevices will have much higher manufacturing defect rates, they will have very low current drive capabilities, and they will be much more sensitive to noise-induced errors [1, 2, 7].

The key advantage to nanotechnology devices is their small size and the resulting unprecedented level of integration expected in designs constructed with these devices.

What is more, as contemporary CMOS devices scale down and multi-gigahertz designs emerge, circuit topologies must change to account for shorter wires and higher densities of noise-induced errors [9]. Manufacturing flawless chips will become prohibitively expensive, if not impossible. Instead of assuming that defects and transient errors are uncommon, future circuits must adapt to, and coexist with, the substantial numbers of manufacturing defects and high transient error rates.

In this work, we introduce the NanoBox, a self-correcting logic block for addressing the increasing error densities in emerging process technologies. The NanoBox consists of a field-programmable gate array (FPGA) style lookup table with appropriate error detection and correction. Due to the scaling of emerging process technologies, and the resulting increase in transistor budgets, the area overhead of adding bit-level fault-tolerance to circuits will now be feasible.

With self-correcting logic blocks, the error coverage of circuit designs can be increased, invisibly to the logic designer. Existing hardware description language code can be synthesized to these self-correcting logic blocks using existing place and route techniques. By correcting errors at the fine-grained bit-level, designs may not need complicated module or system level redundancy. Reducing, and perhaps eliminating, module and system level redundancy will decrease the complexity of designs, making them easier to verify. Ultimately, shifting to fine-grained, self-correcting logic blocks aims to leverage existing intellectual property in logic design to transition these existing designs to emerging fabrication technologies with minimal effort.

2 Background and Motivation

Prior work in fault-tolerance has focused on making relatively large logic blocks or entire systems fault tolerant using such techniques as module-level redundancy in combinational logic [10] and error correction codes in memory [8]. However, for emerging technologies, simple replication of microarchitecture blocks may no longer be sufficient, as all replicated blocks will have errors. Additionally, the frequency of memory faults may require error correcting codes that go substantially beyond conventional single correct, double detect codes.

External fault-tolerance and device reconfiguration has been proposed as a possible way to cope with unstable molecular devices. In the Teramac [6] project, a digital system was built out of unreliable field-programmable gate arrays (FPGAs). An external testing computer was connected to the Teramac to periodically survey the blocks of FPGAs and identify which blocks had incurred errors since the last fault survey. Faulty blocks were disabled and the connections to and from those faulty blocks were rerouted to neighboring blocks. Although Teramac was built out of contemporary FPGAs, not emerging technologies, the unreliable FPGAs had characteristics similar to the trends in emerging technologies.

The Phoenix [5] project proposed using external circuitry (fabricated from reliable devices) to periodically survey the computer system (fabricated with unreliable emerging devices) for faulty logic blocks. As in the Teramac project, once a faulty block is identified, the connections to and from that block are removed or rerouted to functioning blocks.

Periodic system testing becomes a critical bottleneck as computer systems scale in size. If blocks are laid out in a two-dimensional grid, every additional block adds several more connection points to the system with one or more connections into the block and one or more connections out of the block. This super-linear increase in connection points with additional blocks leads to an exponential increase in system checking time as more blocks are added to the system.

Our proposed NanoBox approach addresses the system check bottleneck by distributing the checking circuitry into the logic blocks themselves. In this way, system checks can be performed simultaneously with the actual computation inside a logic function. Errors are identified and corrected on the fly as part of the logic’s normal computation, rather than during a periodic system-wide validation survey.

3 NanoBox Circuit Topology

In this work, we introduce the NanoBox, a self-correcting logic block consisting of a field-programmable gate array (FPGA) style lookup table [3] with appropriate error detection and correction circuitry. The following sections walk through the overall NanoBox concept, a NanoBox imple-

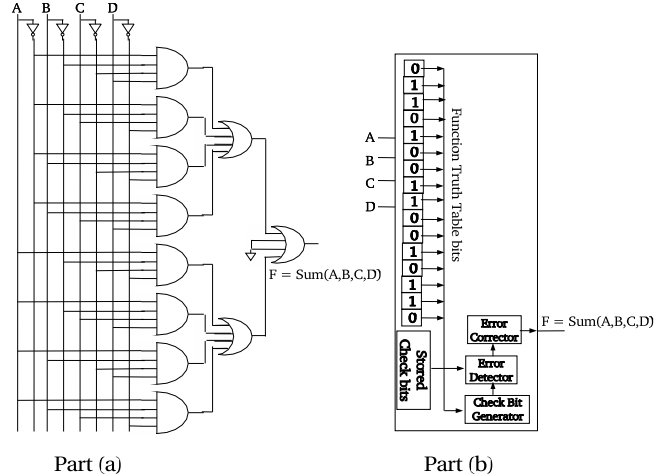


Figure 1: (a) An example combinational logic circuit constructed with conventional devices. (b) The same logic circuit constructed with an error-correcting lookup table. Each bit of the logic function truth table, along with the truth table check bits, is stored in a memory cell.

mentation using triple modular redundancy, and a NanoBox implementation using information coding.

3.1 The NanoBox Concept

To demonstrate how the NanoBox approach could be used in combinational logic design, we present an example using a 4-bit sum operation. Figure 1(a) shows a sum function of four variables as it would be constructed using conventional logic elements. Figure 1(b) shows the same function constructed with an encoded lookup table. The function inputs are fed to a decoder which addresses an SRAM cell array. The value of the addressed SRAM cell is fed to a sense amp which is used as the function output.

In each lookup table, extra SRAM cells are added for the check bits that encode an error correction code of the function truth table bits.

We envision that these NanoBox circuit elements would be created from hardware description language code by the synthesis step of computer aided design (CAD) tools. The synthesis step would determine the contents, 1 or 0, of each truth table SRAM cell, as well as the contents of each check bit cell.

During normal circuit operation, the contents of the SRAM cells do not change. Under transient fault conditions, or in the presence of manufacturing defects, the contents of the SRAM cells may be incorrect. Additionally, there may be noise on the wires within the NanoBox which result in a SRAM cell which appears to have an incorrect value.

Whenever the inputs to the lookup table change, the truth table bits are fed into the check bit generator, which recalcu-

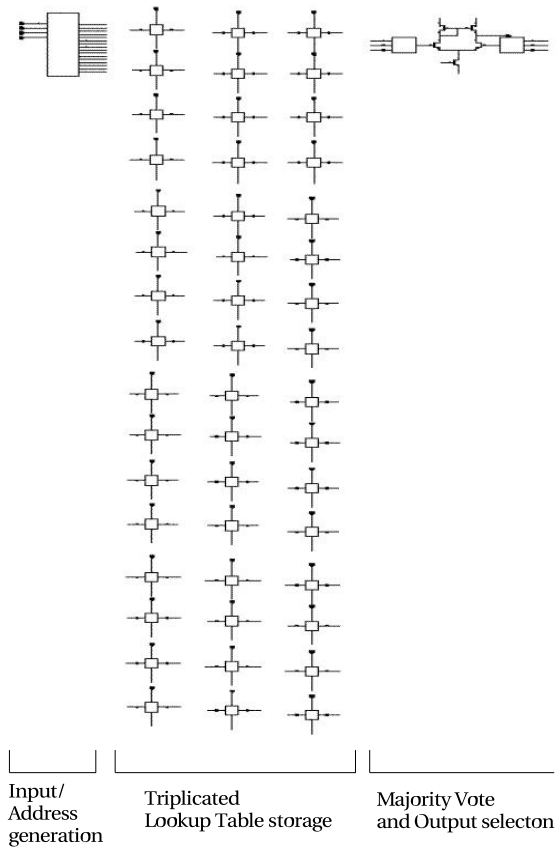


Figure 2: Schematic of a Triple Modular Redundancy NanoBox lookup table, with triplicated SRAM cells and a majority vote error detector/corrector.

lates the check bits. These newly calculated check bits are then compared with the stored check bits in the error detector. The results of the error detector are fed into the error corrector, which makes changes to any flipped bits at the function output. The corrected function output is then used as the actual output of the lookup table.

A point to note is that we do not change the stored value of the SRAM cells. We assume that the errors are either transient and will return to their correct value, or the errors are static and cannot be corrected. Our NanoBox concept aims to identify and correct errors at the function output, not to correct the stored values within the function truth table.

3.2 Triple Modular Redundancy NanoBox

The triple modular redundancy NanoBox implementation, shown in Figure 2, has three copies of the SRAM cell array. Each of the three copies stores an identical copy of the function truth table. The error detection and correction circuitry consists of a simple three input majority gate. Whenever the NanoBox inputs change, the three copies of the SRAM cell addressed by the NanoBox inputs are fed to the ma-

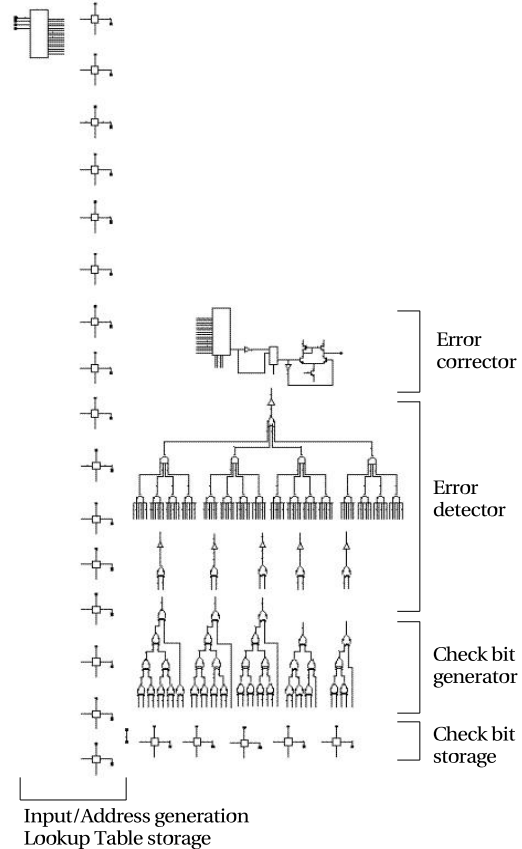


Figure 3: Schematic of an Information Coding NanoBox lookup table, using a single error correct Hamming code. For this initial investigation, the check bit generator, error detector, and error corrector are implemented with full custom CMOS, rather than with additional, smaller, NanoBox circuit elements.

majority gate. The output of the majority gate is used as the NanoBox output. As long as two out of the three copies of the SRAM cell being addressed have the correct value, the NanoBox will produce the correct function output. All of the SRAM cells not addressed by the NanoBox inputs may be in error without affecting the NanoBox output since the non-addressed SRAM cells are not used by the majority gate.

3.3 Information Coding NanoBox

In the information coding NanoBox implementation, shown in Figure 3, a small number of SRAM cells are added to store check bits which encode a function of the truth table bits. The number of check bits varies based on the size of the truth table and the coding being used. In our evaluation, we used 16-bit truth tables with a single error correct Hamming code [8]. This coding was chosen due to its common use in contemporary memory systems. NanoBoxes constructed with emerging technology devices may need a

more robust code than a single error correct code. Some possible alternative information codes include Hsiao codes and Reed-Solomon codes [8].

Our truth table size and coding require five check bits. Whenever the NanoBox inputs change, all of the function truth table bits are fed to the check bit generator circuitry, which recalculates the check bits based on the current, and perhaps faulty, stored truth table bits. These recalculated bits are then fed to the error detector circuitry which compares the recalculated bits to the stored, and perhaps faulty, check bits. The resulting syndrome identifies which stored bit, if any, is in error. If the error is on the SRAM cell being addressed by the NanoBox inputs, the SRAM cell value is inverted before being used as the NanoBox output. If the error is on an SRAM cell not addressed by the NanoBox inputs, the error is ignored.

The information coding NanoBox has relatively small overhead in terms of extra SRAM cells. However, the error detector and error corrector are moderately complex and require significant area. Depending on the characteristics of the devices used to implement the NanoBox, the combinational logic needed for the check bit generator, error detector, and error corrector may also be implemented with NanoBoxes. For example, a very large NanoBox could use information coding on the function truth table, and then the error correction circuitry within the information coding NanoBox could be implemented using smaller, triple modular redundancy NanoBoxes.

4 Evaluation Methodology

We began our evaluation of the triple modular redundancy and information code NanoBoxes by constructing Spice models of the encoded lookup tables. Since nanotechnology devices are not yet mature enough to have specific area, power, and timing models, we constructed our lookup tables from contemporary CMOS devices. The VLSI schematics of our encoded lookup tables will undoubtedly change based on the specifics of a particular emerging technology device. By using CMOS devices, though, we are able to explore proof-of-concept simulations and begin evaluating the relative area, power, and timing trade-offs of the different encoding techniques.

Our simulations used the Cadence software tools suite to develop our schematics and extract HSpice netlists. We then used the Mosis TSMC run T2AF 0.35um process device parameters for our HSpice simulations. NMOS transistors were sized at 2λ and PMOS transistors were sized at 4λ .

In tandem with constructing device-level schematics for the encoded lookup tables, we began exploring how to create computing systems using encoded lookup tables. As a starting point, we chose to implement a simple four instruction arithmetic logic unit (ALU). Given a three bit opcode, this ALU calculates the AND, OR, XOR, and ADD of two bits. This streamlined instruction set was chosen so we could per-

form image processing manipulations such as hue shifts and reverse video with a (future work) full computing system constructed out of NanoBox circuit elements. Each bit slice of the ALU uses four NanoBoxes, as shown in Figure 4. We compare our NanoBox ALU to a full custom CMOS ALU, as shown in Figure 5.

In our simulations, we perform sixteen computations with our four ALUs (full custom CMOS, NanoBox with no coding, NanoBox with triplicated SRAM cells, and NanoBox with information coding). We calculate each of the four ALU instructions, AND, OR, XOR, and ADD, with each of the four 2-bit input combinations. For each instruction, we cycle through the input combinations 00, 01, 10, 11, applying each input combination for 10 nanoseconds. In this way, we exercise all of the computations possible with our ALU.

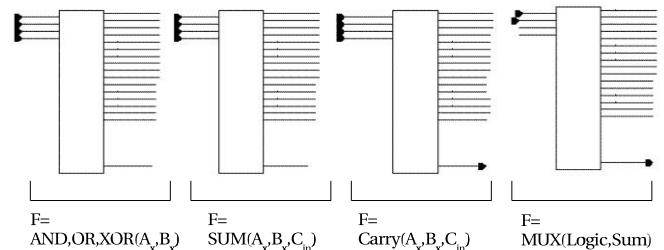


Figure 4: Bit slice of a four instruction ALU, constructed with encoded lookup table NanoBoxes.

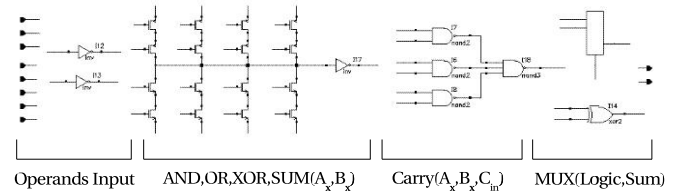


Figure 5: Bit slice of the four instruction ALU from Figure 4, constructed with full custom CMOS devices.

5 Results

We use three different evaluation methods in this study. Section 5.1 discusses our simulation results. Section 5.2 qualitatively discusses our error coverage, and Section 5.3 uses analytical formulas to evaluate how our NanoBox approach scales as the size of the lookup table increases.

5.1 HSpice Simulation Results

Full results of the area, delay, and power of our custom CMOS ALU, No Code NanoBox ALU, triple modular re-

dundancy NanoBox ALU, and information code NanoBox ALU are shown in Table 1.

As we expected, our NanoBox ALUs have significantly higher area and significantly longer delay than a full custom CMOS ALU. These results were expected since NanoBox input decode, as well as error detection and correction logic within each NanoBox, add several levels of logic to the ALU.

Surprisingly, our no code and triple modular redundancy ALU have lower power consumption than the CMOS ALU. We attribute the low power of these NanoBox ALUs to the minimal switching that occurs within these ALUs. Each time the inputs to a NanoBox change, the devices within the NanoBox decoder are the only devices that switch. In a full custom CMOS ALU, in contrast, switching occurs extensively in devices throughout each stage of the CMOS ALU logic.

In this initial study, our information coding NanoBox uses full custom CMOS logic for the check bit generator, error detector, and error corrector. We attribute the significant power consumption of this NanoBox implementation to the device switching in the error correction logic. If this logic within the information coding NanoBox were implemented with no code NanoBoxes, or with triple modular redundancy NanoBoxes, we expect our power consumption would be much less than the results in this initial evaluation (although our area would likely increase over the area results in this initial evaluation).

When comparing our triple modular redundancy and information coding ALU to a NanoBox ALU with no coding, we see that the triple modular redundancy ALU has a modest area (1.9x), power (2.2x) and delay (1.3x) overhead. The information coding ALU, in contrast, had modest delay (1.6x) overhead but significant area (3.7x) and very significant power (88x) overhead.

5.2 Qualitative Error Coverage Results

In this initial study, we do not quantitatively evaluate the error coverage of our triple modular redundancy and information coded NanoBox. Qualitatively, however, the triple modular redundancy NanoBox is expected to have very good coverage for random errors and, depending on the circuit layout, may have good coverage for burst errors. As long as two out of the three SRAM cells addressed by the NanoBox input are error-free, the NanoBox will produce the correct function output. The correct output will be produced even if all of the SRAM cells not addressed by the current NanoBox inputs are in error. To achieve good coverage for burst errors, the three copies of the SRAM cells should be placed far apart in the circuit layout. In this way, a burst error will affect many bits of one copy of the table, rather than all of the copies of a single bit of the table. This dispersion of SRAM cells may make the circuit more difficult to place-and-route, however.

Table 1: Area, Delay, and Power required for custom CMOS and NanoBox ALUs. One area unit is defined as the amount of space required for one NMOS transistor. A PMOS transistor is defined to occupy two area units. Reported delay is the average rise and fall times of the ALU output over a series of sixteen computations. Reported power is average power consumed over the same series of sixteen computations. The sixteen computations are the AND, OR, XOR, and ADD of each of four 2-bit input combinations. Power is reported in nanowatts. CMOS = full custom CMOS ALU. No Code = NanoBox ALU with no coding. TMR = NanoBox ALU with triplicated SRAM cells. Info = NanoBox ALU with information coding. CMOS, TMR, and Info ALUs are compared to the No Code ALU.

Area, Delay and Power of ALUs				
	CMOS	No Code	TMR	Info
Area	86 (0.05x)	1680 (1x)	3176 (1.9x)	6212 (3.7x)
Delay	0.34 (0.14x)	2.39 (1x)	3.12 (1.3x)	3.94 (1.6x)
Power	75.99 (11.1x)	6.87 (1x)	15.13 (2.2x)	604.8 (88.0x)

Our Hamming code implementation of the information coding NanoBox is expected to have good error coverage for random errors, but poor coverage for burst errors. In particular, a burst error would typically cause errors in many of the bits of the lookup table. With a conventional single error correct code, though, the error syndrome bits most likely will flag the wrong bit as being in error.

In both the triple modular redundancy and Hamming code NanoBox implementations, we do not address errors in the input decoder or in the error detector/corrector. The coding schemes instead focus on errors in the SRAM lookup table cells. A quantitative evaluation of whether fully robust circuitry is needed for NanoBox logic other than the SRAM cells is outside the scope of this present evaluation, but is an important item for our future work.

5.3 Analytical Scaling Results

We examine how the error correction overhead of our triple modular redundancy and information coding approaches scale as we increase the size of the lookup table by using analytical equations. For the triple modular redundancy implementation, the total area is approximated as $TMR = D + 3Sn + M$. For the information coding implementation, area is approximately $Info = D + Sn + Slog_2(n) + G + T + C$. In these equations, D is the logic for the decoder, S is the logic for an SRAM cell, M is the majority gate check and correct logic, G is the check bit generator logic, T is the error detector logic, and C is the error corrector logic.

The values of the constants D, S, M, G, T, and C are a strong function of the implementation design and of the de-

vice technology. For our CMOS evaluation, we found that the triple modular redundancy implementation was dominated by the decoder (D) logic as n scaled up. The information coding approach was dominated by the check bit generator logic (G). As n becomes large, the overhead of triplicating the SRAM bits decreases. For a 4-bit bit lookup table, the area overhead of triplicating the SRAM bits is 2.49x. This overhead decreases to 1.47x as n scales to 1024 bits. Since the information coding approach is dominated by the error detection and correction logic, the area overhead increases as we scale n . For a 4-bit Hamming code lookup table, the area overhead is 3.48x. This overhead increases to 3.79x as we scale n to 1024 bits.

6 Future Work

The work described in this paper is an initial investigation into developing self-correcting combinational logic circuits using encoded FPGA style lookup tables. Our foremost future work is to evaluate how the NanoBox concept scales when implementing more complex logic blocks using larger lookup tables. We also are investigating ways to implement the error detection and correction logic using additional lookup tables, rather than with conventional CMOS circuitry, and ways to simplify the correction and detection logic, such as subsetting the lookup table or applying different correction codes.

Another important area that needs to be studied is the level of fault coverage provided by these techniques. To quantitatively test the error coverage of our NanoBox concept, we are developing software tools to inject time-varying faults into HSpice simulations.

To extend the NanoBox approach, we are developing the Recursive NanoBox Processor Grid using the NanoBox circuit elements. This highly parallel, application-specific processor architecture is being used to evaluate whether the self-correcting logic blocks provide adequate error coverage over an entire microarchitecture, or whether modular and system level fault tolerance techniques still need to be used in conjunction with the fine-grained fault tolerance of the NanoBoxes.

7 Conclusion

The yield, reliability, and error characteristics of new device technologies are expected to be substantially different from the corresponding characteristics of conventional CMOS devices. Existing fault tolerance techniques used in current microprocessors assume relatively low manufacturing defect densities and infrequent dynamic faults. Projecting forward, it is expected that future circuit topologies will have substantially higher defect densities and dynamic faults. These differences between current and future devices, and current and future circuit topologies, will require fundamentally new de-

sign techniques in which microprocessors are designed from the start to coexist with many defects and high densities of transient errors.

We introduce The NanoBox, a field programmable gate array style lookup table which uses error correction on the function truth table, thereby achieving fine-grained fault tolerance. These encoded lookup tables are able to dynamically correct faults within the lookup table, thereby presenting a robust logic block to higher levels of circuit integration and microarchitecture. Standard hardware description language code can be mapped to these NanoBox circuit elements using existing synthesis techniques. Logic designers are therefore able to increase the error coverage of their designs without adding complex modular or system level fault tolerance techniques.

Our results show that triplicating the array of static random access memory cells within the lookup table has modest area, power, and delay overhead. The triplicated memory array also has excellent error coverage by looking at only the memory cells addressed by the NanoBox inputs. All of the memory cells not addressed by the NanoBox inputs may be in error. As long as two out of the three memory cells addressed by the NanoBox inputs are correct, the NanoBox will still produce the correct function output.

Our results also show that adding information coding to the NanoBox incurs significant area, delay and power overhead for the simple logic function evaluated in this initial study. However, information coding may still be an option for NanoBox implementations if new, very low overhead, coding techniques can be developed.

The NanoBox approach to fine-grained fault tolerance is targeted at the types of device defects and transient faults that are expected to arise in emerging device process technologies. This approach also is applicable to contemporary field programmable gate array systems, and to deep sub-micron technologies.

References

- [1] Semiconductor Research Corporation. International technology roadmap for semiconductors. Document available at <http://public.itrs.net>, 2001.
- [2] Semiconductor Research Corporation. SRC research needs document for 2002-2007. Document available at <http://www.src.org>, June 2002.
- [3] Xilinx Corporation. Virtex-II pro platform FPGAs: Functional description. Document available at <http://www.xilinx.com>, September 2002.
- [4] Linda Geppert. The amazing vanishing transistor act. *IEEE Spectrum*, pages 28–33, October 2002.
- [5] Seth Copen Goldstein and Mihai Budiu. Nanofabrics: Spatial computing using molecular electronics. In *International Symposium on Computer Architecture (ISCA)*, July 2001.
- [6] James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture:

Opportunities for nanotechnology. *Science*, 280:1716–1721, June 1998.

- [7] Hiroki Iwamura, Masamichi Akazawa, and Yoshihito Amemiya. Single-electron majority logic circuits. *IEICE Transactions on Electronics*, E81-C(1):42–48, 1998.
- [8] Parag K. Lala. *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2001.
- [9] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *International Conference on Dependable Systems and Networks*, 2002.
- [10] Daniel P. Siewiorek and Robert Swarz. *Reliable Computer Systems: Design and Evaluation*. A. K. Peters, third edition, 2001.
- [11] Kang L. Wang. Issues of nanoelectronics: A possible roadmap. *Journal of Nanoscience and Nanotechnology*, 2(3/4):235–266, 2002.