Control and Cybernetics

vol. **26** (1997) No. 2

A new method of fast fault grading[1]

by

Stanisław Deniziak and Krzysztof Sapiecha

Cracow University of Technology, Department of Computer Engineering
Warszawska 24, 31-155 Cracow, Poland

Abstract: The paper presents a new algorithm of fast fault grading for combinational circuits, based on critical path tracing. In the algorithm the information coming from the preprocessing of a circuit structure description is used for static reduction of regions in the circuit where extra analysis of fault propagation is necessary. For most of the benchmarks the reduction exceeds 20%. The pseudoedges are inserted into the circuit structure description so that the multi-stem regions may be processed in the same way as if they were single-stem ones. The forward pass of a classic critical path tracing algorithm is supplemented with extra calculations of the reachability of nodes belonging to the stem regions. Thanks to that, parallel fault propagation analysis may be performed.

The algorithm uses code driven simulation technique and it applies critical path tracing in the whole area of a combinational circuit. No explicit fault simulation is used, at all. The implementation of the algorithm has proven its high efficiency with and even without dynamic reduction of the number of processing steps.

1. Introduction

Fault simulation, Pradhan (1986), is most often used in ATPG (Automatic Test Pattern Generation) systems to evaluate the fault coverage for a given sequence of deterministic or random test patterns. It is also used to build up dictionaries of faults which are essential in fault location. Fault simulation can also be used to analyse the testability of circuits being designed.

Theoretical estimations have showed that the time of fault simulation is proportional to the number of logic gates of the circuit squared, Harel, Krishnamurthy (1987). This means that the bigger the circuit, the longer the time of fault simulation. In the case of VLSI systems it can make fault simulation impossible.

[1]This research was supported by KBN under grant No. 8T11C02709.

Because of the importance of fault simulation in circuit design and testing, much better solutions of the problem are still being looked for. Special equipment used in testing, Ostapko, Barzilai, Silberman (1987), Ozgtner, Daoud (1988), and elaboration of new methods and algorithms, Melgara (1987), Gai, Somenzi, Ulrich (1987), Abramovici, Krishnamurthy, Mathews, Rogers, Schultz, Seth, Waicukauski (1988) are main directions of research. Development of faster algorithms of fault simulation seems to be more effective and more flexible than hardware acceleration.

As far as the faster algorithms are concerned we can single out two directions of research. The modifications of explicit fault simulation methods (parallel, deductive and concurrent) are the first one. The method of implicit fault simulation is the second one, where the results are reached by analyzing a structure of the circuit, true value simulation of it, and critical paths tracing instead of explicit fault simulation.

The interest in fast fault simulation algorithms for combinational circuits has arisen because of scan path designs, Williams, Parker (1982). As a result of the interest, some new fast fault simulation algorithms have been worked out: Critical Path Tracing – Abramovici, Menon, Miller (1984), PPSFP (Parallel Pattern Single Fault Propagation), Waicukauski, Eichelberger, Forlenza, Lindbloom, McCarthy (1985), CDSFP – Daehn, Geilert (1987), etc.

Most effective algorithms of implicit fault simulation are based on critical path tracing, Abramovici, Menon, Miller (1984). In the first step of this method, true-value simulation of the circuit is performed and all sensitive lines are marked. Then, in the next step, all single paths are traced backward from primary outputs of the circuit to its primary inputs, so as to determine criticality of those paths. A path is critical iff a failure propagates from the begining of the path to the end of it.

The algorithm gives correct results for combinational circuits having no nodes with reconvergent fan-out. However, in general, the results may be wrong (if self masking or multiple path sensitization occurs).

An algorithm taking into consideration criticality of nodes with reconvergent fan-out is presented in Abramovici, Menon, Miller (1984). The weak side of the algorithm is the propagation of faults of the nodes with reconvergent fan-out to primary outputs of the circuit to check the criticality of the nodes. This is time consuming but not necessary. Moreover, the criticality of at least one branch of the node is the condition to start analysis the criticality of this node. In some cases (multiple paths propagation), analyzing of some nodes may be neglected and finally the results of simulation will not include the whole set of faults to be detected in a real circuit.

The modification of the algorithm is possible when such cases are detected and handled, Menon, Levendel, Abramovici (1988). Then the results include the whole set of detected faults. However, no implementation of this algorithm is known, yet. Moreover, it seems that it would be rather very time consuming.

Criticality of nodes with reconvergent fan-out may also be graded when

explicit fault simulation is used. It has been observed that for extra cost of preprocessing, which is done only once, the overhead resulting from explicit fault simulation may be considerably limited. It appears that this simulation may be limited to some parts of the circuit (called stem regions) – Maamari, Rajski (1987). In such a case, output lines of these regions are determined and then the propagation of faults of these nodes to output lines of their stem regions is checked using explicit fault simulation. A node is critical if a fault of the node propagates to at least one output line of the stem region of this node.

The above algorithm has been used in Tulip system, see Maamari, Rajski (1988). As a result of the fault simulation limited to the stem regions, the acceleration of up to 10 times has been reached compared with propagation of stem faults to primary outputs of the circuit.

As we mentioned before, in the above algorithm the additional preprocessing step is necessary to determine stem regions and their output lines where the fault propagation for reconvergent fan-out stems to be analyzed. If none of lines that goes out of the stem region is critical, then the stem is not critical too, and fault simulation for this stem is not necessary. From that it appears that the algorithm can be accelerated by analyzing fault propagation of a stem only when at least one of the output lines of its stem region is critical. Dominator Test Detect – Underwood, Ferguson (1989) and Tulip2 – Maamari, Rajski (1990) work according to the above principle.

The weak side of this method lies in that the stem regions may be very large. This is particularly visible when the number of logic levels of the circuit is high. FSIM and FSIM_S algorithms, Hyung Ki Lee, Dong Sam Ha (1991), are free of this weakness because only these parts of a circuit are simulated which propagate at least one fault. In FSIM stem faults propagate to so called dominant nodes. In FSIM_S, however, stem faults propagate to output lines of the stem regions.

All algorithms mentioned above use PPSFP to propagate stem faults through their stem regions.

Critical path tracing is one of the most efficient implicit fault simulation methods, Abramovici, Menon, Miller (1984). Both the time of true value simulation and the time of critical path tracing depend linearly on the number of logic gates in the circuit. However, stem regions grow enormously for many practical circuits and explicit fault simulation, even if limited to stem regions, is still very time consuming.

In general, the critical path tracing method may be used for sequential circuits, Menon, Levendel, Abramovici (1988). It makes this method very interesting in looking for fast fault simulation methods applicable to all types of circuits.

The way of determining the criticality of stems with reconvergent fan-outs decides about the efficiency of an algorithm of implicit fault simulation, Abramovici, Menon, Miller (1984), Maamari, Rajski (1987), Maamari, Rajski (1990), Underwood, Ferguson (1989), Hyung Ki Lee, Dong Sam Ha (1991). Efficiency

of fault grading can be increased by making the time of simulation shorter or decreasing the sizes of regions, where the analysis of fault propagation for stems with reconvergent fan-outs must be done. The aim of this paper is to address these goals.

Static reduction and simultaneous propagation of faults of all stems with reconvergent fan-outs in the forward pass of the algorithm is the essence of a new method of fast fault grading proposed here. To define rules of fault propagation through a combinational circuit, a graph model of the circuit will be introduced and a classification of nodes of the graph will be given. On the basis of this classification the rules of fault propagation through nodes of specific types belonging to the stem regions which do not overlap will be defined. Then, these rules will be extended to nodes belonging to any stem regions. To complete the paper the whole algorithm of fast fault grading will be given and the computational complexity of this algorithm will be theoretically and experimentally estimated.

2. The stem regions

A combinational circuit is to be modelled using an acyclic directed graph $G = \langle N, A \rangle$, where N is a set of nodes and A is a set of edges. Nodes in the graph G correspond to the primary inputs of the circuit and to outputs of the gates in this circuit. Edges in the graph G correspond to connections in the circuit.

DEFINITION 2.1 *If there are at least two paths p_k and p_l between nodes n_i and n_j such that $p_k \cap p_l = \{n_i, n_j\}$ then n_j is called the node of primary reconvergence of n_i and n_i is called the node with reconvergent fan-out (the stem with reconvergent fan-out).*

Let n_j be a node of primary reconvergence of stem n_i.

DEFINITION 2.2 *A minimum subgraph of the graph G including all paths that go out of node n_i and go in any node of primary reconvergence of the stem n_i is called a primary stem region of node n_i and is denoted as $R^1_{n_i}$.*

Primary stem regions of different nodes may overlap. Stem regions of the graph G can be obtained by joining overlapping primary stem regions. Such regions will be called multi-stem regions.

Let the graph G include s nodes with reconvergent fan-out: $n_1, \ldots, n_s$.

DEFINITION 2.3 *A set of subgraphs $R_1, \ldots, R_l, \ldots, R_r$ of the graph G is called the set of stem regions of G if it satisfies the following conditions:*

1. *Each of the R_l subgraph includes at least one primary stem region of the graph G.*
2. *If for any pair of nodes n_i and n_j it holds: $R^1_{n_i} \cap R^1_{n_j} \neq \emptyset$ and $R^1_{n_i} \subseteq R_l$, then $R^1_{n_j} \subseteq R_l$.*

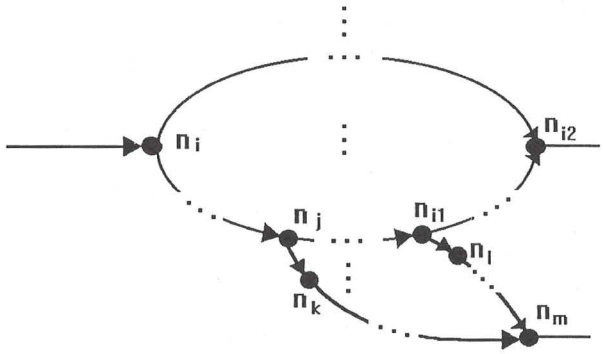

Figure 1. Secondary stem region

DEFINITION 2.4 *A node n_i is called an input node of the stem region R_l if it satisfies the following conditions:*

1. *$n_i \in R_l$, and*
2. *none of the paths that goes in the n_i node does not belong to the R_l region.*

From the above definitions, it follows that an input node of a stem region is the stem with reconvergent fan-out.

DEFINITION 2.5 *A node n_i is called an output node of the stem region R_l if:*

1. *none of the edges that leaves this node belongs to the R_l stem region, and*
2. *at least one edge that enters n_i leaves the node that belongs to the R_l region.*

Output nodes that belong to the stem region are always the reconvergence nodes of at least one stem. Output nodes that do not belong to the stem region R_l are characteristic in that there is exactly one edge between the node and the R_l stem region.

DEFINITION 2.6 *A node n_i is called a focusing node if there are at least two edges entering node n_i and belonging to the same stem region of the G graph.*

3. Stem regions tracing

Input and output nodes of the primary stem regions are defined in the same way as input and output nodes of the stem regions.

DEFINITION 3.1 *Let n_i and n_j be nodes with reconvergent fan-out and n_k and n_l be output nodes of primary stem region of the node n_i (Fig. 1). The region $R^1_{n_j}$ is called a secondary stem region of the node n_i if:*

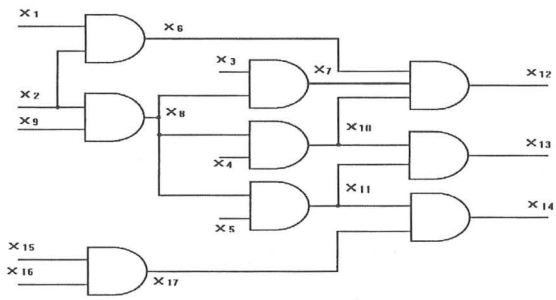

Figure 2. An example circuit

1. $R^1_{n_j} \not\subseteq R^1_{n_i}$, *and*
2. $n_j \in R^1_{n_i}$ *and* $n_k, n_l \in R^1_{n_j}$.

All nodes of primary reconvergence of node n_i that do not belong to region $R^1_{n_i}$ are called the nodes of secondary reconvergence of node n_i.

Proceeding in the same way for the n_j node we can determine third, fourth, fifth etc. stem regions and nodes of third, fourth, fifth etc. reconvergence of node n_i. The necessary condition for the existence of the l-th stem region is that there exists the $(l-1)$-th stem region.

Assume that node n_i has exactly r stem regions $R^1_{n_i}, \ldots, R^r_{n_i}$.

DEFINITION 3.2 *Stem region of node n_i is defined as:*

$$R_{n_i} = R^1_{n_i} \cup \ldots \cup R^r_{n_i}.$$

One can easily observe that if $n_i \in R_l$, then R_l includes all regions: $R^1_{n_i}, \ldots, R^r_{n_i}$.

The circuit in Fig 2 includes two nodes with reconvergent fan-out: x_2 and x_8. The x_{12} node is the node of primary reconvergence of node x_2 and nodes x_{12} and x_{13} are nodes of primary reconvergence of node x_8.

The primary stem regions are the following:

$$R^1_{x_2} = \{x_2, x_6, x_{12}\} \cup \{x_2, x_8, x_7, x_{12}\} \cup \{x_2, x_8, x_{10}, x_{12}\} =$$

$$\{x_2, x_6, x_{12}, x_8, x_7, x_{10}\}$$

$$R^1_{x_8} = \{x_8, x_7, x_{12}\} \cup \{x_8, x_{10}, x_{12}\} \cup \{x_8, x_{10}, x_{13}\} \cup \{x_8, x_{11}, x_{13}\} =$$

$$\{x_8, x_7, x_{12}, x_{10}, x_{13}, x_{11}\}.$$

There is only one secondary stem region:

$$R^2_{x_2} = R^1_{x_8} = \{x_8, x_7, x_{12}, x_{10}, x_{13}, x_{11}\}.$$

Because $R_{x_2} \cap R_{x_8} \neq \emptyset$ there exists exactly one stem region R_1 including the nodes: x_2, x_6, x_8, x_7, x_{12}, x_{10}, x_{13}, x_{11}. Nodes x_{12}, x_{13} and x_{14} are output nodes and x_2 is the input node of the stem region. The nodes x_{12} and x_{13} are the focusing nodes.

4. Node classification

Based on the definitions given above one can distinguish the following types of nodes of the graph G:

N_c – a subset of nodes with reconvergent fan-out,
N_{nc} – a subset of nodes with no reconvergent fan-out,
N_p – a subset of focusing nodes,
N_{np} – a subset of non-focusing nodes,
N_{en} – a subset of input nodes of stem regions,
N_{ex} – a subset of output nodes of stem regions,
N_o – a subset of nodes that do not belong to any stem region and are not the output nodes of any stem region,
N_i – a subset of internal nodes of a stem region.

The following equalities hold:

$$N_c \cup N_{nc} = N \text{ and } N_c \cap N_{nc} = \emptyset \tag{1}$$

$$N_p \cup N_{np} = N \text{ and } N_p \cap N_{np} = \emptyset \tag{2}$$

$$N_{ex} \cup N_{en} \cup N_i \cup N_o = N \text{ and } N_{ex}, N_{en}, N_i, N_o \text{ are pairwise disjoint sets} \tag{3}$$

From these equalities it appears that each node of the graph G belongs to three subsets: N_c or N_{nc}, N_p or N_{np} and one of the subsets denoted as N_{ex}, N_{en}, N_i or N_o. We can derive sixteen subsets of nodes such that each node belongs to exactly one of these subsets:

$N_0 = N_{np} \cap N_{nc} \cap N_o$, $\quad N_1 = N_{np} \cap N_{nc} \cap N_{en}$
$N_2 = N_{np} \cap N_{nc} \cap N_{ex}$ $\quad N_3 = N_{np} \cap N_{nc} \cap N_i$
$N_4 = N_{np} \cap N_c \cap N_o$, $\quad N_5 = N_{np} \cap N_c \cap N_{en}$
$N_6 = N_{np} \cap N_c \cap N_{ex}$ $\quad N_7 = N_{np} \cap N_c \cap N_i$
$N_8 = N_p \cap N_{nc} \cap N_o$ $\quad N_9 = N_p \cap N_{nc} \cap N_{en}$
$N_{10} = N_p \cap N_{nc} \cap N_{ex}$ $\quad N_{11} = N_p \cap N_{nc} \cap N_i$
$N_{12} = N_p \cap N_c \cap N_o$ $\quad N_{13} = N_p \cap N_c \cap N_{en}$
$N_{14} = N_p \cap N_c \cap N_{ex}$ $\quad N_{15} = N_p \cap N_c \cap N_i$.

From the rules of the determination of the stem regions and the Definitions 2.1 to 2.6 it follows that:

- $N_1 = N_4 = N_6 = N_8 = N_9 = N_{12} = N_{13} = N_{14} = \emptyset$
- $N_0 = N_o$
- $N_5 = N_{en}$

Hence, $N_0 \cup N_2 \cup N_3 \cup N_5 \cup N_7 \cup N_{10} \cup N_{11} \cup N_{15} = N$ and N_0, N_2, N_3, N_5, N_7, N_{10}, N_{11}, N_{15} are pairwise disjoint. This means that every node belongs to one and only one subset N_i, where $i \in \{0, 2, 3, 5, 7, 10, 11, 15\}$.

For the example in Fig 2 we get:

$$N_0 = \{x_1, x_3, x_4, x_5, x_9, x_{15}, x_{16}, x_{17}\}$$

$$N_2 = \{x_{14}\}$$

$$N_3 = \{x_6, x_7, x_{10}, x_{11}\}$$

$$N_5 = \{x_2\}$$

$$N_7 = \{x_8\}$$

$$N_{10} = \{x_{12}, x_{13}\}$$

$$N_{11} = \emptyset$$

$$N_{15} = \emptyset.$$

The objective of the preprocessing of the circuit description is to divide the set N into subsets N_i, $i \in \{0, 2, 3, 5, 7, 10, 11, 15\}$ (a sketch of the algorithm of the preprocessing is described in appendix D). In the following sections we will show that membership of a node in the particular subset N_i decides about the way the node is processed when critical paths are traced.

5. Reachability

Let us make the following assumptions:

A1: The reconvergent fan-out stems are the only sources of faults analyzed here.

A2: Fault propagation is analyzed only in stem regions.

To describe fault propagation rules in a circuit the concept of reachability of node n_j for a fault appearing in node n_i will be introduced.

DEFINITION 5.1 *Let V be an input pattern. Reachability of node n_j for a fault appearing in node n_i is defined as follows:*

$$d_i^j(V) = \begin{cases} 1 & \textit{when the fault reaches node } n_j, \\ 0 & \textit{otherwise.} \end{cases}$$

Faults coming from the stems may appear in the nodes that correspond to inputs of the gates in the circuit. The following cases can be distinguished:

1. there are no faults on the inputs (Fig. 3a),
2. there is a single fault on one input (Fig. 3b),

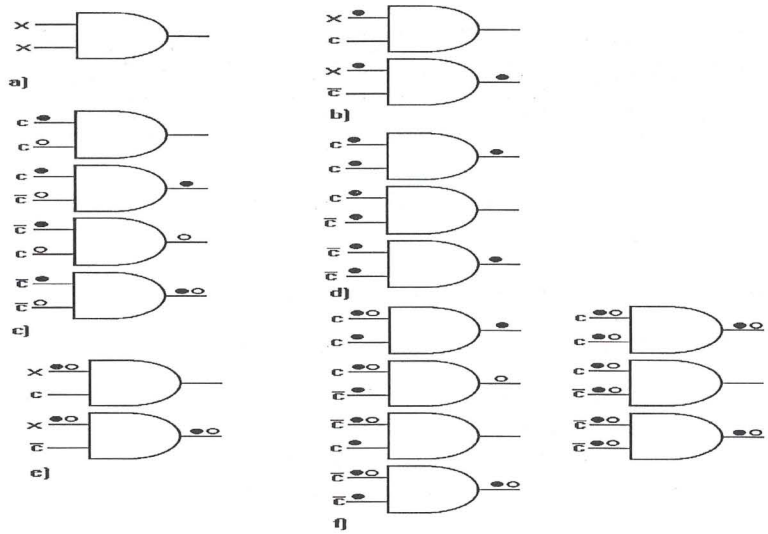

Figure 3. Different cases of fault propagation

where:

o,• – denote faults coming from different stems,

c – is a controlling value of the gate $\overline{c}$ – is its complement,

x – is an arbitrary value of an input.

3. there are multiple faults propagated from different stems on different inputs (Fig. 3c),
4. there are multiple faults propagated from the same stem on different inputs (Fig. 3d),
5. there are multiple faults propagated from different stems on one input (Fig. 3e),
6. there are multiple faults propagated from different stems on both inputs (Fig. 3f).

It is easy to show that the other cases (for gates of greater number of inputs and for greater number of faults) can be reduced to the above listed. Moreover, similar analysis can be done for all the remaining types of logic gates, that is for NOT, OR, NAND, NOR and XOR gates.

For each of the above cases the reachability of the output node for faults appearing on input nodes can be determined according to Table 1.

When $l-1$ stems with reconvergent fan-out drive an output of the gate then l-th pair (logic state, reachability) describes the state of the output of this gate.

Reachability d_i^j indicates if the fault that it is released from node n_i propagates to node n_j. Later on we prove that if node n_j is the output of the gate

Case	a	b1	b2	c1	c2	c3	c4	d1	d2	d3	e1	e2	f1	f2	f3	f4	f5	f6	f7
$d_\circ^j$	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	1	1	0	1
$d_\bullet^j$	0	0	1	0	1	0	1	1	0	1	0	1	1	0	0	1	1	0	1

Table 1. Reachability for AND gate

and nodes $n_1, \ldots, n_k, \ldots, n_m$ are inputs of the gate then:

$$(v_j, d_i^j) = R((v_1, d_i^1), \ldots, (v_k, d_i^k), \ldots, (v_m, d_i^m))$$

where v_k is a logic state of node n_k and R is a function defined by so-called reachability table of this gate.

Reachability d_i^j should be calculated along with the calculation of logic states of the nodes. According to the assumptions A1 and A2, initial values of d_i^i are as follows:
$d_i^i = 1$, if $n_i \in N_5, N_7$ or N_{15} (fault sources),
$d_i^i = 0$, otherwise.

Logic states and reachability are calculated node by node using the reachability tables.

In the next sections, the way reachability tables are defined will be given.

6. Pseudoedges

If the stem regions in a circuit do not overlap, then all faults in a stem region come from the same stem. As concerns logic gates this corresponds to the cases from figures 3a, 3b and 3d. Association of calculated reachability with appropriate stem in the circuit is unique then.

Now, let us assume that the stem regions in a circuit may overlap. As concerns logic gates this corresponds to all cases in Fig. 3. In the cases in Fig. 3a, 3b and 3d (faults of the same stem only) the reachability may be determined as previously. In the case of Fig. 3e we may use the following observation:

OBSERVATION 6.1 *If each path between nodes n_i and n_k goes through node n_j then:*

$$d_i^k = d_i^j \cdot d_j^k$$

In the last two cases of Fig. 3c and 3f, to determine propagation of individual faults, it is necessary to calculate the reachabilities separately for each stem. To aim this, the transformation of cases in Fig. 3c and 3f to the case in either Fig. 3b or 3d will be done. We will do it by appending the G graph with extra edges (so called pseudoedges) which will carry the information that allows us to deal with the multi-stem regions as if all stem regions were disjoint. For example,

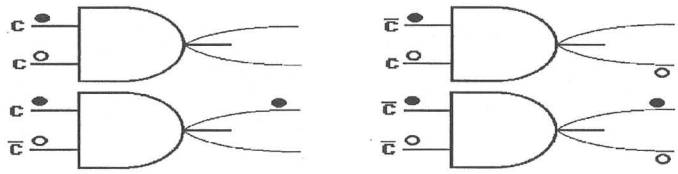

Figure 4. Pseudoedges propagating different faults

for the gate of Fig. 3c two pseudoedges that leave the node corresponding to the output of the gate will be inserted into the G graph. The 'upper' one for the fault coming from the black stem and the 'lower' one for the fault coming from the white one (Fig. 4).

The cases of Fig. 3c,f can only occur for nodes belonging to the sets N_7, N_{11} and N_{15}, and pseudoedges should be inserted into the G graph only for these nodes. The rules of inserting pseudoedges into the graph follow:

R1: when $n_j \in N_7$ or $n_j \in N_{15}$: for each stem with reconvergent fan-out n_i, for which there is at least one path going through node n_j to any reconvergence node, the pseudoedge s_i^j is added.

R2: when $n_j \in N_{11}$ and node n_j is on paths going through some nodes $n_1, \ldots, n_k$ $(k > 1)$ to their reconvergence nodes: pseudoedges $s_1^j, \ldots, s_k^j$ that go out node n_j are added.

A pseudoedge s_i^j goes out of node n_j and it is assigned reachability d_i^j, provided that node n_i belongs to the sets N_5, N_7 or N_{15}. Pseudoedges that go out of node n_j are directed to the nearest nodes from sets N_2, N_7, N_{10}, N_{11} or N_{15} which are on the paths going out of the n_j node. Reachability associated with pseudoedge s_i^j will be denoted ds_i^j.

In the graph of the circuit in Fig. 2 one triple pseudoedge s_{x2}^{x8} should be added. It leaves node x_8 and enters nodes x_{12}, x_{13}, x_{14}.

7. The $\mathcal{R}$-algebra

The $\mathcal{R}$-algebra (Reachability algebra) is introduced to calculate the reachability in a circuit. The $\mathcal{R}$-algebra is defined as follows:

$$\mathcal{R} = \langle W, P, R \rangle,$$

where $W = \{0,1\} \times \{0,1\}$, $P : W \to W$, and $R : W \to W$.

The reachability function R of a logic gate is defined in Table 2.

Table 2 is determined on the basis of the following lemma:

LEMMA 7.1 *The reachability function of any logic gate having m input nodes $n_1, \ldots, n_k, \ldots, n_m$ and output node n_j is as follows:*

1. *logic state v_j of the output is determined by the logic function of the gate,*

v_1, d_i^1	00	00	00	00	01	01	01	01	10	10	10	10	11	11	11	11
v_2, d_i^2	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
AND	00	00	00	00	00	01	01	00	00	01	10	10	00	00	11	11
NAND	10	10	10	10	10	11	11	10	10	11	00	00	10	10	01	01
OR	00	01	10	11	01	01	10	10	10	10	10	10	11	10	10	11
NOR	10	11	00	01	11	11	00	00	00	00	00	00	01	00	00	01
XOR	00	01	10	11	01	01	11	10	10	11	10	01	11	10	01	01

Table 2. Reachability function

2. *d_i^j equals 1 iff:*
   - *$n_j \in R_{n_i}$, and*
   - *for at least one input n_k: $d_i^k = 1$, and*
   - *logic states of all inputs n_k for which $d_i^k = 1$ are the same and*
   - *logic states of all inputs n_k for which $d_i^k = 0$ are not controlling ones.*

Let P denote the reachability function of a pseudoedge. It is defined on the basis of Theorem 7.1.

Let n_k be the node that corresponds to the output of a gate, the n_m node be the node that corresponds to an input of this gate and a s_i^j pseudoedge goes into the n_k node. The s_i^j pseudoedge is said to be connected with the n_m node if there is at least one path from n_j to n_k that goes through the n_m node.

THEOREM 7.1 *Let n_j be the node corresponding to the output of a gate and nodes $n_1, \ldots, n_k, \ldots, n_m$ correspond to its inputs. Let there be at least one path between the node $n_i \in N_5 \cup N_7 \cup N_{15}$ and each of the inputs $n_1, \ldots, n_r$, $(1 \leq r \leq m)$ (which means that $n_k \in R_{n_i}$ for $k = 1, \ldots, r$ and $n_k \notin R_{n_i}$ for $k = r+1, \ldots, m$) and let l nodes (inputs) $n_1, \ldots, n_l$, $(l \leq r)$ be connected with pseudoedges $s_i^{i1}, \ldots, s_i^{il}$ (Fig. 5).*

If the R function is the reachability function of this gate, then the reachability of the pseudoedge s_i^j is calculated as follows:

$$[v_j, ds_i^j] =$$

$$R[[v_1, d_{i1}^1 \cdot ds_i^{i1}], \ldots, [v_l, d_{il}^l \cdot ds_i^{il}], [v_{l+1}, d_i^{l+1}], \ldots, [v_r, d_i^r], [v_{r+1}, 0], \ldots, [v_m, 0]].$$

In Appendixes A and B proofs of the lemma and the theorem are given.

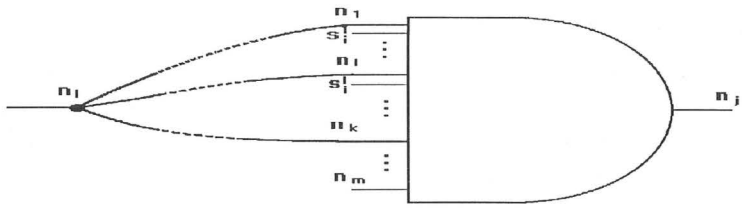

Figure 5.

8. Fast fault grading

The algorithm of fast fault grading consists of three steps:

1. preprocessing of the circuit structure and generation of the code of the fast fault grader,
2. calculating logic states and reachabilities for nodes and pseudoedges of the graph G,
3. critical path tracing.

The first step of the algorithm is executed only once for a given circuit. The second and the third steps are executed for every input pattern.

In the first step of the algorithm the stem regions, the characteristic subsets of nodes and the pseudoedges are determined. This information is used to generate the code of the fast fault grading program. This program consists of two parts that implement the second and third step of the algorithm. In the first part of the program, expressions that allow us to calculate the $W = [vd]$ vectors, for nodes that correspond to outputs of logic gates belonging to the stem regions and for the pseudoedges, are generated. In the second part, expressions are generated that allow us to calculate the criticality of inputs of the gates and criticality of stems with reconvergent fan-out.

From the rules for calculating reachability it appears that the reachability of the nodes that do not belong to any stem regions are always equal 0 (fault propagation analysis is bounded to stem regions) and the reachabilities are not calculated for output nodes of stem regions (this sequence of calculations is executed in the second step of the algorithm only if the output nodes are critical). This way the number of calculations is decreased.

Criticality of stems with reconvergent fan-out is calculated on the basis of criticality and reachability of output nodes of appropriate stem regions. For other nodes criticality is calculated according to the classic CPT algorithm.

In Appendix C, results of execution of the algorithm for the example of Fig. 2 are given.

9. Dynamic fault dropping

To minimize superflous calculations, a method of processing of fanout–free regions(FFRs) and stem regions(SRs) in the forward levelized order has been used Hyung Ki Lee, Dong Sam Ha (1991). Only those regions which are affected by the faults are processed this way.

Simulation consists of two steps:

- calculating fault propagation to stems (calculating local propagation inside FFRs) and calculating stem faults propagation to exit lines of stem regions;
- calculating fault propagation to primary outputs by processing FFRs in the backward order.

For every active FFR (to which faults are propagated), propagation of the faults to a stem (by critical path tracing) is calculated in the first step of the algorithm. If at least one fault is propagated to a stem s, then stem fault propagation is determined by calculating reachability $d_s^{l_i}$ (where l_i are exit lines of the stem region SR(s)). A stem t (such that $l_i \in FFR(t)$) is marked active when a fault of the stem s is propagated to an exit line l_i. Fanout branches are exit lines for stems without reconvergent fan-out. In the second step criticality of a line is calculated as a product of local criticality of this line and criticality of a stem being an FFR output the line belongs to.

The PCPT (Parallel Critical Path Tracing) algorithm follows:

```
procedure PCPT;
{ Where c(l) and c_l(l) are criticality and local criticality of the line l
respectively. }
begin
Mark all stems as active;
for each test pattern do
    begin
    Perform fault free simulation;
    for each stem s in forward levelized order do
        if s is active then
            begin
            Simulate FFR(s) using the critical path tracing;
            if any fault propagates to s then
                for each exit line l of SR(s) do
                    begin
                    Compute d_s^l.
                    if d_s^l = 1 then
                        Mark stem t (where l ∈ FFR(t)) as active.
                    end
            end
    for each active stem s in backward order do
        begin
        if s is a primary output then
```

```
        Mark s as critical;
    if C(s) ≠ 0 then Mark detected faults of stem s;
    for each line l in FFR(s) do
        begin
        C(l) = C_l(l) and C(s);
        if C(l) ≠ 0 then Mark detected faults of line l;
        end
    for each exit line l of the stem region SR(t) in FFR(s) do
        C(t) = C(t) or C_l(l);
    if there are no more undetected faults in FFR(s) then
        Mark s as non active;
    end
end
```

10. Computational complexity of the algorithm

Let the circuit include n nodes with reconvergent fan-out $n_1, \ldots, n_n$ and let $N_k(R_{n_i})$ denote the number of nodes from the set of N_k that belong to the region R_{n_i}. Then the number of pseudoedges can be calculated as follows:

$$S = \sum_{i=1}^{n} (N_7(R_{n_i}) + N_{11}(R_{n_i}) + N_{15}(R_{n_i})).$$

To calculate the criticality of the stem with reconvergent fan-out n_i, it is necessary to calculate the reachability d_i^j for each node n_j which is the output node of the R_{n_i} region. Let t be an average time needed to calculate the state (W vector) of a single node or pseudoedge. To calculate reachability for output nodes, the extra calculations of the state of the pseudoedges that propagate a fault of the n_i node should be performed. This time can be estimated as follows:

$$T(n_i) = t \cdot \overline{S}(n_i) = t \cdot (N_7(R_{n_i}) + N_{11}(R_{n_i}) + N_{15}(R_{n_i}))$$

where $\overline{S}(n_i)$ is the number of pseudoedges that propagate a fault of the n_i node.

Finally, for all the nodes:

$$T = \sum_{i=1}^{n} t \cdot (N_7(R_{n_i}) + N_{11}(R_{n_i}) + N_{15}(R_{n_i})).$$

The above relationship can be expressed in the following way:

$$T = n \cdot t \cdot \sum_{i=1}^{n} \frac{1}{n} \cdot (N_7(R_{n_i}) + N_{11}(R_{n_i}) + N_{15}(R_{n_i})) = n \cdot t \cdot (\overline{N_7} + \overline{N_{11}} + \overline{N_{15}})$$

where $\overline{N_7}$, $\overline{N_{11}}$, $\overline{N_{15}}$ are average numbers of nodes from subsets N_7, N_{11} and N_{15} included in a stem region of one node.

circuits	LANE PC 386SX 25MHz	ASI PC 486DX 33MHz	LANE PC 486DX 50MHz	Apollo HP9000	
				224	102400
C432	0.38	0.17	0.063	0.021	9.58
C499	0.24	0.095	0.039	0.007	3.37
C880	0.32	0.12	0.051	0.010	4.34
C1355	0.43	0.18	0.074	0.020	9.24
C1908	2.02[1)]	1.04[1)]	1.07[1)]	0.073	33.85
C2670	1.09	0.45	0.39	0.063	28.74
C3540				0.239	109.17
C5315	2.37[1)]	1.10[1)]	1.07[1)]	0.091	41.71
C6288				2.2	1005
C7552	7.60[1)]	4.78[1)]	4.59[1)]	0.147	67.09

1) EMS (1MB)

Table 3. Experimental results

From the fact that

$$\sum_{i=1}^{n}(N_7(R_{n_i}) + N_{11}(R_{n_i}) + N_{15}(R_{n_i})) < \sum_{i=1}^{n}\overline{R_{n_i}}$$

where $\overline{R_{n_i}}$ is the number of nodes in the R_{n_i} region, the following conclusion can be drawn:

CONCLUSION 10.1 *Checking the propagation of nodes with reconvergent fan-out to the output nodes of stem regions using pseudoedges is more efficient than the single fault propagation method for these nodes.*

11. Experimental results

Table 3 presents experimental results obtained for well-known benchmark circuits, Brglez, Fujiwara (1985) (time in seconds, no preprocessing time included). The experiments have been performed on LANE PC/386SX-25 MHz, ASI PC/486DX-33 MHz (64kB cache) and LANE PC/486DX-50 MHz (256kB cache) running under MS DOS, and on Apollo HP9000/720 workstation working under HP-UX 8.05 for 224 and 102400 input vectors. In one pass, 16 (PCs) or 32 (Apollo) test vectors have been simulated, no fault dropping technique has been applied.

In Table 4 selected parameters of the benchmarks and results of the preprocessing (performed on LANE PC/486DX-50MHz) are shown.

For benchmarks having similar ratios of the stem region overlapping ($\overline{R}$), the time of simulation grows almost linearly with the size of the circuits. This holds for C499, C1355, C5315 and C7552 benchmarks, for example. For benchmarks

Circuit	Gates	Stems	PIs	POs	$\overline{R}$	Pseudoedges	Prepr. [s]
C432	160	82	36	7	29.5	5212	0.44
C499	202	49	41	32	11.5	1115	0.77
C880	383	82	60	26	8.3	2330	0.82
C1355	546	249	41	32	7.4	2995	3.13
C1908	880	362	33	25	19.7	12054	3.68
C2670	1193	395	233	140	10.3	12905	4.23
C3540	1669	528	50	22	44.2	63293	10.93
C5315	2307	638	178	123	7.2	14656	15.65
C6288	2416	1427	32	32	184.9	493019	75.82
C7552	3512	1090	207	108	10.9	26802	28.18

Table 4. Benchmark characteristics

Circuit	N_0	N_2	N_3	N_5	N_7	N_{10}	N_{11}	N_{15}	Red. [%]	Speed 10^6/s
C432	7	3	23	36	14	4	84	32	26	1.8
C499	40	0	48	33	0	32	106	16	67	1.3
C880	139	7	89	34	3	24	128	45	52	1.9
C1355	72	0	40	33	8	32	226	208	21	1.2
C1908	25	0	232	33	207	25	294	122	23	1.2
C2670	244	20	230	43	235	13	512	117	18	1.6
C3540	32	2	456	46	258	28	695	224	23	1.9
C5315	351	19	492	73	332	74	1034	233	26	1.3
C6288	32	0	17	32	15	32	972	1380	0.6	1.6
C7552	378	4	870	65	553	47	1436	472	25	1.5

Table 5. Reduction ratio

of similar size(G), the time of simulation grows faster than linearly with the growth of the $\overline{R}$ ratio. This holds for C432 and C499, for C2670 and C3540, and for C1355, C1908, C3540 and C6288 benchmarks. A slight dispersion of the values comes from different speeds of simulation for different types of logic gates. Two-input gates are simulated faster than n-input gates, for $n > 2$. This is why the C880 benchmark is simulated faster than predicted by the theoretical estimation.

No fault dropping techniques were used in the algorithm. Hence, the time of simulation grows linearly with the number of input vectors applied.

For most benchmarks, the static reduction of calculations, due to the decrease of the areas in which reconvergent fan-out stem faults are analyzed, is greater than 20% (Table 5). High value of the $\overline{R}$ ratio for C6288 results in relatively low reduction for this benchmark.

Tables 6 and 7 show published results of experiments for eight systems which

Name	SOCRATES		Tulip		PPSFP	CDSFP	
	200	20000	224	102400	30000	224	30016
C432	2.0	21.7	0.9	31.0	21.1	1	2
C499	2.9	30.7	0.8	44.2	40.5	1	8
C880	4.4	27.8	1.5	59.6	45.7	1	3
C1355	7.5	79.4	2.0	124.9	72.3	3	14
C1908	16.8	120.1	3.3	138.5	118.7	5	18
C2670	12.7	431.1	4.7	338.5	410.5	11	487
C3540	38.0	295.8	9.6	810.3	209.7	19	241
C5315	21.5	292.8	8.1	473.4	538.5	37	285
C6288	154.1	387.7	-	2182.6	220.0	36	210
C7552	45.5	673.8	13	1076.2	653.7	75	1539

Table 6.

use different fault simulation algorithms.

Remarks:

1. The algorithm used in SOCRATES, Schulz, Trischler, Sarfert (1987) is a combination of critical path tracing (for nodes having no reconvergent fan-out) and PPSFP (for nodes having reconvergent fan-out). The propagation of faults for nodes with reconvergent fan-out is limited to dominators. The SOFE (Stop On First Error) technique is partly used in the algorithm. The results have been obtained on microVAX for 200 and 20000 random test patterns.
2. In Tulip system, Maamari, Rajski (1988), PPSFP algorithm is used only for stems in their stem regions, and the SOFE technique is partly used, too. The results have been obtained on SUN 3/260 for 224 and 102400 random test patterns.
3. PPSFP algorithm was used for all nodes of the circuit, Briers, Totton (1986). The number of simulated faults was decreased by removing equivalent faults. The results have been obtained on VAX 11/785 computer for 30000 random test patterns.
4. CDSFP algorithm, Daehn, Geilert (1987), reduces the number of calculations for individual faults by simulating just a part of the circuit starting from the faulty node through outputs. All faults detected in each pass of simulation are dismissed (SOFE). The results have been obtained on APOLLO DN330 for 224 and 30016 random test patterns.
5. In FSIM and FSIM_S algorithms, critical path tracing for fan-out free regions (FFRs) and PPSFP for stem regions (SRs) is used. In FSIM algorithm, stem faults propagate to so called dominant nodes. In FSIM_S, however, stem faults propagate to output lines of stem regions. The results have been obtained on Sun 386i for 224 and 102400 random patterns.

Name	FSIM		FSIM_S		DTD		PCPT	
	224	102400	224	102400	224	102400	224	102400
C432	0.38	7.45	0.37	7.32	0.55	5.83	0.01	1.06
C499	0.20	10.60	0.22	11.25	0.81	14.76	0.01	1.19
C880	0.48	3.34[1)]	0.38	3.30[1)]	1.30	13.53	0.02	0.32[2)]
C1355	0.55	23.50	0.52	24.69	1.98	31.70	0.03	2.55
C1908	1.38	25.49	1.28	24.91	3.03	27.84	0.07	4.21
C2670	1.53	146.57	1.37	156.36	3.62	168.49	0.06	12.62
C3540	4.28	98.75	4.10	99.99	7.68	210.99	0.24	20.49
C5315	2.48	94.55	2.15	94.16	7.42	123.26	0.13	13.18
C6288	13.1	101.25	13.48	101.17	55 .45	4114.32	2.34	18.79
C7552	4.37	227.02	4.98	228.77	11. 61	315.32	0.21	30.58

1) for 28.832 patterns
2) for 18.240 patterns

Table 7.

6. Dominator Test-Detect (DTD) is similar to the algorithm applied in Tulip. However, a stem fault is simulated only when at least one output line of its stem region is critical. The results have been obtained on Sun 3/260 for 224 and 102400 random patterns.
7. In PCPT algorithm, Deniziak, Sapiecha (1983), an improved critical path tracing is applied to the entire circuit. Fault dropping technique is used. Results have been obtained on HP Apollo 720 MDL for 224 and 102400 random patterns.

12. Conclusions

In the paper, a new algorithm of fast fault grading has been presented and evaluated. In this algorithm, the information coming from the preprocessing of a circuit structure is used for static reduction of regions where extra analysis of fault propagation is necessary. As a result for most of the benchmarks examined the reduction of computations exceeds 20%. Pseudoedges are inserted into the circuit structure so that the multi-stem regions may be processed in the same way as if they were single-stem ones. The forward pass of the algorithm is supplemented with extra calculations of the reachability of nodes belonging to the stem regions. Thanks to that parallel fault propagation analysis may be performed.

The algorithm uses code driven simulation technique and it applies critical path tracing for the entire combinational circuit. No explicit fault simulation is used, at all. Parallel pattern simulation technique is also applied here, as in most fast fault simulators.

The experimental results confirm the theoretical estimation of the computational complexity of the algorithm. Computational complexity of both forward and backward passes of the algorithm is $O(G+p)$, where G is the number of logic gates that are in the circuit and p is the number of the pseudoedges.

From Table 3 it appears that some of the benchmarks are more difficult than other for simulation using the algorithm discussed (C6288, C432, C3540, C1908 vs. C880, C499, C7552, C1355, C5315). Generally, the higher the $\overline{R}$ ratio, the lower the speed-up of the simulation. Moreover, comparing with CDSFP algorithm, the larger the circuit, the higher the ratio of speeding up the simulation. Hence, the computational complexity of the algorithm seems to be lower than the complexity of the CDSFP algorithm.

No dynamic fault dropping used in the algorithm results in much lower values of the ratio of speeding up the simulation for longer test vector sequences. This is particularly visible when CDSFP algorithm is taken as the basis of comparison (as SOFE is most effective for this algorithm). The influence of SOFE on the speed of the simulation is stronger for C6288 than for C2670 because the first benchmark contains much fewer faults difficult to detect than the second one (fault coverages: 99.43% in the case of C6288 and 82.26% in the case of C2670, both for 224 random tests).

However, it should be noticed that fault dropping is not always acceptable. For example, if fault simulation is used to create the so called fault dictionaries, Breuer, Friedman (1976), or if fault coverage for individual tests is nedeed, then dynamic fault dropping must not be used. Moreover, as far as efficiency is concerned, first experiments with a new version of the algorithm, where dynamic fault dropping is included, seem to be very encouraging.

The contents of Table 5 delivers the information about the efficiency of the static reduction of calculations in the algorithm. The last column of Table 5 shows an average speed of calculations of the W vectors for logic gates and pseudoedges. It can be observed that at the expense of extra calculations in the forward pass of the algorithm (true value simulation) the backward pass of the algorithm (critical path tracing) is executed very fast. Finally, the algorithm is very efficient even without dynamic fault dropping. If one assumes that:

$$t = A \cdot G^n \tag{4}$$

where:
t – time of fault simulation,
G – size of the circuit (number of gates),
A, n – constant coefficients for the given algorithm,
then the algorithm shows what follows:
$n = 1.3$ – for interpolation of individual pairs of results with equation (4),
$n \in \langle 0.5 \ldots 2.3 \rangle$ – the partitions, that include values of ns, most obtained using the interpolation;
$n = 1.39$ – for approximation of the results with equation (4) (details are discussed in Deniziak, Sapiecha, 1983c).

Strong dependence of the speed of algorithm upon the circuit structure results in a high dispersion of the values of coefficient n $(0.5 - 2.3)$. However, it should be mentioned that n does not depend upon the length of a test sequence. Such a dependence is typical for the algorithms that employ fault dropping technique (for example SOFE). When fault dropping is applied, then the actual value of G in equation (4) for the simulated circuit is becoming smaller and smaller. The first 224 input vectors usually detect about 90% of faults. Therefore, the longer the test sequence, the less significant influence of fault dropping on the value of n.

The implementation of the algorithm has proven highly efficient though no dynamic reduction of processing steps was used.

Acknowledgements:

The authors are indebted to anonymous referee for his very helpful suggestions improving the final version of this paper.

References

ABRAMOVICI, M., MENON, P.R., MILLER, D.T. (1984) Critical path tracing: an alternative to fault simulation. *IEEE Design & Test of Computers*, **1**, 83-93, February.

ABRAMOVICI, M., KRISHNAMURTHY, B., MATHEWS, R., ROGERS, B., SCHULTZ, M., SETH, S., WAICUKAUSKI (1988) What is the path to fast fault simulation? (A panel discussion). *Proc. of the IEEE International Test Conference*, 183-192.

BREUER, M.A., FRIEDMAN, A.D. (1976) *Diagnosis & Reliable Design of Digital Systems.* Computer Science Press, Inc.

BRIERS, A.J., TOTTON, K.A.E. (1986) Random pattern testability by fast fault simulation. *Proc. of the IEEE International Test Conference*, 274-281.

BRGLEZ, F., FUJIWARA, H. (1985) A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. *Proc. of the IEEE Int. Symp. on Circuits and Systems*, 663-698.

DAEHN, W., GEILERT, M. (1987) Fast fault simulation for combinational circuits by compiler driven single fault propagation. *Proc. of the IEEE International Test Conference*, 286-292, September.

DENIZIAK, S., SAPIECHA, K. (1993A) *A Method of Fast Fault Grading*, Technical Report No. 3/93, Department of Computer Science, Kielce University of Technology.

DENIZIAK, S., SAPIECHA, K. (1993B) *Stem Region Analysis.* Technical Report No. 4/93, Department of Computer Science, Kielce University of Technology.

DENIZIAK, S., SAPIECHA, J., SAPIECHA, K. (1993) *Is There Hope for Perfect Fault Simulation Algorithm.* Technical Report No. 5/93, Department of Computer Science, Kielce University of Technology.

GAI, S., SOMENZI, F., ULRICH, E. (1987) Advances in concurrent multilevel simulation. *IEEE Trans. CAD*, **CAD-6**, 6, 1006-1012, November.

HAREL, D., KRISHNAMURTHY, B. (1987) Is there hope for linear time fault simulation? *Proc. of the 17th Fault Comp. Symp.*, 28-33.

KE, W., SETH, S., BHATTACHARYA, B.B. (1988) A fast fault simulation algorithm for combinational circuits. *Proc. of the IEEE International Conference on CAD*, 166-169, November.

HYUNG KI LEE, DONG SAM HA (1991) An Efficient, Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation. *Proc. of the IEEE International Test Conference*, 946-955.

MAAMARI, F., RAJSKI, J. (1987) *Reconvergent fanout analysis and fault simulation complexity of combinational circuits.* McGill University, VLSI Design Laboratory, Tech. Rep. 87-3R, August.

MAAMARI, F., RAJSKI, J. (1988) A fault simulation method based on stem regions. *Proc. of the IEEE International Conference on CAD*, 170-173, November.

MAAMARI, F., RAJSKI, J. (1990) A Method of Fault Simulation Based on Stem Regions. *IEEE Trans. on CAD*, **9**, 2, February, 212-220.

MELGARA, M. (1987) Fault simulators at functional level in hardware description languages, ed. Hartenstein, R.W. 337-372, Elsevier Science Publishers B.V. (North Holland).

MENON, P.R., LEVENDEL, Y., ABRAMOVICI, M. (1988) Critical path tracing in sequential circuits. *Proc. of the IEEE International Conference on CAD*, 162-165, November.

OSTAPKO, D.L., BARZILAI, Z., SILBERMAN, G.M. (1987) Fast fault simulation in a parallel processing environment. *Proc. of the IEEE International Test Conference*, 293-298.

OZGTNER, F., DAOUD, R. (1988) Vectorized fault simulation in the CRAY X-MP supercomputer. *Proc. of the IEEE International Conference on CAD*, 198-201.

PRADHAN, D.K. (1986) *Fault tolerant computing: theory and techniques.* ed. Pradhan, D.K. **1**, 184-264, Prentice-Hall, Englewood Cliffs, NJ.

SCHULZ, M.H. (1987) Automatic test pattern generation and fault grading in combinational circuits. *Proc. of CompEuro 1987*, 382-385, May.

SCHULZ, M.H., BRGLEZ, F. (1987) Accelerated transition fault simulation. *Proc. of the 24th ACM/IEEE Design Automation Conference*, 237-243, June.

SCHULZ, M.H., TRISCHLER, E., SARFERT, T.M. (1987) SOCRATES: a highly efficient automatic test pattern generation system. *Proc. of the IEEE International Test Conference*, 1016-1026.

UNDERWOOD, B., FERGUSON, J. (1989) The Parallel-Test Detect Fault Simulation Algorithm. *Proc. of the International Test Conference*, 801-808.

WAICUKAUSKI, I.A., EICHELBERGER, E.B., FORLENZA, D.O., LINDBLOOM, E., MCCARTHY, T. (1985) Fault simulation for structured VLSI. *VLSI Systems Design*, **6**, 12, 20-32, December.

WILLIAMS, T.W., PARKER, K.P. (1982) Design for testability – a survey. *IEEE TC*, **C-31**, 1, 2-15, January.

Appendix A

Proof of Lemma 7.1 (by contradiction):

Let us assume that $d_i^j = 0$ but still a fault of node n_i propagates to node n_j. We are concerned only with the stem regions (assumption A2). Hence, $n_j \in R_{n_i}$. If $d_i^j = 0$, then according to the assumption:

- $d_i^1 = \ldots = d_i^m = 0$, or
- there exist nodes n_l and n_r $(l, r \in \langle 1 \ldots m \rangle)$, such that $d_i^l = d_i^r = 1$ and $v_l \neq v_r$, or
- there exists a node n_l $(l \in \langle 1 \ldots m \rangle)$, such that $d_i^l = 0$ and the state of v_l is the controlling one.

On the other hand, by the same assumption, the fault of the n_i node propagates to the n_j node. This occurs iff:

C1 there is the n_l $(l \in \langle 1 \ldots m \rangle)$ node such that $d_i^l = 0$ and the fault of the node n_i propagates to the n_l node, or

C2 there is the n_l $(l \in \langle 1 \ldots m \rangle)$ node such that $d_i^l = 1$ and the fault of node n_i does not propagate to node n_l.

The same requirements should be met if it is assumed that $d_i^j = 1$ and that the fault of the n_i node does not propagate to the n_j node. This means that proofs of the necessary condition and the sufficient condition are the same.

Let us analyse the requirements for the assumption to be true. The following cases can be distinguished:

1. $n_l = n_i$: then $d_i^l = 1$, which means that C1 is not satisfied; moreover the fault of the n_i node propagates to the n_l node which means that C2 is not satisfied, neither.
2. $n_l \notin R_{n_i}$: then a directed path from the n_i node to the n_l node does not exist and therefore the fault of the n_i node cannot propagate to the n_l node, which means the C1 is not satisfied. Moreover, $d_i^l = 0$ which means the C2 is not satisfied, neither.
3. $n_l \in R_{n_i}$ and $n_l \neq n_i$: after the reasoning for nodes n_l and n_j are done in the same way, this case can be finally reduced to the above cases. ■

Appendix B

Proof of Theorem 7.1 (inductive for the number of logic levels between nodes n_i and n_j):

" => "

Let assume that the fault associated with the n_i node reaches the nodes $n_1, \ldots, n_{l1}$ and $n_{l+1}, \ldots, n_{l+k1}$ $(l1 \leq l, l1 + k1 \leq r)$.

1° Let the n_j node be the first node on the path or paths that go out of the n_i node and let the fault of n_i propagate to the n_j node. From the assumptions it follows that:

(1) $n_1, \ldots, n_r$ and n_i are the same node
(2) $l1 + k1 = r$
(3) $v_1 = \ldots = v_{l1} = v_{l+1} = \ldots = v_{l+k1}$
(4) $v_{l1+1} = \ldots = v_{l-1} = v_{l+k1+1} = \ldots = v_m = \overline{c}$
(5) $l1 = l$

From the rules of reachability calculation (Lemma 7.1) and from (3), (4) and (5), it follows that $ds_i^j = 1$ iff:

C1: $d_{i1}^1 = \ldots = d_{i_{l_1}}^{l_1} = ds_i^{i1} = \ldots = ds_i^{l_1} = d_{i_{l+1}}^{l+1} = \ldots = d_{i_{l+k}}^{l+k1} = 1$, and
C2: $d_{i_{l1+1}}^{l1+1} = \ldots = d_{i_{l-1}}^{l-1} = d_{i_{l+k1+1}}^{l+k1+1} = \ldots = d_{i_r}^r = 0$ or $ds_i^{l1+1} = \ldots = ds_i^{l-1} = d_{i_{l+k1+1}}^{l+k1+1} = \ldots = d_{i_r}^r = 0$.

From (1) and (2) it results that $d_{i1}^1 = 1, \ldots, d_{i_{l_1}}^{l_1} = 1$, $ds_i^{i1} = 1, \ldots, ds_i^{l_1} = 1$ and $d_{i_{l+1}}^{l+1} = 1, \ldots, d_{i_{l+k1}}^{l+k1} = 1$ (because n_i has reconvergent fan-out) which means the first condition is satisfied.

From (2) it results that there are no nodes that need the second condition to be satisfied. Hence, $ds_i^j = 1$.

2° Let a fault of the n_i node reach the n_j node and let:

A1. $d_i^1 = \ldots = d_i^{l1} = d_i^{l+1} = \ldots = d_i^{l+k1} = 1$ and
A2. $d_i^{l1+1} = \ldots = d_i^{l-1} = d_i^{l+k1+1} = \ldots = d_i^r = 0$.

From the assumptions it follows that:

(1) $d_{i_k}^k \cdot s_i^{i_k} = d_i^k = 1$ for $k = 1, \ldots, k1$
(2) $d_{i_k}^k \cdot s_i^{i_k} = d_i^k = 0$ for $k = k1 + 1, \ldots, l$
(3) $v_1 = \ldots = v_{l1} = v_{l+1} = \ldots = v_{l+k1}$
(4) $v_{l1+1} = \ldots = v_{l-1} = v_{l+k1+1} = \ldots = v_m = \overline{c}$

From the rules of reachability calculation (Lemma 7.1) and from (3) and (4) it follows that $ds_i^j = 1$ iff:

C1: $d_{i1}^1 = \ldots = d_{i_{l_1}}^{l_1} = ds_i^{i1} = \ldots = ds_i^{l_1} = d_{i_{l+1}}^{l+1} = \ldots = d_{i_{l+k}}^{l+k1} = 1$, and
C2: $d_{i_{l1+1}}^{l1+1} = \ldots = d_{i_{l-1}}^{l-1} = d_{i_{l+k1+1}}^{l+k1+1} = \ldots = d_{i_r}^r = 0$ or $ds_i^{l1+1} = \ldots = ds_i^{l-1} = d_{i_{l+k1+1}}^{l+k1+1} = \ldots = d_{i_r}^r = 0$.

From (1) and (A1) it appears that condition C1 is satisfied. From (2) and (A2) it results that condition C2 is satisfied. Hence, $ds_i^j = 1$.

" <= "

Let $d_i^k = 1$ for $k = 1, \ldots, l1$ and $k = l+1, \ldots, l+k1$ $(l1 \leq l, l1 + k1 \leq r)$.

1° Let the n_j node be the first node on the path that goes out of the n_i node and let $ds_i^j = 1$. From the assumptions and the rules of reachability calculation it follows that:

(1) $n_1, \ldots, n_r, n_i$ are the same node
(2) $l1 + k1 = r$
(3) $v_1 = \ldots = v_{l1} = v_{l+1} = \ldots = v_{l+k1}$
(4) $v_{l1+1} = \ldots = v_{l-1} = v_{l+k1+1} = \ldots = v_m = \overline{c}$
(5) $l1 = l$

From (1) it follows that the fault of the n_i node propagates to the nodes $n_1, \ldots, n_{l1}, n_{l+1}, \ldots, n_{l+k1}$. From (2) it results that the nodes $n_{l+k1+1}, \ldots, n_m$ do not belong to the R_{n_i} stem region and therefore the fault of the n_i node cannot propagate to these nodes. From the rules of fault propagation and from (3) and (4) it results that the fault of the n_i node propagates to the n_j node.

2° Let $ds_i^j = 1$ and let the fault of the n_i node propagate only to the nodes $n_1, \ldots, n_{l1}, n_{l+1}, \ldots, n_{l+k1}$. From the assumptions it appears that:

(1) $v_1 = \ldots = v_{l1} = v_{l+1} = \ldots = v_{l+k1}$,
(2) $v_{l1+1} = \ldots = v_{l-1} = v_{l+k1+1} = \ldots = v_m = \overline{c}$

From (1) and (2) and rules of fault propagation it results that the fault of the n_i node propagates to the n_j node. ■

Appendix C

Results of the execution of the algorithm for the example in Fig. 2 are as follows:

1. During the preprocessing, the stem regions of nodes x_2 and x_8 are determined:

   $$R_{x_2} = \{x_2, x_6, x_8, x_7, x_{10}, x_{11}, x_{12}, x_{13}\}$$
   $$R_{x_8} = \{x_8, x_7, x_{10}, x_{11}, x_{12}, x_{13}\}.$$

   The output nodes for both regions are nodes x_{12},x_{13} and x_{14}.
   The following characteristic subsets of nodes are determined:

   $$N_0 = \{x_1, x_3, x_4, x_5, x_9, x_{15}, x_{16}, x_{17}\}$$
   $$N_2 = \{x_{14}\}$$
   $$N_3 = \{x_6, x_7, x_{10}, x_{11}\}$$
   $$N_5 = \{x_2\}$$
   $$N_7 = \{x_8\}$$
   $$N_{10} = \{x_{12}, x_{13}\}$$
   $$N_{11} = \emptyset$$
   $$N_{15} = \emptyset$$

   and the pseudoedge s_{x2}^{x8} connected with nodes x_7, x_{10} and x_{11}, is added.
   Next, an appropriate code of the fast fault grading program is generated.
2. Let the input pattern of the circuit be equal:

   $$(x_1, x_2, x_3, x_4, x_5, x_9, x_{15}, x_{16}) = (0, 1, 1, 1, 1, 1, 1, 0).$$

The execution of the program that was generated in step 1 will determine logic values and the reachability evaluated for nodes and pseudoedges (fig. 5.1). The reachability of pseudoedge s_{x2}^{x8} is $ds_{x2}^{x8} = 1$ (the fault of the x_2 node propagates to the x_8 node).

3. Calculating of criticality is done as follows:

   (a) It is to be assumed that nodes x_{12}, x_{13} and x_{14} are critical (primary outputs of the circuit).

   (b) Because x_{12} is the output node of the stem region for x_8, criticality of the x_8 node should be checked. To obtain this, the reachability d_{x8}^{x12} is calculated:

   $$[v_{x12}, d_{x8}^{x12}] = D_{AND}[[v_{x6}, 0], V_{x7}, V_{x10}] = [0, 0]$$

   Because $d_{x8}^{x12} = 0$, the fault of the x_8 node does not propagate to the x_{12} node.

   (c) The x_{12} node is also the output node of stem region of the x_2 node. Criticality of the x_2 node should be checked. To this goal the reachability d_{x2}^{x12} is calculated:

   $$[v_{x12}, d_{x2}^{x12}] = D_{AND}[V_{x6}, [v_{x7}, ds_{x2}^{x8} \cdot d_{x8}^{x7}], [v_{x10}, ds_{x2}^{x8} \cdot d_{x8}^{x10}]] = [0, 0]$$

   Because $d_{x2}^{x12} = 0$ then the x_2 fault does not propagate to the x_{12} node.

   (d) Analogical calculations should be done for the nodes x_{13} and x_{14}. As the result of these calculations follows:

   $$[v_{x13}, d_{x8}^{x13}] = D_{AND}[V_{x10}, V_{x11}] = [1, 1]$$

   $$[v_{x13}, d_{x2}^{x13}] = D_{AND}[[v_{x10}, ds_{x2}^{x8} \cdot d_{x8}^{x10}], [v_{x11}, ds_{x2}^{x8} \cdot d_{x8}^{x11}]] = [1, 1]$$

   $$[v_{x14}, d_{x8}^{x14}] = D_{AND}[V_{x11}, [v_{x17}, 0]] = [0, 0]$$

   $$[v_{x14}, d_{x2}^{x14}] = D_{AND}[[v_{x11}, ds_{x2}^{x8} \cdot d_{x8}^{x11}], [v_{x17}, 0]] = [0, 0]$$

   Because the faults of the x_2 and x_8 nodes propagate to the x_{13} output, nodes x_2 and x_8 are critical.

   (e) Sensitive inputs of the x_{12} gate are determined. The only sensitive input of the x_{12} gate is the x_6 node and so it is critical.

   (f) Sensitive inputs of the x_{13} gate are determined. The x_{13} gate has both inputs sensitive. Nodes x_{10} and x_{11} are critical.

   (g) Sensitive inputs of the x_{14} gate are determined. The only sensitive input of the x_{14} gate is the x_{17} node. This node is critical.

   (h) Sensitive inputs of the x_6 gate are determined. The only sensitive input of the x_6 gate is the x_1 node. This node is critical.

v_j, d_i^j / v_k, d_i^k	0 0	0 1	1 0	1 1
0 0	0 0	0 0	0 0	0 0
0 1	0 0	0 1	0 1	0 0
1 0	0 0	0 1	1 0	1 1
1 1	0 0	0 0	1 1	1 1

Table 8. Reachability function for AND gate

(i) Because an output of the x_7 gate is not critical so inputs of this gate are not analyzed.

(j) Sensitive inputs of the x_{10} gate are determined. This gate has both inputs sensitive but the x_8 node has reconvergent fan-out so only the x_4 node is marked as critical.

(k) Sensitive inputs of the x_{11} gate are determined. This gate has both inputs sensitive but the x_8 node has reconvergent fan-out. The only critical node is the x_5 one.

(l) Sensitive inputs of the x_8 gate are determined. This gate has both inputs sensitive but the x_2 node has reconvergent fan-out. The only node marked as critical is the x_9 node.

(m) Sensitive inputs of the x_{17} gate are determined. The x_{16} node is the sensitive input of the x_{17} gate. This node is critical.

Finally, the following list of critical nodes is determined: x_1, x_2, x_4, x_5, x_6, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{16}, x_{17}.

From this it appears that the test detects the following faults: $x_1/1$, $x_2/0$, $x_4/0$, $x_5/0$, $x_6/1$, $x_8/0$, $x_9/0$, $x_{10}/0$, $x_{11}/0$, $x_{12}/1$, $x_{13}/0$, $x_{14}/1$, $x_{16}/1$, $x_{17}/1$.

In the example mentioned above the table D_{AND} is used to determine the reachability of AND gate output. It is defined as in Table 8.

Appendix D

Algorithm of the preprocessing. A draft of the preprocessing is as follows:

procedure Preprocessing;
begin
$N_o := N;$
$N_{nc} := N;$
$N_{np} := N;$
$N_i := \emptyset;$

```
N_ex := ∅;
N_en := ∅;
N_c := ∅;
N_p := ∅;
Creation of a G_o graph for the given G graph and determination spanning tree
and unique cycles of this graph.
for each node n_i ∈ G_o do
  begin
  if a n_i node is not an input node of any unique cycle then
    a n_i node has no reconvergent fan-out
  else
    begin
    Reducing the G_o(n_i) graph to the G(n_i) graph modyfing adequate unique
    cycles;
    Determination a set of unique cycles creating a stem region of the n_i node
    (according to the Theorem 7.1);
    N_c := N_c ∪ {n_i};
    N_nc := N_nc − {n_i};
    if n_i ∈ N_i then
      begin N_en := N_en ∪ {n_i}; N_o := N_o − {n_i} end;
    for each node n_j belonging to R_{n_i} do
      begin
      N_o := N_o − {n_j};
      if origin nodes of at least two edges going into n_j belong either to N_i or
      N_en then
        begin
        N_p := N_p ∪ {n_j}; N_np := N_np − {n_j}
        end;
      if a n_j node is an output node of the R_{n_i} graph then
        N_ex := N_ex ∪ {n_j}
      else
        begin N_i := N_i ∪ {n_i}; N_ex := N_ex − {n_j} end;
      if n_j ∈ N_en then N_en := N_en − {n_j}
      end
    end
  end;
  for each node n_j ∈ N_o do
    if a origin node of at least one edge going into n_j belongs either to N_i
    or N_en then
      begin N_ex := N_ex ∪ {n_j}; N_o := N_o − {n_j} end;
  Determining sets N_0, N_2, N_3, N_5, N_7, N_10, N_11, N_15.
  Creating pseudoedges for nodes belonging to sets N_7, N_11 and N_15.
end;
```