On Teaching Circuit Reliability

Azam Beg and Walid Ibrahim
College of Information Technology, UAE University
{abeg, walidibr}@uaeu.ac.ae

Abstract - **Integrated circuits in the coming years are expected to be based on nano-scaled devices, such as single electron transistors, self-assembled DNA, carbon nano-tubes, and resonant tunnel diodes. Future designs based on such nano-devices will exhibit high integration densities, and might be either low power or fast switching but not both. Unfortunately, nano-devices suffer heavily from fabrication inconsistencies, and transient and permanent failures due to external causes. Therefore, circuit reliability will have to be added to the design space currently consisting of timing, area, and power. This also means that the reliability calculation/estimation could soon be an important topic in the undergraduate/graduate courses on circuit design. Probability transfer matrix numerical method has been used as an exact way to calculate the reliability of a circuit. Traditionally, the matrices are created manually, making the process tedious, time-consuming, and error-prone. This paper proposes an automatic tool (*AutoPTMate*) for generating ready-to-use MATLAB m-files for calculating the circuit's reliability. The tool allows users to significantly speed-up the reliability assessment of a large number of circuits. The first potential users of such a tool are members of academia and R&D community.**

Index Terms - Active learning, teaching tools, curricula, computer-aided design, nano-architecture, reliability, probability transfer matrix (PTM), fault tolerance

CHALLENGES FOR FUTURE CIRCUIT DESIGNERS

The development of ever-smaller devices brings promise for further improvement in the performance of future integrated circuits (ICs), yet also leads to several new technical challenges, including the need for architectures that reduce the uncertainty inherent to computations at very small scales [1]–[3]. In particular, as feature sizes are aggressively scaled, the design and manufacturing of ICs becomes more complex and inevitably introduces more defects. The devices' small sizes—and consequently the tiny amounts of energy required in their switching—make them susceptible to transient failures [4]. Unfortunately, architectures built from emerging nanodevices, such as resonant tunnel diodes (RTDs), single electron transistor (SET), carbon nanotubes (CNTs), silicon nanowires, molecular devices, etc., will be even more vulnerable to parameter variations, fabrication noise, and transient failures induced by environmental/external causes [2], [5]. Device failure rates are predicted to be as high as 10% in case of single-electron transistor (SET) [6] and going up to 30% in case of self-assembled DNA (SADNA) [7]. As a fresh example, [8] has reported defect rates of 60% for a 160-kilobit molecular electronic memory. Clearly, achieving 100% correctness at the system level using such devices (and their associated interconnects) will be not only outrageously expensive, but might be plainly impossible! Therefore, relaxing the requirement of 100% correctness for devices and interconnects might significantly reduce the costs of manufacturing, verification, and test. This will lead to more transient and permanent failures of signals, logic values, devices, and interconnects. Therefore, most (if not all) of these failures will have to be compensated by architectural level techniques (e.g., redundancy in space, time, and information) [9]-[15].

An accurate calculation of the reliability of nano-architectural circuits through simulations will become essential for future designs. It will allow designers to estimate the yield and the behavior of their designs under different operating conditions. It is also crucial in designing/selecting the most suitable (nano) architecture that optimally makes a trade-off among all the delay, power, area, and reliability requirements.

In Section 2, we present some ideas about a course on circuit reliability (for example, target audience, delivery method, topics, etc.) Section 3 delves into the details of *AutoPTMate* (pronounced as *auto-P-T-mate*), a tool – for use by designers and students – that aids in reliability evaluation of circuits. A complete example on PTM methodology is also included. And finally, we present the conclusions.

TEACHING NANO-CIRCUIT RELIABILITY – STRUCTURE AND CONTENTS OF A COURSE

Circuit reliability calculation (estimation) could soon be an important part of VLSI courses; the students will have to consider reliability as a new design pillar. Reliability-driven digital design can be taught both in undergraduate and graduate level classes. An undergraduate class may only cover reliability as one of the topics in a course on digital design; whereas a complete graduate level course can be offered on the reliability of sub-10 nm designs. The students learning nano-circuit reliability may also come from industry/R&D background.

The teaching format of a nano-circuit reliability class may differ among different audiences. For example, the topic can be a part of a university level digital design course or it can be offered as a stand-alone course; a university

course can proceed at a moderate pace with some take-home assignments and/or projects. Whereas, for industrial users, a semester-long format may not be suitable, so the course could be limited to duration of 5 days or less; conflict with work schedule or urgent needs may still interfere with the course progression, although an off-site class may alleviate some disruptions. Such students would like to see practical information that can be readily applied in work situations [14]. They may prefer some methodologies that are applicable to the work-related situations. Unlike academic students, the practicing engineers would rather not be taught theory in depth. Any managers attending the course may just be interested in the overall picture and the upcoming trends, instead of detailed methods [16].

Both in academic and industrial environments, emphasis should be placed on the course offering being interactive. The university students' background and hands-on experience (if any) is not as varied as the practitioners; level of prior knowledge needs to be taken into account. Some take-away material or a carefully selected book is obviously quite helpful for future reference.

Normally, an academic student may design a circuit, and run simulations to verify its functionality; however, the *correctness* of his/her design does not warrant that the design is fulfilling its reliability requirements. A common approach taken by students while working on small projects is designing-before-thinking. This method may work fine in the academic environment, but may not help the students learn to tackle the real-life design tasks of much larger sizes. The students, therefore, need to be taught a pro-active frame of mind of problem solving [16] [17].

While developing reliable hardware, one can borrow lifecycle techniques from the software realm: prevention, removal, tolerance, and prediction. As mentioned before, the reliability issues in nanometric circuits are bound to happen, so it will become quite critical to estimate and predict them. *Reliability-driven* design methodology could begin by first establishing reliability goals, and then by the meeting other constraints such as timing and power/area. An example would be Tosun *et al*'s [17] methodology that searches through a library of design alternatives in order to yield a design with best reliability figures. Their framework first finds the most reliable configuration and then tries to meet the area/power and timing constraints.

We propose that the course on reliability-driven circuit design include this general design methodology: design definition, setting criteria for optimal reliability/power/timing, design/reliability test vector generation, design implementation, test execution; and design modifications, if needed. The following is a proposed outline of a (graduate-level) course offered on nano-scaled devices and their associated reliability [21][22]:

1. Review of CMOS devices; and device feature reduction and its implications.
2. Nanometric devices and models: for example, resonant tunneling devices, quantum devices, solid-state nano-electronic devices, SETs, CNTs, etc.
3. Nanoscale architecture models and designs: crossbar, CMOS molecular (CMOL), quantam cellular automata (QCA), etc.
4. Interconnection network topologies: crossbar, hyper-mesh, cube-connected cycle, spanning bus hypercube, etc.
5. Elements of reliability: theory, statistics, characterization, etc.
6. Applicability of reliability: identifying unreliable/fault-prone components/sub-circuits, computing reliability, fault-tolerance techniques, and related automated/CAD tools, etc.

AUTOPTMATE – A TOOL FOR RELIABILITY EVALUATION

Probability transfer matrix (PTM) is a powerful modeling tool for problems involving uncertainty [19]. (The methodology falls under item 5 listed above). The PTMs can be used to evaluate a circuit's overall reliability by combining the PTMs of elementary gates. A PTM performs simultaneous computation over all possible input combinations, and calculates the exact probabilities of errors. Another advantage (besides being exact) is that it is very easy to use different probabilities of failures for different gates. However, the circuit's overall probability of failure requires dividing the circuit manually into several stages (as shown in Figure 1), generating the PTMs for the individual stages, and finally combining all the PTMs to construct the circuit's overall PTM. This process is fairly simple for small circuits, but becomes quite intricate and error-prone as the circuit size increases. If performed manually, the PTM method can limit the number of circuits, a user may design and analyze during a limited amount of time.

In this paper, we introduce *AutoPTMate* (pronounced as *auto-P-T-mate*), a tool that fully automates the PTM methodology. The current version of the tool outputs a set of MATLAB commands included in a single m-file. The user simply needs to run this file to calculate the circuit overall probability of failure.

In the PTM methodology, each type of gate has its unique matrix. For example, a PTM that represents a 2-input NAND gate is shown in Figure 2; *pf* represents the probability of incorrect output for any given input, also called *probability of failure.* The output values are shown on the left and the inputs on the top of the matrix. The probability of each output value is given for each input set in the matrix. By combining PTMs that represent the probability of failure of the individual gates and connection wires, one can determine a circuit's overall probability of failure. Gates and sub-circuits are combined using the following three rules [20]:

1. If two gates (or sub-circuits) with PTM_1 and PTM_2 are connected in series, then their combined PTM is the product of the individual gates (or sub-circuits) PTMs:

$$PTM_s = PTM_1 \times PTM_2 \quad (1)$$

2. If two gates (or sub-circuits) with PTM_1 and PTM_2 are connected in parallel, then their combined PTM is the Kronecker (tensor) product of the individual gates (or sub-circuits):

$$PTM_p = PTM_1 \otimes PTM_2 \quad (2)$$

3. If two gates (or sub-circuits) are connected with fan-out (an output of one gate/sub-circuit is connected to more than one input of the following gates (or sub-circuits), the combined PTM is constructed by calculating the Kronecker product of the two gates (or sub-circuits) and eliminating all the columns that have different values for the fan-out inputs.

The first step toward the calculation of the circuit's probability of failure is to divide the circuit into a series of *stages* (*stg*1-4 in Figure 1). Stages can be classified into a *computing stage*, a *connecting stage*, or a mix of computing and connecting stages. A connecting stage contains only a group of connecting wires (or simply connections). Connecting stages prepare the inputs for the following computing stage. Connecting stages can be used to augment the number of connections using the fan-out rule. A separate PTM is calculated for each stage using the previously described rules. The PTM for the complete circuit is then calculated by successive multiplication of PTMs of individual stages (starting from the input stage):

$$PTM_{S1\text{-}2} = PTM_{S2} \times PTM_{S1}, \quad (3)$$

$$PTM_{S1\text{-}3} = PTM_{S3} \times PTM_{S1\text{-}2}, \ldots, \quad (4)$$

$$PTM_{S1\text{-}n} = PTM_{Sn} \times PTM_{Sn\text{-}1} = \prod_{i=n}^{1} PTM_{Si} \ . \quad (5)$$

In the next section, we include a detailed example (based on the circuit of Figure 1) to demonstrate the full process of evaluating the probability of failure.

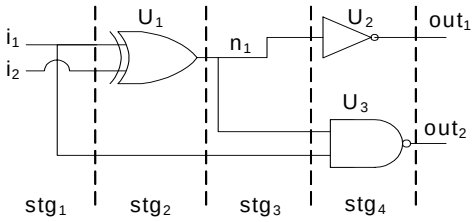

FIGURE 1
A CIRCUIT SPLIT INTO *STAGES* IN ORDER TO CREATE EACH STAGE'S OWN PTM.

$$\begin{array}{cc} & \text{Input} \\ & \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \text{Output}\begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} pf & pf & pf & (1\text{-}pf) \\ (1\text{-}pf) & (1\text{-}pf) & (1\text{-}pf) & pf \end{bmatrix} \end{array}$$

FIGURE 2
PTM FOR A 2-INPUT NAND GATE WITH *pf* PROBABILITY OF FAILURE.

AUTOPTMATE DESCRIPTION AND USAGE

This section describes the *AutoPTMate* tool (using the circuit in Figure 1). The primary usage of the tool is to calculate the overall probability of failure of a given circuit (pf_{CIRC}) using the probability of failure of its individual gates (pf_{GATE}). The tool is developed in the popular scripting language PERL, which makes it fully portable and flexible, so that it can easily run on different host systems, e.g., Windows, Linux, etc. Obviously, no compilation is required, and also no installation is required (as long as a recent version of PERL is available on the host system). The input to the tool is a Verilog file in structural format. (Refer to the two sample files in Appendix A). The tool's output is a MATLAB m-file; MATLAB has been chosen for its efficient matrix handling capabilities. To calculate pf_{CIRC}, the user needs to simply run the generated m-file using a pre-installed MATLAB package. The syntax for running *AutoPTMate* from the command-line follows:

```
perl autoPTMate.pl -f filename -pf pfnand
```

As an example, the command line for creating an m-file that has pf_{GATE} (for NAND gate) ranging from 0 to 0.05, would be like this:

```
perl autoPTMate.pl -f eg1.v -pfnand 0:0.005:0.05
```

A graphical overview of *AutoPTMate* is shown in Figure 3. The tool first scans the circuit description (presented by the input Verilog file) to identify the inputs, outputs, and nets, by parsing the lines starting with keywords *input*, *output*, and *wire*. (The Verilog file corresponding to the circuit in Figure 1 is included in Appendix A).

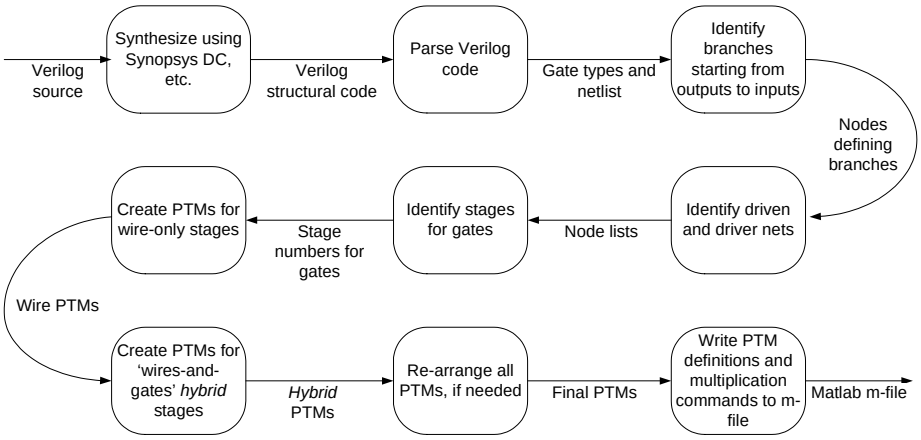

FIGURE 3
THE STEPS *AUTOPTMATE* TAKES TO GENERATE THE M-FILES FROM VERILOG FILES.

In the following step, the tool identifies the gates and their types from the gate (or sub-circuit) instantiations. The tool then builds a complete list of *branches* by starting at output nodes and back-tracking towards input nodes. For example, the circuit of Figure 1 has the following four branches: out_1-n_1-i_1; out_2-n_1-i_1; out_2-i_1; out_1-n_1-i_2; out_2-n_1-i_2. These branches help the tool divide up the circuit into different stages; in other words, the gates belonging to each stage are now known. In medium and large size circuits, the creation of the PTMs corresponding to the wire stages is usually the most difficult and error-prone step toward the calculation of the pf_{CIRC}. This is because the wires usually intersect and cross over each other, which complicates the definition of the matrices that describe the probability of failure of the output signals in terms of the input signals. In order to overcome this difficulty, we introduce the concept of an *Input-Output Mapping Matrix* (IOMM). The IOMM describes the relationship between the input and output signals in a stage; the IOMM concept is utilized by the tool to generate the corresponding PTM.

A stage with c inputs signals and r outputs signals results in an IOMM of dimensions $r \times c$ (see Figure 4(a)). We assume that the wires are perfect, and therefore have $pf_{WIRE} = 0$. Any connection between an input and output is denoted by 1, while no connection is denoted by 0. In case of a fan-out of n, the count of 1's appearing in a column is n (see column i_1 in Figure 4(a)).

The next step involves creating PTMs for wires and gates in *hybrid* stages (*stg*2 in Figure 1) and gate-only stages (*stg4* in Figure 1). For the hybrid stages and gate-only stages, the tool creates MATLAB commands for instantiating Kronecker products, in order to determine the PTM for the whole stage. Finally, commands for equations such as (3), (4), and (5) are created, so that the circuit's overall PTM can be determined. Creation of the IOMM and the PTM corresponding to each stage is discussed next (starting with the wire-only stages, due to the simplicity of their PTM calculations).

stg1: This wire-only stage has IOMM shown in Figure 4(a). An IOMM is used to create the PTM which has $2^2 = 4$ input columns and $2^3 = 8$ rows (see Figure 4(b)). Notice how the fan-out of the wires affects the IOMM and its corresponding PTM.

stg3: This stage has a fan-out to increase the number of signals from 2 to 3. The corresponding IOMM and PTM are shown in Figure 5.

***stg2*:** This stage has a single 2-input XOR gate (U1) and a single wire. The PTM of the stage is generated by performing a Kronecker product of the PTMs corresponding to XOR gate and the wire. Since we assume that all the wires are perfect ($pf_{WIRE} = 0$), the PTM corresponding to a single wire is simply a 2×2 identity matrix. As widely known, if A is an $m \times n$ matrix and B is a $p \times q$ matrix, the Kronecker product of A and B is a matrix of dimensions $mp \times nq$. The size of the XOR gate's PTM is 2×4, while the size of the wire's PTM is 2×2. Therefore the PTM for this stage is a 4×8 matrix (Figure 6).

***stg4*:** This stage has an inverter (U2) and a 2-input NAND (U3) gate. The PTM corresponding to this stage is also obtained by performing a Kronecker product for the PTMs

	i_1	i_2
o_1	1	0
o_2	0	1
o_3	1	0

(A)

	00	01	10	11
000	1	0	0	0
001	0	0	0	0
010	0	1	0	0
011	0	0	0	0
100	0	0	0	0
101	0	0	1	0
110	0	0	0	0
111	0	0	0	1

(B)

FIGURE 4
THE IOMM AND PTM FOR STG1: (A) IOM; (B) PTM.

	i_1	i_2
o_1	1	0
o_2	1	0
o_3	0	1

(A)

	00	01	10	11
000	1	0	0	0
001	0	1	0	0
010	0	0	0	0
011	0	0	0	0
100	0	0	0	0
101	0	0	0	0
110	0	0	1	0
111	0	0	0	1

(B)

FIGURE 5. THE IOMM AND PTM FOR STG3: (A) IOM; (B) PTM.

corresponding to the inverter and the NAND gates. The PTM for this stage is shown in Figure 7.

To calculate the pf_{CIRC}, for a given pf_{GATE}, the user should run the tool twice. In the first run, $pf_{GATE} = 0$, while in the second run, the desired value of pf_{GATE} is used. These two runs produce two matrices. The next step would be to find the location of the value 1 in each column of the first matrix (see Figure 8(a)) and then sum up the corresponding values in the second matrix (i.e., $S = 0.85975 + 0.85975 + 0.8575 + 0.8575 = 3.4345$. Finally, pf_{CIRC} is calculated:

$$pf_{CIRC} = 1 - \frac{S}{n} \tag{6}$$

where n is the number of columns. pf_{CIRC} for the circuit (of Figure 1), when $pf_{GATE} = 0.05$, is: $1 - 3.4345 / 4 = 0.1414$.

To confirm the tool's correctness, we used it to generate the PTMs for several different circuits; and then we verified that the automatically generated PTMs are identical to the manually generated ones.

In order to demonstrate the insight one can quickly gain by using *AutoPTMate*, we used a somewhat larger circuit which has 4 gates spread over 8 stages; the circuit is shown in Figure 9 and its corresponding Verilog code is listed in Appendix A. In our analysis, we varied pf_{GATE} (from 0 to 0.5) for one gate-type at a time, while keeping pf_{GATE} for all other gate-types fixed at zero. Figure 10 shows the

	000	001	010	011	100	101	110	111
00	1	0	0	0	0	0	1	0
01	0	1	0	0	0	0	0	1
10	0	0	1	0	1	0	0	0
11	0	0	0	1	0	1	0	0

FIGURE 6
PTM FOR STG2 WHERE $PF = 0$.

	000	001	010	011	100	101	110	111
00	0	0	0	0	0	0	0	1
01	0	0	0	0	1	1	1	0
10	0	0	0	1	0	0	0	0
11	1	1	1	0	0	0	0	0

FIGURE 7
PTM FOR STG4 WHERE $PF = 0$.

	00	01	10	11
00	0	0	1	0
01	0	1	0	0
10	0	0	0	0
11	1	0	0	1

(A)

	00	01	10	11
00	0.00475	0.04525	0.8575	0.0475
01	0.09025	0.85975	0.0475	0.0475
10	0.04525	0.00475	0.0475	0.0475
11	0.85975	0.09025	0.0475	0.8575

(B)

FIGURE 8
THE CIRCUIT'S OVERALL PTM FOR: (A) $pf = 0$; (B) $pf = 0.05$.

cumulative results from multiple runs of the tool. In the figure, we can see that the circuit's reliability is the most affected by the NOR gate, whereas the circuit reliability is the least vulnerable to the inverter. This type of information can help a circuit designer locate and *harden* the gates/parts of a circuit that are the most susceptible to failure.

CONCLUSION

Traditionally, reliability calculations using PTM methodology to create the PTM and all related commands

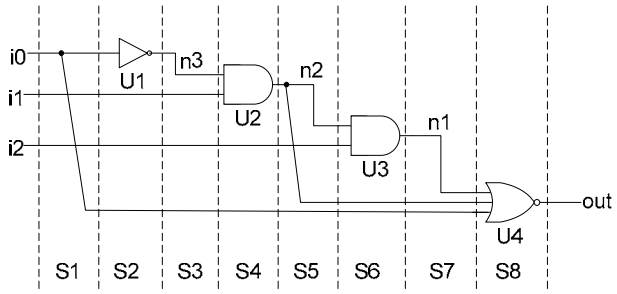

FIGURE 9
A SAMPLE CIRCUIT WITH SEVERAL *STAGES*.

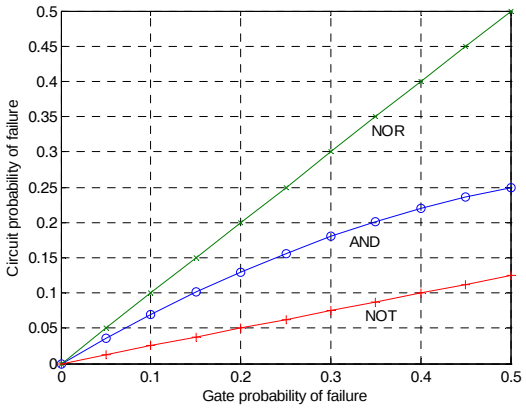

FIGURE 10
EFFECT OF VARYING A SINGLE GATE'S *pf* WHILE ALL OTHERS ARE KEPT CONSTANT AT ZERO, FOR CIRCUIT IN FIGURE 9. THE CIRCUIT'S *pf* IS THE MOST AFFECTED BY THE NOR GATE WHEREAS THE NOT GATE AFFECTS THE CIRCUIT-*pf* THE LEAST.

have been performed manually, which is both a time-consuming and an error-prone process. The proposed *AutoPTMate* simplifies the task of reliability calculations by automatically generating MATLAB m-files that include all the command to calculate the circuit's PTM. The tool allows the users to significantly speed-up the reliability assessment of a large number of circuits. Potential users of such a tool are members of academia and R&D community alike.

We have started to use *AutoPTMate* to generate the PTMs for other larger and more complex circuits. The results from our ongoing experiments will be reported in the future. We are also planning to enhance the tool so that it helps create mathematical models for circuit reliability.

APPENDIX A

Verilog code for the circuit of Figure 1:

```
module EG1 (out1, out2, i1, i2);
  input  i1, i2;
  output out1, out2;
  wire   n1;
  xor2_1x  U1 (.B(i1), .A(i2), .Y(n1));
  not_1x   U2 (.A(n1), .Y(out1));
  nand2_1x U3 (.B(n1), .A(i1), .Y(out2));
endmodule
```

Verilog code for the circuit of Figure 9:

```
module EG2 (out, i0, i1, i2);
  input i0, i1, i2;
  output out;
  wire   n1, n2, n3;
  inv_1x U1 (.A(i0), .Y(n3) );
  and_1x U2 (.A(n3), .B(i1), .Y(n2));
  and_1x U3 (.A(n2), .B(i2), .Y(n1));
  nor_1x U4 (.A(n1), .B(n2), .C(i0), .Y(out));
endmodule
```

REFERENCES

[1] G. Krishnaswamy, G. Viamontes, I. Markov, and J. P. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," *Proc. Des., Automation & Test in Europe Conf. & Exhib. (DATE 2005)*, Munich, pp. 282-287, Mar. 2005.

[2] Semiconductor Industry Association (SIA), *Intl. Tech. Roadmap for Semicond* (ITRS), SEMATECH, Austin, TX, USA, 2005 Edition and 2006 Update.

[3] J. D. Meindl, Q. Chen, and J.A. Davis, "Limits on silicon nanoelectronics for terascale integration," *Science*, vol. 293, 14 Sep. 2001, pp. 2044–2049.

[4] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, Jul.-Aug. 2003, pp. 14–19.

[5] P. Sivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on soft error rate of combinatorial logic," *Proc. Intl. Conf. Dependable Sys. & Networks DSN'02*, Washington, DC, USA, Jun. 23-26, 2002, pp. 389–398.

[6] K. K. Likharev, "Single-electron devices and their applications," *Proc. IEEE …*, vol. 87, Apr. 1999, pp. 606–632.

[7] U. Feldkamp, and C. M. Niemeyer, "Rational design of DNA nanoarch.," *Angew. Chem. Intl. Ed.*, vol. 45, 13 Mar. 2006, pp. 1856–1876.

[8] J. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu , Y. S. Shin, H. R. Tseng, J. F. Stoddart, and J. R. Heath, "A 160-kilobit molecular electronic memory patterned at 1011 bits per square centimeter," *Nature*, vol. 445, 25 Jan. 2007, pp. 414–417.

[9] K. Nicolić, A. Sadek, and M. Forshaw, "Arch. for reliable computing with unreliable nanodevices," *Proc. IEEE Conf. Nanotech. (IEEE-NANO'01)*, Maui, HI, USA, Oct. 2001, pp. 254–259.

[10] J. Dai, L. Wang, and F. Jain, "Analysis of Defect Tolerance in Molecular Electronics Using Information-Theoretic Measures," *IEEE Intl. Symp. on Nanoscale Arch. (NANOARCH 2007)*, San Jose, CA, USA, Oct. 2007, pp. 21-26.

[11] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation," *IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD 2007)*, San Jose, CA, USA, Nov. 2007, pp. 149-154.

[12] S. Krishnaswamy, S. M. Plaza, and J. P. Hayes, "Tracking Uncertainty with Probabilistic Logic Circuit Testing," *IEEE Design & Test of Computers*, Jul.-Aug., 2007, pp 312-321.

[13] H. Li, J. Mundy, W. Patterson, D. Kazazis, A. Zaslavsky, and R. I. Bahar, "Thermally-induced soft errors in nanoscale CMOS circuits," *IEEE Intl. Symp. on Nanoscale Arch. (NANOARCH 2007)*, San Jose, CA, USA, Oct. 2007, pp. 62 – 69.

[14] A.W. Mayers, S. K. Kurtz, "Teaching reliability eng. to working engineers," *Proc. 30th Annual Frontiers in Education Conf., (FIE 2000)*, Kansas City, MO, USA, Vol. 2, pp. F2E/14 - F2E/19

[15] F. Martorell, S. D. Cotofana, and A. Rubio, "An Analysis of Internal Parameter Variations Effects on Nanoscaled Gates," *IEEE Trans. on Nanotechnology*, vol. 7, No. 1, Jan. 2008, pp. 24-33.

[16] M. Garzia, J. Hudepohl, W. Snipes, M. Lyu, J. Musa C. Smidts, and L. Williams "How should software reliability eng. (SRE) be taught?" *ACM SIGSOFT Software Eng. Notes*, Vol. 31, Issue 4, Jul. 2006, pp. 1-5.

[17] M. R. Lyu, "Software Reliability Eng. : A Roadmap," *Intl. Conf. on Software Eng. (FOSE '07)*, May 2007, pp. 153-170.

[18] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W-L. Hung, "Reliability-Centric Hardware/Software Co-design," *Proc. of the 6th Intl. Symp. on Quality of Electronic Design (ISQED'05)*, San Jose, CA, USA, Mar. 2005, pp. 375 – 380.

[19] K. N. Patel, I. L. Markov, and J. P. Hayes, "Evaluating circuit reliability under probabilistic gate-level fault models," *Proc. Intl. Workshop Logic Synthesis (IWLS'03)*, Laguna Beach, CA, USA, May 2003, pp. 59–64.

[20] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," *Proc. Intl. Test Conf. (ITC 2003)*, Sept. 2003, Charlotte, NC, USA, pp. 893-901.

[21] D. Kim, R. Kamoua, and A. Pacelli, "Design-oriented introduction of nanotechnology into the electrical and computer eng. curriculum" *Journal of Educational Tech. Sys.*, Vol. 34, No. 2, 2005-2006, pp. 155-164.

[22] R. Kamoua, D. Kim, and G. Roach, "Incorporating nanoscale system design into the undergraduate electrical and computer eng. curriculum," *Proc. of 9th Intl. Conf. Eng. Educ.*, Jul. 2006, San Juan, Peurto Rico, pp. M4K4-M4K9.

AUTHOR INFORMATION

Azam Beg Assistant Professor, College of Information Technology, UAE University, Al-Ain, UAE, abeg@uaeu.ac.ae

Walid Ibrahim, Assistant Professor, College of Information Technology, UAE University, Al-Ain, UAE, walidibr@uaeu.ac.ae