Accurate CMOS Bridge Fault Modeling With Neural Network-Based VHDL Saboteurs

Don Shaw
Gennum Corporation
970 Fraser Drive
Burlington, Ontario, L7L 5P5
email: dshaw@gennum.com

Dhamin Al-Khalili and Côme Rozon
Dept. of Elec. and Comp. Engineering
Royal Military College of Canada
Kingston, Ontario, K7K 7B4
email: alkhalili-d, rozon-c@rmc.ca

ABSTRACT

This paper presents a new bridge fault model 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 existing approaches. The new model computes exact bridged node voltages and propagation delay times with due attention to surrounding circuit elements. This is significant since, with the exception of full analog simulation, no other technique attempts to model the delay effects of bridge defects. Yet, compared to these analog simulations, the new approach is orders of magnitude faster and achieves reasonable accuracy; computing bridged node voltages with an average error near 0.006 volts and propagation delay times with an average error near 14 ps.

Keywords

Bridge Defects, Fault Models, Neural Networks, VHDL, CMOS ICs, Fault Simulation

1. INTRODUCTION

An area of concern related to the rapid advancement of IC technologies is the development of new fault models to study the effects of defects on IC designs. This paper concerns the development of a new bridge fault model. 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 can be 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 [4], [3], [12]. 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 routing between cells, it has been found that the majority of bridge defects occur between the output signals of logic gates [2], [9], [12]. 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. 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. 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, the bridge defect usually has an impact on the propagation time of the bridged signals.

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 intermediate voltage levels occur, downstream logic gates with varying input voltage thresholds can interpret the same voltage level differently. This is known as the *Byzantine General's Problem* and can significantly complicate bridge defect modeling issues [1].

Bridge defects exist at various levels of severity, depending on the electrical resistance of the short circuit caused by the defect. Due to the progressive nature of wearout mechanisms that can cause bridge defects, they often start at high resistance levels and continue to become more severe as time passes. For model tractability, the range of bridge resistances is often divided into two groups, hard and soft. If the resistance is relatively low, a hard defect is said to have occurred and a logical fault may be introduced into the circuit. Conversely, a higher resistance bridge causes a soft defect with performance degradation effects such as delay faults. The actual resistance value that distinguishes between hard and soft defects varies based on the process technology [8], [10], [13].

2.2. Bridge Fault Model Goals

An accurate bridge fault model must consider the effects of resistive bridges at various levels of severity. Then it can represent 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 have a means to deal with the Byzantine General's problem in the logic interpretation of the node voltages. Also important is the ability to model the delay fault effects of bridge defects with consideration to the load characteristics and other relevant factors. Finally, the bridge fault model must be efficient to allow for analysis of real circuit modules within a reasonable amount of CPU time.

3. A NEW BRIDGE FAULT MODEL

3.1. VHDL Fault Models - The Saboteur

Fault modeling with VHDL is a compelling idea that has been evolving steadily over the past several years. 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 [6]. It is placed on the nets between existing cells and can be used to model a wide variety of fault cases. A significant drawback of the saboteur model as originally proposed in [6] is that it has no direct access to the input ports of the preceding gates and, therefore, cannot model faults that depend on the drive strength of the signal. Thus, a modification is proposed such that additional signal lines are added to the circuit, enabling the saboteur to determine the fault behavior based on the inputs of the cells driving the bridged nodes as shown in Figure 1.

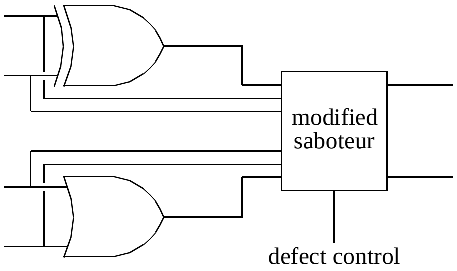

Figure 1. The modified saboteur fault model.

3.2. Multilayer Feedforward Neural Networks

Neural networks are used for classification, time-series prediction, noise reduction, and general function mapping in a wide variety of different applications. There are many different types of neural networks. The fault model presented in this paper uses the popular multilayer feedforward neural network (MLFN).Various references can be consulted to provide more details about its architecture and operation [7], [11].

With respect to the capabilities the MLFN, it is often referred to as a *universal function approximator* [5]. Specifically, given an appropriate network architecture and sufficient training, the MLFN can learn any deterministic function to an arbitrary degree of accuracy. Clearly then, 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 output vector that it is supposed to produce. During this process, a cumulative measure of error is computed for the training set. Then, the weights are updated, using an optimization algorithm, to reduce the measure of error for the network. Training must be conducted for many passes 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, relatively fast algorithms now exist for weight optimization. In the application presented in this paper, the conjugate gradient algorithm is used for training [7].

3.3. Bridge Fault Model Implementation

The new bridge fault model uses a MLFN implemented within the framework of the modified saboteur fault model. 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. The defect resistance (hard, soft, or no-defect) is controlled by a defect control signal and the fault model is executed in response to a transition on the nodes that it is bridging. If there is no defect, it simply passes the logic signals in zero physical time. However, 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 those cells driving the bridged nodes. The flowchart shown in Figure 2 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 physical time and is 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 labeled B_1 and B_2 are assumed 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.

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 is passed to the bridge cell through generic parameters during the instantiation of the 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.

3.4. Fault Model Neural Network Architecture

Figure 3 shows the top-level architecture of the bridge fault neural network. The output variables define the voltage, *V*, and a delay factor, *t*, of the two bridged nodes with respect to the input

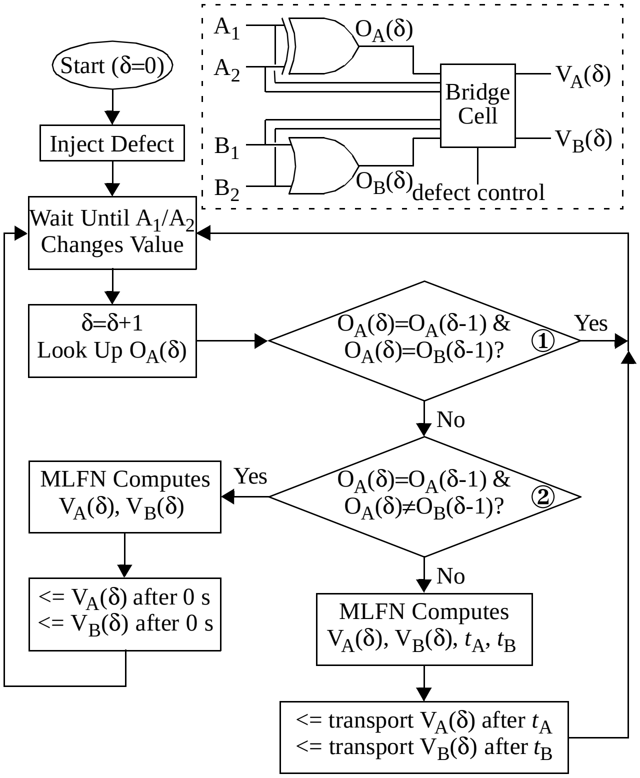
Start (δ=0)
Inject Defect
Wait Until A_1/A_2 Changes Value
$\delta=\delta+1$ Look Up $O_A(\delta)$
A_1
A_2
$O_A(\delta)$
Bridge Cell
$V_A(\delta)$
$V_B(\delta)$
B_1
B_2
$O_B(\delta)$
defect control
$O_A(\delta)=O_A(\delta-1)$ & $O_A(\delta)=O_B(\delta-1)$? ①
Yes
No
MLFN Computes $V_A(\delta)$, $V_B(\delta)$
Yes
$O_A(\delta)=O_A(\delta-1)$ & $O_A(\delta)\neq O_B(\delta-1)$? ②
<= $V_A(\delta)$ after 0 s
<= $V_B(\delta)$ after 0 s
No
MLFN Computes $V_A(\delta)$, $V_B(\delta)$, t_A, t_B
<= transport $V_A(\delta)$ after t_A
<= transport $V_B(\delta)$ after t_B

Figure 2. Operation of saboteur bridge cell.

variables. These values are computed in response to an appropriate change on the inputs to the preceding cells as described in Figure 2. 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.

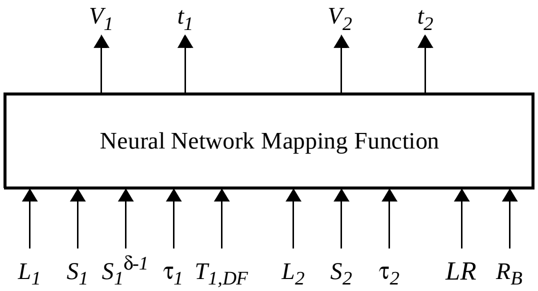

Figure 3. Neural network input and output vectors.

The input vector for the neural network has 10 elements as shown in Figure 3. The inputs L_1 and L_2 are the defect-free logic values driven by the cells. 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. Both the final voltage and the propagation delay of the bridged nodes depend on these values. However, the propagation delay computations require additional parameters. The values τ_1 and τ_2 are input-pattern dependent charge 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, $S_1^{\delta-1}$, is the drive strength of the active cell prior to the transition. This provides an indication of the node voltages before the transition, allowing a better estimate of the transition time 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. 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 actual load on the node driven by the inactive cell.

3.5. Bridged Node Voltage Interpretation

As discussed previously, the Byzantine General's problem occurs when the bridged lines adopt an intermediate voltage level, with downstream cells interpreting this voltage at different logic values. The new bridge fault model handles 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 are placed immediately preceding every cell on a bridged node, along with the bridge saboteur cell, as shown in Figure 4. When a voltage change is sensed by the voltage-level interpreter, it compares it to the voltage transfer characteristics for 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_x, are used to determine the logic values as shown in Figure 4.

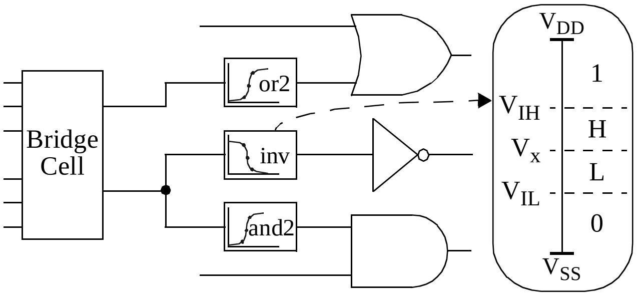

Figure 4. Voltage-Level Interpreter Components.

4. VALIDATING THE BRIDGE MODEL

4.1. Deriving the Training andTest 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 0.35µ, 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. For instance, 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 defects are considered. Finally, for each of these combinations, various logic and drive strength transitions must also be included.

A general purpose defect to fault (D2F) translation tool has been developed to fully automate the process of deriving this data. The D2F tool is implemented using the Tcl scripting language and uses Cadence Spectre version 4.4.3 for all analog simulations. It is capable of performing bridge defect analysis using any device models supported by the simulator. All simulations conducted for the sample cell library were performed using BSIM3v3.1 device models at 3.3 Volts supply and typical process/temperature parameters. The procedure would be identical for data collected at worst/best case voltage/process/temperature parameters. Hard and soft defects were studied at bridge resistances of 750Ω and 3000Ω respectively. These values are within the ranges suggested by previous studies, [8], [10], [13], and were precisely selected for this technology based on results of extensive 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 prudent 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 arbitrarily allocated to the training set and the remaining 60 divided between the test and validation sets.

Before training commences, the input and output data for the neural network must be scaled to ensure that it is within acceptable ranges. Then, using a custom C program, training of the bridge fault model for the sample cell library was conducted for 10000 iterations, with the voltage and delay error evaluated after every 100 iterations. Figure 5 shows the performance of the neural network, evaluated using the test set, at 100 iteration intervals throughout the training. The node voltage line 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 delay time line shows the average delay error for the 30 defects in the test set. The rate of training for this network, with the 70 defect training set, is approximately 1800 iterations/hour on a Sun Ultra^{TM} workstation. It should be noted that training is a one time process and, therefore, is not a significant issue for deriving the model.

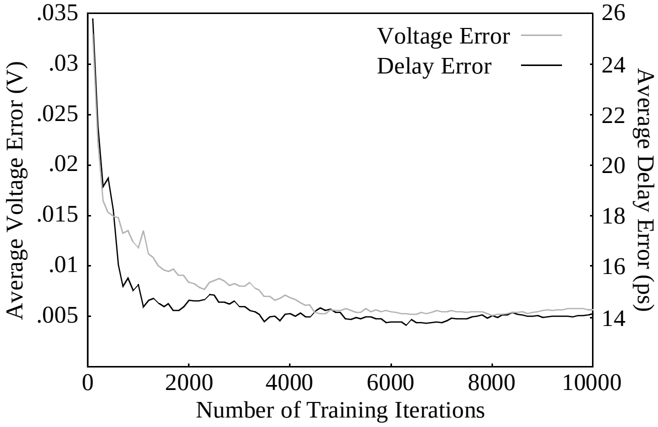

Figure 5. Neural network test set error during training.

The results presented in Figure 5 show that training significantly reduces the test set prediction error until around 6000 iterations (200 minutes). Subsequent training, which continues to reduce the error for the training set, shows mixed performance benefits for the test set. During the training beyond 6000 iterations, the network learns irrelevant details about the training set with respect to the general bridge defect population. Increasing the size and diversity of the training set would provide further accuracy for test set predictions. Nonetheless, it is clear that the existing network is excellent at predicting the node voltages. Furthermore, achieving an average delay prediction error below 14 ps, the neural network produces respectable results in a domain where no other model, short of full analog simulation, has even made an attempt in the past. Considering that the gate loads are typically only an estimate at this point, and the significant timing variations due to process/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 Figure 5 suggests that the network is optimally trained somewhere between 6000 and 7000 training iterations. The trained weight states were arbitrarily extracted at 6600 iterations for subsequent experiments. Testing the 30 bridge defects in the validation set at this point, the average voltage error was found to be 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 confirmation of accuracy, 20 more bridge defects were selected randomly by an independent 3rd party and the actual behavior was derived using the D2F Tool for comparison with the neural network predictions. The average voltage error for this set was 0.0071 V and the average delay error was 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 randomly selected bridge defects in either of these validation sets, it can be concluded that the network is adequately trained for this cell library.

5. IMPLEMENTATION AND RESULTS

5.1. Conducting a Bridge Fault Analysis

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 automated by another specialized software tool.

The bridge simulation process starts by reading a structural VHDL circuit description and its most recent SDF timing file. It also reads a library technology file, written by the D2F tool, which contains the necessary information regarding cell I/O pins, logic functions, drive strengths, charge times, and voltage transfer characteristics. Then, after itemizing the ports and cells listed in the VHDL circuit description, a test pattern file is generated (pseudorandomly) or read from a file if available. Testbench processes are then added to the structural VHDL circuit description. These processes apply the test patterns to the circuit inputs and record the outputs in ASCII result files. A defect-free simulation is conducted at this point to enable comparison with defect-injected results.

Before the bridge defect injection process can begin, a list of 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. 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 of the defect site. When the fault model instances are inserted into the netlist, logic behavior, charge time, and drive strength tables for the driving cells are loaded into 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, $T_{1,DF}$, 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 enable injection of defects into the fault models. This process controls whether each bridge fault model is dormant, or is modeling a soft or hard defect. Then, the netlist is compiled, 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 observable circuit outputs are written to a file for analysis. The circuit is then reset and each subsequent defect in the list is individually injected and simulated. Following each simulator run, the output file is compared to the defect free simulation results and a concise summary file is created. After all of the bridge defects are simulated, the summary files are parsed and relevant statistics are tabulated.

5.2. 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 0.35µ cell library described previously. Some characteristics of these benchmark circuits are presented in Table 1.

Table 1: Characteristics of benchmark circuits.

Circuit	Input Pins	Output Pins	# Cells
b11	7	6	439
b14	32	54	4651
b21	32	22	11224

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 2. All simulations were run on a *Sun Ultra™ Enterprise 4500* server, utilizing 10x400MHz *UltraSPARC II* processors.

Table 2: Benchmark circuit test results.

Circuit & Clock Period	#Bridge Defects	Sim Time	Soft/Hard Def Cov (%)
b11-12 ns	10000	43 min	37.4 / 60.3
b11- 7.5 ns	10000	39 min	40.8 / 63.7
b14- 25 ns	10000	293 min	20.1 / 51.4
b14- 20 ns	10000	256 min	21.3 / 54.1
b21- 30 ns	40000	30 hr	9.7 / 31.5
b21- 22 ns	40000	28 hr	9.9 / 32.1

The number of bridge defects reported in the table includes an equal number soft and hard defects. Note that 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. The results show that, for all of the benchmark circuits, a faster clock causes a notably higher incidence of observable faults. This would not occur using other bridge fault models. Instead, only the faults occurring at 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 and may remain undetected.

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 5000 test vectors. These results are presented in Table 3 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.

Table 3: Effects of varying the number of test vectors.

Circuit & Test Set Size	#Bridge Defects	Sim Time	Hard Def Cov (%)
b11- 100 vec	5000	11 min	57.5
b11- 200 vec	5000	20 min	63.7
b11- 500 vec	5000	46 min	67.1
b11- 5000 vec	5000	427 min	83.5

The issue of simulation time remains a concern with the proposed bridge fault modeling scheme. However, considering that it implements the sign-off quality VITAL specification, observed simulation times are within reason. To qualify this, simulation time comparisons were conducted between the defect-free circuits 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 4. For these circuits, the added overhead required to implement the fault model induce a reasonably small simulation time penalty beyond the industry standard VITAL specification.

Table 4: Sim. time for defect-injected and defect-free circuits.

Circuit	Defect-Free	Defect-Injected	% increase
b11	03:01	03:42	22.65%
b14	28:53	33:59	17.66%
b22	54:45	57:30	5.02%

6. 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 has been 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. Finally, several benchmark circuits were tested at different clock speeds 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.

7. REFERENCES

[1] J.M. Acken and S.D. Millman, “Fault model evolution for diagnosis: accuracy vs. precision,” *Proc. IEEE Custom Integrated Circuits Conference*, 1992, pp. 13.4.1-13.4.4.

[2] M. Calha, M. Santos, F. Goncalves, and J.P. Teixeira, “Back annotation of physical defects into gate-level, realistic faults in digital ICs,” *Proc. Int’l Test Conf.*, 1994, pp. 720-728.

[3] F.J. Ferguson and T. Larrabee, “Test pattern generation for realistic bridge fault models in CMOS ICs,” *Proc. Int’l Test Conference*, IEEE, 1991, pp. 492-499.

[4] F.J. Ferguson and J.P Shen, “A CMOS fault extractor for inductive fault analysis,” *IEEE Trans. Computer-Aided Design*, Vol. 7, No. 11, Nov. 1988, pp. 1181-1194.

[5] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, 2:5, 1989, pp. 359-356.

[6] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, “Fault Injection into VHDL Models: The MEFISTO Tool,” *Proc. 24th Int’l Symposium Fault-Tolerant Computing*, IEEE, 1994, pp.66-75.

[7] T. Masters, *Practical Neural Network Recipes in C++*, Academic Press, Inc., San Diego, CA, 1993, 493 pp. ISBN 0-12-479040-2.

[8] S.D. Millman and J.M. Acken, “Special applications of the voting model for bridging faults,” *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 3, Mar. 1994, pp. 263-270.

[9] M. Renovell, P. Huc, and Y. Bertrand, “CMOS bridging fault modeling,” *Proc. 12th IEEE VLSITest Symposium*, Cherry Hill, New Jersey, Apr. 1994, pp. 393-397.

[10] R. Rodriguez-Montanes, J.A. Segura, V.H. Champac, J. Figueras, J.A. Rubio, “Current vs. Logic Testing of Gate Oxide Short, Floating Gate and Bridging Failures in CMOS,” *Proc. Int’l Test Conference*, IEEE, 1991, pp. 510-519.

[11] D. Rumelhart and J. McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts, 1986.

[12] J.J.T. Sousa, F.M. Gonçalves, and J.P. Teixeira, “IC defects-based testability analysis,” *Proc. Int’l Test Conference*, IEEE, 1991, pp. 500-509.

[13] H. Vierhaus, W. Meyer, and U. Glaser, “CMOS bridges and resistive transistor faults: I_{DDQ} versus delay effects,” *Proc. Int’l Test Conference*, IEEE, 1993, pp. 83-91.