

# Fault Tolerant Nanocomputing

Bharat Joshi, Dhiraj K. Pradhan, and Saraju P. Mohanty

**Abstract** This chapter provides an overview of fault tolerant nanocomputing. In general, fault tolerant computing can be defined as the process by which a computing system continues to perform its specified tasks correctly in presence of faults with the goal of improving the dependability of the system. Principles of fault tolerant nanocomputing as well as applications of the fault tolerant nanocomputers are discussed. The chapter concludes by observing trends in the fault tolerant nanocomputing.

**Keywords** Availability · Coverage · Dependability · Fault tolerance · Redundancy Reliability

## Introduction

Fault tolerant computing can be defined as the process by which a computing system continues to perform its specified tasks correctly in presence of faults. These faults could be transient, permanent, or intermittent faults. A fault is said to be transient if it occurs for a very short duration of time. While permanent faults are the faults that continue to exist in the system, intermittent faults repeatedly appear for a period of time. They could be either hardware or software faults caused by errors in specification, design, or implementation, or faults caused by manufacturing defects. They could also be caused by external disturbances or simply due to ageing of the components in a system.

---

B. Joshi (✉)  
University of North Carolina-Charlotte, USA  
e-mail: [bsjoshi@uncc.edu](mailto:bsjoshi@uncc.edu)

D.K. Pradhan  
University of Bristol, UK  
e-mail: [pradhan@compsci.bristol.ac.uk](mailto:pradhan@compsci.bristol.ac.uk)

S.P. Mohanty  
University of North Texas, USA  
e-mail: [saraju.mohanty@unt.edu](mailto:saraju.mohanty@unt.edu)

The goal of fault tolerant computing is to improve the dependability of a system where dependability can be defined as the ability of a system to deliver service at an acceptable level of confidence in either presence or absence of faults. Among all the attributes of dependability reliability, availability, fault coverage, and safety are commonly used to measure dependability of a system. Reliability of a system is defined as the probability that the system performs its tasks correctly throughout a given time interval [18, 49, 52]. Availability of a system is the probability that the system is available to perform its tasks correctly at time  $t$ . Fault coverage, in general, implies the ability of the system to recover from faults and continue to operate correctly given that it still contains a sufficient complex of functional hardware and software. Finally, safety of a system is the probability that the system will either operate correctly or switch to a safe mode if it operates incorrectly in event of a fault.

The concept of improving dependability of computing systems by incorporating strategies to tolerate faults is being attempted before. Earlier computers, such as the Bell Relay Computer built in 1944, used ad-hoc techniques to improve reliability. The first systematic approach for fault tolerant design to improve system reliability was published by von Neumann in 1956 [58]. Earlier designers improved the computer system reliability by using fault tolerance techniques to compensate for the unreliable components that included vacuum tubes and electromechanical relays that had high propensity to fail. With the advent of more reliable transistor technology the focus shifted from improving reliability to improving performance [18, 38, 50]. While component reliability has drastically improved over the past 40 years increases in device densities have led to realization of complex computer systems that are more prone to failures. Researchers in industry and academe have proposed techniques to reduce the number of faults. However, with the advent of nanotechnology where reliability of the nanoelectronic devices is not fully convincing, it has been recognized that a shift in design paradigm to incorporate strategies to tolerate faults is much more important and necessary in order to improve dependability.

The invention of nanometer-scale devices suggest that extremely large scale integration of these devices, in the order of  $10^{12}$  devices/m<sup>2</sup>, will be potentially feasible in future. A chip with 2.9 billion of 32 nm nanoscale CMOS (complementary metal oxide) devices has already been demonstrated [17]. However, to fully realize the benefits offered by such systems it is prudent to address the uncertainties introduced due to inherent characteristics of the devices, stochastic nature of the fabrication processes, and external disturbances. These uncertainties will lead to faulty devices and interconnects.

The high level of unreliabilities common to the nanoscale devices stem from two different origins. The probabilistic nature of the self-assembly fabrication process will inevitably result in a fairly large percentage of defective devices. For example, while the manufacturing failure rate of the conventional complementary metal oxide semiconductor (CMOS) device is approximately  $10^{-7} - 10^{-6}$ , the failure rate of a typical nanoscale device is just under unity. Moreover, these devices are more sensitive to external disturbances, such as radiation effects, electromagnetic influence, parameter fluctuations, and high temperatures. In addition,

process variations introduced due to uncertainty and complexity of nanoscale device fabrication process can affect leakage dissipation, power consumption, reliability, and defect [30, 32, 33]. Hence permanent and transient [28] faults or defects will occur during the manufacturing phase and during the field operation due to such factors as aging, while transient faults will appear in the field due to external disturbances. Thus successful realization of effective systems consisting of a trillion unreliable devices per square centimeter has created interesting opportunities and unique challenges in incorporating fault tolerance techniques in nano-computing.

In the current VLSI processes the defect (fault) density is in the range of 1 part per billion and thus the manufacturer can afford to discard the defective chip. Some manufacturers also add redundancy that can replace the defective (faulty) resource with the goal of increasing the yield. While the exact levels of fault densities in nanoelectronic systems are not known researchers have assumed that up to 15% of the resources on a chip will be faulty at the end of the manufacturing cycle. Hence, discarding such large percentage can over burden the cost of healthy chips which will be eventually sold to costumers. With this high level of fault densities none of the techniques used in the current VLSI processes are applicable. Thus, reliability issues need to be addressed at each level of circuit abstraction, from system level to device level. Handling reliability issues may be preferred at higher levels of design abstraction, such as system or architectural level for detection and prevention of faults at the early stage of design cycle. Current fault tolerance research for nanoelectronic systems can be broadly divided into three classes, namely: (1) masking techniques, (2) reconfiguration techniques, and (3) designs that are inherently fault tolerant.

## Principles of Fault Tolerant Nanocomputer Systems

The goal of fault tolerant nano-computer design is to improve the dependability of the system by improving the reliability during the manufacturing phase as well as field operation. Any fault tolerance technique requires the use of some form of redundancy which increases both the cost and the development time. Furthermore, redundancy can also have impact on performance, power dissipation, weight, and size of the system. Thus a good fault tolerant design is a trade-off between the level of dependencies provided and the amount of redundancies used. For nanocomputers the redundancy could be in various forms including: (1) hardware, (2) information, and (3) temporal.

One of the most challenging tasks in the design of fault tolerant systems is the evaluation of the design for the dependability requirements [18,21,39,42,48,50,59]. All the evaluation methods can be divided into two main groups, namely the qualitative and quantitative methods. Qualitative techniques, which are typically subjective, are used when certain parameters that are used in the design process cannot be quantified. As the name implies, in the quantitative methods numbers are derived for a certain dependability attribute of the system and different systems

can be compared with respect to this attribute by comparing the numerical values. However, this requires development of appropriate models. At present, several probabilistic models have been developed based on combinatorial techniques and Markov and semi-Markov stochastic processes.

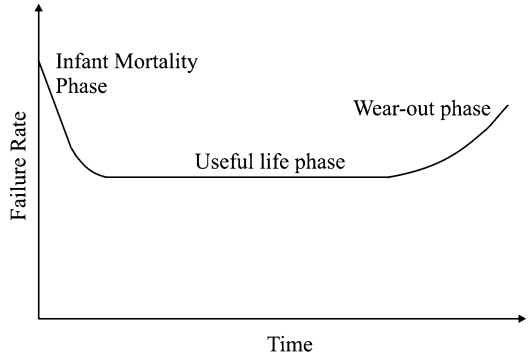
Fundamental to all these models are the probability of failure,  $p_m$ , of a nanoscale device during manufacturing and the failure rate defined as the expected number of failures per a time interval during field operation. Failure rates of most of the electronic devices follow bathtub curve shown in Fig. 1. Usually the useful life phase is the most important phase in a system's life and during this time the failure rate is assumed to be constant and it is typically denoted by  $\lambda$ . During this phase reliability of a system and the failure rate are related by the exponential failure law expressed as:

$$R(t) = e^{-\lambda t}.$$

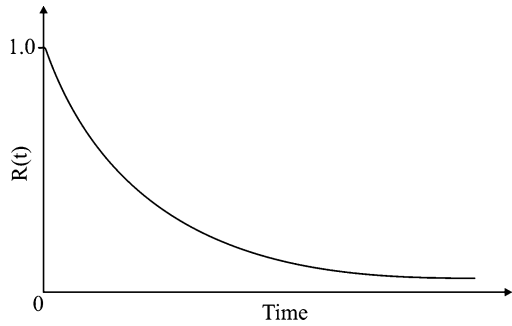
Figure 2 shows the reliability function with the constant failure rate. The failure rate  $\lambda$  is related to the mean time to failure (MTTF) as follows:

$$MTTF = 1/\lambda,$$

where MTTF is defined as the expected time until the occurrence of the first failure of the system.



**Fig. 1** Bathtub curve



**Fig. 2** Reliability function with constant failure rate

Researchers have developed computer-aided design (CAD) tools based on quantitative methods, such as CARE III (NASA's Langley Research Center), SHARPE developed at the Duke University, DEPEND (University of Illinois at Urbana Champaign), and HIMAP (Iowa State University), to evaluate dependability requirements.

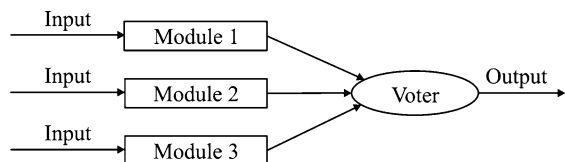
In the last several years there has been a considerable interest in the experimental analysis of computer system dependability. In the design phase CAD tools are used to evaluate the design by extensive simulations that include simulated fault injection. On the other hand, during the prototyping phase the system is allowed to run under a controlled environment. During this phase physical fault injection is used for the evaluation purposes. Several CAD tools are available including FTAPE developed at the University of Illinois at Urbana Champaign, NANOLAB developed at Virginia Polytechnic and State University, and Ballista developed at the Carnegie-Mellon University. In the following sections several redundancy techniques for fault tolerance will be briefly described.

## Hardware Redundancy

It is one of the most common forms of redundancy in which functionally identical modules replicated so that a faulty module can be replaced by a fault free module. Since each module acts as a fault-containment region the size of the fault containment region is dependent on the complexity of the module. These modules can be in various forms, such as, logic gates, logic blocks, functional units, or subsystems.

### *N-Modular Redundancy*

The basic hardware redundancy that can be used are known as passive or static techniques, where the faults are masked or hidden by preventing them from producing errors [34, 39]. Fault masking is accomplished by using voting mechanisms, typically majority voting, and the technique does not require any fault detection or system reconfiguration. The most common form of this type of redundancy is triple modular redundancy (TMR). As shown in Fig. 3, three identical modules are used and majority voting is performed to determine the output. Such a system can tolerate one fault. However, the voter is the single-point-of-failure, that is, the failure of the voter leads to the complete failure of the system.



**Fig. 3** Basic triple modular redundancy

A generalization of the TMR technique is the  $N$ -modular redundancy (NMR) where  $N$  modules are used, where  $N$  is typically an odd number. Such a system can mask up to  $\lfloor \frac{N-1}{2} \rfloor$  faulty modules.

In general, NMR system is a special case of  $M$ -of- $N$  system that consists of  $N$  modules out of which at least  $M$  of them should work correctly in order for the system to operate correctly. Thus, the reliability of  $M$ -of- $N$  system is given by,

$$R(t) = \sum_{i=M}^N \binom{N}{i} R^i(t) [1 - R(t)]^{N-i}.$$

For a TMR system  $N = 3$  and  $M = 2$  and if an non-ideal voter is assumed ( $R_{voter} < 1$ ) then the reliability is given by:

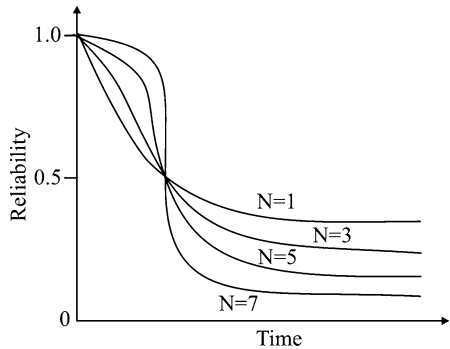
$$R(t) = \sum_{i=2}^3 \binom{3}{i} R^i(t) [1 - R(t)]^{3-i} = R_{voter}(t) (3R^2(t) - 2R^3(t)).$$

Figure 4 shows reliability plots of a simplex, TMR, and NMR when  $N = 5$  and 7. It can be seen that higher redundancy leads to higher reliability in the beginning. However, the system reliability sharply falls at the end for higher redundancies.

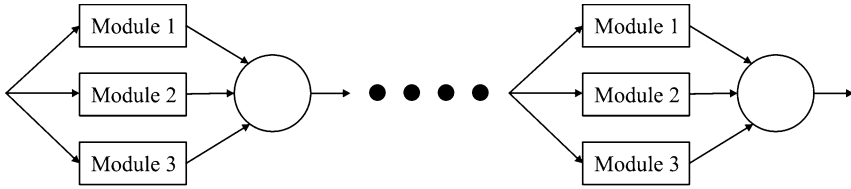
Extension of the simple TMR system is cascaded TMR (CTMR). A simple CTMR scheme with simplex voters is shown in Fig. 5. The reliability of the scheme is expressed by the following:

$$R_{CTMR}(t) = \left( R_{voter}(t) \left( R^3(t) + \binom{3}{2} R^2(t) (1 - R(t)) \right) \right)^n$$

The NMR scheme for safety usually requires trade-off between reliability and safety. For example, one can use  $k$ -out-of- $n$  system where a voter produces the output  $y$  only if at least  $k$  modules agree on the output  $y$ . Otherwise, the voter asserts the unsafe flag. Noting that the reliability refers to the probability that the



**Fig. 4** Reliability plots of NMR system with different values of  $N$ . Note that  $N = 1$  implies a simplex system



**Fig. 5** Cascaded TMR with simplex voters

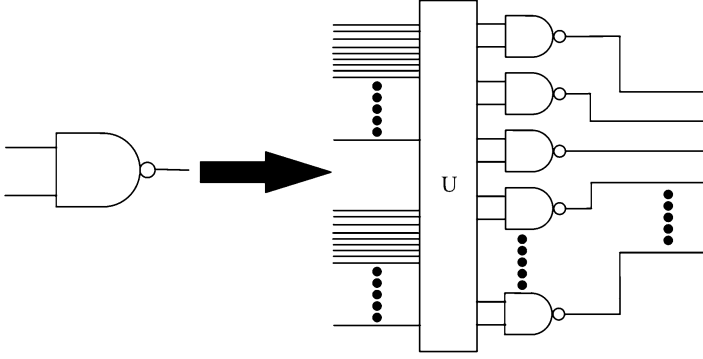
system produces correct output and safety refers to the probability that the system either produces the correct output or the error is detectable. Thus, high reliability implies high safety. However, the reverse is not true.

Voting of the NMR system becomes complicated when non-classical faults are present. Systems requiring ultra-high reliability a voting mechanism should be able to handle arbitrary failures, including the malicious faults where more than one faulty units may collaborate to disable the system. The Byzantine [24] failure model was proposed to handle such faults.

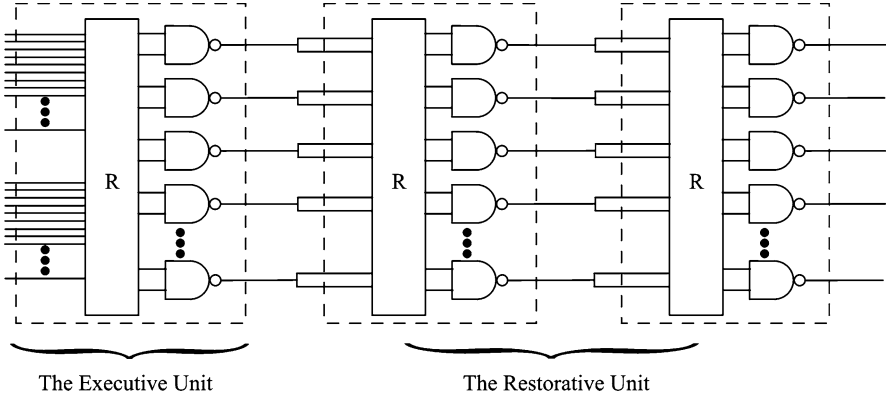
The reliability equations presented above can be used if the faulty devices are independent and uniformly distributed. For example, these equations are reasonable under the transient faults that are expected to dominate during the useful lifetime of the nanoelectronic chip. However, they are not applicable for manufacturing defects and permanent faults since they tend to cluster on a chip rather than being statistically independent. Although it is premature to predict the processes that will be used for manufacturing nanocomputers many researchers, such as in [15, 34], have attempted to provide insight into reliability issues based on the current processes.

## ***NAND Multiplexing***

In 1956 von Neumann [58] proposed a scheme, called the multiplexing technique, in which redundant unreliable components were used to create a reliable system. There has been a renewed interest in this concept since researchers are attempting to develop reliable systems based on unreliable nanoelectronic devices. The problem of enabling reliable computation using unreliable components is typically termed as probabilistic computation. NAND multiplexing refers to a construction shown in Fig. 6. It consists of  $N$  NAND gates and a randomization unit that randomly pairs each input from the first set of inputs with an input from the second set of inputs to form a pair of inputs to one of the NAND gates. In any system based on such a construction each signal is carried on a bundle of  $N$  wires and each logic computation is performed by  $N$  gates. The state of a bundle (0 or 1) is decided by whether the fraction of the excited wires are below or above a predetermined threshold value.



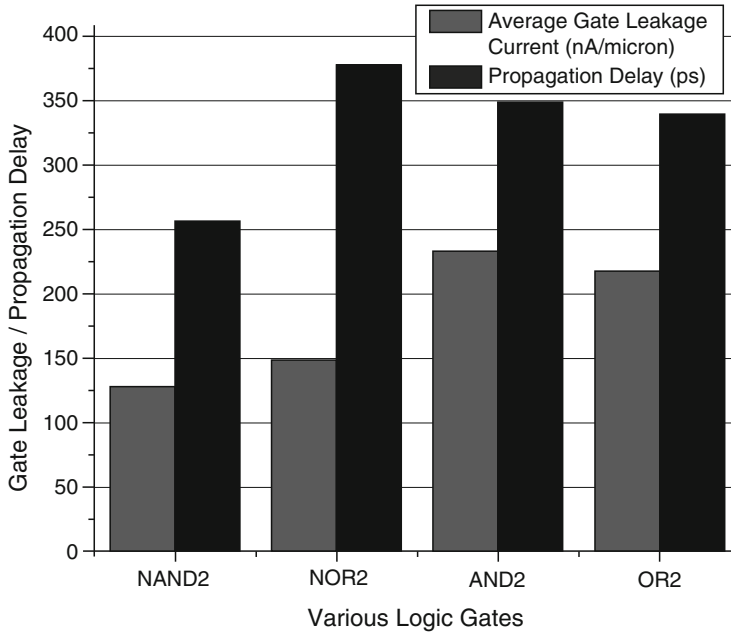
**Fig. 6** NAND multiplexing unit



**Fig. 7** NAND multiplexing system

As shown in Fig. 7, the computation is carried out by three cascaded NAND multiplexers. The executive unit performs the NAND operation while the restorative unit compensates for the degradation caused in the executive unit for a certain types of input configurations. von Neumann had concluded that by increasing the bundle size computation by such constructions can be performed such that the probability of the system failure is less than the probability of each NAND gate failing. Recently several researchers have extensively analyzed and proposed designs based on NAND multiplexing, such as the ones in [3,4,15,41,44]. Use of NAND instead of other logic gates can be of further advantage from leakage dissipation point of view. It has been recently observed (Fig. 8) that that NAND has minimal average gate-oxide leakage and propagation delay compared other logic gates of same number of inputs [31–33]. Leakage is a major issue nanoscale circuits and system in which gate-oxide leakage is predominating for sub-65 nm CMOS technology [30,31,33].





**Fig. 8** Gate-oxide leakage and delay of 45 nm CMOS logic gates

## ***Reconfiguration***

The main motivations for incorporating fault tolerance at chip level is yield enhancement and enable real-time and compile-time reconfiguration with the goal of isolating faulty unit, such as, logic unit, processor or memory cells, during field operation.

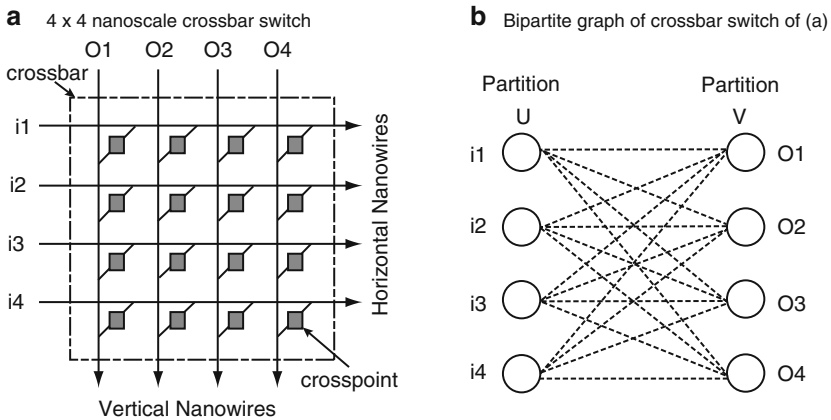
Strategies for manufacturing time and compilation time reconfigurations are similar as they do not affect the normal operation of the system and there are no real-time constraints imposed on reconfiguration time. Real-time reconfiguration schemes are difficult to design since the reconfiguration has to be performed without affecting the operation of the system. If the erroneous results are not acceptable at all then static or hybrid techniques are typically used, otherwise cheaper dynamic techniques would suffice. The effectiveness of any reconfiguration scheme is measured by two aspects: (1) the probability that a redundant unit can replace a faulty unit and (2) the amount of reconfiguration overhead involved.

Numerous schemes have been proposed for all the three types of reconfiguration in the literature [2–9, 16, 20, 22, 38, 45, 46]. One of the first schemes that was proposed for an array of processing elements allowed each processing element to be converted into a connecting element for signal passing so that a failed processor resulted in converting other processors in the corresponding row and column to connecting elements and they ceased to perform any computation. This simple scheme is not

practical for multiple faults since an entire row and column must be disabled for each fault. This problem has been addressed by several reconfiguration schemes that either use spare columns and row or redundant processing elements are dispersed throughout the chip, such as the interstitial redundancy scheme.

Hewlett-Packard has recently demonstrated an  $8 \times 8$  crossbar switches using molecular switches at the cross-points. It was observed that 15% of the switches were defective. Such a high rate of defect rate has drawn the attention of several researchers to design effective strategies to bypass the defective switches in order to perform reliable computation on the remaining fault-free structure. As shown in Fig. 9 [56], a 2-D crossbar system can be represented by a bipartite graph  $B = (U, V, E)$ , where  $U$  is the set of inputs,  $V$  is the set of outputs, and  $E$  is the set of switches of the crossbar system. When the system is defect-free the corresponding bipartite graph is complete. Given a defective  $n \times n$  crossbar switches the problem is to find the largest  $k \times k$  defect-free subset. This corresponds to finding a balanced complete bipartite subgraph which has been proven to be NP complete. Thus researchers have attempted to design efficient heuristics to find such subsets [1, 56].

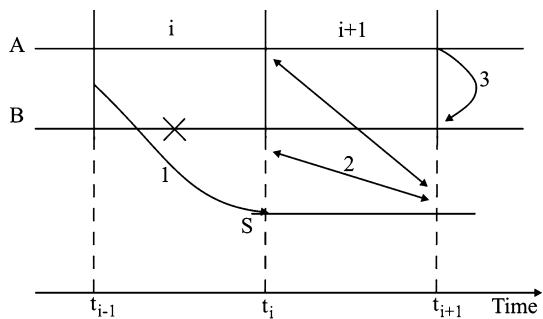
Compile-time and run-time reconfiguration requires built-in fault detection capability. Such strategies typically follow three basic steps: (1) fault detection and location, (2) error recovery, and (3) reconfiguration of the system. When a fault is detected and a diagnosis procedure locates the fault [19, 23, 25, 26, 36] the system is usually reconfigured [37, 39, 50] by activating a spare module or if the spare module is not available then the reconfigured system may have operate under degraded mode with respect to the resources available. Finally, error recovery is performed in which the spare unit takes over the functionality of the faulty unit from where it left off. While various approaches have been proposed for these steps some of them have been successfully implemented on the real systems. For example, among the recovery strategies the most widely adopted strategy is the rollback recovery using checkpoints [13, 47, 60]. The straight forward recovery strategy is to terminate execution of the program and re-executing the complete program from the beginning on all the processors known as the global restart. However, this leads to severe



**Fig. 9** 4 × 4 Crossbar system and its corresponding bipartite graph

performance degradation and increased power requirements since several tasks may have to be repeated. Thus, the goal of all the recovery strategies is to perform effective recovery from the faults with minimum overheads. The rollback recovery using checkpoints involves storing adequate amount of processor state information at discrete points during the execution of the program so that the program can be rolled back to these points and restarted from there in the event of a failure. The challenge is when and how these checkpoints are created. For interacting processes inadequate check pointing may lead domino effect where the system is forced to go to global restart state. Over the years researchers have proposed strategies that effectively address this issue.

Although rollback recovery schemes are effective in most of the systems they all suffer from substantial inherent latency that may not be acceptable for real-time systems and increase power requirements. For such systems forward recovery schemes are typically used [40, 57]. The basic concept common to all the forward recovery schemes is that when a failure is detected the system discards the current erroneous state and determines the correct state without any loss of computation. These schemes are either based on hardware or software redundancy. The hardware based schemes can further be classified as static or dynamic redundancy. Several schemes based on dynamic redundancy and check-pointing that avoid rollback have been proposed in the literature. Unlike rollback schemes, when a failure is detected the roll-forward check-pointing schemes (RFCS) attempt to determine the faulty module and the correct state of the system is restored once the fault diagnosis is done. For example, in one RFCS the faulty processing module is identified by retrying the computation on a spare processing module. During this retry the duplex modules continue to operate normally. Figure 10 shows execution of two copies, *A* and *B* of a task. Assume that *B* fails in the check-pointing interval *i*. The checkpoints of *A* and *B* will not match at the end of the check-pointing interval *i* at time  $t_i$ . This mismatch will trigger concurrent retry of the checkpoint interval *i* on a spare module while *A* and *B* continue to execute into the check-pointing interval *i* + 1. The previous checkpoint taken at  $t_{i-1}$  together with the executable code is loaded



- 1: Copy state and executable code to spare
- 2: Compare states A and S and B and S
- 3: Copy state from A to B
- X: A fault

**Fig. 10** A RFCS scheme

into the spare and the checkpoint interval  $i$  in which the fault occurred is retired on the spare module. When the spare module completes the execution at time  $t_{i+1}$  the state of the spare module is compared with the states of  $A$  and  $B$  at time  $t_i$ , and  $B$  is identified faulty. Then, the states of  $A$  are copied to  $B$ . A rollback is required if both  $A$  and  $B$  have failed. In this scheme one fault is tolerated without paying the penalty of the rollback scheme. Several versions of this scheme have been proposed in the literature.

## Information Redundancy

Hardware redundancy is very effective, however it is expensive. In the information redundancy approach, redundant information is added to enable fault detection and sometimes fault tolerance by correcting the affected information. For certain systems, such as memory and bus, error-correcting codes are cost effective and efficient. Information redundancy includes all error-detecting codes, such as various parity codes,  $m$ -of- $n$  codes, duplication codes, checksums, cyclic codes, arithmetic codes, Berger codes, and Hamming and Bose-Chaudhuri-Hocquenghem (BCH) error-correcting codes.

Selection of a particular coding technique involves tradeoffs among several factors. For example, encoding and decoding requires time and hardware while incorporation of extra bits requires extra storage and circuitry. An important issue is to select a code that meets the requirement of error detection and/or correction and at the same time keeps the cost within the acceptable level [17]. Important factors that have to be considered in the selection process of a code include whether the code should be separable, if the code should have the capability to detect error, correct error, or both, and the number of bits errors that need to be detected or corrected. Several researchers have addressed these issues for nanoelectronic digital memories [11, 53–55].

## Time Redundancy

Both hardware and information redundancy methods require a substantial amount of extra hardware which tends to increase size, weight, cost, and power consumption of the system. Time redundancy attempts to reduce the hardware overhead by using extra time. The basic concept is to repeat execution of a software module to detect faults. The computation is repeated more than once and the results are compared. An existence of discrepancy suggests a fault and the computations are repeated once more to determine if the discrepancy still exists. Earlier this scheme was used for transient fault detection. However, with a bit of extra hardware permanent faults can also be detected. In one of the schemes, the computation or transmission of data is done in the usual way. In the second round of computation or transmission the data is encoded and the results are decoded before comparing them to the previous

results. The encoding scheme should be such that it enables detection of the faults. Such schemes may include complementation and arithmetic shifts.

## Coverage

The coverage probability varies depending on the circumstances. The systems in which the failure rate is dominated by the reliability of the hardware modules being protected through redundancy, their coverage probability may be of relatively minor importance and treating it as a constant may be adequate. However, if the goal is to provide extremely high reliability, then it is easy to provide sufficient redundancy to guarantee that a system failure due to the exhaustion of all hardware modules is arbitrarily small. In this case, failures from imperfect coverage begin to dominate and more sophisticated models are needed to account for this fact.

The fault coverage consists of three components: (1) the probability that the fault is detected, (2) the probability that it is isolated to the defective module, and (3) the probability that normal operation can be resumed successfully on the remaining healthy modules. These probabilities are often time dependent, particularly in cases in which uninterrupted operation is required or in which loss of data or program continuity cannot be tolerated. If too much time elapses before a fault is detected, for example, erroneous data may be released with potentially unacceptable consequences.

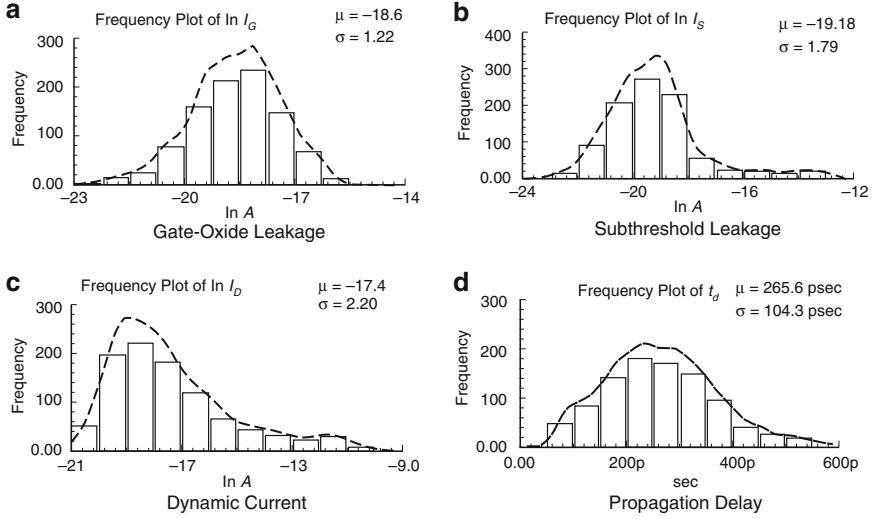
Markov models can be extended to account for coverage effects by inserting additional states. Following a fault three changes happen: (1) the system transitions to a state in which a fault has occurred, but has not yet been detected, (2) from there to a state in which it has been detected but not yet isolated, and (3) from there to a state in which it has been isolated but normal operation has not yet resumed successfully and finally to a successful recovery state. Transitions out of each of those states can be either to the next state in the chain or to a failed state, with the probability of each of these transitions most likely dependent on the time spent in the state in question. State transition models in which the transition rates are dependent on the state dwelling time are called semi-Markov processes. These models can be analyzed using techniques similar to those illustrated in the earlier Markov analysis, but the analysis is complicated by the fact that differences between the transition rates that pertain to hardware or software faults and those that are associated with coverage faults can be vastly different. The mean time between hardware module failures, for example, typically is on the order of thousands of hours whereas the time between the occurrence of a fault and its detection may well be of the order of microseconds. The resulting state-transition matrix is called “stiff” and any attempt to solve it numerically may not converge. For this reason, some of the aforementioned reliability models use a technique referred to as behavioral decomposition. Specifically, the model is broken down into its component parts, one subset of models used to account for coverage failures and the second incorporating the outputs of those models into one accounting for hardware and software faults.

## Process Variations in Nanoscale Circuits

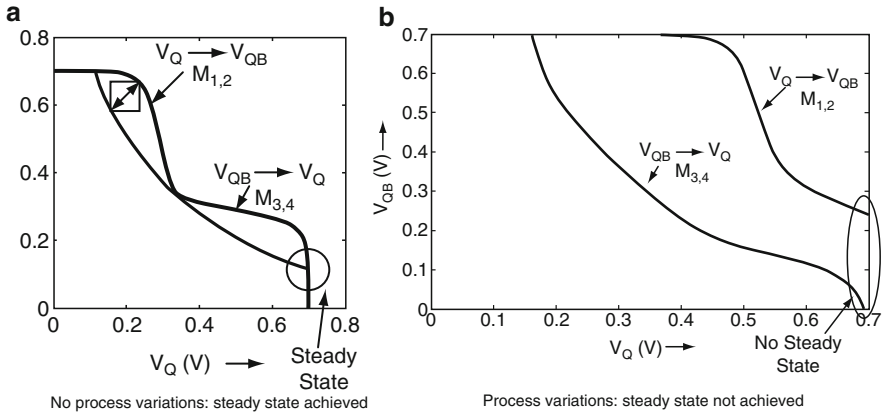
A single important factor that affects leakage dissipation, power consumption, reliability, defect, yield, and circuit optimization, etc. in nanoscale devices, circuits, and systems is process variation [12, 14, 30, 32, 33, 51]. For the fabrication of circuits using nanoscale devices more and more sophisticated lithographic, chemical, and mechanical processing steps are adopted. The uncertainty in the processes, such as ion implantation, chemical mechanical polishing (CMP), chemical vapor deposition (CVD), etc. involved in nanoscale device fabrication has caused variations in process parameters [12, 14] such as channel length, gate-oxide thickness, threshold voltage, metal wire thickness, via resistance, etc. These variations are categorized as inter-die variations and intra-die. This may include systematic as well random variations. The process variations are treated as global or local variations. These variations can be either temporal or spatial in nature [14]. They ultimately affect design margins and yield and may lead to loss of money in the ever reducing time-to-market scenario.

It has been observed that the major components power dissipation, such as gate-oxide leakage ( $I_{gate}$ ), and subthreshold leakage ( $I_{sub}$ ), and dynamic current ( $I_{dyn}$ ) as well as propagation delay predominantly depend on the device geometry namely gate-oxide thickness ( $T_{gate}$ ), device length ( $L_e$ ), threshold voltage ( $V_{th}$ ), and supply voltage ( $V_{DD}$ ), etc. Thus, in order to incorporate the process variation effects, statistical modeling of leakage current components is a key area to characterize nano-CMOS. Monte Carlo simulations provide a method to analyze the effect of process variation exhaustively. Via Monte Carlo simulations, the process and design variations are translated into gate-oxide leakage, subthreshold leakage, dynamic current, and propagation delay probability density distributions for a two-input NAND logic gate. For Gaussian input process and design variations, it is observed that gate-oxide leakage, subthreshold leakage, and dynamic currents are lognormally distributed [30, 32, 51]. On the other hand, the delay follows normal distribution. This is demonstrated in Fig. 11 for a 45 nm CMOS logic gate.

In addition to the variations in power, leakage and delay profile of devices and circuits, the variations can have impact on circuit operation. One important example is read-failure that can develop in a SRAM circuit realized using nano-CMOS devices. With increase in device density a larger fraction of a system on a chip (SoC) area is devoted to SRAM, because on-chip memory considerably offers the high system performance in exchange of space and power they consume. Correct operation of the SRAM circuit in both hold and read states are important for its faithfully operations. Let us consider a SRAM with two cross coupled inverted  $M_{1,2}$  and  $M_{3,4}$  with the two inputs of the inverters are Q and QB, respectively. From the plot the voltage transfer characteristics (VTC) as shown in Fig. 12, it is evident that read stability failure because there is no steady stable state. This happens when variations is introduced in the threshold voltage ( $V_{th}$ ) of transistors. Thus, a 6-transistor SRAM cell is not stable during read operation under process variations.



**Fig. 11** Effects of statistical process variations on gate-oxide leakage, subthreshold leakage, dynamic power, and propagation delay in a two-input 45 nm CMOS NAND logic gate



**Fig. 12** Voltage Transfer Characteristic (VTC) in read state of a 45 nm CMOS 6-Transistor static random-access memory (SRAM) cell showing read failure introduced due to process variation

## Fault Tolerant Nanocomputer Applications

Incorporation of fault tolerance in nanocomputers will have tremendous impact on the design philosophy. A good design is a trade-off between the cost of incorporating fault tolerance and the cost of errors that, includes losses due to downtime and the cost of erroneous results.

Historically the applications of the fault tolerant computers were confined to military, communications, industrial, and aerospace applications where the failure of a computer could result in substantial financial losses and possibly loss of life. Currently most of the fault tolerant computers developed can be into five general categories based on the design requirements and challenges. As nanocomputers pervade these areas innovative fault tolerance strategies will have to be developed since these computers will be prone to different types and patterns of faults.

### ***General-Purpose Computing***

General-purpose computers include workstations, and personal computers. These computers require minimum level of fault tolerance since errors that disrupt the processing for a short period of time is acceptable provided the system recovers from these errors. The goal is to reduce the frequency of such outages and increase reliability, availability, and maintainability of the system.

Since the attributes of each of the main components, namely the processor, memory, input, and output, are unique, the fault tolerant strategies employed for these components are in general different. Moreover, the nanoelectronic devices based systems will be highly prone to transient failures than other failures and it will be desirable to facilitate rapid recovery. One of the most effective approaches for rapid error recovery for transient faults would be instruction retry in which the appropriate instruction is retried once the error is detected [35].

### ***Long-Life Applications***

Applications typical of unmanned space flight and satellites require that the computers have reliability of 0.95 or greater at the end of a 10-year period. Computers in these environments are expected to use the hardware in an efficient way due to limited power availability and constraints on weight and size. Long-life systems can sometimes allow extended outages provided the system becomes operational again.

### ***Critical-Computation Applications***

In these applications errors in computations can be catastrophic, such as human safety. Applications include certain type of industrial controllers, space applications, such as the space shuttle, aircraft flight control systems, and military applications. A typical requirement for such a system is to have reliability of 0.9999999 at the end of 3-h time period.

One of the most important applications of nanocomputing would be in healthcare. It has been envisioned that in future it would be possible to deploy



large number of nanorobots to eliminate diseases and relieve a patient of physical trauma. This will require a high level of reliability and safety involving innovative strategies, such as a nanorobot with self-destruction capability to ensure the safety of a patient in the event of a nanorobot failure.

### ***High-Availability Applications***

The applications under this category include time-shared systems, such as banking and reservation systems where providing the prompt services to the users is critical. In these applications while system-wide outages are unacceptable occasional loss of service to individual users are acceptable provided the service is restored quickly so that the downtime is minimized. In fact sometimes the downtime needed to update software and upgrade hardware may not be acceptable and so well coordinated approaches are used to minimize the impact on the users.

### ***Maintenance Postponement Applications***

Applications where maintenance of computers is either costly or difficult or perhaps impossible to perform, such as some space applications, fall under this category. Usually the breakdown maintenance costs are extremely high for the systems that are located remote areas. Maintenance crew can visit these sites at a frequency that is cost-effective. Fault tolerance techniques are used between these visits to ensure that the system is working correctly. Applications where the systems may be remotely located include telephone switching systems, certain renewable energy generation sites, and remote sensing and communication systems.

## **Trends and Future**

While the hardware and software costs have been dropping the cost of computer downtime has been increasing due to increased complexity of the systems. This problem will be compounded by the nanoelectronic devices due to high device densities and high failure rates. Studies suggest that the state-of-the-art fault tolerant techniques are not adequate to address this problem. Thus, a shift in the design paradigm may be needed. Power, leakage, variability, performance, yield, and fault-tolerance have to be considered simultaneously for fault-tolerant system realization using nano-devices. This paradigm shift is driven in large part by the recognition that hardware is becoming increasingly reliable for any given level of complexity. This is due to the fact that, based on empirical evidence, the failure rate of an integrated circuit increases roughly as the 0.6th power of the number of its equivalent gates. Thus, for a given level of complexity, the system fault rate decreases as the

level of integration increases. If, say, the average number of gates per device in a system is increased by a factor of 1,000, the fault rate for that system decreases by a factor of roughly 16.

In contrast, the rapid increase in the complexity of software over the years, due to the enormous increase in the number and variety of applications implemented on computers, has not been accompanied by a corresponding reduction in software bugs. Significant improvements have indeed been made in software development [27, 43], but these are largely offset by the increase in software complexity and in the increasing opportunity for software modules to interact in unforeseen ways [29] as the number of applications escalate. As a result, it is expected that the emphasis in future fault-tolerant computer design will shift from techniques for circumventing hardware faults to mechanisms for surviving software bugs.

A good viable alternative approach for fault tolerance would be to mimic biological systems. There is some activity in this area, but it is still in its infancy. For example, an approach to design complex computing systems with inherent fault tolerance capabilities based on the embryonics was recently proposed. Similarly, a fault detection, location, and reconfiguration strategy was proposed last year based on cell signaling, membrane trafficking, and cytokinesis [10].

Furthermore, the existing external infrastructure, such as Automatic Test Equipment (ATE), are not capable of dealing with the new defect levels that nanometer scale technologies create in terms of fault diagnosis, test time, and handling the amount of test data generated. In addition, continual development of ATE facilities that can deal with the new manufacture issues is not realistic. This has considerably slowed down the design of various system-on-a-chip efforts [61]. For example, while the embedded memory typically occupies half of the integrated circuits area their defect densities tend to be twice that of logic. Thus, to achieve and maintain cost advantages requires improving memory yields. As described earlier, one commonly used strategy to increase yield is to incorporate redundant elements that can replace the faulty elements during repair phase. The exiting external infrastructure cannot perform these tasks in a cost effective way. Thus, there is a critical need for on-chip (internal) infrastructure to resolve these issues effectively. The semiconductor industry has attempted to address this problem by introducing embedded intellectual property blocks called the infrastructure intellectual property (IPP). For embedded memory, the IPP may consist of various types of test and repair capabilities.

## References

1. Al-Yamani, A., Ramsundar, S., and Pradhan, D., "A Defect Tolerance Scheme for Nanotechnology Circuits," *IEEE Transactions on Circuits and Systems*, vol. 54, no. 11, November 2007, pp. 2402–2409.
2. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., and Piuri, V., "Detecting and Locating Faults in VLSI Implementations of the Advanced Encryption Standard," in *Proc. of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 105–113, Nov. 2003.

3. Bhaduri, D. and Shukla, S., "NANOLAB – A tool for evaluating reliability of defect-tolerant nanoarchitectures," *IEEE Transactions on Nanotechnology*, vol. 4, no. 4, July 2005, pp. 381–394.
4. Bhaduri, D., Shukla, S., Graham, P., and Gokhale, M., "Comparing reliability-redundancy tradeoffs for two von neuman multiplexing architectures," *IEEE Transactions on Nanotechnology*, vol. 6, no. 3, May 2007, pp. 265–279.
5. Bowen N. and Pradhan, D., "Issues in Fault Tolerant Memory Management", *IEEE Transactions on Computers*, August 1996, pp. 860–880.
6. Bowen, N. and Pradhan, D., "Processor and memory-based checkpoint and rollback recovery," *IEEE Computer*, Vol. 26, No. 2, Feb. 1993, pp. 22–31.
7. Chatterjee, M. and Pradhan D., "A GLFSR-Based Pattern Generator for Near-Perfect Fault Coverage in BIST", *IEEE Transactions on Computers*, December 2003, pp. 1535–1542.
8. Chen, C. and Somani, A., "Fault Containment in Cache Memories for TMR Redundant Processor Systems," *IEEE Transactions on Computers*, Vol. 48, No. 4, March 1999, pp. 386–397.
9. Chillarege, R. and Iyer, R., "Measurement-Based Analysis of error latency," *IEEE Transactions on Computers*, pp.529–537, May 1987.
10. Datta, K., Karanam, R., Mukherjee, A., Joshi, B., and Ravindran, A., "A Bio-Inspired Self-Repairable Distributed fault tolerant Design Methodology with Efficient Redundancy Insertion Technique," in *Proceedings of the 16<sup>th</sup> IEEE NATW*, 2006.
11. DeHon, A., Goldstein, S., Kuekes, P., and Lincoln, P., "Nanophotolithographic nanoscale memory density prospects," *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, May 2005, pp. 215–228.
12. Drennan P. and McAndrew, C., "Understanding MOSFET mismatch for analog design," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 450–456, March 2003.
13. Elnozahy, E., Alvisi, L., Wang, Y., and Johnson, D., "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, September 2002, pp. 375–408.
14. Gattiker, A., Haensch, W., Ji, B., Nassif, S., Nowak, E., Pearson, D., Bernstein, K., Frank, D., and Rohrer, N., "High-performance CMOS variability in the 65-nm regime and beyond," *IBM Journal of Research and Development*, vol. 50, no. 4/5, pp. 433–449, July–September 2006.
15. Han, J. and Jonker, P., "A system architecture solution for unreliable nanoelectronic devices," *IEEE Transactions on Nanotechnology*, vol. 1, no. 4, December 2002, pp. 201–208.
16. Hayne, R. and Johnson, B., "Behavioral Fault Modeling in a VHDL Synthesis Environment", *Proceedings of the IEEE VLSI Test Symposium*, Dana Point, California, April 25–29, 1999, pp. 333–340.
17. Intel chips to shrink to 32-nanometer process, [http://www.infoworld.com/article/07/09/18/Intel-chips-shrink-to-32-nanometer-process\\_1.html](http://www.infoworld.com/article/07/09/18/Intel-chips-shrink-to-32-nanometer-process_1.html) or <http://www.intel.com/idf/>.
18. Johnson, B., "Design and Analysis of Fault Tolerant Digital Systems," Addison-Wesley, 1989.
19. Joshi, B. and Hosseini, S. "Diagnosis Algorithms for Multiprocessor Systems," in *Proc. IEEE Workshop on Embedded Fault-Tolerant Systems*, Boston, MA, 1998.
20. Kaufman, L., Bhide, S., and Johnson, B., "Modeling of Common-Mode Failures in Digital Embedded Systems", in *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*, Los Angeles, California, January 24–27, 2000, pp. 350–357.
21. Koren, I. and Mani Krishna, C., "Fault-Tolerant Systems," Morgan Kauffman Publishers, 2007.
22. Krishnaswamy, S., Markov, I., and Hayes, J., "Logic circuit testing for transient faults," in *Proceedings of the European Test Symposium (ETS'05)*, 2005.
23. Lala, P., "Self-Checking and Fault Tolerant Digital Design," Morgan Kaufmann, 2000.
24. Lamport, L., Shostak, S., and Pease, M., "The Byzantine Generals Problem," *ACM Transactions of Programming Languages and System*, July 1982, pp. 382–401.
25. Li, M., Goldberg, D., Tao, W., and Tamir, Y., "Fault-Tolerant Cluster Management for Reliable High-Performance Computing," in *Proc. of the 13th International Conference on Parallel and Distributed Computing and Systems*, Anaheim, CA, pp. 480–485 (August 2001).
26. Liu, C. and Pradhan, D., "EBIST: A Novel Test Generator with Built-In-Fault Detection Capability", *IEEE Transactions on CAD*, Volume 24, No. 8, August 2005.

27. Lyu, M., editor, "Software Reliability Engineering," McGraw Hill, 1996.
28. Maheshwari, A., Koren, I., and Burleson, W., "Techniques for Transient Fault Sensitivity Analysis and Reduction in VLSI Circuits," *Defect and Fault Tolerance in VLSI Systems*, 2003.
29. Messer, A., Bernadat, P., Fu, G., Chen, D., Dimitrijevic, Z., Lie, D., Mannaru, D., Riska, A., and Milojicic, D., "Susceptibility of commodity systems and software to memory soft errors," *IEEE Transactions on Computers*, Vol. 53, No. 12, pp.1557–1568, Dec. 2004.
30. Mohanty, S. and Kougianos, E., "Simultaneous Power Fluctuation and Average Power Minimization during Nano-CMOS Behavioral Synthesis", in *Proceedings of the 20th IEEE International Conference on VLSI Design (VLSID)*, pp. 577–582, 2007.
31. Mohanty, S. and Kougianos, E., "Steady and Transient State Analysis of Gate Leakage Current in Nanoscale CMOS Logic Gates", in *Proceedings of the 24th IEEE International Conference on Computer Design (ICCD)*, pp. 210–215, 2006.
32. Mohanty, S., Kougianos, E., Ghai D., and Patra, P., "Interdependency Study of Process and Design Parameter Scaling for Power Optimization of Nano-CMOS Circuits under Process Variation", in *Proceedings of the 16th ACM/IEEE International Workshop on Logic and Synthesis (IWLS)*, pp. 207–213, 2007.
33. Mohanty, S., Ranganathan N., Kougianos, E., and Patra, P., *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*, 1<sup>st</sup> Edition, Springer, 2008.
34. Nikolic, K., Sadek, A., and Forshaw, M., "Fault-tolerant techniques for nanocomputers," *Nanotechnology*, vol. 13, 2002, pp. 357–362.
35. Oh, N., Shirvani, P., and McCluskey, E., "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, Vol. 51, pp. 63–75.
36. Omari, R., Somani, A., and Manimaran, G., "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *Journal of Parallel and Distributed Computing*, Vol. 65, no. 5, pp. 595–608, May 2005.
37. Papadopoulou, E., "Critical Area Computation for Missing material Defects in VLSI Circuits," *IEEE Transactions on CAD*, vol. 20, pp. 503–528, May 2001.
38. Peper, F., Lee, J., Abo, F., Isokawa, T., Adachi, S., Matsui, N., and Mashiko, S., "Fault tolerance in nanocomputers: a cellular array approach," *IEEE Transactions on Nanotechnology*, vol. 3, no. 1, March 2004, pp. 187–201.
39. Pradhan, D. (editor), "Fault-Tolerant Computer System Design," Prentice Hall, 1996.
40. Pradhan, D. and Vaidya N., "Roll-Forward Checkpointing Scheme: A Novel fault-Tolerant Architecture" *IEEE Transactions on Computers*, vol. 43, no. 10, October 1994, 1163–1174.
41. Qi, Y., Gao, J., and Fortes, J., "Markov chains and probabilistic computation – a general framework for multiplexed nanoelectronic systems," *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, March 2005, pp. 194–205.
42. Ramesh, A., Twigg, D., Sandadi, U., Sharma, T., Trivedi, K., and Somani, A., "Integrated Reliability Modeling Environment," *Reliability Engineering and System Safety*, Elsevier Science Limited; UK, Volume 65, Issue 1, March 1999, pp. 65–75.
43. Reis, G., Chang, J., Vachharajani, N., Rangan, R., August, D., and Mukherjee, S., "Software-Controlled Fault Tolerance," *ACM Transactions on Architecture and Code Optimization*, pp. 1–28, December 2005.
44. Roy, S. and Beiu, V., "Majority multiplexing – economical redundant fault-tolerant designs for nanoarchitectures," *IEEE Transaction on Nanotechnology*, vol. 4, no. 4, July 2005, pp. 441–451.
45. Schmid, A. and Leblebici, Y., "A Highly Fault Tolerant PLA Architecture for Failure-Prone Nanometer CMOS and Novel Quantum Device Technologies," *Defect and Fault Tolerance in VLSI Systems*, 2004.
46. Schwab, A., Johnson, B., and Bechta-Dugan, J., "Analysis Techniques for Real-Time, Fault-Tolerant, VLSI Processing Arrays", in *Proceedings of the 1995 IEEE Annual Reliability and Maintainability Symposium*, Washington, DC, January 17–19, 1995, pp. 137–143.
47. Sharma, D. and Pradhan, D., "An Efficient Coordinated Checkpointing Scheme for Multicomputers," *Fault-Tolerant Parallel and Distributed Systems*, IEEE Computer Society Press, 1995.
48. Shooman, M., "Reliability of Computer Systems and Networks," Wiley Inter-Science, 2002.

49. Siewiorek, D. (editor), "Fault Tolerant Computing Highlights from 25 Years," in *Special volume of the 25<sup>th</sup> International Symposium on Fault-Tolerant Computing FTCS-25*, Pasadena, CA.
50. Siewiorek, D. and Swartz, R., "The Theory and Practice of Reliable System Design," A. K. Peters, 1999.
51. Singh, J., Mathew, J., Mohanty, S., and Pradhan, D., "Statistical Analysis of Steady State Leakage Currents in Nano-CMOS Devices", in *Proceedings of the 25th IEEE Norchip Conference (NORCHIP)*, 2007.
52. Somani, A. and Vaidya, N., "Understanding fault tolerance and reliability," *IEEE Computer*, April 1997, pp. 45–50.
53. Strukov, D. and Likharev, K., "Defect-tolerant architectures for nanoelectronic crossbar memories," *Journal of Nanoscience and Nanotechnology*, vol. 7, no. 1, January 2007.
54. Strukov, D. and Likharev, K., "Prospects for terabit-scale nanoelectronic memories," *Nanotechnology*, vol. 16, January 2005, pp. 137–148.
55. Sun, F. and Zhang, T., "Defect and transient fault-tolerant system design for hybrid CMOS/nanodevice digital memories," *IEEE Transactions on Nanotechnology*, vol. 6, no. 3, May 2007, pp. 341–351.
56. Tahoori, M., "Defects, Yield, and Design in Sublithographic Nano-Electronics," *Proceedings of the 2005 20<sup>th</sup> IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005.
57. Vaidya, N. and Pradhan, D., "Fault-Tolerant Design Strategies for High Reliability and Safety," *IEEE Transactions on Computers*, vol. 42, no. 10, October 1993.
58. von Neumann, J. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies, Annals of Mathematical Studies*, Princeton University Press, No. 34, 1956, pp. 43–98.
59. Welke, S., Johnson, B., and Aylor, J., "Reliability Modeling of Hardware-Software Systems", *IEEE Transactions on Reliability*, Vol. 44, No. 3, September 1995, pp. 413–418.
60. Zhang, Y. and Chakrabarty, K., "Fault Recovery Based on Checkpointing for Hard Real-Time Embedded Systems," in *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, 2003.
61. Zorian, Y. and Chandramouli, M., "Manufacturability with Embedded Infrastructure IPs, [http://www.evaluationengineering.com/archive/articles/0504/0504embedded\\_IP.asp](http://www.evaluationengineering.com/archive/articles/0504/0504embedded_IP.asp).



<http://www.springer.com/978-90-481-8539-9>

Robust Computing with Nano-scale Devices

Progresses and Challenges

Huang, C. (Ed.)

2010, VIII, 180 p., Hardcover

ISBN: 978-90-481-8539-9