1988

A Fast Fault Simulation Algorithm for Combinational Circuits

Wuudiann Ke
University of Nebraska-Lincoln

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Bhargab B. Bhattacharya
Indian Statistical Institute Calcutta, India

Digest of Technical Papers., IEEE International Conference on Computer-Aided Design, 1988. ICCAD-88.
doi: 10.1109/ICCAD.1988.122486

A Fast Fault Simulation Algorithm for Combinational Circuits

Wuudiann Ke & Sharad Seth
Department of Computer Science
University of Nebraska, Lincoln, NE 68588-0115

Bhargab B. Bhattacharya
Indian Statistical Institute
Calcutta, India

ABSTRACT

The performance of a fast fault simulation algorithm for combinational circuits, such as the critical path tracing method, is determined primarily by the efficiency with which it can deduce the detectability of stem faults (stem analysis). We propose a graph based approach to perform stem analysis. A dynamic data structure, called the criticality constraint graph, is used during the backward pass to carry information related to self masking and multiple-path sensitization of stem faults. The structure is updated in such a way that when stems are reached their criticality can be found by looking at the criticality constraints on their fanout branches. Compared to the critical path tracing method, our algorithm is exact and does not require forward propagation of individual stem faults. Several examples are given to illustrate the power of the algorithm. Preliminary data on an implementation is also provided.

1. INTRODUCTION

Ideally, a fast fault simulation algorithm should be able to complete its job in two passes: logic simulation of the (good) circuit in a forward walk and line criticality determination in a backward walk through the circuit. The first of these, logic simulation is straightforward and takes linear time in the size of the circuit. The second pass would also be straightforward if the circuit had no reconvergent fanout stems. The criticality of such stems can not be deduced directly from the criticalities of is fanout branches (FOB's). Due to *self masking* (that is, cancellation of the effect of a stem fault propagating along multiple paths at a reconvergent gate) the stem may be non-critical when one or more of its FOB's are critical. Conversely, a stem fault may be detectable only because its effect propagates through a reconvergent gate along more than one path (*multiple path sensitization*). In this case, the stem is critical while its FOB's may all be non-critical.

The stem analysis carried out in critical path tracing [1] has two characteristic aspects, each of which has its own drawbacks as noted below:

> Dynamic memory management is avoided by fault-simulating stems serially; for many circuits stems constitute a significant fraction of the total number of lines hence this solution could be quite expensive.
>
> The number of stems that must be fault-simulated is minimized by making the simplifying assumption that a stem is non-critical whenever all its FOB's are non-critical. This simplification sacrifices the exactness of the fault simulation algorithm.

The algorithm reported here is similar to critical path tracing with one major difference: *it integrates the process of determining the non-stem and the stem line criticalities in a single backward walk of the circuit.* This is achieved by the introduction of a dynamic data structure, called the criticality constraint graph (CCG), which carries enough information, along with the line criticalities, to allow determination of a stem's criticalities from its FOB's. That is, we pay the price of dynamic data management but avoid separate and individual consideration of stems. While the algorithm is not as simple as critical path tracing, we expect it to run faster and yet produce exact results. Other recent proposals for speeding up fault simulation have dealt with the problem of reducing the amount of computation required in stem fault propagation [2,3].

The following sections give details of the graph (CCG) notation, rules for its construction and manipulation, and the algorithm for fault simulation. The algorithm is illustrated with several small examples. We include preliminary data from an implementation currently underway.

2. THE CRITICALITY CONSTRAINT GRAPH

The nodes in the criticality constraint graph (CCG) represent lines in the circuit. They are dynamically labeled with the criticality values (C for critical and N non-critical) of the corresponding lines (recall that a line l with the good-circuit value v is critical if and only if the fault l *stuck at* $\bar{v}$ is detectable at a circuit output.) The non-critical values are further divided into two classes depending on whether or not the effect of a *preceding* stem fault reaching the line in question can be blocked at a subsequent gate. If the effect is known to be blocked we call it a *negatively non-critical* (NN) value, otherwise, it is called a *positively non-critical* (PN) value.

The directed edges in a CCG, denoting criticality constraints between lines, come in two flavors as well. The first type denotes the situation of fault-effect cancellation, e.g., at the input of the AND gate shown in Fig. 1 (a). The effect of a stem fault arriving at input A would propagate to the output only if it does not reach input B also. In such a case we say that B *cancels* A and show it by an edge directed from B to A. Fig. 1 (b) depicts a different situation. Here, the stem fault-effect will propagate to the

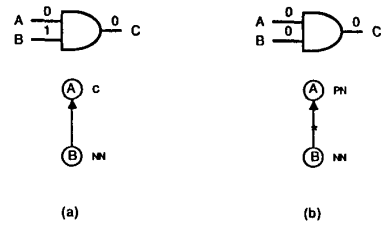

Figure 1

gate output only if it reaches both A and B. In this case, we arbitrarily mark one node (say B) with the criticality NN, the other node (A) with the criticality PN, and draw an *enabling* type of edge from B to A (enabling arcs are shown starred in the graph.) The interpretation is as follows: the stem fault-effect arriving at A is propagated if and only if B enables A. Note that B would enable A if and only if the stem fault-effect also reaches B. Because of the symmetry, we could have exchanged the roles of A and B in the above discussion. Our fault simulation algorithm will produce identical results in either case.

In the CCG we define a node to be *free* if there are no edges (enabling or cancelling) pointing to it. A node is said to be *independent* if there are no edges pointing to or away from it, that is, the node is isolated. An independent node is also free but the reverse is not always true.

For a fault *f*, the *reachability function* R(N) is true for node N in the CCG if the effect of *f* can reach the line represented by node N. We can use the CCG to determine if the fault effect would propagate to a circuit output via N. To this end, we define the *influence expression* of a node N as follows:

> If N is an independent node then IE(N) = R(N), otherwise, let N_1, ..., N_p be the nodes that cancel N and N_{p+1}, ..., N_q be the nodes that enable N. Then

$$IE(N) = R(N) \wedge \left[\sim(IE(N_1) \vee \cdots \vee IE(N_p))\right] \wedge \left[IE(N_{p+1}) \vee \cdots \vee IE(N_q)\right]$$

Note that IE is defined recursively but we maintain the CCG in such a way that a cycle is never created. Thus there is no circularity in the above definition.

The backward walk of the circuit in our algorithm starts at the primary output and proceeds in a breadth-first fashion towards the primary inputs. Thus no gate input is processed before the gate output and no fanout stem is considered before all its FOB's have been considered. Initially, the CCG consists of just the independent primary output nodes each of which is assigned the value C. There are no constraints (edges) in the graph at this point since each output is unconditionally observable. As the walk proceeds, the CCG is dynamically updated.

There are two aspects to the dynamic adjustment of CCG. First, as we proceed from the output of a gate towards its inputs, we must create new nodes for the input lines, assign them the correct criticalities, and *move* the constraints from the gate output to its inputs. Second, when we go back from FOB's to a fanout stem, the stem's criticality must be correctly determined and the CCG must be adjusted so that the walk could proceed from the stem. The first aspect refers to processing of fanout-free regions [4] of the circuit; during this time new nodes and edges are added while some old ones are deleted. Typically, however, the CCG grows in size while going through a fanout-free region (FFR). On the other hand, rules for propagating from FOB's to stems have the effect of generally reducing the size of the CCG. The specific rules for updating CCG's in FFR's and for stems are discussed in the next two sections.

3. BACK PROPAGATION THROUGH FANOUT-FREE REGIONS

3.1. Determining Line Criticalities

The fault simulation algorithm must assign criticalities to the inputs of a logic gate knowing the criticality of the gate output, the gate type, and the signal values at the gate. The rules for this computation will be discussed for a two-input AND gate. These are easily generalized to other gate types and for more than two inputs. A summary of the rules appears in the table below. These will now be further explained.

Back Propagation of Criticalities through a Two-input AND

Input Values		Input Criticalities when the Output is:		
A	B	C	PN	NN
0	0	(NN,PN)*	(NN,PN)*	(NN,NN)
0	1	(C,NN)	(PN,NN)	(NN,NN)
1	0	(NN,C)	(NN,PN)	(NN,NN)
1	1	(C,C)	(PN,PN)	(NN,NN)

* These values can be interchanged.

The simplest situation occurs when the gate output is marked as negatively non-critical (NN). In this case no stem fault-effect can propagate through this gate and the inputs are also marked as NN.

Next, assume the output of the AND gate is critical (C). If both its inputs are zero then we assign the criticalities NN and PN to the two inputs and draw an enabling edge between them as described in the last section. The case of two opposite input values was also discussed in the last section. Assume, for example, that input A is one and input B is zero. A fault effect reaching A can not possibly change the output independent of whether it reaches B or not. Thus A is marked negatively non-critical (NN). On the other hand, a fault effect reaching B would propagate to the output unless it is cancelled by A. Thus B is marked as critical (C) with a cancelling edge coming from A. Lastly, if both inputs are one, the fault effect arriving at either input would unconditionally reach the output. Thus both inputs are marked as critical (C).

Finally, assume the output of the AND gate is positively non-critical (PN), that is, the output itself is non-critical yet the effect of a multiply-sensitized fault effect may propagate through this gate. It is easily seen that this case is very similar to the previous one, with PN replacing the role of C as shown in the table.

3.2. Updating CCG

In going from the output to the inputs of a gate, the CCG is updated as follows. First, new input nodes are created, their criticalities are determined, and enabling or cancelling edges are introduced between them as described in the previous section (exception: if the output node is independent and NN then no edges between input nodes need to be introduced). Next, the gate output node is deleted and the edges incident on it are moved to the gate input node(s) marked as C or PN. The specific set of rules for a two-input AND gate are shown in Fig. 2 where shaded arrows are used to represent the collection of edges incident on node c. In the figure G_{-c} represents the graph (before updating) with node c deleted. There are two cases distinguished in Figs. 2(c) and 2(c'). In both cases *a* and *b* are zero but in 2(c) there is no enabling link to node c. If there is an enabling link to c (case 2(c')), we trace chains of enabling links terminating at node c and attach an enabling link from a to each node which is at the beginning of a chain. It can be verified that these rules correctly translate enabling and cancelling constraints from the output to the inputs of a gate. The rules for other gate types can be easily derived in the same manner. Also, the rules for a multiple input gate can be derived by treating it as a cascade of two-input gates. The result may depend on the different ways in which a cascade connection can be formed, however, the different CCG's are equivalent in terms of their constraints.

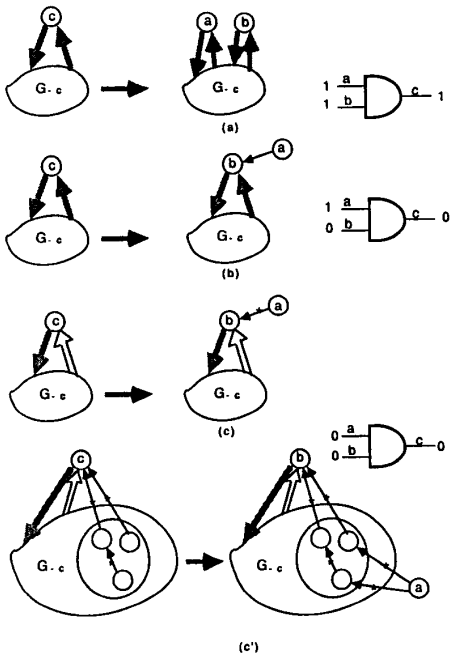

Figure 2

4. STEM ANALYSIS

A stem can be processed as soon as all its FOB's have been reached through the FFR processing described in the last section. It is easily verified that by the time a stem S is processed, the branches of S are the only nodes in the CCG which can be reached from S. The main part of stem analysis has to do with checking the branch nodes for applicability of certain rules and performing rule-dependent reductions of the graph. A skeleton algorithm for stem analysis is as follows:

Stem Analysis - Skeleton Algorithm

```
Apply Rule R1;
do until none of R2 through R4 are applicable;
    Apply Rule R2;
    Apply Rule R3;
    Apply Rule R4;
Apply Rule R5;
```

where, R1 through R5 are reduction rules for branches of S described in the table below:

These rules may be rigorously proved (see [5]) to preserve the constraints described by the CCG. We omit the proofs here for lack of space.

These rules are repeatedly applied to all the stems until no further reduction of CCG is possible. At this point, some stems may have their criticality assigned according to Rule R1, R2, or R4. For the other stems, we must apply the following procedure on the reduced CCG to complete the stem analysis. We assume S to be the stem whose criticality needs to be determined and B(S) is the set of its FOB's.

CCG Reduction Rules for the Branches of Stem S		
Rule	**Condition**	**Action**
R1	S is a PI	Determine criticality of S; Remove all branches of S
R2	A branch of S is free and critical	Remove all branches of S; Insert independent critical node S in CCG.
R3.1	A branch b1 of S is free, noncritical, and cancels another branch b2 of S.	Remove b2.
R3.2	A branch b1 of S is free, noncritical, and enables another branch b2 of S.	Remove all enabling links to b2.
R4	A critical branch of S has cancelling links coming only from other critical branches of S.	Remove all branches of S. Insert independent critical node S in CCG.
R5	There are multiple free NN branches of S.	Combine these nodes and form a single free NN node.

Step 1: Set the reachability function R(b) to be true for each branch b in B(S).

Step 2: If there is any branch b in B(S) labeled C or PN such that its influence function IE(b) is true then stem S is critical otherwise it is non-critical.

To propagate the constraints to S from its branches, we examine the reduced CCG. First, if only one of the branches of S survives the reduction process, its criticality and constraints are transferred to S. The branch can then be deleted from the CCG. Next, if there are two or more branches of a stem that survive we create a single ***supernode*** of their combination after deleting independent nodes (if any). This supernode inherits all the edges incident on its constituent nodes as well as the criticality of S.

The complete fault simulation algorithm, based on the ideas introduced above, can be described at a high level as follows.

5. FAULT SIMULATION ALGORITHM AND EXAMPLES

The algorithm contains the following steps:

Step 1: Read in the circuit description.

Step 2: If there are no input vectors then stop, otherwise, read a (binary) vector and do the true-value simulation. Insert all the POs in the CCG with label C.

Step 3: Assign line criticalities and update CCG in the FFR. If CCG contains only PIs goto step 2, or if CCG contains only NN nodes, label the remaining circuit lines NN and goto step 2.

Step 4: Do stem analysis and reduce CCG.

Step 5: Assign criticalities to stems and propagate CCG through them.

Step 6: If some lines are not yet assigned criticalities, goto step 3, otherwise goto step 2.

We will consider two examples to show how the CCG is modified during the algorithm. The first example illustrates the process of updating the criticality constraint graph (CCG) for a simple circuit. The second example is a 3-input exclusive-or circuit. It shows how complex masking relationships can be captured quite simply by the CCG. This class of circuits is known to be difficult for fault simulation.

Example 1: Fig. 3 shows a part of a circuit connected to the two outputs. A sequence of four graphs represent the CCG for this circuit during different stages of back propagation. Initially, the graph contains just the isolated primary output vertices which

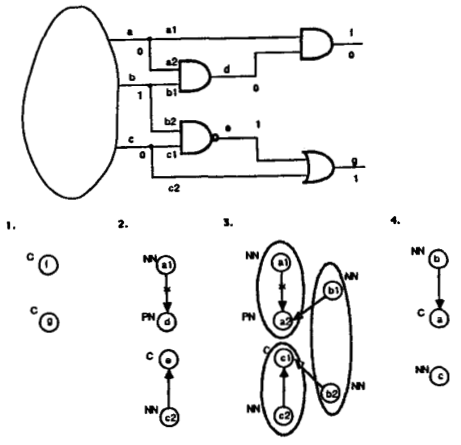

Figure 3

are labeled as critical. The second graph shows the CCG after the two output gates have been processed. The FFR propagation stops at the FOB's and the CCG at this stage is shown in the third graph which becomes the starting point for the stem analysis discussed in Section 4. The rule R3.1 is applicable to node c2 as the canceling node; the result is the deletion of node c1 and its two incoming edges. Also, the rule R3.2 is applicable to a1 as the enabling node, and results in the deletion of the enabling link from a1. No further reduction is possible. Next, we determine the criticality of the three stem lines. The effect of a fault on stem a reaches only a1 and a2. Of these a2 is PN and not canceled by b1 (since the the reachability R(b1) is false), hence the stem a is marked as critical. Both the branches of b are marked as NN so b should also be NN. Finally, only the single independent branch node c2 survives the reduction process for the FOB's of c which inherits its criticality. No supernodes need to be created for propagation of constraints from the branches to the respective stems in this example and the CCG at the three stems is shown in the last graph in the figure.

Example 2: Fig. 4 shows a subcircuit with two exclusive-or gates in series. The CCG for the L1 interface is shown in the first graph. Both the nodes h and c are represented by supernodes and are critical. Suppose a fault effect from a preceding stem arrives at node h but not at node c. The canceling edge from c' to h is ineffective hence the effect will be propagated to the circuit output. On the other hand, if the effect arrives at both the nodes, mutual cancellation occurs and the effect can not propagate to the primary output.

After back propagating through the second exclusive-or we will get the CCG shown in the second graph for the interface L2. It is interesting to see how the CCG captures the notion of masking for this example. Suppose, a fault effect arrives only at node a. Since a is critical and not canceled by b or c, we can conclude that the effect will propagate to the primary output. If, on the other hand, effect arrives at a and b but not at c, the critical nodes a and b are cancelled respectively by b'' and a''. Thus the effect is masked. Finally, suppose the effect arrives at all the three nodes. In this case, a' and b' are canceled respectively by b'' and a''. Therefore, they can not cancel c which is a critical node. Thus the fault will be detectable.

6. PRELIMINARY RESULTS

An implementation of the fault simulation algorithm described

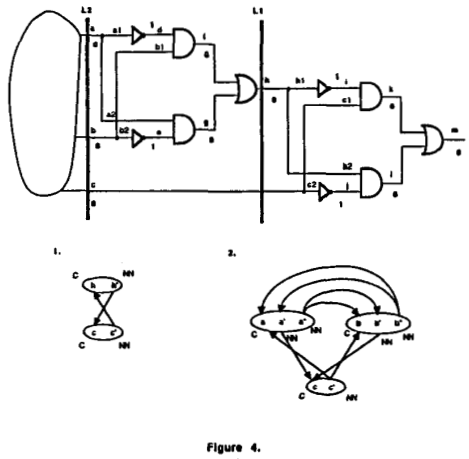

Figure 4.

above has just been completed. It consists of some 1,000 lines of C code. The implementation is yet to be fully debugged and optimized for performance. Early results on its performance are provided for single vector fault simulation on some of the benchmark circuits in the following table. More extensive data on its performance will be available shortly.

Single-Vector Fault Simulation Times
(in seconds of Apollo DN4000 CPU time)

Circuit	Init. Time	Total Time	Fault-sim. Time
C880	1.13	1.83	0.70
C1908	2.67	26.8	24.13
C2670	3.02	5.65	2.63
C5315	6.47	77.5	71.03

7. CONCLUSION

While the basic idea of critical path tracing is retained in our algorithm, it has been modified in significant ways so as to make the method exact and still run fast because the need for individual stem-fault propagation is avoided.

References

1. M. Abramovici, P. R. Menon, and D. T. Miller, "Critical path tracing: An alternative to fault simulation," *IEEE Design & Test of Computers*, pp. 83-93, February, 1984.
2. F. Maamari and J. Rajski, "Reconvergent fanout analysis and fault simulation complexity of combinational circuits," *McGill Univ., VLSI Design Lab., Tech. Rep. 87-9R*, August, 1987.
3. K. J. Antreich and M. H. Schulz, "Accelerated fault simulation and fault grading in combinational circuits," *IEEE Trans. CAD*, vol. CAD-6, pp. 704-712, September 1987. also Proc. Int. Conf. CAD (ICCAD-86), November 1986, pp. 330-333
4. S. J. Hong, "Fault simulation strategy for combinational logic networks," *Proc. 8-th Int. Symp. on Fault Tolerant Comp. (FTCS-8)*, pp. 96-99, Tolouse, France, June 1978.
5. W. Ke, "A new graph based fault simulation algorithm for combinational circuits," Master's Thesis, Department of Computer Science, Univ. of Nebraska, Aug. 1988.