An efficient CMOS bridging fault simulator with SPICE accuracy

An Efficient CMOS Bridging Fault Simulator: With SPICE Accuracy

Chennian Di and Jochen A. G. Jess

Abstract— **This paper presents an alternative modeling and simulation method for CMOS bridging faults. The significance of the method is the introduction of a set of *generic-bridge tables* which characterize the bridged outputs for each bridge and a set of *generic-cell tables* which characterize how each cell propagates a logically undefined input. These two sets of tables are derived dynamically for a specific design by using a SPICE circuit simulator. Then they can be used by any logic fault simulator to simulate bridging faults. In this way, the proposed method can perform very fast bridging fault simulation yet with SPICE accuracy. The paper shows how these two sets of tables are derived and used in a parallel pattern fault simulator. Experimental results on ISCAS85 benchmarks are promising.**

I. Motivation

In last decade, the gap between the so-called realistic faults caused by manufacturing defects and practically used Single Stuck-At (SSA) faults has been emphasized strongly [1], [4], [6]–[8] for CMOS Integrated Circuits (IC). It becomes evident that accurate modeling and efficient simulation of defect induced faults are essential for high quality testing of IC's. Particularly the bridging faults, one of the most frequently occurring faults, attract a lot of attention.

The complexity of modeling and simulating bridging faults has been very well studied. It is widely known that one of the difficulties of modeling a bridging fault is with the conducting circuit created from power supply to ground. Such a conducting circuit changes a digital circuit into one with undefined behavior. To illustrate this, Fig. 1 shows a zero-ohm bridge between the outputs of a complex cell and a 2-in-NAND (the number next to each transistor indicating its size). Its bridged output obtained by SPICE is shown in Table I. It is seen that the output may vary from 0.63 to 4.71 V for different inputs. These values cannot easily be accepted as logic "1" or "0" since the propagated logic value depends on the conditions of the following cells. Usually a node in such a state is said to be in "undefined" state. Fig. 2 shows a possible fanout structure of the bridge in Fig. 1. The bridged output bearing a value 2.15 V can drive one 2-in-NAND to 0.64 V which can be read as "0" and drives another structurally equivalent 2-in-NAND to 3.99 V which can be read as "1." Thus any simple

Manuscript received October 29, 1993; revised June 5, 1995 and April 4, 1996. This work was supported by the ESPRIT project 2318 entitled: EVEREST". This paper was recommended by Associate Editor W. Maly.

C. Di was with the Faculty of Electrical Engineering, Eindhoven University of Technology, The Netherlands. He is now with the IBM Microelectronics Division, Essex Junction, VT 05452 USA.

J. A. G. Jess is with the Faculty of Electrical Engineering, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands.

Publisher Item Identifier S 0278-0070(96)06728-0.

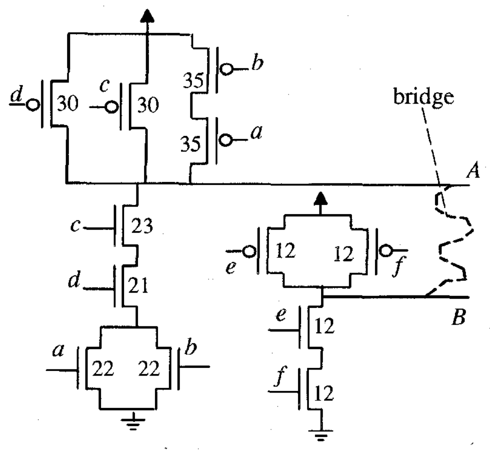


Fig. 1. An example of a bridge.

model, such as wired-and or wired-or, is not sufficient here. It is obvious that the behavior of a bridge can only be accurately modeled if following two issues can be resolved efficiently:

As it can be seen from the example that few centivolts difference in the input voltage can cause different logic outputs, thus any approximate method is not sufficient. To guarantee correct results, circuit-level accuracy must be considered. Obviously a circuit simulator, such as SPICE, can accurately fulfill the tasks. However, for large circuits, this seems computationally intolerable.

Early solutions of using a switch level model [2], [3] have been widely [4], [10], [12] recognized as inadequate to model and simulate CMOS bridging faults.

Many suggested methods attempt to improve the modeling accuracy by using an approximate model, such as the resistive network model [10] or the voting model [4], [11], [14], [16]. These methods allow very fast logic fault simulation but have the drawback that only the bridged outputs are analyzed without carefully considering how the fault propagates. A case of incorrect modeling by using such method can be illustrated by Fig. 2. For the conducting circuit created by the inputs shown in Fig. 2, the pull-down conductance is stronger than the pull-up conductance. The voting model would predict the bridged output as "0" but, in fact, it can be "1" or "0" depending on the condition of the fanout cells as shown in Fig. 2.

A more accurate method using mixed-level or multilevel simulation techniques is proposed in [13]. This method switches from normal logic simulation to circuit-level simulation whenever a bridging fault is encountered. The

TABLE I
BRIDGED OUTPUT OBTAINED BY SPICE

inputs		outputs	
a b c d	*e f*	*A B*	bridged(V)
1 1 1 1	0 1	0 1	0.63
1 0 1 1	1 0	0 1	1.42
0 0 1 1	1 1	1 0	2.15
1 1 1 1	0 0	0 1	2.47
0 1 0 1	1 1	1 0	3.35
0 0 0 0	1 1	1 0	4.71

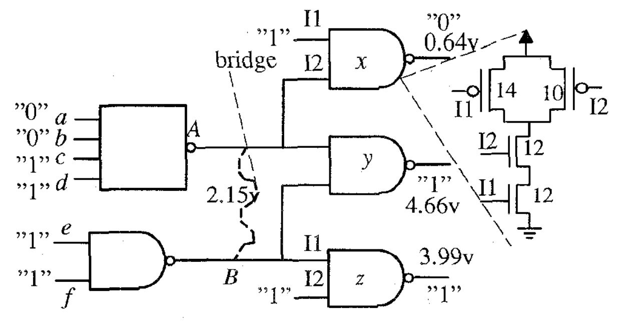


Fig. 2. Impact of an undefined input on fanout cells.

bridge is simulated at circuit-level through its fanout cells until the undefined signals can be safely read as logic values. Then the simulation is switched back to logic level. This method is very accurate. But for lengthy test patterns, a large circuit may not be efficiently simulated. For instance, while the bridge in Fig. 1 is simulated, the inputs $abcdef = 100\,111$ and $abcdef = 110\,111$ cause the same conducting circuit. In such a case this method would invoke the expensive circuit simulator twice while it is unnecessary. It is also not efficient to evaluate all the bridges connecting two cells having the same combination of cell types, such as a 2-in-NOR to a 3-in-NAND. Some improvements [17] use so-called "precomputed tables" derived by a circuit simulator to avoid some unnecessary computations and use cell/gate logic threshold voltages to propagate an input voltage. However, the precomputed tables that are derived by enumerating all the combinations of a cell library may be both time and memory consuming. They may contain redundant informations and make it not easy to maintain and to use such a huge database. Furthermore the fault propagation is still not accurate since the cell logic threshold voltages when one signal drives more than one input terminal of a cell are not considered. This may introduce many errors. An improvement of the voting model [14], [18] is unfortunately still approximate in nature and the fault propagation has the same shortcoming.

1) the accurate evaluation of the bridged output voltage;
2) the accurate propagation of an undefined input.

Based on our development in [19], this paper presents a more accurate modeling and yet very fast fault simulation approach. Section II presents two new concepts and the general strategy of the proposed method. Section III and IV discuss some implementation issues. Section V presents some experimental results.

II. FAULT SIMULATION USING GENERIC-BRIDGE AND GENERIC-CELL TABLES

This paper concentrates on CMOS combinational circuits. A CMOS circuit can be viewed as an interconnection of CMOS cells. A CMOS **cell** has a network of serial-parallel PMOS transistors as pull-up and, its dual part in terms of NMOS transistors, as the pull-down part. The bridging faults analyzed are nonfeedback bridges between outputs of two cells. The feedback bridges are not treated in this paper since they show much complex behavior and need a different treatment. For more information of feedback bridges, we refer to the recent advances documented in [21]. The resistance of the bridges is assumed to be negligible. Furthermore only static analysis is performed.

2.1. Evaluation of a Bridging Fault

In order to guarantee the circuit-level accuracy and yet to obtain high efficiency, let us examine the design procedure first. Modern digital CMOS designs are mostly based on a standard cell library. In a specific design, the number of instantiated cells is usually much larger than the size of the cell library. One type of a cell may be used many times in the design. Thus it is very likely that many bridges may connect the same combination of the cell types in the same manner. Such a set of bridges represents one bridge type, called a **generic-bridge**. A set of generic-bridges can be derived for all the extracted bridges in a design. Then the evaluation of all the bridges can be restricted to the generic-bridges. Usually the number of generic-bridges is far smaller than the number of all extracted bridges. Each generic-bridge can be evaluated by using a circuit-level simulator, such as SPICE in our case. Then the bridged output is computed with the accuracy of SPICE. Yet a large amount of computation is avoided.

For each generic-bridge, a **generic-bridge-table** is introduced for the set $\mathcal{B}$ of all input vectors of the two bridged cells that activate the bridge. A generic-bridge-table consists of a set of pairs $\langle b, d\rangle$ as its entries. Let T_{bri} be a set denoting all the entries. For each $\langle b, d\rangle \in T_{\text{bri}}, b$ is the one of the distinct output voltages in the presence of the bridge. The entity d is a Boolean expression exactly covering all input vectors of the two bridged cells generating b at the bridged output. Obviously the expressions d_i induce a partition of $\mathcal{B}$. For any two $\langle b_1, d_1\rangle$ and $\langle b_2, d_2\rangle$ in T_{bri}, if d_1 is true, then d_2 is not true and vice versa. The generic-bridge-table can be viewed as the "function" with the symbols "+" and "$\cdot$" interpreted as addition and multiplication of real numbers as well

$$V_{\text{bri}} = b_1 \cdot d_1 + \cdots + b_n \cdot d_n \tag{1}$$

if d_i is satisfied, the V_{bri} takes a voltage value b_i.

For the bridge shown in Fig. 1, its generic-bridge-table is obtained as

$$\begin{aligned} V_{\text{bri}} = &\, \mathbf{0.00} \cdot (e \oplus f) \cdot a \cdot b \cdot c \cdot d \\ &+ 1.42 \cdot (e \oplus f) \cdot (a \oplus b) \cdot c \cdot d \end{aligned}$$

$$\begin{aligned}&+2.15\cdot e\cdot f\cdot\overline{a}\cdot\overline{b}\cdot c\cdot d\\&+2.45\cdot\overline{e}\cdot\overline{f}\cdot a\cdot b\cdot c\cdot d\\&+2.89\cdot\overline{e}\cdot\overline{f}\cdot(a\oplus b)\cdot c\cdot d\\&+3.35\cdot e\cdot f\cdot(a+b)\cdot(c\oplus d)\\&+\mathbf{5.00}\cdot e\cdot f\cdot(\overline{a}\cdot\overline{b}\cdot\overline{c\cdot d}+\overline{c}\cdot\overline{d}).\end{aligned}\tag{2}$$

The entries having boldfaced 0.00 and 5.00 indicate that for those inputs, the bridged output voltage is actually close enough to the potential of ground and power supply respectively.

2.2. *Propagation of Undefined Inputs*

Let us examine how a CMOS cell transfers an input voltage. First the **logic (switch) threshold voltage** of a cell is defined. For an invertor, it is defined as the input voltage value such that the output voltage is equal to the input voltage. It can be defined in the similar way for a cell having more than one input terminal. Such a cell may have several different logic threshold voltages. For instance, a NAND with two inputs a and b has a logic threshold voltage 1.89 V when a switches while $b = 1$. Vice versa, it has a logic threshold voltage 2.20 V. When both a and b are driven by the same signal, the logic threshold voltage is 2.60 V. In the sequel, a logic threshold voltage when only one input terminal switches is classified as **single-input** logic threshold voltage. Otherwise it is classified as **multiple-input** logic threshold voltage.

In modern technology, it is known that the CMOS cell has a very high gain around its logic threshold voltage. A small variation at the input will yield a very big swing at the output. It is very likely that an input voltage lower than the input logic threshold voltage will be read as a logic "0" and vice versa. It is possible that an input voltage is equal to or very close to the logic threshold voltage. Then the output can be still not safely determined. In our experiments on the ISCAS85 benchmarks, we chose a margin of 0.02 V around the respective logic threshold voltage. Any input voltage within this range was considered as not safely propagated. Still such situations add up to only 0.2% of all the cases investigated. Therefore, to obtain fast fault simulation, it is sufficient to propagate a bridging fault just up to the outputs of its immediate fanout cells. Further propagation by circuit simulation to the next level of fanout gates does not resolve the uncertainty.

Usually a specific design uses only a subset of cells from the given cell library. To be consistent with the definition of the generic-bridge, each cell in such a subset of a cell library is called a **generic-cell** of this design. Then for a specific design, only the logic threshold voltages of each generic-cell are required for the fault propagation. They can be computed accurately by a circuit-level simulator, in our case SPICE. Again a large amount of computations can be avoided.

To formulate and keep the derived logic threshold voltages of each generic-cell, a **generic-cell-table** is introduced. The generic-cell-table of a generic-cell consists of a set of labeled pairs $\langle w, O\rangle_l$ as its entries. The label l represents some subset of the input terminals of the generic-cell. For each $\langle w, O\rangle_l$, the entity w is the value of the threshold voltage when the inputs l are all driven by the same signal. O is a Boolean expression representing the set of input vectors such that input terminals l are observable at the output of the generic-cell. Or, in other words, the input established by connecting together all the inputs of the set l is a controlling input to the generic-cell. Let $T_{\text{cell}}(l)$ denote the union of all labeled pairs $\langle w, O\rangle_l$ when the terminals l are driven by the same signal

$$T_{\text{cell}}(l) = \bigcup_{i=1}^{m}\langle w_i, O_i\rangle_l.$$

Then, for any two $\langle w_1, O_1\rangle_l$ and $\langle w_2, O_2\rangle_l$ in $T_{\text{cell}}(l)$, if O_1 is true, then O_2 is not true and vice versa. This is because for an input vector such that terminals l are observable, the generic-cell cannot have two different logic threshold voltages simultaneously when inputs at l are driven by the same signal.

Let L be a set denoting all the combinations of the input terminals of a generic-cell. Then the set containing all the entries in the generic-cell-table can be expressed as

$$T_{\text{cell}} = \bigcup_{l\in L} T_{\text{cell}}(l). \tag{3}$$

It is not difficult to prove that any two entries of a generic-cell-table are also mutual exclusive. Thus the generic-cell-table can be viewed as the function V_{cell} defined by

$$V_{\text{cell}} = \sum_{l\in L}((w_1\cdot O_1)_l + (w_2\cdot O_2)_l + \cdots + (w_m\cdot O_m)_l). \tag{4}$$

When specific terminals l are driven by the signal and O_i is satisfied, V_{cell} takes the logic threshold voltage w_i. For the NAND with two inputs a and b mentioned at the beginning of this section, its generic-cell-table can be expressed as

$$V_{\text{cell}} = (1.89\cdot b)_{\{a\}}(2.20\cdot a)_{\{b\}} + (2.60)_{\{a,b\}}. \tag{5}$$

2.3. *Fault Simulation Strategy*

With the introduction of the generic-bridge-table and generic-cell-table, the whole modeling and simulation can be performed in the procedure illustrated in Fig. 3. Given the circuit layout and defect statistics, the transistor netlist is extracted. Simultaneously, all possible bridging faults are extracted. Then the circuit is further condensed to a model on logic-level. All the generic-bridges and generic-cells are extracted as well. Their tables are then computed by SPICE simulations. After this step, the bridging faults can be simulated for a given test pattern set through the manipulation of these two sets of tables. The fault simulations can be done exclusively at logic level. Consequently both high modeling accuracy and simulation efficiency are obtained. The following sections discuss how these two sets of tables are derived and give details about the fault simulation.

III. Dynamic Derivation of Generic-Bridge and Generic-Cell Tables

The derivation of the generic-bridge-tables and the generic-cell-tables is performed by analyzing the extracted bridging faults for a specific design instead of exploring the given cell library by complete enumeration. Thus the derivation is

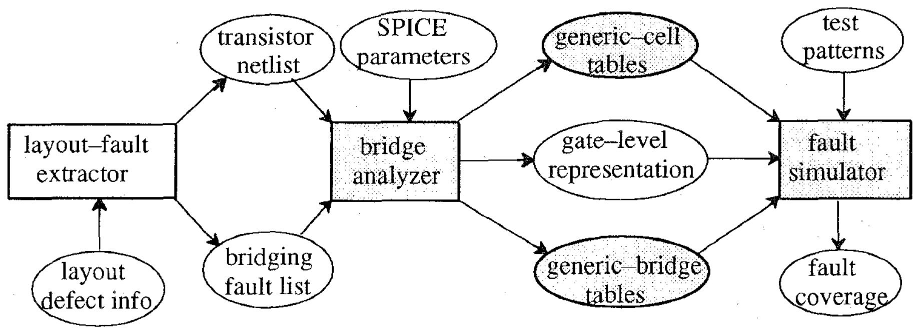


Fig. 3. Modeling and simulation system overview.

dynamic to each design. The reasons of using this strategy are discussed below:

1) The number of the generic-cells in a specific design is usually smaller than the size of a given cell library. Consequently the number of all possible generic-bridges in the design is small. Thus, the task of characterizing both tables for a design is easier.
2) The occurrence of bridging faults depends highly on the layout topology of a specific design. It is very likely that a generic-bridge derived by enumerating the cell library may actually never occur in a design. Such information can only be obtained by analyzing the extracted bridging faults for a specific design.
3) The number of all possible multiple-input logic threshold voltages for a set of cells is usually very large. The actual number of multiple-input situations depends on how many bridging faults actually connect more than one input of a cell and how a cell is actually connected in a design. Again such information can only be obtained by analyzing the extracted bridging faults for a specific design.

In a practical situation with evolving history of designs, an incremental strategy may be still more appropriate. That is, all the derived generic-bridge-tables and generic-cell-ta bles of previous designs are kept in a database. If some generic-bridge-tables or generic-cell-tables are not present in the database while considering a new design, only the missing new generic-bridges and generic-cells are analyzed on SPICE level and the database is updated. This way, the database evolves according to the needs of the design team and no redundant information is computed and accumulated.

The inputs of the *bridge analyzer* (Fig. 3) are a flat representation of the transistor netlists and all possible bridging faults. Both are extracted from the layout of a design. The SPICE parameters for a specific fabrication process are also taken as an input.

The first step is the extraction of all the cells. The CMOS circuit can be represented by a connection graph $G(V, E)$. Each node $v \in V$ represents a network node which can be the drain, source or gate of a transistor. An undirected edge $e \in E$ represents a transistor and has a Boolean variable (defined by its gate input variable) and a weight representing its transistor size associated with it. The Boolean function of a cell can be easily extracted by exploring the pull-up and pull-down paths in the cell. Then the set of generic-cells for this design has to be identified. The connection graph corresponding to a cell is relatively small. Therefore the checking of the isomorphism of two cell graphs can be done efficiently. After all the instantiated cells in the design are checked, then the generic-cell set of this design is obtained. After this step, the bridging fault list can be passed to derive the generic-bridge-tables and generic-cell-tables.

3.1. Derivation of Generic-Bridge-Table

The derivation procedure of a generic-bridge-table is rather straightforward. For each identified generic-bridge, first all the possible input combinations of the two involved generic-cells that create a conducting circuit from power supply to ground are enumerated. The respective SPICE format input of each conducting circuit is accumulated in a file. Then, a SPICE call is invoked to compute the voltages at the bridged output. Upon the completion of the SPICE computation, the results are collected to construct the table. This procedure is repeated for every generic-bridge. The major cost of this procedure is obviously the execution of SPICE. To speed up the derivation, the following techniques are used.

The first technique makes use of the fact that an output voltage very close to the potential of power supply or ground can be safely interpreted as logic value. For example, for a typical 5 V CMOS technology, an input above 4 V (which may be considered as the lowest "hard" logic "1" value V^1_{hard}) or below 1 V (which my be considered as the highest "hard" logic "0" V^0_{hard}) can definitely be interpreted as "1" or "0", respectively. In such a case, we use an estimation method developed in [15] to predict the voltage range so that the SPICE runs can be avoided. This method uses a simplified transistor model to estimate the voltage. Using this model, the dc-characteristic of an NMOS transistor is characterized as

$$\begin{cases} I_{ds} = k_n \frac{W}{L}\left(V_{gs} - V_{tn} - \frac{1}{2}V_{ds}\right)V_{ds}, & V_{ds}<V_{gs} - V_{tn} \\ I_{ds} = 0, & \text{otherwise} \end{cases} \tag{6}$$

(PMOS transistors are handled analogously). Here k_n is process dependent. V_{tn} is the zero-bias transistor threshold. The subscripts g, d, s indicate the gate, drain and source of a transistor. W/L is the transistor width to length ratio. In the model, a transistor works in a linear region if it conducts, otherwise it is off. This is because in a conducting circuit

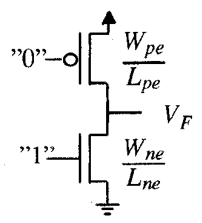


Fig. 4. The equivalent conducting circuit.

the voltage level at any drain (source) cannot be higher than V_{dd} when the gate of the transistor is driven by a logic "1." Thus $V_{ds} < V_{gs} - V_{tn}$ is always true. The model also neglects the body-effect of the MOS transistor. Using this model, any conducting circuit can be simplified to the one shown in Fig. 4 with W_{ne}/L_{ne} and W_{pe}/L_{pe} as equivalent transistor sizes of pull-down and pull-up parts respectively. The output V_F can be derived by solving (7)

$$(\beta-1)V_F^2+2(V_{dd}-V_{tn}-\beta V_{tp})V_F-\beta(V_{dd}-2V_{tp})V_{dd}=0 \tag{7}$$

($\beta = (k)p(W_{pe}/L_{pe})/(k_n(W_{ne}/L_{ne}))$. It is not difficult to prove that V_F is an increasing function of β. Two values β^1_{hard} and β^0_{hard} corresponding to V^1_{hard} and V^0_{hard} exist. Therefore

$$\beta > \beta^1_{\text{hard}} \Rightarrow V_F > V^1_{\text{hard}} \text{ and } \beta < \beta^0_{\text{hard}} \Rightarrow V_F < V^0_{\text{hard}} \tag{8}$$

($\Rightarrow$ denoting implication) holds which implies that, for a specific technology, it is not even necessary to actually solve all equations for the output voltage. Only the equivalent β is needed. Consequently the estimation is very fast. The computation of the equivalent β grows linearly with the number of transistors in a cell and can be very fast. For the cases considered, this estimation method appears accurate enough [15].

The second reduction technique is based on equivalent structures. For a bridge, many conducting circuits from power supply to ground activated by a different combination of input excitations have the same structure. Consequently, the bridged output voltage for these different excitations is the same. The conducting circuits are then said to be "structurally equivalent" for these inputs. For instance, with the bridge in Fig. 1, the four different input combinations shown in Fig. 5(a) imply the same conducting circuit as shown in Fig. 5(b) with the bridged output being 1.42 V. For those structurally equivalent conducting circuits, there is no need to repeat the SPICE simulation. In the course of analyzing a bridging fault, all the conducting circuits for which the output voltages are already obtained by simulation, are kept in a temporary set. During the enumeration of conducting circuits, if a new conducting circuit is found to be equivalent to one already in the temporary set, the SPICE simulation is skipped. Only its input condition is merged with the corresponding one in the temporary set.

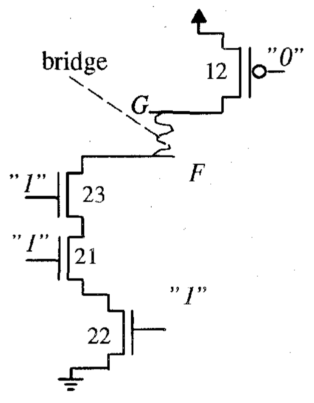
It will be shown by experimental results that the above two techniques are every effective.

3.2. Derivation of Generic-Cell-Table

A generic-cell-table is constructed in two steps. In first step, the single-input logic threshold voltages of each generic-cell are derived. The derivation procedure is straightforward. For each input terminal, all possible generic-cell configurations which may lead to different logic threshold voltages are enumerated. Their respective SPICE input formats are generated in a file. Then a SPICE call is invoked to compute the logic threshold voltages. Upon the completion, the results are collected to construct the table. This procedure is repeated for each input terminal of every generic-cell. Note that generic-cell table is restricted for single output cells. For a multiple output cell, the circuit can be partitioned according to each output cone. The generic-cell table for each output terminal can be derived as for a single output cell.

a b c d e f	bridged output
0 1 1 1 0 1	1.42V
0 1 1 1 1 0	1.42V
1 0 1 1 0 1	1.42V
1 0 1 1 1 0	1.42V

(a)



(b)

Fig. 5. Illustration of structural equivalence.

It seems that other methods did not pay enough attention to the phenomenon that a cell may have many different logic threshold voltages when a single input terminal switches. To demonstrate this effect, Fig. 6(a) shows a generic-cell which has three different logic threshold voltages when input a switches. Their values are listed in Table II. Assume the input voltage of a is 2.15 V, then it can be propagated as a "0" (the threshold is 2.08 V), undefined (the threshold is 2.15 V) or "1" (the threshold is 2.17 V) to the output. Thus those effects cannot be ignored.

In a second step, the multiple-input logic threshold voltages of each generic-cell are derived. To illustrate the situation where the multiple-input logic threshold voltages are needed, Fig. 6(b) shows a possible use of the cell in Fig. 6(a) in an actual design. It can be seen that one signal can drive two inputs (a and b in the original cell). Assume that a bridge between a and c in Fig. 6(b) occurs. Then one signal can drive three inputs (a, b, and c in the original cell). Table II also lists the logic threshold voltages for those situations. The table shows clearly that ignoring the dependencies between various inputs can be very deceptive. Thus it is essential to know

TABLE II
MULTIPLE-INPUT LOGIC THRESHOLDS

input a		input a & b		input a & b & c	
$V_{\text{threshold}}$	$b\ c\ d$	$V_{\text{threshold}}$	$c\ d$	$V_{\text{threshold}}$	d
2.08V	1 0 1	1.89V	1 0	1.58V	1
2.15V	1 0 0	2.11V	0 1	2.44V	0
2.17V	0 0 1	2.62V	0 0		

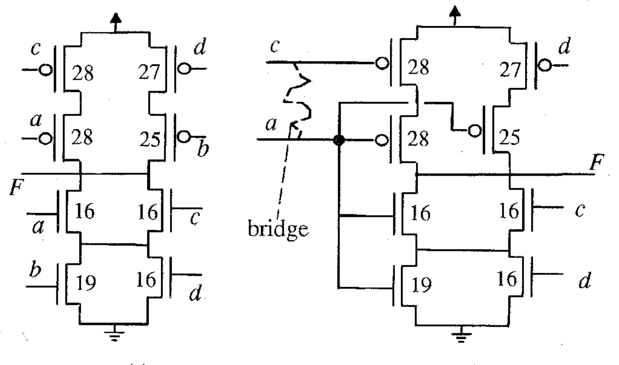


Fig. 6. (a) A complex cell. (b) Illustration of multiple-input thresholds.

the multiple-input threshold voltages in order to propagate the input correctly.

The derivation is computed while the bridging faults are analyzed. In the course of the analysis, each multiple-input case is individually identified. If more than one input in the fanout cell is bridged or one signal drives more than one input terminal of a cell, then all the possible configurations are enumerated. Their logic threshold voltages are computed by SPICE and the generic-cell-table is updated. The procedure is repeated for every bridge. Eventually all the necessary multiple-input logic threshold voltages are obtained in the tables.

The effects of multiple-input logic threshold voltages are not considered by the methods in [17] and [18]. Instead, the single-input logic threshold voltage is used for the fault propagation. This can easily lead to a wrong decision. For instance, the two inputs of a 2-in-NAND are bridged together. This NAND has two single-input logic thresholds 1.89 and 2.20 V and a multiple-input logic threshold 2.60 V. If the input voltage is 2.47 V, using the multiple-input logic threshold voltage 2.60 V, the input is propagated as "1" to the output which is consistent with the real value 4.13 V. But if the single-input logic threshold voltage, either 1.86 V or 2.20 V, is used, then a "0" would be propagated to the output which is incorrect.

3.3. Boolean Function Representations

During the analysis and the derivation of the two sets of tables, the Boolean function of each generic-cell and each table entry involves symbolic Boolean expressions and manipulations. The results need to be stored for the simulations. This seems not an important issue since it is claimed before that the number of generic-bridges and generic-cells for a design

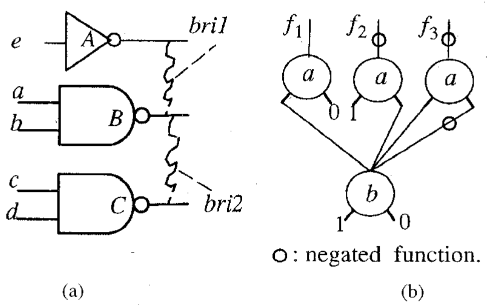


Fig. 7. (a) Example bridges. (b) Illustration of compact storage.

is small. However, if the issue is not properly handled, it may still cost unnecessary memory. To be efficient, ROBDD data structures [9] are used. It is not difficult to observe that the Boolean expression in each entry of a generic-bridge-table is established by a pull-up term of one generic-cell and a pull-down term of another generic-cell. Let each of them be stored separately. Then the canonical property of the ROBDD can result in a very compact representation.

To illustrate this, Fig. 7(a) shows a generic-cell B involved with two generic-bridges (they are not supposed both to occur simultaneously). After analysis, all the pull-down ($f_1 = a \cdot b$) and pull-up terms ($f_2 = \overline{a} \cdot \overline{b}$ and $f_3 = a \cdot \overline{b} \cdot \overline{a} \cdot b$) of B are required to construct the tables. Their ROBDD representations are shown in Fig. 7(b). The generic-bridge-tables are obtained as:

$$V_{\text{bri1}} = 1.35 \cdot \overline{e} \cdot f_1 + 1.57 \cdot e \cdot f_3 + 3.39 \cdot e \cdot f_2 \tag{9}$$

$$V_{\text{bri2}} = 1.45 \cdot g_3 \cdot f_1 + 2.67 \cdot g_2 \cdot f_1 + 1.89 \cdot g_1 \cdot f_3 + 3.45 \cdot g_1 \cdot f_2. \tag{10}$$

Here, $g_1 = c \cdot d$ is the pull-down term of C and $g_2 = \overline{c} \cdot \overline{d}$ and $g_3 = c \cdot \overline{d} + \overline{c} \cdot d$ are the pull-up terms of C.

During the whole process, the generic-cell B is only needed to be processed once to create f_1, f_2 and f_3). They are shared by both the generic-bridge-tables. The g_1, g_2 and g_3 are also created once. f_1 and g_1 are also shared inside V_{bri2}. Thus in theory, the upper bound of the memory requirement for all the tables is the number of the different pull-up and pull-down terms of all the generic-cells in a design. Consequently the memory required grows linearly with the number of generic-bridges and generic-cells.

IV. FAULT SIMULATION

With the introduction of the generic-bridge-table and generic-cell-table, bridging faults can be simulated by the procedure described below:

1) after fault free simulation, for each bridge, find its respective generic-bridge-table. Evaluate the table for the applied input pattern. If no entry is satisfied, stop. Otherwise obtain the respective output voltage value.
2) for each fanout cell of the bridged outputs, find its generic-cell-table. For the applied input, evaluate the entries labeled with the inputs that are connected with the bridged outputs. If one entry is satisfied, obtain the logic threshold voltage value; compare the bridged

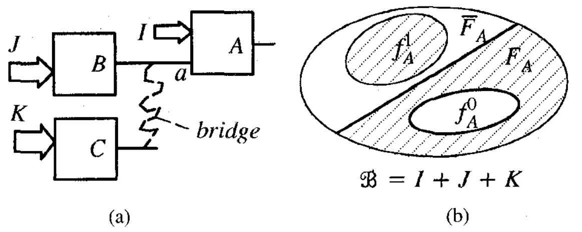


Fig. 8. Illustration of a bridging fault propagation procedure.

output voltage with the logic threshold voltage and interpret it as logic value at the output.

3) after all the fanout cells are processed, start the normal logic fault simulation from these fanout cells carrying faulty values.

In the above procedure, except for the evaluation at the fanout cells from a bridged output, the bridging fault simulation works essentially like any other logic fault simulator. Thus any efficient technique can be applied. In this paper, the well-known parallel pattern and single fault propagation (PPSFP) [5] technique is adapted. The first two steps of the fault simulation can be executed as in [5]. That is, in the forward traversal, the fault free simulation is carried out for applied patterns in parallel. In the backward traversal, the observability of each node is determined for the applied patterns in parallel as well. Then the detectability of each bridging fault is determined. In many cases, a voltage value at a bridged output can be propagated as a set of different faulty values to different fanout cells. The fanout branches carrying faulty values may reconverge later at some point. That is, regarding the fault propagation, a nonreconvergent node may behave like a reconvergent node. Therefore the detectability of a bridging fault should be determined by explicit fault simulation. To carry out this procedure for parallel patterns, it is essential to characterize the Boolean function of each fanout cell from the bridged outputs. Such a Boolean function characterizing the faulty behavior of a fanout cell should be derived from the respective generic-bridge-table and generic-cell-tab le symbolically so that its evaluation can be done via bit-vector operation for parallel patterns. Below it is shown how a faulty Boolean function at a fanout cell from the bridged output is derived.

First, let us examine how a faulty Boolean function is constructed. A cell is said having a **faulty-on** behavior if the fault free value is "0" but in case of a bridge is "1". A cell is said having a **faulty-off** behavior in the opposite case. For ease of the discussion, a generic-bridge between B and C and one of the fanout cell A as depicted in Fig. 8(a) are used for illustration.

Let F_B and F_C be the fault free functions of B and C, respectively. The input spaces I, J, and K are independent of each other. The fault free function of A can be viewed as a function of I and J as

$$F_A = \{x \in (I + J) | A \text{ is "on"}\}. \tag{11}$$

Due to the bridge, A becomes a function of inputs I, J and K. Let $\mathcal{B} = I + J + K$. Then, the Boolean function $\tilde{F}_A$ of A in the presence of the bridging fault is defined as

$$\tilde{F}_A = \{x \in \mathcal{B} | A \text{ is "on"}\}. \tag{12}$$

The **faulty-on set** and the **faulty-off set** of A are defined as

$$f_A^1 = \{x \in \mathcal{B} | \overline{F}_A \wedge \tilde{F}_A\} \quad \text{and} \quad f_A^0 = \{x \in \mathcal{B} | F_A \wedge \overline{\tilde{F}}_A\}. \tag{13}$$

The complement of f_A^0 is then obtained as $\overline{f_A^0} = \{x \in \mathcal{B} | \overline{F}_A \wedge \tilde{F}_A\}$.

The set $\mathcal{B}$ is then split into three parts: f_A^0, f_A^1, and the rest of the inputs. On the other hand, obviously $\mathcal{B}$ can also be viewed as the union of F_A and $\overline{F}_A$ considering the inputs in K as "don't cares". Fig. 8(b) illustrates the relation of f_A^1 and f_A^0 with respect to F_A and $\overline{F}_A$. With the above definitions, the following theorem holds.

Theorem: Assume a cell with its fault free function as F_A is affected by a bridge. Let f_A^1 and f_A^0 be the faulty-on set and faulty-off set of the cell. Then

$$\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1. \tag{14}$$

Proof: Equation (12) can be partitioned into two parts

$$\tilde{F}_A = \{x \in \mathcal{B} | \overline{F}_A \wedge \tilde{F}_A\} \cup \{x \in \mathcal{B} | F_A \wedge \tilde{F}_A\}. \tag{15}$$

The first part is exactly the faulty-on set f_A^1. In the part containing original "on" set F_A, except for the inputs in f_A^0, the output A is still "on". Thus the second subset in (15) should be the original "on" set F_A minus the faulty-off set f_A^0 (the shaded part in Fig. 8(b)). Therefore, $\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1$. □

The above theorem implies that once the faulty-on and the faulty-off set of a fanout cell are obtained, they are sufficient to characterize the faulty behavior of this cell. Below it will be shown how the faulty-on and the faulty-off set of a fanout cell are derived from the generic-bridge and generic-cell tables.

For the bridge in Fig. 8(a), let all entries of its generic-bridge-table be represented by a set T_{bri}. For the fanout cell A in Fig. 8(a), let all the entries of its generic-cell-table labeled with a be represented by a set $T_{\text{cell}}(a)$.

The generic-bridge-table T_{bri} can be partitioned into two parts T_{bri}^0 and T_{bri}^1

$$T_{\text{bri}}^0 = \{\langle b, d\rangle \in T_{\text{bri}} | d \Rightarrow \overline{F}_B \wedge d \Rightarrow F_C\} \tag{16}$$
$$T_{\text{bri}}^1 = \{\langle b, d\rangle \in T_{\text{bri}} | d \Rightarrow F_B \wedge d \Rightarrow \overline{F}_C\}. \tag{17}$$

For any $\langle b, d\rangle \in T_{\text{bri}}^0$, it is known that the fault free value of B is "0" $(a = 0)$. Suppose a is observable at the output of A in the presence of the bridge. Obviously the input voltage at a should be higher than the logic threshold voltage of A for the propagation of the bridging fault. Let the Boolean expression representing all the input vectors generating the bridged output higher than a value w be expressed as

$$C^0(w) = \sum_{\langle b,d\rangle \in T_{\text{bri}}^0(w) | b > w} d. \tag{18}$$

Then for any $\langle w, O\rangle_{\{a\}} \in T_{\text{cell}}(a), a$ is observable if O is satisfied. Thus a faulty-off behavior is caused at A for any input vector satisfying $C^0(w) \cdot O$.

TABLE III
SOME CIRCUIT DATA AND RESULTS OF BRIDGE ANALYSIS

circuit	#trans.	#cell	#GC	#bridge	#GB	size(Kb)	time(s)
c432	728	152	18	1025	68	34	7.3
c499	1396	284	9	2625	36	17	2.5
c880	1164	236	20	3254	146	74	26.5
c1355	1768	366	10	3421	44	24	6.4
c1908	2058	411	20	4132	111	51	20.9
c2670	2974	604	31	13483	238	141	58.0
c3540	4122	791	29	14499	283	134	48.4
c5315	6734	1288	36	39412	345	238	91.0
c6288	8464	1848	7	14298	24	10	4.7
c7552	8854	1795	31	51773	309	197	83.0

GC: Generic–Cell; GB: Generic–Bridge.

TABLE IV
REDUCTION OF SPICE SIMULATIONS

circuit	#conducting circuits			#multiple–input threshold		
	actual	total	reduce	actual	enumerate	reduce
c432	458	1237	62.9%	31	120	74.0%
c499	128	275	53.0%	18	34	47.0%
c880	1276	3310	61.4%	68	157	56.7%
c1355	317	569	44.3%	28	58	51.7%
c1908	845	1861	84.6%	52	178	70.8%
c2670	2414	6420	62.4%	82	274	70.1%
c3540	2173	6788	68.0%	107	246	56.5%
c5315	3589	12223	70.6%	131	446	70.6%
c6288	139	224	38.0%	4	12	66.7%
c7552	3369	10110	66.7%	155	368	57.9%

By complementary reasoning, let the Boolean expression representing all the input vectors that generate the bridged output lower than a value w be expressed as

$$C^1(w) = \sum_{\langle b,d\rangle \in T^1_{\text{bri}}(w) | b < w} d. \tag{19}$$

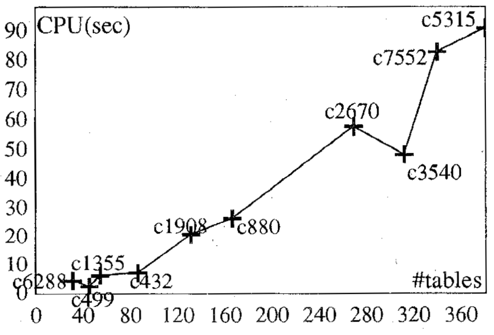
Then for $\langle w, O\rangle_{\{a\}} \in T_{\text{cell}}(a)$, if any input vector satisfies $C^1(w) \cdot O$, a faulty-on behavior is caused at A.

Consider A has more than one logic threshold voltage when a switches, then the final faulty-off and faulty-on sets of A are obtained as

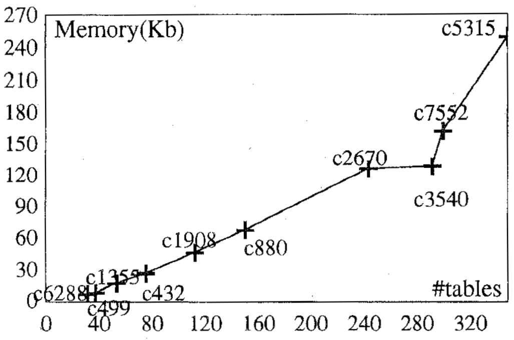
$$f^0_A = \sum_{\langle w,O\rangle_{\{a\}} \in T_{\text{cell}}(a)} C^0(w) \cdot O \tag{20}$$

$$f^1_A = \sum_{\langle w,O\rangle_{\{a\}} \in T_{\text{cell}}(a)} C^1(w) \cdot O. \tag{21}$$

That is, if any input satisfies (20), then the output A has a faulty value "0." Vice versa any input satisfying (21) introduces a faulty "1" at the output A. Therefore, according



(a)



(b)

Fig. 9. (a) Analysis time versus size of tables. (b) Memory requirement versus size of tables.

to the above theorem, the faulty behavior of A is characterized as $\tilde{F}_A = F_A \cdot \overline{f^0_A} + f^1_A$ with f^0_A and f^1_A derived from (20) and (21).

For the case that more than one input of a fanout cell is connected together, the propagation procedure is very similar. The function $\tilde{F}_A$ of each fanout cell can be evaluated. After all the fanout cells are processed, the logic fault simulation can be started from the fanout cells carrying faulty values.

It is not difficult to observe that above formulas can be evaluated for patterns in parallel via bit-vector operations. Thus the whole procedure can be done for parallel patterns.

V. EXPERIMENTAL RESULTS

The whole system is implemented in C on a HP-9000/755 workstation. For experiments, the ISCAS85 benchmark circuits are used. They are implemented in a standard cell design approach for a 2 μm CMOS technology at MCNC (Microelectronics Center of North Carolina's). The cell library consists of both simple (such NAND and NOR) and complex (such as AOI and OAI) cells. The bridging faults are extracted from the circuit layout by using a system described in [20]. For the SPICE simulator, the level 3 MOS SPICE model is used for the analysis.

Table III summarizes some circuit data and the analysis results for the bridging faults. In general, the number of generic-cells in each circuit is far less than the actual cell library. The circuit c6288 having 1848 instantiated cells has only 7 generic-cells. The actual number of generic-bridge tables derived from the extracted bridging faults is also far less

TABLE V
RESULTS OF PPSFP SIMULATION

circuit	SSA test pattern set					21x32 random patterns		
	#pat.	SSA%	bridge%	time(s)	error%	SSA%	bridge%	time(s)
c432	75	99.7	96.9	0.3	0.14	99.7	97.2	0.6
c499	71	100	98.0	0.3	0.04	99.5	98.9	0.6
c880	95	100	99.3	0.8	0.30	98.7	99.2	1.1
c1355	101	100	99.3	0.6	0.25	99.4	99.2	0.9
c1908	147	100	98.4	1.2	0.11	94.5	96.4	1.8
c2670	160	98.8	98.6	2.5	0.39	87.7	96.6	3.3
c3540	242	98.4	99.3	6.0	0.25	98.0	99.2	7.3
c5315	211	100	98.9	7.7	0.29	99.9	98.9	8.8
c6288	44	99.9	99.8	8.5	0.18	99.9	99.9	22.7
c7552	318	99.7	99.4	11.3	0.28	92.9	98.6	12.7

than the number of extracted bridges. It is even less than the number of combinations of the generic-cells for each design. For instance, the circuit c7552 has 51 773 possible bridges but only 309 generic-bridges are derived. The circuit exhibits 31 generic-cells and the number of pairs of those would already amount to 465.

Table IV shows the effectiveness of using the techniques described in Section III. The table shows the total number of conducting circuits caused by the set of generic-bridges in each design. They have to be analyzed by SPICE to compute generic-bridge-tables. The actual number of them after using the reduction techniques in Section 3-1 is also shown. On average 65% of the SPICE computations are bypassed. The dynamic derivation of multiple-input logic threshold voltages also bypasses on average about 65% of the SPICE simulations compared to exhaustive analysis of the set of generic-cells in each design. The table clearly exposes the economy of our methods.

The times listed in Table III are the actual CPU times in seconds used for the complete analysis for each circuit. In Table III the total size of the tables containing both the generic-bridges and the generic-cells is listed for each circuit. Only up to about 240 kbytes are required for the largest circuit. Those numbers are significant for the assessment of the cost incurred by assembling the fault model information from the library data. Both the CPU time and the memory requirement exhibit an almost linear relation with the number of generic-bridges and generic-cells as shown in Fig. 9(a) and (b).

The fault simulation results are shown in Table V. The bridges are simulated for the test pattern sets generated for single stuck-at faults. These test pattern sets are obtained from MCNC. The fault simulation is performed in one run for both stuck-at and bridging faults. The fault coverage for bridging faults is the percentage of detected bridging faults divided by the number of all simulated bridging faults. In general, the bridging fault coverages are slightly lower than the respective single stuck-at fault coverages. The simulation time is very short. The column *errors%* indicates the possible false interpretation percentages during the whole fault simulation. That is, the percentage of the situations where the input is the same as or very close to the cell logic threshold voltage. We choose

$$|V_{\text{input}} - V_{\text{logic threshold}}| \leq 0.02 \text{ V}.$$

V_{input} is the input voltage value and $V_{\text{logic }threshold}$ is the logic threshold voltage. This error is not a substantial problem for this set of benchmarks. The above margin value 0.02 V is chosen hypothetically. In a computational experiment, we choose the margin value as 0.5 V without observing a dramatic decrease of the fault coverage (less than 3%). The simulation time remains almost the same. In practice, the margin value can be chosen according to the actual process parameters.

A set of 21× 32 randomly generated test patterns are also simulated for the bridging faults. The results are shown in Table V as well. The results may indicate that the random testability of the bridging faults is rather good.

VI. CONCLUDING REMARKS

It is hard to make a comparison with other methods since most of those methods do not include the time and memory use of the fault modeling process. The way of selecting the bridging faults, the test pattern sets, design approach and process parameters (SPICE parameters) can make a lot of difference as well. Nevertheless, with the introduction of two new concepts, the generic-bridge and the generic-cell, the paper demonstrates an accurate, fast and memory efficient modeling and simulation method for CMOS bridging faults. Since a SPICE simulator is used and the multiple-input logic threshold voltages are considered, the method is much more accurate than any other approximation method. The idea can be easily used for intracell bridging faults.

This paper does not include the analysis of feedback bridging faults. This kind of bridging fault may cause a circuit to oscillate. In this situation, it is difficult to decide if a bridge is actually observable. Our recent work [21] shows that feedback bridging faults may or may not oscillate under certain input conditions. An approach has been developed to identify oscillatory and nonoscillatory situations and resolve the modeling problem by using generic-bridge and generic-cell tables. This approach again has circuit-level accuracy and can lead to reliable simulation results. A clear document of the associated theory, however, requires more space. For more detail, we refer to [21].

References

[1] Y. K. Malaiya and S. Y. H. Su, "A new fault model and testing technique for CMOS devices," in *Proc. Int. Test Conf.*, 1982, pp. 25–34.

[2] R. E. Bryant and M. D. Schuster, "Fault simulation of MOS circuits," *VLSI Design*, vol. 4, no. 6, pp. 24–30, Oct. 1983.

[3] D. Saab and I. Hajj, "Parallel and concurrent fault simulation of MOS circuits," in *Proc. Int. Conf. Comput. Design*, pp. 752–756, 1984.

[4] J. M. Acken, "Driving accurate fault models," Comput. Syst. Lab., Standford Univ., pp. CSL-TR-88-365, Oct. 1985.

[5] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and Th. McCarthy, "Fault simulation for structured VLSI," *VLSI Syst. Design*, pp. 20–32, Dec. 1985.

[6] W. Maly, "Realistic fault modeling for VLSI testing," in *Proc. 24th Design Automatic Conf.*, 1987, pp. 173–180.

[7] F. J. Ferguson, and J. P. Shen, "A CMOS fault extractor for inductive fault analysis," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1181–1194, Nov. 1988.

[8] J. M. Soden and C. F. Hawkins, "Electrical properties and detection methods for CMOS IC defects," in *Proc. Euro. Test Conf.*, 1989, pp. 159–167.

[9] K. S. Bracs, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 40–45.

[10] T. M. Storey and W. Maly, "CMOS bridging fault detection," in *Proc. Int. Test Conf.*, 1990, pp. 842–851.

[11] S. D. Millman, and J. P. Garvey, "An accurate bridging fault test pattern generation," in *Proc. Int. Test Conf.*, 1991, pp. 411–418.

[12] F. J. Ferguson and T. Larrabee, "Test pattern generation for realistic bridge faults in CMOS IC's," in *Proc. Int. Test Conf.*, 1991, pp. 492–499.

[13] G. S. Greenstein and J. H. Patel, "E-PROOFS: A CMOS bridging fault simulator," in *Proc. Int. Conf. Comput.-Aided Design*, pp. 1992.

[14] J. M. Acken and S. T. Millman, "Fault model evolution for diagnosis: Accuracy vs precision," in *Proc. Custom Integrat. Circuits Conf.*, 1992, pp. 13.4.1-13.4.4.

[15] C. Di and J. Jess, "On CMOS bridge fault modeling and test pattern evaluation," in *Proc. 11th IEEE VLSI Test Symp.*, 1993, pp. 116–119.

[16] B. Chess and T. Larrabee, "Bridge fault simulation strategies for CMOS integrated circuits," in *Proc. 30th ACM/IEEE Design Automation Conf.*, 1993, pp. 1503–1507.

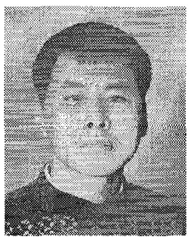
[17] J. Rearick and J. H. Patel, "Fault and accurate CMOS bridging fault simulation," in *Proc. Int. Test Conf.*, 1993, pp. 54–62.

[18] P. C. Maxwell and R. Aitken, "Biased voting: A method for simulating CMOS bridging faults in the presence of variable gate logic thresholds," in *Proc. Int. Test Conf.*, 1993, pp. 63–72.

[19] C. Di and J. Jess, "On the development of a fast and accurate bridging fault simulator," Tech. Res. Rep. EUT Rep. 93-E-277, ISBN 90-6144-277-1, 1993.

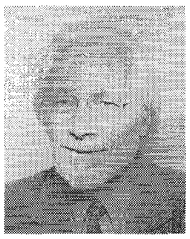
[20] H. Xue, C. Di, and J. A. G. Jess, "A net-oriented method for realistic fault analysis," *IEEE/ACM Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 78–83.

[21] C. Di and J. Jess, "On accurate and reliable testing of CMOS bridging faults," Tech. Rep., Design Automation Section, Eindhoven Univ. Technol., 1995.



Chennian Di was born in Xi'an, China. He received the B.S. degree in information engineering from Xidian University, China, the M.S. degree in electrical engineering, and the Ph.D. degree from Eindhoven University of Technology, The Netherlands, in 1985, 1988, and 1995, respectively.

Since 1989, he was a Research Assistant with the Design Automation group, Eindhoven University of Technology. He is now with IBM Microelectronics Division. He is interested in various aspects of computer-aided design and testing of VLSI circuits. Currently, his focus is on microprocessor testing and diagnosis.



Jochen A. G. Jess was born on April 13, 1935 in Dortmund, Germany. He received the M.S. degree from the Rheinish-Westfalische Technische Hochschule Aachen, Germany, in 1960, and the Ph.D. degree from the Aachen University of Technology, Germany, in 1963.

From 1963 to 1968, he was a Research Staff Member with the Institute Fur Nachrichtensystem, Karlsruhe University of Technology. From 1968 to 1969, he was a Visiting Professor with the Department of Electrical Engineering, University of Maryland. From 1969 to 1971, he was a Senior Staff Member with the Karlruhe University of Technology. Since 1971, he has been the Professor and Head of the Design Automation Section with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. His current interests are in the design and automation of integrated circuits, in particular layout design, logic design, design of architectures, and formal verification. He is the coauthor of more than 75 papers.

Dr. Jess is a member of the Board of the European Design and (Design) Automation Association (EDAA) and he has served as a Program and General Chair for ICCAD'93 and ICCAD'94.