An efficient CMOS bridging fault simulator with SPICE accuracy
Di, C.; Jess, J.A.G.

Published in:
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

DOI:
10.1109/43.536713

Published: 01/01/1996

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
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 ISCASS5 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 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 conducance is stronger than the pull-up conducance. 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 1
bridged output obtained by spice

<table>
<thead>
<tr>
<th>inputs</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c d e f</td>
<td>A B (brided(V))</td>
</tr>
<tr>
<td>1 1 1 1 0 1</td>
<td>0 1 0.63</td>
</tr>
<tr>
<td>1 0 1 1 1 0</td>
<td>1 0 1.42</td>
</tr>
<tr>
<td>0 0 1 1 1 1</td>
<td>1 0 2.15</td>
</tr>
<tr>
<td>1 1 1 1 0 0</td>
<td>0 1 2.47</td>
</tr>
<tr>
<td>0 1 0 1 1 1</td>
<td>1 0 3.35</td>
</tr>
<tr>
<td>0 0 0 0 1 1</td>
<td>1 0 4.71</td>
</tr>
</tbody>
</table>

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 abedef = 100111 and abedf = 110111 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. Yet a large amount of computation is avoided.

For each generic-bridge, a generic-bridge-table is introduced for the set B of all input vectors of the two bridged cells that activate the bridge. A generic-bridge-table consists of a set of pairs (b, d) as its entries. Let $T_{brid}$ be a set denoting all the entries. For each (b, d) $\in T_{brid}$, 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 $B$. For any two $(b_1, d_1)$ and $(b_2, d_2)$ in $T_{brid}$, 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 "·" interpreted as addition and multiplication of real numbers as well

$$V_{brid} = b_1 \cdot d_1 + \ldots + b_n \cdot d_m$$  \hspace{1cm} (1)

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

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

$$V_{brid} = 0.00 \cdot (c \oplus f) \cdot a \cdot b \cdot c \cdot d + 1.42 \cdot (c \oplus f) \cdot (a \oplus b) \cdot c \cdot d$$
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 $T_{cell}(l)$ denote the union of all labeled pairs $(w,O)_l$ when the terminals $l$ are driven by the same signal

$$T_{cell}(l) = \bigcup_{i=1}^{m} (w_i,O_l)_l.$$ (3)

Then, for any two $(w_1,O_1)_l$ and $(w_2,O_2)_l$ in $T_{cell}(l)$, 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

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

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

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

$$V_{cell} = (1.89 \cdot b)_{(a)} (2.20 \cdot a)_{(b)} + (2.60)_{(a,b)}.$$ (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
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-tables 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 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 \( V_{th,low} \) or below 1 V (which may be considered as the highest “hard” logic “0” \( V_{th,high} \) ) 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

\[
\begin{cases}
I_{ds} &= k_n \frac{W}{L} (V_{gs} - V_{th} - \frac{1}{2} V_{ds}) V_{ds}, \quad V_{ds} < V_{gs} - V_{th} \\
I_{ds} &= 0, \quad \text{otherwise}
\end{cases}
\]

(PMOS transistors are handled analogously). Here \( k_n \) is process dependent, \( V_{th} \) 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
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 $V_{dd} < V_{gs} - V_{tn}$ 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)

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

(7)

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

$$\beta > \beta_{hard} \Rightarrow V_F > V_{hard} \text{ and } \beta < \beta_{hard}^0 \Rightarrow V_F < V_{hard}^0$$

(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 $\beta$ is needed. Consequently the estimation is very fast. The computation of the equivalent $\beta$ 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.

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 I1. 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 I1 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

\[
\begin{array}{c|c}
 a & b & c & d & e & \text{bridged output} \\
 \hline
 0 & 1 & 1 & 0 & 1 & 1.42V \\
 0 & 1 & 1 & 1 & 0 & 1.42V \\
 1 & 0 & 1 & 1 & 0 & 1.42V \\
 1 & 0 & 1 & 1 & 0 & 1.42V \\
\end{array}
\]
TABLE II
MULTIPLE-INPUT LOGIC THRESHOLDS

<table>
<thead>
<tr>
<th>input a</th>
<th>input a &amp; b</th>
<th>input a &amp; b &amp; c</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_threshold</td>
<td>b c d</td>
<td>V_threshold</td>
</tr>
<tr>
<td>2.08V</td>
<td>1 0 1</td>
<td>1.89V</td>
</tr>
<tr>
<td>2.15V</td>
<td>1 0 0</td>
<td>2.11V</td>
</tr>
<tr>
<td>2.17V</td>
<td>0 0 1</td>
<td>2.62V</td>
</tr>
</tbody>
</table>

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 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 \((f_2 = \overline{a} \cdot b)\) and \((f_3 = a \cdot \overline{b} \cdot \overline{c} \cdot d)\) of B are required to construct the tables. Their ROBDD representations are shown in Fig. 7(b). The generic-bridge-tables are obtained as:

\[
V_{br1} = 1.35 \cdot \overline{c} \cdot f_1 + 1.57 \cdot e \cdot f_3 + 3.39 \cdot e \cdot f_2 \quad (9)
\]

\[
V_{br2} = 1.45 \cdot g_3 \cdot f_1 + 2.67 \cdot g_2 \cdot f_1 + 1.89 \cdot g_1 \cdot f_3 \\
+ 3.45 \cdot g_1 \cdot f_2. \quad (10)
\]

Here, \(g_1 = c \cdot d\) is the pull-down term of C and \(g_2 = \overline{c} \cdot d\) 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 \(V_{br2}\). 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
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-table 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

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

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

\[
\tilde{F}_A = \{ x \in B | A \text{ is "on"} \}. \quad (12)
\]

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

\[
f^1_A = \{ x \in B | F_A \land \bar{F}_A \} \quad \text{and} \quad f^0_A = \{ x \in B | F_A \land \bar{F}_A \}. \quad (13)
\]

The complement of \( f^0_A \) is then obtained as \( \bar{f}^0_A = \{ x \in B | F_A \land \bar{F}_A \} \).

The set \( B \) is then split into three parts: \( f^0_A, f^1_A, \) and the rest of the inputs. On the other hand, obviously \( B \) can also be viewed as the union of \( F_A \) and \( \bar{F}_A \) considering the inputs in K as “don’t cares”. Fig. 8(b) illustrates the relation of \( f^1_A \) and \( f^0_A \) with respect to \( F_A \) and \( \bar{F}_A \). 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 \( f^1_A \) and \( f^0_A \) be the faulty-on set and faulty-off set of the cell. Then

\[
\tilde{F}_A = F_A \cdot f^1_A + f^0_A. \quad (14)
\]

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

\[
\tilde{F}_A = \{ x \in B | F_A \land \bar{F}_A \} \cup \{ x \in B | F_A \land \bar{F}_A \}. \quad (15)
\]

The first part is exactly the faulty-on set \( f^1_A \). In the part containing original “on” set \( F_A \), except for the inputs in \( f^0_A \), 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 \( f^0_A \) (the shaded part in Fig. 8(b)). Therefore, \( \tilde{F}_A = F_A \cdot f^0_A + f^1_A \). \( \square \)

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 \( T_{\text{br}} \). 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 \( T_{\text{br}} \) can be partitioned into two parts \( T^0_{\text{br}} \) and \( T^1_{\text{br}} \)

\[
T^0_{\text{br}} = \{ (b,d) \in T_{\text{br}} | d \Rightarrow F_B \land d \Rightarrow F_C \} \quad (16)
\]

\[
T^1_{\text{br}} = \{ (b,d) \in T_{\text{br}} | d \Rightarrow F_B \land d \Rightarrow \bar{F}_C \}. \quad (17)
\]

For any \( (b,d) \in T^0_{\text{br}} \), 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

\[
C^0(w) = \sum_{(b,d) \in T^0_{\text{br}}(w)} b > w. \quad (18)
\]

Then for any \( (w,O) \in T_{\text{cell}}(a) \), \( a \) 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

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#trans.</th>
<th>#cell</th>
<th>#GC</th>
<th>#bridge</th>
<th>size(Kb)</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>728</td>
<td>152</td>
<td>18</td>
<td>1025</td>
<td>68</td>
<td>34</td>
</tr>
<tr>
<td>c499</td>
<td>1396</td>
<td>284</td>
<td>9</td>
<td>2625</td>
<td>36</td>
<td>17</td>
</tr>
<tr>
<td>c880</td>
<td>1164</td>
<td>236</td>
<td>20</td>
<td>3254</td>
<td>146</td>
<td>74</td>
</tr>
<tr>
<td>c1355</td>
<td>1768</td>
<td>366</td>
<td>10</td>
<td>3421</td>
<td>44</td>
<td>24</td>
</tr>
<tr>
<td>c1908</td>
<td>2058</td>
<td>411</td>
<td>20</td>
<td>4132</td>
<td>111</td>
<td>51</td>
</tr>
<tr>
<td>c2670</td>
<td>2974</td>
<td>604</td>
<td>31</td>
<td>13483</td>
<td>238</td>
<td>141</td>
</tr>
<tr>
<td>c3540</td>
<td>4122</td>
<td>791</td>
<td>29</td>
<td>14499</td>
<td>283</td>
<td>134</td>
</tr>
<tr>
<td>c5315</td>
<td>6734</td>
<td>1288</td>
<td>36</td>
<td>39412</td>
<td>345</td>
<td>238</td>
</tr>
<tr>
<td>c6288</td>
<td>8464</td>
<td>1848</td>
<td>7</td>
<td>14298</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>c7552</td>
<td>8854</td>
<td>1795</td>
<td>31</td>
<td>51773</td>
<td>309</td>
<td>197</td>
</tr>
</tbody>
</table>

GC: Generic-Cell; GB: Generic-Bridge.

TABLE IV
REDUCTION OF SPICE SIMULATIONS

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#actual</th>
<th>#reduce</th>
<th>#reduce</th>
<th>#actual</th>
<th>#reduce</th>
<th>#reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>458</td>
<td>1237</td>
<td>62.9%</td>
<td>31</td>
<td>120</td>
<td>74.0%</td>
</tr>
<tr>
<td>c499</td>
<td>128</td>
<td>275</td>
<td>53.0%</td>
<td>18</td>
<td>34</td>
<td>47.0%</td>
</tr>
<tr>
<td>c880</td>
<td>1276</td>
<td>3310</td>
<td>61.4%</td>
<td>68</td>
<td>157</td>
<td>56.7%</td>
</tr>
<tr>
<td>c1355</td>
<td>317</td>
<td>569</td>
<td>44.3%</td>
<td>28</td>
<td>58</td>
<td>51.7%</td>
</tr>
<tr>
<td>c1908</td>
<td>845</td>
<td>1861</td>
<td>84.6%</td>
<td>52</td>
<td>178</td>
<td>70.8%</td>
</tr>
<tr>
<td>c2670</td>
<td>2414</td>
<td>6420</td>
<td>62.4%</td>
<td>82</td>
<td>274</td>
<td>70.1%</td>
</tr>
<tr>
<td>c3540</td>
<td>2173</td>
<td>6788</td>
<td>68.0%</td>
<td>107</td>
<td>246</td>
<td>56.5%</td>
</tr>
<tr>
<td>c5315</td>
<td>3588</td>
<td>12223</td>
<td>70.6%</td>
<td>131</td>
<td>446</td>
<td>70.6%</td>
</tr>
<tr>
<td>c6288</td>
<td>139</td>
<td>224</td>
<td>38.0%</td>
<td>4</td>
<td>12</td>
<td>66.7%</td>
</tr>
<tr>
<td>c7552</td>
<td>3369</td>
<td>10110</td>
<td>66.7%</td>
<td>155</td>
<td>368</td>
<td>57.9%</td>
</tr>
</tbody>
</table>

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

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

\[
C^1(w) = \sum_{(b,d) \in \mathcal{T}^{1}(w) | b < w} d, \quad (19)
\]

Then for \( (w, O) \in \mathcal{T}_{\text{cel}}(a) \), 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

\[
f_A^0 = \sum_{(w,O),(a) \in \mathcal{T}_{\text{cel}}(a)} C^0(w) \cdot O \quad (20)
\]

\[
f_A^1 = \sum_{(w,O),(a) \in \mathcal{T}_{\text{cel}}(a)} C^1(w) \cdot O. \quad (21)
\]

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 to the above theorem, the faulty behavior of \( A \) is characterized as \( \hat{F}_A = F_A^1 \cdot f_A^0 + F_A^0 \cdot f_A^1 \) with \( f_A^0 \) and \( f_A^1 \) 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 \( \hat{F}_A \) 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 \( \mu \)m 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.
DI AND JESS: CMOS BRIDGING FAULT SIMULATOR

TABLE V
RESULTS OF PPSFP SIMULATION

<table>
<thead>
<tr>
<th>circuit</th>
<th>#pat.</th>
<th>SSA%</th>
<th>bridge%</th>
<th>time(s)</th>
<th>#pat.</th>
<th>SSA%</th>
<th>bridge%</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>75</td>
<td>99.7</td>
<td>96.9</td>
<td>0.3</td>
<td>14</td>
<td>99.7</td>
<td>97.2</td>
<td>0.6</td>
</tr>
<tr>
<td>c499</td>
<td>71</td>
<td>100</td>
<td>98.0</td>
<td>0.3</td>
<td>04</td>
<td>99.5</td>
<td>98.9</td>
<td>0.6</td>
</tr>
<tr>
<td>c880</td>
<td>95</td>
<td>100</td>
<td>99.3</td>
<td>0.8</td>
<td>30</td>
<td>98.7</td>
<td>99.2</td>
<td>1.1</td>
</tr>
<tr>
<td>c1355</td>
<td>101</td>
<td>100</td>
<td>99.3</td>
<td>0.6</td>
<td>25</td>
<td>99.4</td>
<td>99.2</td>
<td>0.9</td>
</tr>
<tr>
<td>c1908</td>
<td>147</td>
<td>100</td>
<td>98.4</td>
<td>1.2</td>
<td>11</td>
<td>94.5</td>
<td>96.4</td>
<td>1.8</td>
</tr>
<tr>
<td>c2670</td>
<td>160</td>
<td>98.8</td>
<td>98.6</td>
<td>2.5</td>
<td>39</td>
<td>87.7</td>
<td>96.6</td>
<td>3.3</td>
</tr>
<tr>
<td>c3540</td>
<td>242</td>
<td>98.4</td>
<td>99.3</td>
<td>6.0</td>
<td>25</td>
<td>98.0</td>
<td>99.2</td>
<td>7.3</td>
</tr>
<tr>
<td>c5315</td>
<td>211</td>
<td>100</td>
<td>98.9</td>
<td>7.7</td>
<td>29</td>
<td>99.9</td>
<td>98.9</td>
<td>8.8</td>
</tr>
<tr>
<td>c6288</td>
<td>44</td>
<td>99.9</td>
<td>99.8</td>
<td>8.5</td>
<td>18</td>
<td>99.9</td>
<td>99.9</td>
<td>22.7</td>
</tr>
<tr>
<td>c7552</td>
<td>318</td>
<td>99.7</td>
<td>99.4</td>
<td>11.3</td>
<td>28</td>
<td>92.9</td>
<td>98.6</td>
<td>12.7</td>
</tr>
</tbody>
</table>

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_{\text{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 21x 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


