IC Bridge Fault Modeling for IP Blocks Using Neural Network-Based VHDL Saboteurs

Donald B. Shaw, Member, IEEE, Dhamin Al-Khalili, Senior Member, IEEE, and Côme N. Rozon

Abstract—This paper presents a new bridge fault model, suitable for IP blocks, that is based on a multiple layer feedforward neural network and implemented within the framework of a VHDL saboteur cell. Empirical evidence and experimental results show that it satisfies a prescribed set of bridge fault model criteria better than any existing approach. The new model computes bridged node voltages and propagation delay times with due attention to surrounding circuit elements. This is especially significant since, with the exception of full analog defect simulation, no other technique even attempts to model the delay effects of bridge defects. Yet, compared to these analog simulations, the new approach is several orders of magnitude faster and, for a 0.35μm cell library, is able to compute bridged node voltages with an average error near 0.006 volts and propagation delay times with an average error near 14 ps.

Furthermore, dealing with a concept that has not previously been considered in related research, the new model is validated with respect to deep-submicron technologies for limited gate-count circuit modules.

Index Terms—Bridge defects, fault models, neural networks, VHDL, CMOS ICs, fault simulation, IP blocks.

1 INTRODUCTION

The advancement of integrated circuit (IC) technologies is proceeding at a swift pace with respect to speed, density, and design complexity. Accordingly, issues concerning manufacturer defect testing, device wearout, and fault secure circuit validation become increasingly challenging. One area of particular concern is the existence of fault models to study the effects of defects on IC designs [1]. This paper introduces a new bridge fault model that is appropriate for circuit modules and intellectual property (IP) blocks in CMOS ICs. For this discussion, a defect is any physical imperfection that may exist within a circuit. Defects lead to the occurrence of faults, which are defined as any type of abnormal circuit behavior, such as an incorrect logic level or increased signal delay. By judicious development and use of accurate fault models, based on actual defects, circuit simulation is used to reveal the relevant effects of the various defects.

It has been shown that bridge defects account for the majority of all defects in CMOS ICs [2], [3], [4]. A bridge defect is defined simply as a short between two normally unconnected nodes in a circuit. Considering that a high percentage of layout area is used by interconnect between cells, it has been found that the majority of bridge defects occur between the output signals of logic gates [4], [5], [6], [7]. These are the bridge defects considered in this paper.

The paper begins with a discussion about the fault effects of interconnect bridge defects. The scope is limited to CMOS standard cell-based circuits as these are currently prevalent in the IC industry. Then, existing techniques for modeling bridge defects are discussed with respect to their ability to model actual fault effects and simulation efficiency. A set of goals is established for developing accurate and efficient bridge fault models. This is followed by a description of a completely new bridge fault modeling scheme for CMOS standard cell designs. The model achieves nearly the accuracy of analog defect simulation using an efficient neural network implemented in VHDL. It is validated by comparing results with those obtained using analog simulation. Then, it is validated with respect to deep submicron (DSM) IC technologies for moderate size IP blocks using a series of specific analog simulations. Finally, implementation issues are addressed and some benchmark circuits are studied using the new model.

2 BRIDGE DEFECTS

2.1 Bridge Defect Effects

A bridge defect between two gate outputs appears dormant as long as the gates are driving the same logic value. However, when the two gates attempt to adopt different logic values, logic contention occurs. Depending on factors such as the drive strength of the two gates, their individual input patterns, and the characteristics of the bridge defect, the bridged node may adopt either logic value or settle at some intermediate voltage level. Also, in many cases, the bridge defect can have a significant impact on the propagation time of the bridged signals. This is particularly problematic when large parasitic delays are introduced due to the defect.

Bridge defects also cause less obvious effects that can introduce further modeling complications. If a bridge defect creates a feedback loop, a formerly stable combinational circuit may take on oscillatory or sequential properties. Also, when an intermediate voltage level occurs, downstream logic gates with varying input voltage thresholds...
Switch-level simulators have been used for modeling bridge defects as permanent conducting transistors at the defect site [18], [19], [20]. This is much more accurate than the permanent wired-logic model, but also has some serious limitations. For instance, inherently nonlinear transistors are treated as linear resistance elements. Also, timing degradation characteristics and intermediate voltages cannot be modeled using this approach. This can cause the Byzantine General’s Problem to appear, as well as produce an excessive and pessimistic number of unknown states for the bridged nodes. Finally, due to the added complexity of switch-level circuit netlists, simulations are prohibitively slow compared to gate-level.

More recent research has developed bridge models that compute the node value at the defect site as a Boolean function of the inputs driving the bridged wires. One such model, the voting model, determines the node value based on the number, type, size, and state of the transistors that are driving the bridged signals [17]. This model employs the AND/OR resolution functions only in cases where the pull-down/pull-up networks driving the shorted nodes always overpower the opposite network in the other cell. Similarly, it allows for a dominant driver function that always forces the bridged node to the value of the stronger gate, when such a condition exists. Otherwise, the driving gate inputs and a series of look-up tables determine the dominant network. The look-up tables are specific to a given cell library and built using data obtained from analog simulations prior to fault simulation. It is notable that, since the tables grow only linearly with the number of gates in the library, the analog simulation time and table size remain manageable. Additionally, the model includes provisions for the Byzantine General’s Problem and complex gate designs. However, this model suffers from a considerable weakness such that only shorts with negligible resistance (hard shorts) can be modeled. Thus, the performance degradation effects of soft shorts, such as increased propagation delay, cannot be simulated. Furthermore, the capacitive loading and resistance contributed by the interconnect is not considered.

A similar model, the Primitive Bridge Function (PBF), creates a general fault block that replaces the gates driving the bridged node as well as several downstream gates [21]. The PBF performs a mapping from the inputs of the driving gates to the outputs of the driven gates. The mapping can be determined using a variety of different approaches, including analog circuit simulation of the bridge defect and surrounding circuit components. A strength of this approach is that it can take into account critical parameters of the transistors driving the bridged node as well as the varying logic thresholds of downstream gates, thereby eliminating the Byzantine General’s Problem. Also, during the analog simulation phase, any electrical resistance can be set for the bridge defect, thus allowing hard and soft defects to be modeled. A significant drawback of this approach is that the number of cell combinations grows exponentially as new cells are added to the library, potentially requiring huge setup and simulation time requirements to derive the PBFs.

Mixed-mode simulators, capable of both analog and digital simulation on different portions of a circuit, have also
been investigated for modeling bridge defects [22], [23]. Using this approach, switch or gate-level simulation is conducted throughout most of the circuit, except in the region immediately surrounding the bridge defect. In this region, the simulator switches to the analog domain, where the electrical parameters of nearby transistors and the exact bridging resistance are accounted for. The only bridge fault model more accurate than a mixed-mode approach is full analog simulation, which is clearly impossible for most digital circuits. However, despite significant time-savings over full analog simulation, mixed-mode bridge fault simulation is still time consuming compared to other methods.

A totally different, yet interesting, approach to bridge modeling uses fuzzy logic processing to compute the effects of bridge defects on a circuit [24]. This approach begins by determining the voltage transfer characteristics for each cell in a given library using analog simulation and building a fuzzy model of it. Then, a resistive bridge defect is inserted into a test circuit and the voltage of the bridged nodes is computed using voltage division. If the voltage of the bridged nodes is in an intermediate range, it propagates through the subsequent gates using the fuzzy models until the voltage is within normal levels. This technique is efficient and introduces some novel ideas. However, the initial voltage computation of the bridged nodes, upon which subsequent calculations are based, has insufficient accuracy. Furthermore, no attempt is made to model delay effects of bridges and interconnect parasitics are completely neglected.

The most accurate bridge models considered so far all have in common the need for analog circuit simulation at some point in the modeling process. However, methods for achieving reasonable levels of accuracy without time-consuming analog simulations have also been proposed [25], [26]. In general, these methods make some simplifying assumptions about the circuit, then use first order equations to compute the voltage of the bridged node. The premise is that negligible accuracy is lost in determining the voltage of the bridged node, while simulation speed is significantly better than with commercial analog simulators. A drawback of this model is that there is no mechanism for computing timing characteristics for the bridged node, thus neglecting a common fault effect. Furthermore, such an approach is clearly inaccurate for use with deep submicron technologies.

### 2.3 Bridge Fault Model Goals

After analysis of existing approaches for bridge fault modeling, several issues are apparent. First, the resistive properties of bridge defects are often neglected, except in those models that are based entirely on analog simulation in the defect region. Thus, the degraded signal and propagation delay effects of soft bridge defects are virtually ignored. Also, several of the models totally neglect the existence of the Byzantine General’s Problem. Equally disturbing is that none of the models have been validated for interconnects of various length. This totally ignores the significant parasitic resistance and capacitance of the interconnects in DSM technologies.

An accurate bridge fault model must consider the effects of resistive bridges at various levels of severity. Then, it can determine the exact voltage of the two bridged nodes with due consideration to the resistance and the input-pattern dependent drive strength of the cells driving those nodes. Furthermore, the model must consider the Byzantine General’s problem in the logic interpretation of the node voltages. It is also important to model the delay effects of bridge defects with consideration to the load characteristics and other relevant factors. For bridge fault modeling in deep submicron circuits, the length and geometry of the bridged interconnect lines must be considered as well as the position of the defect along the interconnect. Finally, the bridge fault model must be efficient to allow for analysis of realistic size circuit modules within a reasonable amount of processing time.

For any bridge modeling scheme, it is clear that there is a trade off between model efficiency and accuracy. Only analog circuit simulators provide sufficient accuracy, while only digital simulators have the speed to analyze realistic circuits. Thus, a hybrid approach to bridge fault modeling is required. The next section presents such an approach, one that considers all of the desired characteristics of accurate bridge fault models.

### 3 A NEW BRIDGE FAULT MODEL

#### 3.1 VHDL Fault Models—The Saboteur

Considering the prevalence of hardware description languages throughout all phases of the IC design process, fault modeling with VHDL is a compelling idea that has been evolving steadily. A popular technique for VHDL fault modeling is based on the saboteur, which is a controllable component that is physically added to the VHDL netlist of a design [27]. It is placed on the nets between existing components and can be used to model a wide variety of fault cases. For instance, a saboteur could cause the output of the XOR gate to appear as a stuck-at fault to subsequent circuit elements. Alternatively, additional delays or reduced voltage swings could be modeled. The saboteur can also be used to model bridge defects between two interconnects by placing it across them. The saboteur component is implemented using a behavioral VHDL description that completely defines its output response based on the defect control state and input pattern sequence.

A significant drawback of the saboteur model as originally proposed in [27] is that it has no direct access to the input ports of the preceding gates and, therefore, cannot model faults that are input pattern dependent or determine the drive strength of the signal. Thus, a modification is proposed such that additional signal lines are added to the circuit to enable the saboteur to determine the circuit behavior based on the inputs of the cells driving the bridged nodes, as shown in Fig. 2. In this way, it realizes increased data analysis capabilities and is used to implement the new bridge fault model.

#### 3.2 Multilayer Feedforward Neural Networks

Neural networks are used for tasks such as classification, time-series prediction, noise reduction, and general function mapping in a wide variety of different scientific applications. There are many different types of neural network architectures in existence. The bridge fault model presented in this paper uses the popular multilayer feedforward neural network (MLFN) or backpropagation...
network. This type of network has been selected for the bridge defect model because it performs well as a function approximator. Interested readers should refer to various references for more details about its architecture and operation [28], [29], [30].

With respect to the capabilities of the MLFN, it is often referred to as a universal function approximator [31]. Specifically, given an appropriate network architecture and sufficient training, the MLFN can learn any deterministic function to an arbitrary degree of accuracy. Thus, since the voltage and timing characteristics of electronic circuits are deterministic functions by nature, the MLFN can learn them, assuming the other conditions are met. To conduct training, the input vectors in the training set are presented to the network individually and the network’s outputs are compared to the correct output vector. This allows a cumulative measure of error to be computed for the training set. Then, the weights are updated, using an optimization algorithm, to reduce the error for the network. Training must be conducted for many iterations through the training set until the network reaches a suitable level of accuracy. Historically, this has been a time-consuming process and is perhaps the most significant drawback to practical implementations of neural networks. However, faster algorithms now exist for weight optimization in MLFNs. In the application presented in this paper, the conjugate gradient algorithm is used for training [28].

3.3 Bridge Fault Model Implementation

The new bridge fault model uses an MLFN implemented within the framework of the modified saboteur fault model. It takes advantage of the modeling features of VHDL as well as the computational strengths of the neural network. A single saboteur cell is developed to model all bridge faults between the different components in a cell library. Instance-specific information, such as cell drive strength and propagation delay, is provided to the saboteur through generic parameters in the VHDL entity. Furthermore, the defect resistance (hard, soft, or no-defect) is controlled by a defect control signal. When there is no active defect, the saboteur fault model is executed in response to a transition on the nodes that it is capable of bridging. However, since there is no defect, it simply passes the logic signals in zero simulation time.

When the defect control signal is set to an active defect state, the bridge model is executed in response to a transition on one of the input signals to the cells driving the bridged nodes. Fig. 3 details the operation of the defect-injected saboteur cell in response to a logic transition on inputs $A_1$ and/or $A_2$. It is important to note that the bridge fault model itself executes in zero simulation time and is typically invoked in response to a single event on one of its inputs. Thus, if an event on $A_1$ and/or $A_2$ occurs, the input nodes $B_1$ and $B_2$ are usually inactive and will remain that way for at least as long as the fault model takes to execute. An equivalent flowchart can be constructed for transitions occurring on $B_1$ and/or $B_2$, with $A_1$ and $A_2$ inactive. The scenario when the $A$ inputs and $B$ inputs change at the exact same time is a special case and is discussed at the end of this section.

In this flowchart, the cell outputs, $O_A$ and $O_B$, are the normal driven logic values (i.e., no defect). Since the bridge cell must determine $O_A$ and $O_B$ immediately after a change in the driving cell inputs, it must know specific information with respect to the logic function of these cells. This behavioral information is passed to the bridge cell through generic parameters during the instantiation of each saboteur cell. Also, note that the bridge cell outputs are analog voltage values, which are seemingly out of place in a digital simulation environment. However, it will be shown later how this is used to resolve the Byzantine General’s problem. Finally, the model flags bridge defects as being $I_{DDQ}$ detectable when the bridged nodes are driven to opposite logic values, allowing a direct path for current to flow from power supply to ground.
Referring to Fig. 3, the symbol, $\delta$, represents a unit time step that increments each time the bridge model is executed. Thus, after each transition on $A_1$ and/or $A_2$, the time step increments and the bridge cell looks up the new value for $O_A$. Then, the $I_{DDQ}$ flag is set if $O_A(\delta) \neq O_B(\delta - 1)$. The model then compares the new output of the active cell, $O_A(\delta)$, to the previous output, $O_A(\delta - 1)$, and the previous output of the other cell driving the node, $O_B(\delta - 1)$, to determine how to compute the next state. In scenario 1, if the output of the active cell, $O_A$, does not change value and it is equal to the output of the other cell, $O_B$, then no action is required because there is no logic contention. If, as in condition 2, $O_A$ is unchanged but is not equal to $O_B$, new node voltages are computed because the relative drive strength of the cells may be different. The new node voltages are then assigned immediately because, in reality, the actual propagation time between the two intermediate voltage states would be minimal. Finally, if neither of the previous conditions have been satisfied, then the node voltages will change under conditions of either logic contention or with both nodes driving to the same logic value. The neural network is used to compute the new node voltages and propagation delay times in these situations.

To resolve timing glitches, VHDL’s transport delay mechanism is used. Specifically, if a transition is to occur after all other scheduled events, then it simply occurs at that time, after the other events. However, if the transition is supposed to occur before other scheduled events, then it preempts these events. In the rare case when the $A$ inputs change at the exact same time as the $B$ inputs, the bridge cell is executed twice, in the order determined by the VHDL simulator event handler, and both responses will be individually scheduled. This approach ensures that the final voltage is correct and provides a reasonable timing approximation as the node voltages switch through the various states.

### 3.4 Fault Model Neural Network Architecture

Fig. 4 shows a port-level diagram of the neural network for the bridge fault model. The output variables define the voltages, $V_1$ and $V_2$, and the delay factors, $t_1$ and $t_2$, of the two bridged nodes with respect to the input variables. Outputs are computed in response to an event on the inputs to the preceding cells, as described in Fig. 3. As a convention, the node driven by the active cell is node 1, or the active node. Conversely, the other node is referred to as node 2 or the inactive node.

The input vector for the neural network has 10 specific elements, as shown in Fig. 4. Selection of these specific inputs was based on the requirement, as discussed in Section 3.2, to develop a deterministic function for the output vector. As such, each parameter selected for inclusion in the input vector was found to be relevant for computing the output voltages and propagation delay times. The inputs $L_1$ and $L_2$ are the defect-free logic values driven by the active and inactive cells, respectively. Elements $S_1$ and $S_2$ are input-pattern dependent measures of relative drive strength for the two cells. The electrical resistance of the bridge defect is $R_B$. It is clear that both the final voltage and the propagation delay of the bridged nodes depend on these values. However, the propagation delay computations depend on several additional parameters. The values $\tau_1$ and $\tau_2$ are input-pattern dependent change times for each cell driving one standard load. These parameters contribute information about the capacitive characteristics of the pull-up/pull-down network within each cell. The parameter, $S_{L,R}^T$, is the drive strength of the active cell prior to the transition. This provides an indication of the bridge node voltages before the transition, allowing a better estimate of the delay to the next state. The parameter, $T_{1,DF}$, is the defect-free propagation time of the current logic transition on the active cell. It is determined from the timing information in a Standard Delay Format (SDF) file. Since this data is backannotated from various tools in the ASIC design flow, it infers the most accurate loading factors available. Furthermore, provision of an SDF timing file allows the entire VHDL circuit simulation to implement a sophisticated timing algorithm, such as the IEEE Standard 1076-4 VITAL specification. With respect to the neural network outputs, $T_{1,DF}$ is also used as a time base multiplier for the delay factors output by the network. For instance, if $T_{1,DF}$ is equal to 200 ps and the delay factor, $t_{\tau}$, is computed by the neural network as 2.0, then the propagation delay for bridged node is modeled as 400 ps. The last parameter, $LR$, is referred to as the load ratio and is simply a ratio of the fanout for the active cell to the fanout of the inactive cell. This provides the network with a concise estimate of the load on the node driven by the inactive cell.

It is generally not difficult to find a neural network node configuration that works for a given problem. However, the total number of nodes (and connection weights) should remain as small as possible to enable fast training and to avoid overfitting the data. If the data is overfitted, the network learns irrelevant details of individual training cases and, subsequently, loses its ability to generalize when presented with unknown cases. For this bridge defect modeling application and a sample cell library discussed in Section 4, both three and four layer networks were found to be acceptable. Also, better performance was observed when two separate neural networks were used to implement the bridge model. One network covers all situations when the active node changes to the opposite state of the static node, causing an increased propagation delay and intermediate voltage level. The other network is used when the active node is changing to the same logic state as the static node, causing a reduced propagation delay. It was found that networks with less than 300 connection weights are unable.
to learn the functions to an acceptable degree of accuracy. Conversely, networks with more than 600 connection weights tend to overfit the data very easily. Accordingly, the networks were configured with four layers, having 20 nodes in the second layer and 10 nodes in the third layer. The total number of weights for each network is $10 \times 20 + 20 \times 10 + 10 + 4 = 440$.

3.5 Determining Drive Strength Data
The drive strength variables, $S_1$ and $S_2$, are determined using the simple test circuits shown in Fig. 5. In these circuits, the cell output is connected to an ideal voltage source through a 1 KΩ resistor. This resistance value was selected such that it could be held constant, yet yield consistent relative circuit effects, between different cells, influenced by the entire range of critical bridge resistances. Then, for every input pattern driving the logic value opposite to the external source, the node voltage at the output of the cell is measured as the drive strength. From the example shown in Fig. 5, it is clear that different input patterns cause the same logic response do indeed have different drive capabilities.

The charge time parameters, $\tau_1$ and $\tau_2$, are collected in a similar manner. Specifically, individual defect-free cells are loaded with a capacitance equivalent to one standard load. Then, for each input pattern combination, the charge/discharge time at the output node is determined. Again, since the arbitrary load circuit is kept constant across all tests, a suitable relative measure of charge time will be obtained.

3.6 Bridged Node Voltage Interpretation
As discussed in Section 2, the Byzantine General’s problem occurs when the bridged lines adopt an intermediate voltage, with downstream cells interpreting it at different logic values. The new bridge model eliminates this problem by using voltage-level interpreter components, which are specifically derived for each input of each cell in the library. The voltage-level interpreters immediately precede every cell on a bridged node, along with the bridge cell, as shown in Fig. 6. When a voltage change is sensed by the voltage-level interpreter, it compares it to the voltage transfer characteristics of the downstream cell and determines an appropriate logic value. Specifically, the lowest input voltage interpreted as logic high, $V_{IH}$, the highest input voltage interpreted as logic low, $V_{IL}$, and the switching voltage, $V_s$, are used to determine the logic values, as shown in Fig. 6. Depending on the application of the fault model, the "H" and "L" logic values may be interpreted as "X" to account for process variations, noise, and environmental factors.

4 VALIDATING THE BRIDGE MODEL
4.1 Deriving the Training and Test Data
Training and validation of the neural network for the bridge fault model requires data from several sample bridge defects for a given cell library. An example cell library based on TSMC’s 0.35μm, 3 metal layer, CMOS technology has been provided by the Canadian Microelectronics Corporation. It contains several dozen combinational logic cells and a wide variety of latches and D flip-flops, including scannable cells. Bridge defect data is derived using analog simulation and must cover a broad range of possible bridge defect scenarios. Bridge defects between cells of varying drive strength combinations must be included in the training set to ensure that the network can “learn” the effects of cell drive strength on the circuit behavior. Furthermore, it is imperative that a wide range of loading conditions and both soft and hard bridge defect resistances are considered. Finally, for each of the combinations possible from these variables, various logic transitions must also be included.

Given the large number of cases possible from the conditions listed above, a significant amount of time is spent deriving the training data before the bridge fault model can be completed. Thus, a general purpose defect to fault (D2F) translation tool has been developed to automate the process. The D2F tool is a 15,000 line program implemented using Tcl and the Tk Toolkit to provide an intuitive graphical user interface. The current version uses Cadence Spectre version 4.4.3 to conduct all analog simulations and runs on a Sun Solaris Unix platform. To conduct the bridge defect simulations, the user simply selects a cell from the cell library and specifies a load circuit. Then, another cell in the library is selected along with its load circuit. The D2F tool automatically builds the defect circuit, determines appropriate input stimulus, and conducts simulation of hard and soft bridge defects. Then, the simulation results are analyzed by the tool and output voltage and propagation delay data is written to a file.

The D2F tool is capable of performing bridge defect analysis using any simulation device models supported by the analog simulation engine. All simulations conducted for the sample cell library were performed using BSIM3v3.1
Hard and soft defects were studied at bridge resistances of 750Ω and 3,000Ω, respectively. These values are within the ranges suggested by previous studies [11], [12], [13] and were precisely selected for this technology based on results of extensive analog simulation. Using a greater number of different resistances would result in a more flexible neural network, at the cost of more time to derive the training set and conduct the training.

4.2 Training the Neural Network

Training of the neural network is conducted using a random subset of the bridge defect data samples derived using the D2F tool. The remaining samples are used for testing and validating the neural network. The rationale behind this approach is that the neural network should certainly be capable of learning its training set, assuming that the architecture is appropriate and the learned function is deterministic. However, the true measure of performance can only be assessed when it is tested using a set of data that it has not yet been exposed to. Furthermore, as the performance of particular network is continually evaluated using the separate test data set, architecture and training decisions eventually become influenced by the particular characteristics of this set as well. Thus, it is useful to have yet another set of data to perform a final validation of the neural network. To summarize, the set of derived bridge defect data is separated into three sets. These sets are referred to as the training set, the test set, and the validation set. For the sample cell library, data was collected for 130 of the several thousand possible bridge defects; with 70 referred to as the training set, the test set, and the validation set. For the sample cell library, data was collected for 10,000 iterations. Fig. 7 shows the performance of the neural network, evaluated using the test set, at 100 iteration intervals throughout the training. The voltage error curve shows the average voltage deviation, from the analog simulation results, of both bridged nodes across all input pattern combinations for the 30 defects in the test set. Similarly, the error curve shows the average delay error, compared to analog simulation, of both bridged nodes across all input patterns for the 30 defects in the test set. The rate of training for this network, with the 70 defect training set, is approximately 1,800 iterations/hour on a Sun Ultra workstation. It should be noted that training is a one time process and, therefore, is not a significant issue for deriving the model.

Fig. 7 shows that training significantly reduces the test set error until around 6,000 iterations (200 minutes). Subsequent training, which continues to reduce the error for the training set, shows mixed benefits for the test set. During the training beyond 6,000 iterations, the neural network is learning irrelevant details, with respect to the general bridge defect population, about the training set. This can negatively affect performance for samples not in the training set. Increasing the size and diversity of the training set would alleviate this problem. Nonetheless, Fig. 7 clearly indicates that the existing network is excellent at predicting the node voltages at bridge defect sites. Furthermore, achieving an average delay prediction error below 14 ps, the neural network performs well in a domain where no other model, short of full analog simulation, has even made an attempt. In fact, considering that cell load parameters are typically only an estimate at this point and there are significant timing variations due to process and environmental conditions, further attempts to reduce the delay computation error would be pointless.

To confirm the accuracy of the neural network, the validation set is used. Inspection of Fig. 7 suggests that the network is optimally trained somewhere between 6,000 and 7,000 iterations. The trained weight states are arbitrarily extracted at 6,600 iterations for subsequent validation experiments. Testing the 30 bridge defects in the validation set, after 6,600 iterations of training, the average voltage error was 0.0059 V and the average delay error was 14.15 ps. These figures are in the same range as those found using the test set. As yet another accuracy confirmation, 20 more bridge defects were selected randomly by an independent third party. The actual defect behavior was derived using the D2F Tool for comparison with the neural network predictions. Results show that the average voltage error for this set is 0.0071 V and the average delay error is 12.14 ps, both of which are within the same range as the previous error computations. Since the neural network had no prior exposure to the bridge defects in either of these validation sets, it can be concluded with reasonable assurance that the network is adequately trained for this cell library.

5 DEEP SUBMICRON TECHNOLOGY CONSIDERATIONS

5.1 Determining an Upper Limit on Design Size

Recent research in DSM technology design principles suggests that, to sustain the current trends in scaling, future ICs will be comprised mainly of moderate-sized modules in the 50,000 gate count range [32]. Specifically, it
is argued that wire lengths are small enough at this module size that line resistance can be neglected for delay calculations. Furthermore, as scaling continues, shrinking gate pitches will allow local wire lengths to shrink. Subsequently, a decrease in average wire capacitance is forecast, within the 50K gate module, for each generation down to 0.1\mu \text{m} and beyond. Notwithstanding these predictions, given that a list of possible defects for a large circuit is huge, the idea of comprehensive fault modeling is possible only for limited gate count designs. Thus, for the new bridge fault model, a circuit module limit of 50K gates is established. This module size is more than sufficient to include many IP blocks. This section will prove that the new model is accurate for a 50K gate IP block using the 0.35\mu \text{m} technology. A cumulative wire-length distribution function, CIDF, a high average fanout of 3.0 is assumed [35]. Also, the average number of terminals per gate, \( K \), is set to 4.0 and Rent’s exponent, \( p \), is set to 0.75. These values are also in the conservative (high) range for estimating interconnect lengths. The CIDF for a 50K gate module using 0.35\mu \text{m} technology is presented in Fig. 8.

From the CIDF, the average wire length is estimated to be 63 \mu \text{m} and the longest wire length is estimated at 2,073 \mu \text{m}. Furthermore, 99.9 percent of all interconnects are less than 1,282 \mu \text{m}, 99 percent are less than 709 \mu \text{m}, and 98 percent are less than 514 \mu \text{m}. Via resistances, at around 1 \Omega \text{ per contact in the 0.35\mu m technology, are negligible at these wire lengths. Accordingly, various analog bridge defect simulations are conducted using the \pi 3 interconnect model, at these wire lengths, and ignoring via resistance. These experiments enable analysis of the effects of defect position along the interconnect. This will provide reasonable justification for ignoring parasitic resistive effects during collection of the training and test data.

The first series of tests conducted uses the bridge defect circuit shown in Fig. 9. This circuit represents a worst case scenario, whereby the node drivers are minimum drive strength buffers loaded by a long segment of interconnect. Since there are very few long wires in a given circuit, the chance that two such wires are in close enough proximity to cause a bridge defect at such a troublesome site is minimal. Furthermore, wires of this length are typically not driven by minimum strength cells. Still, this represents a possible, albeit unlikely, scenario. Tests are conducted with each segment of wire, on both sides of the defect site, being modeled by the \pi 3 model in metal layer 3 (M3). The position of the defect is reasonably far from the node driver, allowing a nearly maximal voltage drop along the interconnect. Results obtained from this circuit, at various wire lengths, are shown in Table 1.

Results reported in Table 1 are the timing and voltage variations observed between the \pi 3 interconnect model circuits and the lumped capacitance loaded test circuits. Specifically, \( \Delta V_1 \) and \( \Delta V_2 \) are the voltage deviations observed on nets 1 and 2 for the \pi 3 interconnect model circuits versus the lumped capacitance loaded test circuits. The values \( \Delta d_1 \) and \( \Delta d_2 \) are percentage delay deviations observed on nets 1 and 2 for the \pi 3 interconnect model circuits versus the lumped capacitance loaded test circuits. Actual voltage transitions are driven on net 1 only and include both rise and fall cases as indicated. The delay deviations reported as NT (no transition) could not be measured because there was no distinct logic transition occurring. Furthermore, tests performed with the bridge
defect closer to the node driver, on either or both nets, resulted in smaller voltage deviations from the capacitance only test circuit. Delay figures collected with the defect position closer to the driver were not significantly different from those reported in Table 1.

From the results in Table 1, it is observed that node voltages computed with the more accurate interconnect model are reasonably close to the purely capacitive model. However, for the delay computations, a wide range of variations are observed. This is primarily due to the higher capacitance of the longer line lengths compared to the more typical capacitance values used throughout for the lumped capacitance load model. It is again emphasized that this series of experiments represents an unlikely scenario due to the small number of long wires in a 50K gate circuit. A more realistic defect involving long wires would likely be with a shorter wire, simply due to their relative abundance, and perhaps in an adjacent metal layer. Thus, an identical series of experiments was conducted except that net 2 is an average length interconnect (63 μm) and is routed in the M2 layer because it is shorter. Also, net 1 is driven by a 4X strength buffer because of the large load. Finally, the bridge defect is positioned at the 50 percent distance along each interconnect, which represents the mean, assuming a uniform distribution of possible positions. Results obtained from these experiments are presented in Table 2.

These results show us that the voltage deviations are insignificant for typical bridge defects in a 50K gate circuit. For the delay computations, negative values are reported due to the use of a conservative wire load in the lumped capacitance simulations. Despite this, the delays are close to those achieved using the simpler load circuit, even for smaller wires. Considering the delay effects of temperature, supply voltage, and process variation, the approximation provided by the lumped capacitance load circuit is certainly within an acceptable range for most applications.

The previous two experiments have shown, for a 50K gate module using the 0.35 μm cell library, that it is indeed reasonable to ignore the defect position and the parasitic voltage effects of resistive interconnects when modeling bridge defects. However, with respect to parasitic delay effects, some consideration to the total interconnect load is prudent, especially for the longer wires. The defect-free propagation time parameter, $T_{1,DF}$, in the new bridge fault model provides this. Specifically, assuming the tool that generated the SDF data considered parasitic resistances, then $T_{1,DF}$ will infer these details for the delay time calculation.

### 5.2 Augmentation of Neural Network Model

When a given application does not fit the criteria of the limited size module, the neural network architecture as presented is insufficient for modeling various bridge defects in long interconnects. Furthermore, even under the 50K IP module assumption, bridges between global and semiglobal interconnects within a larger design cannot be modeled. Thus, for using the new bridge fault model under these conditions, a conceptually straightforward augmentation is suggested.

First, as discussed previously, the length of the wire between the driving cells and the bridge defect site can have a significant impact on the node voltages. Thus, the distances between the driving cells and the bridge defect site, $D_1$ and $D_2$, should become inputs to the neural network. For instance, if an IFA process reveals that two interconnects cross each other in adjacent layers at a particular point, then the distances, $D_1$ and $D_2$, from this point to the signal drivers would become inputs to the neural network. This data would be in addition to the data already used by the neural network. Furthermore, if the network is to be trained to handle different metal layers within the same process technology, some indication of the resistive properties of the bridged interconnect layers will be required by the neural network.

The primary problem with this augmented model is in deriving an appropriate training set. With at least 13 independent input variables, complex statistical techniques

---

### TABLE 1

<table>
<thead>
<tr>
<th>wire lengths</th>
<th>$\Delta V_1$ rise/fall</th>
<th>$\Delta d_1$ (%) rise/fall</th>
<th>$\Delta V_2$ rise/fall</th>
<th>$\Delta d_2$ (%) rise/fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>2073 μm</td>
<td>+0.18V / -0.03V</td>
<td>+99 / +73</td>
<td>-0.03V / +0.18V</td>
<td>NT / +97</td>
</tr>
<tr>
<td>1282 μm</td>
<td>+0.11V / -0.02V</td>
<td>+49 / +36</td>
<td>-0.02V / +0.11V</td>
<td>NT / +48</td>
</tr>
<tr>
<td>709 μm</td>
<td>+0.06V / -0.01V</td>
<td>+12 / +8</td>
<td>-0.01V / +0.06V</td>
<td>NT / +12</td>
</tr>
<tr>
<td>514 μm</td>
<td>+0.05V / -0.01V</td>
<td>0.0 / -2.4</td>
<td>-0.01V / +0.05V</td>
<td>NT / -0.5</td>
</tr>
</tbody>
</table>

### TABLE 2

<table>
<thead>
<tr>
<th>net 1/2 lengths</th>
<th>$\Delta V_1$ rise/fall</th>
<th>$\Delta d_1$ (%) rise/fall</th>
<th>$\Delta V_2$ rise/fall</th>
<th>$\Delta d_2$ (%) rise/fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>2073 μm 63 μm</td>
<td>+0.008V / -0.001V</td>
<td>+25 / +18</td>
<td>-0.09V / +0.07V</td>
<td>+25 / +12</td>
</tr>
<tr>
<td>1282 μm 63 μm</td>
<td>+0.005V / -0.001V</td>
<td>+12 / +9</td>
<td>-0.06V / +0.04V</td>
<td>+6 / 0</td>
</tr>
<tr>
<td>709 μm 63 μm</td>
<td>+0.003V / -0.000V</td>
<td>+1 / +1</td>
<td>-0.04V / +0.03V</td>
<td>-7 / -9</td>
</tr>
<tr>
<td>514 μm 63 μm</td>
<td>+0.002V / -0.000V</td>
<td>-3 / -3</td>
<td>-0.03V / +0.02V</td>
<td>-11 / -12</td>
</tr>
<tr>
<td>6 μm 63 μm</td>
<td>+0.000V / -0.000V</td>
<td>-13 / -10</td>
<td>-0.010V / +0.007V</td>
<td>-21 / -18</td>
</tr>
</tbody>
</table>
will be required to ensure that variable interrelationships and domain corners are adequately represented in the training set. Furthermore, the time required to derive the training set will become large, but probably not unmanageable. Another limitation with this model is that it can only be applied late in the design process, after the routing stage. At this point, design changes to avoid fatal or untestable defects are much more costly than those discovered earlier. Despite these drawbacks, however, this augmented neural network bridge fault model has considerable potential and is worthy of future research efforts.

6 IMPLEMENTATION AND RESULTS

After the neural network has been trained and validated, the D2F tool automatically generates the defect-injectable VHDL saboteur cell. Also, the voltage interpreter components used for resolving the Byzantine General’s Problem are generated based on information collected during defect-free cell characterization. The remaining steps in employing the new bridge fault model for analysis of digital circuit designs are outlined in this section. Specifically, a specialized software tool for automating the defect-injection process is described. Then, using several benchmark circuits, results are provided with respect to observed error response to the bridge faults as well as a discussion on simulation time issues. Finally, it is shown that the overhead required to derive the new bridge fault model and the simulation time requirements are indeed justified when model accuracy is a concern.

6.1 The Bridge Fault Analysis Tool

Given that an IP block has thousands of individual cells, the task of manually inserting bridge faults and analyzing the simulation results is impossible. Thus, a specialized software tool, written in Tcl/Tk, has been developed to automate these administrative tasks and invoke the commercial VHDL simulation tools. Fig. 10 outlines the operation of the bridge fault analysis tool.

The bridge simulation process starts by reading a structural VHDL circuit description and its SDF timing file as written by a commercial synthesis tool. A technology file is also read which contains all of the necessary information regarding cell I/O pins, logic functions, drive strengths, charge times, and voltage transfer characteristics. Then, after conducting an inventory of the ports and cells listed in the VHDL circuit description, a test pattern file can be generated (pseudorandomly) or read from a file. Testbench processes are then added to the structural VHDL circuit description to apply the test patterns to the circuit inputs and record the outputs in ASCII result files. A defect-free simulation is then conducted to enable comparison with defect-injected results.

Before the bridge defect injection process can begin, a list of bridge defects is required. An option is provided to build an exhaustive set of bridge defects, a pseudorandom set of bridge defects, or to read a list of bridge defects from a file. Selecting an exhaustive list, the total number of bridge defects to consider is of \( O(n^2) \), or \( n(n - 1)/2 \) to be exact, where \( n \) is the number of nodes in the circuit. Clearly, for anything but the smallest circuits, it would be impossible to simulate this many defects. Selecting a random subset of these defects is a reasonable approach to use when no layout information is available or for validating a fault modeling technique. However, since close physical proximity between nodes is essential for bridge defects to occur, most cases in the exhaustive or random subset are physically impossible. Thus, the most accurate and efficient set of bridge defects would be derived using an inductive fault analysis approach whereby likely bridge defects are selected based on the proximity and length of circuit interconnects [3], [36], [37], [38]. Despite the fact that this is beyond the scope of the present research, the tool is capable of importing bridge defect lists from other tools.

Once the bridge defect list is selected, the defect-injectable saboteurs are inserted into the VHDL netlist along with the voltage interpreter components for each connection downstream from the defect site. However, to avoid overcomplicating the circuit and bogging the simulation down with dormant bridge fault models, only a subset of saboteurs are inserted during each iteration. The size of the subset is set by the user, but, as a maximum, each node may have only one bridge fault model attached to it. For the
bridge fault model instances in the VHDL netlist, logic behavior, charge time, and drive strength tables for the driving cells are passed to the entity through generic parameters. The gate fanouts of the two bridged cells are also passed to enable computation of the load ratio parameter. Finally, the defect-free propagation times, $T_{DF}$, for the driving cells are provided from the timing information in the SDF file.

After the bridge fault model instances are inserted into the circuit, a VHDL process is added to the circuit netlist to enable individual injection of defects into the bridge fault models. This defect injection process controls whether each bridge fault model is dormant or is modeling a soft or hard defect. Dormant fault models will simply translate the logic values at their inputs to the correct voltage level, which will in turn be translated back to the logic domain by the voltage interpreter cells in zero simulation time.

Before invoking the simulator, the netlist is compiled using a commercial VHDL compiler. Then, the simulator is invoked and the first bridge defect from the list is injected into its corresponding bridge fault model. The entire set of test patterns is applied sequentially to the circuit and the circuit outputs are written to a file for analysis. The circuit is then reset and subsequent defects in the list are individually injected and simulated. Following each simulator run, the output file is compared to the defect free simulation results by the defect analysis tool and a concise summary file is created. After all of the bridge defects have been simulated, the summary files are combined and relevant statistics are tabulated.

### 6.2 Benchmark Circuits and Results

To test the new bridge fault model and defect injection mechanism, a few sample circuits were selected from the ITC '99 benchmark suite available from Politecnico di Torino. Specifically, RTL VHDL descriptions of circuits b11, b14, and b21 were synthesized using the $0.35\mu m$ cell library described previously. Some characteristics of these benchmark circuits are presented in Table 3.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Input Pins</th>
<th>Output Pins</th>
<th># Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
<td>7</td>
<td>6</td>
<td>439</td>
</tr>
<tr>
<td>b14</td>
<td>32</td>
<td>54</td>
<td>4651</td>
</tr>
<tr>
<td>b21</td>
<td>32</td>
<td>22</td>
<td>11224</td>
</tr>
</tbody>
</table>

For each benchmark circuit, a test pattern file with 200 test vectors was generated pseudorandomly using the bridge fault analysis tool. Then, after the defect-free simulation, bridge defect lists were randomly generated and simulations were conducted for the three circuits as detailed in Table 4. All simulations were run on a Sun Ultra Enterprise 4500 server, utilizing 10x400MHz UltraSPARC II processors.

<table>
<thead>
<tr>
<th>Circuit &amp; Clock Period</th>
<th>#Bridge Defects</th>
<th>Sim Time</th>
<th>Soft/Hard Def Cov%</th>
<th>IDDQ Cov%</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11-12 ns</td>
<td>10000</td>
<td>43 min</td>
<td>37.4 / 60.3</td>
<td>98.85</td>
</tr>
<tr>
<td>b11-7.5 ns</td>
<td>10000</td>
<td>39 min</td>
<td>40.8 / 63.7</td>
<td>98.85</td>
</tr>
<tr>
<td>b14- 25 ns</td>
<td>10000</td>
<td>293 min</td>
<td>20.1 / 51.4</td>
<td>94.17</td>
</tr>
<tr>
<td>b14-20 ns</td>
<td>10000</td>
<td>256 min</td>
<td>21.3 / 54.1</td>
<td>99.22</td>
</tr>
<tr>
<td>b21- 30 ns</td>
<td>40000</td>
<td>30 hr</td>
<td>9.7 / 31.5</td>
<td>96.03</td>
</tr>
<tr>
<td>b21- 22 ns</td>
<td>40000</td>
<td>28 hr</td>
<td>9.9 / 32.1</td>
<td>96.91</td>
</tr>
</tbody>
</table>

The number of bridge defects reported in the table includes an equal number soft and hard defects. Each circuit was simulated at two different clock speeds, one with plenty of slack time and the other with virtually no slack time in the critical path. The results reported in the Soft/Hard defect coverage column are the percentage of soft and hard defects that caused logical errors at the circuit outputs. It should be noted that no effort was made to derive optimal test patterns for these circuits, hence the low defect coverages. Finally, the last column shows the expected IDDQ coverage for the set of test patterns, assuming that background current is minimal and that adequate measurement resolution can be achieved. The simulator flags bridge defects as IDDQ detectable when the bridged nodes are driven to opposite logic values, allowing a direct path for current to flow from power supply to ground.

Results presented in the above table show that, for all of the benchmark circuits, faster clock speeds cause a notably higher incidence of observable faults. This situation would not occur using other bridge fault simulation schemes. Instead, only the faults occurring using the slower clock speeds would be observed. This provides a strong argument for the use of a “timing aware” bridge fault simulator. As a specific example, if the defect coverage data was intended for use as the cost function in a test pattern generation scheme, those defects causing delay faults only are ignored. Despite the fact that several of them would likely be fortuitously covered by the test set, many others would go undetected due to a lack of specific tests generated for them.

To provide an indication of the effects of increasing the number of test patterns and to show that these bridge defects are indeed testable, the hard defect tests were repeated for circuit b11 at a 7.5 ns clock period using test pattern files of 100, 500, and 5,000 test vectors. These results

### TABLE 5

<table>
<thead>
<tr>
<th>Circuit &amp; Test Set Size</th>
<th>#Bridge Defects</th>
<th>Sim Time</th>
<th>Hard Def Cov (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11- 100 vec</td>
<td>5000</td>
<td>11 min</td>
<td>57.5</td>
</tr>
<tr>
<td>b11- 200 vec</td>
<td>5000</td>
<td>20 min</td>
<td>63.7</td>
</tr>
<tr>
<td>b11- 500 vec</td>
<td>5000</td>
<td>46 min</td>
<td>67.1</td>
</tr>
<tr>
<td>b11- 5000 vec</td>
<td>5000</td>
<td>427 min</td>
<td>83.5</td>
</tr>
</tbody>
</table>

### TABLE 3

Characteristics of Benchmark Circuits

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Input Pins</th>
<th>Output Pins</th>
<th># Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
<td>7</td>
<td>6</td>
<td>439</td>
</tr>
<tr>
<td>b14</td>
<td>32</td>
<td>54</td>
<td>4651</td>
</tr>
<tr>
<td>b21</td>
<td>32</td>
<td>22</td>
<td>11224</td>
</tr>
</tbody>
</table>
TABLE 6
Simulation Time for Defect-Injected and Defect-Free Circuits

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Defect-Free</th>
<th>Defect-Injected</th>
<th>% increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
<td>03:01</td>
<td>03:42</td>
<td>22.65%</td>
</tr>
<tr>
<td>b14</td>
<td>28:53</td>
<td>33:59</td>
<td>17.66%</td>
</tr>
<tr>
<td>b22</td>
<td>54:45</td>
<td>57:30</td>
<td>5.02%</td>
</tr>
</tbody>
</table>

are presented in Table 5 along with the results reported previously for the 200 vector test pattern set. Note that increasing the number of test patterns improves the defect coverage considerably. However, as with any pseudorandom test pattern generation scheme, increasing the number of test patterns inevitably reaches a point where benefits are offset by the increased simulation time requirements. Also, we see that the simulation time scales almost proportionately with the number of test vectors.

The issue of simulation time remains a notable concern with the proposed bridge fault modeling scheme. However, considering that the simulation engine implements the sign-off quality VITAL modeling specification, observed simulation times are certainly within reason. To demonstrate this, simulation time comparisons are conducted between the defect-free VHDL netlists and the netlists with 100 bridge fault model instances, including all associated signal and interpreter cell overhead. The circuits with the fault models were simulated using the 200 vector test pattern set, individually injecting both soft and hard defects into each fault model. The defect-free circuits were simulated for an equivalent number of clock cycles. Simulation time results from these tests are reported in Table 6. For these circuits, the added overhead and calculations required to implement the fault model induce a reasonably small simulation time penalty beyond the industry standard VITAL specification.

7 CONCLUSIONS AND FUTURE WORK

This paper has presented a totally new approach to bridge fault modeling. The new model uses a neural network implemented within the framework of a VHDL saboteur cell. It was shown that this bridge fault model is able to compute the voltage and propagation delay times for the bridged node signals with negligible error. Furthermore, computations are achieved in a fraction of the time required by comparably accurate approaches. Consideration of its applicability for DSM technologies has also been provided, with tests confirming accuracy in IP blocks up to 50,000 gates. Finally, several benchmark circuits were tested to demonstrate the strengths and limitations of the new bridge fault model.

To fully realize the capabilities of the new bridge fault model, an automated test pattern generator must be developed. This would produce test patterns that specifically target the realistic logical faults and relevant delay faults that are neglected by other bridge fault models. Also, the suggested extension to consider positional data of bridge defects should be studied. This augmentation could enable accurate bridge fault models for global signals between IP blocks and in individual modules beyond 50,000 gates.

References


Donald B. Shaw received the BSc (electrical engineering) and MSc (computer engineering) degrees from the University of Manitoba, Winnipeg, Manitoba, Canada, in 1994 and 1997, respectively. In 2001, he received the PhD degree from the Royal Military College of Canada, conducting research in CMOS device/interconnect defects, fault modeling, and fault secure systems. He is currently employed as a video ASIC design engineer at Gennum Corporation in Burlington, Ontario. He is a member of the IEEE.

Dhamin Al-Khalili received the BSc degree in 1966 and the MSc and PhD degrees in electrical engineering from the University of Manchester, United Kingdom, in 1970 and 1972, respectively. He joined the Ontario Centre for Microelectronics, OCM, in 1982, as a senior consultant, then worked with the Canadian Semiconductor Design Association as a research director. Before joining OCM, he worked at the Winnipeg Microelectronics Center as a senior engineer and project leader. He also worked for one year as a research industrial fellow at Northern Telecom, Ottawa. He is presently a professor with the Department of Electrical and Computer Engineering at the Royal Military College of Canada and adjunct professor at Concordia University, Montreal. His research interests include VLSI architecture, low power electronics, testability analysis and design automation. He is a senior member of the IEEE.

Côme N. Rozon received the MSc degree (June 1977) in solid state physics from Sherbrooke University, Sherbrooke, Quebec, Canada, and the PhD degree (August 1987) in electrical engineering from Queen’s University, Kingston, Ontario, Canada. From 1975 to 1983, he served as a combat systems engineer in the Royal Canadian Navy and retired at the rank of Naval Lieutenant. In 1983, he joined the teaching staff of the Electrical and Computer Engineering Department at the Royal Military College of Canada, Kingston, Ontario. He has worked as a consultant for Newbridge Networks and Nortel Networks. He has teaching commitments in electronics and digital design and research interests in VLSI and design for testability applied to both binary and multivalued logic systems. He held the position of Director of Computing Services at the Royal Military College of Canada and currently he is the head of the Department of Electrical and Computer Engineering at the same institution.

For more information on this or any computing topic, please visit our Digital Library at http://computer.org/publications/dlib.