Trends and Future Directions in Nano Structure Based Computing and Fabrication

R. Iris Bahar
Division of Engineering
Brown University
Providence, RI 02912
Email: iris_bahar@brown.edu

***Abstract*— As silicon CMOS devices are scaled down into the nanoscale regime, new challenges at both the device and system level are arising. While some of these challenges will be overcome in the near future, nanoscale devices will have high manufacturing defect rates and will operate at reduced noise margins, exposing computation to higher soft error rates. Thus, a key challenge for the future will be building fault and defect-tolerant computing systems. Researchers are looking to develop hybrid systems that combine on the same chip CMOS-based circuitry with any number of alternatives, including circuits composed of nanowire or carbon nanotube devices. The big advantage of including these new devices on the same chip is the increased device densities, and potential drop in fabrication costs. On the other hand, integrating very large numbers of devices on a single chip leads to questions of how to manage so many devices with tight constraints on cost, performance, power, and reliability, without having it become a design complexity nightmare. In this paper, we review some key issues and trends arising from nanostructure based computing and fabrication, while providing a few examples of defect-tolerant circuits and architectures currently being proposed as alternatives to "traditional" computing based exclusively on CMOS technology. These include hybrid nanowire/CMOS designs, reconfigurable or redundant architectures, and designs based on probabilistic computing. We end with a discussion on future challenges and direction in nanoscale computing.**

I. Introduction

As silicon CMOS devices are scaled down into the nanoscale regime, new challenges at both the device and system level are arising. In particular, although the semiconductor industry has successfully overcome many hurdles (including the current transition to silicon-on-insulator (SOI) technology [1]), there remain many challenges for CMOS, including developing new materials (e.g., high-κ and low-κ dielectrics [2]), dealing with new device geometries (dual-gate or fin-FET devices [3]), and managing power dissipation, and economics of commodity manufacturing resulting from further downscaling of devices and supply voltages [2]. Longer term challenges include the development of a hybrid system that combines CMOS FET-based digital logic with any number of alternative devices, ranging from analog circuits, to more exotic alternatives (optical sources and detectors, quantum or molecular transistors, nanowire or carbon nanotube devices, *etc.*), all on the same chip [4]. The big advantage of including these molecular scale devices is the increased device densities — between 10X and 1000X greater than those predicted by ITRS for silicon-based CMOS [5]. On the system side, integrating very large numbers of devices on a single chip with tight constraints (cost, performance, power, and reliability), will increase the design complexity greatly.

While there is no clear consensus on how far and how fast CMOS will downscale, and which of the emerging hybrid technologies will eventually enter production, it is certain that future nanodevices will have high manufacturing defect rates. It is also clear that the supply voltage, V_{DD}, will be aggressively scaled down to reduce dynamic power dissipation. Supply voltage at $V_{DD} = 0.5V$ is the current prediction for low-power CMOS in 2018 [6], although extrapolations to even lower $V_{DD} = 0.3V$ have appeared in the literature [4]. This reduction in noise margins will expose computation to higher soft error rates. Another point to consider is the economic practicality of manufacturing devices at such small scale. Economics may dictate the use of regular layouts or self-assembly techniques rather than the use of arbitrary structures.

In this paper, we review some key issues and trends arising from nanostructure based computing and fabrication. We start in Section II with a discussion on issues facing lithographic and self-assembly techniques for device fabrication. Next, we give a taste of a few hybrid nanoscale/CMOS circuits and architectures being proposed today in Section III. All these structures include some form of fault or defect tolerance, specific to the structural implementation. In Section IV we consider general strategies for reliable computing in the presence of both static and transient errors. These strategies include triple modular redundancy, error correcting codes, and dynamic reconfiguration. In Section V, we discuss a new paradigm based on probabilistic computation, appropriate for systems operating under very noisy (i.e., fault-prone) conditions. We end with a discussion on future challenges and direction in nanoscale computing.

II. Building Top-down vs. Bottom-up Designs

Today's approach to designing integrated circuits uses a top-down methodology. That is, layers are added on top of a silicon wafer, requiring hundreds of steps before the final circuit is complete. Although this process has allowed the manufacture of reliable circuits and architectures, future scaling will make the production of reliable mask sets extremely expensive. In the near future, a major shift from top-down lithography-based fabrication may be needed in order to cost-effectively fabricate devices at true nanoscale dimension.

As an alternative, bottom-up approaches rely on self-assembly for defining feature size and may offer opportunities to drastically reduce the number of steps required to produce a circuit. However, the biggest impact in going from top-down designs to bottom-up is the inability to arbitrarily determine placement of devices or wires. Instead, we are restricted to simple structures. Therefore, self-assembly techniques are more easily applied to fabrication of two-terminal devices. Since these devices are usually non-restoring, a major design challenge is providing signal restoration between nanoscale logic stages. Furthermore, this self-assembly approach also lends itself to defect rates orders of magnitude higher than traditional top-down approaches largely because of the inability to precisely control the design of these devices. Therefore some means of defect and/or fault tolerance (whether at the circuit, logic, or architecture level) will be needed if reliable computation is to be achieved. Taking a hybrid approach that uses self-assembled structures as an add-on to a CMOS subsystem may create a design framework where fault tolerant techniques can be more effectively applied.

Bottom-up assembly techniques require fabrication regularity. However, even if a hybrid approach is used, it may be advantageous, in terms of fabrication costs, to build the photo-lithographically manufactured components from regular structures as well. Not only will this allow for an easier fabrication process, it will also lend itself more easily to reconfigurable architectures. That is, the desired circuit may be designed by configuring around faulty structures; since all structures are identical, one faulty element can be easily swapped out and replaced with an operational one, therefore creating a reliable system out of an unreliable substrate.

Of the molecular-scale devices being developed using these self-assembly techniques, the non-volatile programmable switch has gained much attention. Such a switch can be fabricated using two layers of parallel nanowires, with the two layers perpendicular to each other, forming a 2D array. At every crosspoint, the wires are connected together via a two-terminal nanodevice formed by the layering. These crossbar arrays are similar to programmable logic arrays (PLAs) and can be used as building blocks for implementing logic. The array can then be interconnected, using CMOS circuitry, as part of a hybrid nanoscale/CMOS design architecture. We describe some of these hybrid architectures in the following section.

III. Building Architectures from Hybrid Nanowire/CMOS Circuits

In this section, we provide three different examples of hybrid nanoscale/CMOS circuits and architectures being proposed today. These hybrid designs combine nanoscale devices and nanowires with larger CMOS components. The main advantage of these approaches is that the CMOS subsystem can serve as a reliable medium for connecting nanoscale circuit blocks, providing long interconnects and I/O functions.

A. nanoPLA

In [7], [8] DeHon *al.* have proposed a programmable interconnect architecture built from hybrid components. The main building block, called the *nanoPLA*, is built from a crossed set of N-type and P-type nanowires. An electrically switchable diode is formed at each crosspoint. The diodes then provide a programmable wired-OR plane that can be used to configure, or program, arbitrary logic into the PLA. The nanoPLA is programmed using lithographic scale wires along with stochastically coded nanowire address [9]. The programmability also allows defective devices to be avoided as a means of fault tolerance. DeHon and Naemi have shown that when 20% of devices (i.e., crossbar diodes) were defective, only a 10% overhead in devices was needed to correctly configure the array around the defects [10]. Note that the nanoPLAs are build on top of a lithographic substrate. Lithographic circuitry and wiring provides a reliable means of probing for defects, and configuring the logic.

The nanoPLA is composed of two stages of programmable crosspoints. The first stage defines the logical product terms (*pterms*) by creating a wired-OR of appropriate inputs. The outputs of this wire-OR plane are restored through field-effect controlled nanowires that invert the outputs (thus creating the logical NOR of the selected input signals). These restored signals are then sent to the inputs of the next stage of programmable crosspoints. Each nanowire in this plane computes the wired-OR of one or more restored *pterms*. The outputs of the stage are then restored in the same manner as the first stage. The two stages together provide NOR-NOR logic (logically equivalent to AND-OR logic of a conventional PLA) [8].

The nanoPLA blocks are interconnected by overlapping the restored output nanowires from each block with the wired-OR input region of adjacent nanoPLA blocks. This organization allows each nanoPLA block to receive inputs from a number of different nanoPLA blocks. With multiple input sources and outputs routed in multiple directions, the nanoPLA block can also serve as a switching block by configuring the overlap appropriately. Their experiments mapping benchmark circuits onto the proposed architecture have suggested that device density could be one to two orders of magnitude better than what is projected for the 22nm roadmap node [8]. See [11] for additional discussion on this architecture.

B. NASIC

Other work in hybrid systems combining nanowires and CMOS includes the NASIC designs described in [12]. Two-dimensional nanowire grids are used as the basis, and at each junction, the crossing wires act as FETs or diodes when activated. A circuit is broken up into *tiles*, where each tile contains enough circuitry to implement flip-flops, multiplexors, adders, etc. A NASIC design is effectively a connected chain of AND-OR tiles.

An inefficiency of this 2D array structure is that logic ends up along the diagonal, with large wasted spaces away from the diagonal. Another limitation is that the circuits have been based on static-ratioed logic, which requires careful sizing and may have undesirable effects in terms of static power dissipation. One solution to the static ratioed logic problem is to use dynamic, nanogrid adapted, cascaded circuits. The dynamic circuits require one type of doping in each dimension and allow latching functionality without actually adding an explicit latch circuit. The gate-level latching also provides opportunities for pipelining. By using dynamic circuits and pipelining on the wires, NASICs eliminates the need for explicit flip-flops to store intermediate values, therefore considerably improving density.

The NASIC group has used its proposed architecture to create a simple processor with a 5-stage pipeline built using 2D nanowire fabric, called the wire-streaming processor, or WISP-0 [13]. Each stage is implemented in its own tile and nanowires are used to provide communication between adjacent nanotiles. Each nanotile is surrounded by microwires that carry V_{dd}, ground, and control wires.

In order to get a sense of the kind of area advantage that might be achieved with this architecture, the authors compared the WISP-0 with a CMOS implementation that is scaled to 30nm technology. Initial estimates puts the CMOS implementation at about 12.5X larger than the WISP-0, with only 9% of the total area taken up by nanowires (i.e. microwires dominate relative to nanotiles) [13]. However, they claim that with better layout, this area advantage could go up to 100X. In their later work, they explore defect-tolerant approaches for the WISP-0 processor [14], [15]. Their solution for defect-tolerance is based mainly on built-in circuit-level redundancy in the cascaded AND-OR planes of the nanotiles. They combine this with system-level CMOS voting using triple modular redundancy (TMR) to further improve the yield (TMR will be discussed further in Section IV). Overall, they show a 3X density advantage over CMOS, even with all the built-in fault tolerance. With 14% defective transistors, the WISP-0 still had a yield of 30%.

C. Molecular CMOS (CMOL)

The molecular-CMOS, or CMOL, circuits proposed in [16], [17] are designed using the same crossbar array structure consisting of two levels of nanowires as the nanoPLA and NASIC designs. The main difference with CMOL is how the CMOS/nanodevices are interfaced. Pins are distributed over the circuit in a square array, on top of the CMOS stack to connect to either lower or upper nanowire levels. The nano crossbar is turned by some angle $< 90°$ relative to the CMOS pin array.

By activating two pairs of perpendicular CMOS lines, two pins, and the two nanowires they contact, are connected to the CMOS lines. Each nanodevice may be uniquely accessed using this approach. That is, each device may be switched ON/OFF by applying some voltage to the selected nanowires such that the total voltage applied to the device exceeds the switching threshold of the selected nanodevices. By angling the nanoarry the nanowires do not need to be precisely aligned with each other and the underlying CMOS layer in order to be able to uniquely access a nanodevice.

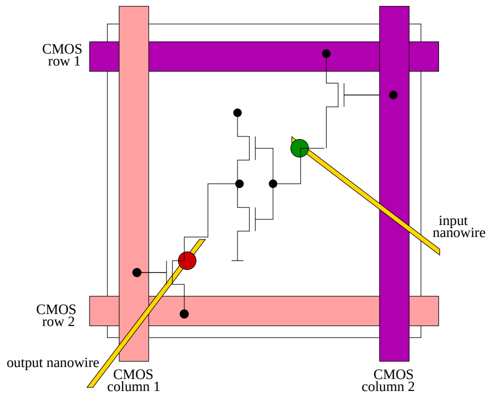

Fig. 1. The CMOS logic cell consisting of two pass transistors and in inverter (taken from [18]). The function is determined by how the overlaying nanowires are programmed. Note that typically, there are many nanowires (and thus many nanodevices) available per CMOS cell.

The most straightforward application of CMOL would be for memories (embedded or stand alone). The authors project that a CMOL based memory chip about 2×2 cm^2 in size will be able to store about 1Tb (Terabits) of data [17]. To improve reliability of the memory array, the authors propose adding spare lines and error correction codes (ECC), a standard procedure in memory array design, to improve yield.

The CMOL circuits have also been proposed for building FPGA-like architectures for implementing random logic [18]. A CMOS cell comprised of an inverter and two pass transistors are connected to the nanowire crossbar via two pins, as shown in Figure 1. This essentially creates a configurable logic block (CLB) structure, similar to that found in an FPGA. The CMOS cell is then programmed by disabling the inverter and selectively switching devices ON in the crossbar array. After configuration, the pass transistors act as pull-down resistors while the nanodevices programmed to be in the ON state serve as pull-up resistors. In this way wire-NOR gates may be formed within a CMOS cell. Note that the inverter provides signal restoration. Any arbitrary Boolean function (represented as a product-of-sums) may be implemented as a connection of two or more CMOS cells. Further, the idea is to have many nanodevices per CMOS cell. This allows gates with high fanin and/or high fanout to be formed, with extra devices available as "spares" for reconfiguring around faulty devices. Their Monte Carlo simulations of a 32-bit adder and a 64-bit full crossbar switch have shown that the reconfiguration allows the circuit yield to be increased to above 99% when the fraction of bad nanodevices is above 20%.

IV. Design Strategies for Reliable Computation

Reliable computation requires systems to be resilient to both static errors (e.g., static device defects), and transient errors (e.g., those arising from noise and signal coupling and alpha particle emissions, leading to single-event upsets (SEUs)). Of the example nanoscale systems described in the previous section, the nanoPLA and CMOL architectures are designed to handle only static errors, while the NASIC

WISP-0 architecture is designed to handle both static and transient errors. Traditional design strategies for handling transient errors included redundant execution, error-correcting codes, and dynamic reconfiguration. In this section, we briefly review these techniques and provide some additional examples of nanosystem designs that employ one or more of these techniques. A more detailed discussion on this topic can be found in [19].

Triple modular redundancy (TMR), or more generally N-modular redundancy (NMR), was first proposed by von Neumann as a method for building reliable systems from unreliable components (which was the prevalent concern for computing systems of his time) [20]. NMR is based on the idea of executing N copies of the same hardware modules, where N is odd and $N \geq 3$ and then taking the outputs and feeding them into a majority voter to determine the most likely correct value at the output of the circuit. The modules can be at a single gate, logical block or functional unit depending on the amount of error tolerance required in the system. While NMR provides relief from errors caused in a single unit, errors in more than $\lfloor \frac{N}{2} \rfloor$ modules will cause the logic to fail. In order to allow the circuit to withstand more errors, the NMR process can be repeated by taking each NMR module as a single module, making N copies and then combining the outputs of the individual NMR units with another majority gate. This method of redundancy is called cascaded NMR (CNMR). An NMR module can be considered to be a 0^{th} order CNMR. While NMR and CNMR greatly increase the reliability of the system in the presence of transient errors, the main assumption made in these redundancy processes is that the *final* majority voting gate is free from the same failures that the rest of the circuitry is facing.

The nanocomputing community has been interested in N-modular redundancy techniques because they can provide high reliability even in the presence of high device failure or soft error rates. Although the redundancy overhead can be very high, nano-assembly techniques promise to produce devices with such high density that the redundancy requirements become reasonable. Note that there may be some assumptions made on the reliability of the majority voter itself in order to improve the reliability of the whole system. Although redundant voters may be used to improve reliability, it comes at a cost of even greater redundancy overhead.

As an alternative to fully replicating hardware, error coding adds extra encoded bits in the hardware to help distinguish error-free from erroneous data. Error-correcting codes (ECC) are commonly used in storage arrays and communication. In addition to hardware needed to store the coding bits, extra hardware is needed for encoding, decoding, and correcting; however, in general, this overhead will be less than that required for full replication (e.g., when implementing a TMR scheme). Note that there is an implicit assumption that this extra encoding/decoding/correcting hardware is reliable, which may be a significant issue when applying this to nanocomputing.

The reconfigurable architectures described in Section III all required some means of testing the components and creating a defect "map" or database to avoid programming a circuit onto faulty devices. In general, the reconfiguration is done up front, in the initial programming stage, and therefore is set up for handling only static faults. In addition, the defect map may require a non-negligible amount of area to store. On the other hand, if dynamic reconfiguration is possible, then the system may be to handle dynamic faults, as well as static faults that cannot be located easily during an initial test stage.

An example of an architecture that uses dynamic reconfiguration is the Cell Matrix [21]. Instead of using a static defect discovery process, the Cell Matrix (CM) design uses dynamic defect/fault discovery and recovery. The idea is to design a distributed, parallel system such that failure detection is a hierarchical set of increasingly simple, local tasks run while the system is running. The Cell Matrix is a fine-grained reconfigurable fabric composed of simple homogeneous cells and nearest-neighbor interconnects. Cells can be configured to operate in either *data* or *configuration* mode. With this set up, a collection of cells can monitor neighbor's activities, detect erroneous behavior, disable defective cells, or relocate damaged portions of the circuit to other locations. While the CM architecture allows for autonomous, self-repairing circuits constructed from simple, locally interconnected homogeneous fabric, the dynamic reconfigurability adds another level of complexity to the design and also requires a high overhead in interconnection and control.

One example of an architecture that uses error codes, triple modular redundancy and reconfiguration is the *Recursive NanoBox* [22]. The *NanoBox* is a generic black box that employs a specific fault-tolerant technique for any computation done within the box. The NanoBox Processor Grid uses a hierarchical organization of these boxes such that different fault-tolerant techniques can be used at the bit, module, and system levels. At the bit level, error correction techniques are implemented by using FPGA-style lookup tables, while at the module or system level, redundancy is used by computing an operation multiple time across different cells. At the bit level, the error detection/correction scheme can vary from stored check bits, to triple modular redundancy. At the next level of hierarchy, simple ALUs coupled with memory and a communication router are constructed from the fault-tolerant lookup tables to create a processor cell. A single instruction will be executed multiple times on the processor cells (using either space or time redundancy) and results are fed into a majority voter to determine the output. A processor cell may be permanently disabled if it has exceeded it error threshold. At the highest level, the NanoBox Processor Grid consists of a 2D grid of processor cells. Their experiments showed that using redundancy at both the bit and module levels incurred a 9X area overhead; however, their simulation results showed that they can achieve 100% correct computation when operating at FIT rates above 10^{23}. While their analysis assumed that logic for error detection/correction and voters were error free, this is still an impressive result.

V. Using a Probabilistic Computing Paradigm

As discussed in the previous section, if a system is susceptible to high rates of transient errors, then reconfiguration may be rather costly in terms of added complexity or high interconnection and control overhead. Furthermore, applying N- modular redundancy may lead to an extremely high device overhead and requires some restrictions on the reliability of the majority voters. Instead, to more effectively protect against transient errors, a different computing paradigm may be needed for reliable operation of logic circuits and architectures. In this section, we explore the use of probabilistic computing based on principles of Markov Random Fields [23]. Under this framework, logic states are considered to be random variables, and one no longer expects a correct logic signal at all nodes at all times, but only that the joint probability distribution of signal values has the highest likelihood for valid logic states. Markov Random Fields have been used extensively in other areas outside of logic design, most notably pattern recognition and communication, as a way of effectively handling fault-prone data. Its successful application to such problems has led to the exploration of its application to logic design as well [23], [24].

The Markov Random Field defines a set of random variables that can each take on various values and interact with other similar random variables in a finite neighborhood [25]. Logic circuits can be expressed in terms of such neighborhoods and the interaction of the logic states and variables can be represented as a dependence graph. The crucial factor for probabilistic circuit design is that the full set of nodes (logic variables) in the circuit can be factored into a product of joint probabilities in the set of cliques that describe the local interactions.

There are two key requirements for the MRF model to be mapped onto a CMOS circuit:

- Each logic state, X_i, should be represented as a *bistable storage element*, taking on logical values of “0” an “1” with equal probability. The probability for any other signal value should be low.
- The constraints of each logic graph clique should be *enforced by feedback* to the appropriate storage elements, implementing the logic compatibility functions to maximize the joint probability of the correct logical values.

The first requirement insures that the MRF logic states are maintained so that the conditional probabilities among the neighboring elements can propagate. The feedback paths, required by the second design principle, are based on conditional probabilities and insure that the correct logic state is the most probable state.

Consider, for example, a NAND gate. The constraints for the gate can be expressed using the following logic compatibility function:

$$U_c(x_0, x_1, x_2) = x_0'x_1'x_2 + x_0'x_1x_2 + x_0x_1'x_2 + x_0x_1x_2', \quad (1)$$

or more concisely as:

$$U_c(x_0, x_1, x_2) = (x_0' + x_1')x_2 + x_0x_1x_2' \quad (2)$$

Using this factored form given in Equation 2, a mapping of the NAND gate can be created, as shown in Figure 2. The mapping consists of an OAI (OR-AND-INV) gate implementing the first term $(x_0' + x_1')x_2$ and a 3-input static CMOS NAND gate implementing the second term $x_0x_1x_2'$. Note that the correct output values on x_2 and x_2' are always reinforced for every possible input assignment.

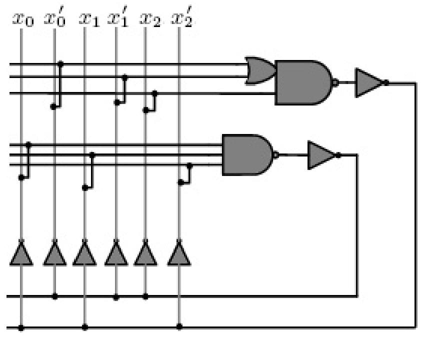

Fig. 2. An MRF NAND gate implementation. The inputs are x_0 and x_1, the output is x_2.

This new implementation of a NAND was simulated in SPICE using the 70 nm Berkeley predictive technology model [26] at $V_{DD} = 0.15V$ [27]. Operating the circuit at sub-threshold voltages allowed the effects of noise on the transient error rate to be better evaluated. They found that their MRF-inspired circuits were very resilient to noise, whereas circuits designed using standard CMOS logic gates experienced very high error rates. Being able to operate correctly at ultra low voltages can also provide a significant power advantage. Their preliminary results estimate an average 33% savings in power dissipation using this approach [27].

This MRF-inspired approach has also been applied to handle soft errors and single event upsets in sequential logic. The approach has some similarities with TMR, in that state is replicated three times. However, instead of using a majority voting mechanism to determine the correct state value, feedback is used to reinforce the value of the dominant state assignment using MRF-inspired circuitry. In this way, the scheme allows the entire circuit to be resilient to soft errors, single event upsets, and other sources of signal noise. This approach has also been extended to circuits that use error-correcting codes to protect data. The MRF ECC differs from traditional ECC in two fundamental ways: (i) both data and parity bits are treated equally in the MRF graph node, and (ii) error detection and correction is done naturally in the system without explicit decoding [28]. They estimate that the MRF approach for a (6,3) Hamming decoder is comparable to a traditional ECC implementation in CMOS in terms of transistor count. Furthermore, their simulations estimate that for the traditional Hamming decoder to operate reliably, it would have to dissipate 12X more power compared to the MRF technique [27].

VI. Future Directions in Nanocomputing

Emerging nanoscale technologies will enable extremely high levels of devices to be integrated onto a single substrate.

These nanoscale devices may consist of nanowire, carbon nanotubes, single electron transistors, or even CMOS devices shrunk down to ultimate dimensions. Regardless of which devices may be used to design these nanoscale systems, some sort of fault tolerance will have to be built into the designs at multiple levels in order to guarantee reliable computation. Essentially, reliability needs to become one more constraint added to the optimization equation, along with area, performance, power, and cost.

In this paper, we have reviewed a few hybrid schemes that combine CMOS and nanoscale devices. While mixing reliable elements among unreliable devices appears to have some advantages, there are still issues that need to be explored in this area. For instance, how do you best integrate alternative nanodevices into CMOS technology? What kind of constraints will these hybrid designs place on the architecture designs? Will random technology layouts become less desirable as a fabric for handling defective devices? Also, how will interconnect and memory bottlenecks limit the ability to handle high fault and defect rates?

As we have discussed, design strategies for reliable computation require various means of redundancy. What is not yet clear is at what point will a designer be better off using reliable, micro-scale devices rather than unreliable nanoscale devices that require high levels of redundancy?

Computation with nanoscale devices implies computing close to the thermal limit, where, computation becomes probabilistic in nature. New computational paradigms need to be developed that take probabilistic behavior into account.

There are many other issues related to reliability of nanoscale computing that were not discussed in this paper. Below, we touch on some of these issues and discuss how they may influence future directions in nanocomputing.

Asynchronous circuits and self-timed or clockless logics have been tried with CMOS technology, but are generally not found in mainstream architecture designs. Asynchronous computation may have a larger role in reliable computation since these designs may simplify global communication and power issues.

More realistic assumptions should be made about the nature of faults for these new devices, and how the faults may manifest themselves in the logic or system behavior. These new fault models may change theoretical results as well as the way reliability estimation is done for these nanoscale systems.

In order to aid the architecture, software for fault management also needs to be developed. The software should have some awareness of the faults in the system and should also have some part in managing the error rates, testing, and recovery schemes. For instance, the software may be responsible for driving the architectural reconfiguration when faults are detected. Programming languages that map high-level program descriptions onto nanoscale devices also need to be developed.

REFERENCES

[1] G. K. Celler and S. Cristoloveanu, "Frontiers of silicon-on-insulator," *Journal of Applied Physics*, vol. 93, pp. 4955–4978, May 2003.

[2] S. Luryi, J. M. Xu, and A. Z. eds., *Future Trends in Microelectronics: The Nano, the Giga, and the Ultra*. New York: Wiley, 2004.

[3] H. S. P. Wong, "Beyond the conventional transistor." *IBM Journal of Research and Development*, vol. 46, no. 2-3, pp. 133–168, 2002.

[4] H. Iwai, *The future of CMOS downscaling*. New York: Wiley, 2004, ch. in: S. Luryi, J. M. Xu, and A. Zaslavsky, eds., *Future Trends in Microelectronics: The Nano, the Giga, and the Ultra*, pp. 23–33.

[5] M. Butts, A. DeHon, and S. Goldstein, "Molecular electronics: Devices, systems and tools for gigagate, gigachips," in *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, November 2002, pp. 433–440.

[6] The latest 2004 update to the ITRS is available at http://www.public.itrs.net.

[7] A. DeHon and M. J. Wilson, "Nanowire-based sublithographic programmable logic arrays," in *International Symposium on Field-Programmable Gate Arrays*, February 2004, pp. 123–132.

[8] A. DeHon, "Design of programmable interconnect for sublithographic programmable logic arrays," in *International Symposium on Field-Programmable Gate Arrays*, February 2005, pp. 127–137.

[9] A. DeHon, P. Lincoln, and J. Savage, "Stochastic assembly of sublithographic nanoscale interfaces," *IEEE Transactions on Nanotechnology*, pp. 165–174, 2003.

[10] A. DeHon and H. Naeimi, "Seven strategies for tolerating highly defective fabrication," *IEEE Design and Test of Computers*, vol. 22, no. 4, pp. 306–315, July-August 2005.

[11] A. DeHon, "Nanowire-based progammable architecture," *ACM Journal on Emerging Technologies in Computing Systems*, 2005.

[12] T. Wang, Z. Qi, and C. A. Moritz, "Opportunities and challenges in application-tuned circutis and architectures based on nanodevices," in *First ACM Conference on Computeirn Frontier*, New York, NY, 2004, pp. 503–511.

[13] T. Wang, M. Ben-Naser, Y. Guo, and C. A. Moritz, "Wire-streaming processor on 2-D nanowire fabrics," in *NSTI (Nano Science and Technology Institute) Nanotech 2005*, California, May 2005.

[14] C. A. Moritz and T. Wang, "Towards defect-tolerant nanoscale architectures," in *IEEE Nano2006*, 2006.

[15] T. Wang, M. BenNaser, Y. Guo, and C. A. Moritz, "Self-healing wire streaming processor on 2-D semiconductor nanowire fabrics," in *NSTI (Nano Science and Technology Institute) Nanotech 2006*, May 2006.

[16] K. K. Likharev and D. B. Strukov, *Introducing Molecular Electronics*. Springer, 2005, ch. 16: CMOL: Devices, circuits, and architectures.

[17] X. Ma, D. B. Strukov, J. H. Lee, and K. K. Likharev, "Afterlife for silicon: Cmol circuit architectures," *IEEE Transactions on Nanotechnology*, 2005.

[18] D. B. Strukov and K. K. Likharev, "CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, no. 16, pp. 888–900, 2005.

[19] K. Nikolić, A. Sadek, and M. Forshaw, "Fault-tolerant techniques for nanocomputers," *Nanotechnology*, vol. 13, no. 3, pp. 357–362, 2002.

[20] J. von Neumann, *Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components*, ser. Automata Studies. Princeton University Press, 1956.

[21] L. Durbeck and N. Macias, "The cell matrix: An architecture for nanocomputing," *Nanotechnology*, pp. 217–230, 2001.

[22] A. KleinOsowski, K. KleinOsowski, V. Rangarajan, P. Ranganath, and D. J. Lilja, "The recursive nanobox processor grid: A reliable system architecture for unreliable nanotechnology devices," in *International Conference on Dependable Systems and Networks*, June 2004.

[23] R. I. Bahar, J. Mundy, and J. Chen, "A probabilistic-based design methodology for nanoscale computation," in *Proceedings of International Conference on Computer Aided Design*, Nov. 2003.

[24] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," in *Proceedings of Design Automation Conference*, June 2005.

[25] S. Z. Li, *Markov Random Field Modeling in Computer Vision*. Berlin: Springer - Verlag, 1995.

[26] Available at http://www-device.eecs.berkeley.edu/∼ptm/.

[27] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Optimizing noise-immune nanoscale circuits using principles of markov random fields," in *Proceedings of Great Lakes Symposium on VLSI*, April 2006.

[28] ——, "Designing MRF based error correcting circuits for memory elements," in *Proceedings of Design Automation and Test in Europe*, March 2006.