# Fast Identification of Robust Dependent Path Delay Faults

U. Sparmann*, D. Luxenburger*, K.-T. Cheng†, S.M. Reddy‡

Technical Report 08/1994
SFB 124–B1

* Fachbereich 14 – Informatik, Universität des Saarlandes,
D 66041 Saarbrücken, Germany

† Department of ECE, University of California,
Santa Barbara, CA 93106, USA

‡ Department of ECE, University of Iowa,
Iowa City, IA 52242, USA

# Fast Identification of Robust Dependent Path Delay Faults**

U. Sparmann*, D. Luxenburger*, K.-T. Cheng†, and S.M. Reddy‡

* Computer Science Dept., University of Saarland, D 66041 Saarbrücken, Germany

† Department of ECE, University of California, Santa Barbara, CA 93106, USA

‡ Department of ECE, University of Iowa, Iowa City, IA 52242, USA

**Abstract**

*Recently, it has been shown in [LSBSV93] and [CC93] that in order to verify the correct timing of a manufactured circuit not all of its paths need to be considered for delay testing. In this paper, a theory is developed which puts both of the above papers in a common framework, thus allowing for a better understanding of their relation. In addition, we consider the computational problem of identifying large sets of such not-necessary-to-test paths. Since the approach of [LSBSV93] can only be applied for small scale circuits, we develop a new algorithm which trades quality of the result against computation time, and allows to handle large circuits with tens of millions of paths. Experimental results show that enormous improvements in running time are only paid for by a small decrease in quality.*

## 1 Introduction

The purpose of delay testing is to ascertain that a manufactured digital circuit meets its timing specifications. Two fault models have been proposed in this context, namely gate delay [BR83] and path delay faults [Smi85]. In this paper we will focus on path delay faults, which are more powerful since they also model distributed defects. Test generation and fault simulation methods with respect to this fault model have been studied considerably in the literature. (See for example [Smi85, RLP87, FFS91, MSS+91, BAA92, PRU93, HWA94, CC94, ULKK94, Var94].)

A major problem in path delay fault testing is that the number of paths in a circuit is often extremely large. Thus, the question of whether really all of these paths have to be tested in order to verify the temporal correctness of a manufactured circuit is of great interest. Two important results with respect to this question have been reported recently. In [LSBSV93] ([CC93]) the class of robust dependent (functional unsensitizable) paths has been introduced, and it has been shown that these paths need not be considered for the purpose of delay testing.

The contribution of this paper with respect to the above work is twofold. First, a theory is developed which allows to put the results of [LSBSV93] and [CC93] in a common framework, and thus, helps for a better understanding of their relation. Our approach is based on the idea of choosing for each input vector $v$ to circuit $C$ a 'stabilizing system', i.e. a subcircuit $S_v$ of $C$ which can stabilize the primary outputs of $C$ on their final values under $v$ independent of the remaining circuitry. For a complete delay test of $C$, it can be shown that it is sufficient to only check the logical paths included in these stabilizing systems robustly. The remaining paths which need not be tested form a robust dependent path set (RD-set) as computed by the approach of [LSBSV93]. They include the functional unsensitizable paths of [CC93] as a subset.

Usually, there is a huge number of different possibilities for choosing the stabilizing system $S_v$ for a given input $v$. Thus, we arrive at the optimization problem of selecting $S_v$ for every input $v$ such that the overall set of logical paths which need to be tested is minimized (the corresponding RD-set is maximized). The second major contribution of this paper is motivated by the fact that the procedure of [LSBSV93] for identifying a near maximum RD-set is very time and space consuming, and thus, can only be applied for small scale circuits. (As an example, for circuit c499 of the ISCAS85 benchmarks [BF85] this algorithm was not able to complete in a running time of 69 hours on a SUN SPARC 10 workstation.) We will show that the approach of [CC93] can be generalized to allow for a much faster identification of large RD-sets. (For circuit c499 our new algorithm runs less than 4 minutes, which corresponds to a speed-up factor of over

**This work has been supported by DFG, SFB 124-"VLSI Entwurfsmethoden und Parallelität" and Grant No. Sp431/1-1.

1000 compared to the method of [LSBSV93].) This speed-up is achieved by considering only a restricted search space for selection of the RD-set, and applying an approximation method for computing the elements of this set. As a consequence, the savings in computation time have to be paid by a decrease in quality of the result, i.e. size of the identified RD-set. Eperimental results show that this loss in quality is only small.

The paper is organized as follows: In Section 2 some basic definitions concerning delay testing are reviewed. The theoretical framework allowing for an easy comparison of the results presented in [LSBSV93] and [CC93] is developed in Section 3. Section 4 shows how to generalize the methodology of [CC93] for fast RD-set identification. Heuristics for improving the quality of this approach are discussed in Section 5. Section 6 gives experimental results and compares them to the approach of [LSBSV93].

## 2 Basic Definitions

We will restrict to single output combinational circuits in the following. For multi output circuits the theory is applied for each output cone separately. In our model a combinational circuit $C$ consists of leads, and gates. As gate types we will consider simple gates (And, Or, Nand, Nor, and Not), as well as primary inputs (Pis) and primary outputs (Pos). A lead is a wire connecting two gate pins with each other.

Let $P = (g_0, l_0, g_1, \ldots, l_{m-1}, g_m)$ be a *physical path* in $C$ (given by an alternating sequence of gates and leads) leading from a Pi $g_0$ to a Po $g_m$. The primary input $g_0$ of $P$ is denoted by $\mathrm{PI}(P)$. As usual in delay fault test generation we will associate two *logical paths* with each physical path. A logical path is given as a tuple $(P, t)$ with $t = \overline{x} \rightarrow x$, $x \in \mathbf{B} = \{0, 1\}$, being a transition at the primary input $\mathrm{PI}(P)$.

A manufactured implementation of circuit $C$ is denoted by $C_m$. We assume that $C_m$ has the gate level structure of $C$, but the delays of its gates and lines may take arbitrary values due to variations of the fabrication process or manufacturing defects. Let $\tau$ be the required operation time for $C_m$. $C_m$ is said to have a *delay fault* $> \tau$ on logical path $(P, t)$ iff its delay for propagating $t$ over $P$ exceeds $\tau$. A *robust test* for $(P, t)$ [LR87] is a two pattern sequence which can be used to measure the delay of $(P, t)$ in any implementation $C_m$ of $C$. From the fact that $C_m$ does (does not) operate correctly for this test sequence under clock period $\tau$ it can be concluded that the delay of $C_m$ for propagating $t$ over $P$ is $\leq \tau$ ($> \tau$).

In [CC93, LSBSV93] it has been shown that in order to guarantee that the delay of a manufactured circuit meets its specification not all logical paths need to be checked robustly.

**Definition 1 ([LSBSV93])** *Let $\mathcal{LP}(C)$ be the set of all logical paths in a circuit $C$ and $\mathcal{R}$ a subset of $\mathcal{LP}(C)$.*

*$\mathcal{R}$ is said to be* robust dependent (RD). $\Longleftrightarrow$
*For all implementations $C_m$ of $C$ and all clock periods $\tau$ it holds that: The absence of delay faults $> \tau$ on the logical paths from $\mathcal{LP}(C) \setminus \mathcal{R}$ implies the delay of $C_m$ is $\leq \tau$.*

In [LSBSV93] the authors reduce the problem of identifying a (maximum) RD-set $\mathcal{R}$ to the problem of finding (maximum) redundant multiple stuck-at-0 (stuck-at-1) faults in the leaf-dag of $C$. Since the leaf-dag is the 'unfolded' version of $C$ with fan-out only allowed at the Pis, its size is exponential in the size of $C$ for circuits with large amount of internal fan-out.

To cope with this problem, a heuristic is developed in [LSBSV93] which gradually unfolds the circuit, searches for redundant single stuck-at faults, and removes them to reduce the size of the unfolded circuit. But even this heuristic algorithm has very large running times and may result in a circuit of exponential size. Thus, it can only be applied for small scale circuits.

The purpose of this paper is to develop an algorithm for the computation of large RD-sets which is based on [CC93], and does not rely on unfolding the circuit. As will be seen, this approach is much faster than the heuristic of [LSBSV93]. In order to develop our new algorithm for RD-set identification, we first have to introduce the notions of *stabilizing system* and *complete stabilizing assignment*. These notions will also be helpful for a better understanding of the results in [LSBSV93, CC93] and their relation.

## 3 Stabilizing Systems

Consider a circuit $C$ realizing function $f$. For a given input vector $v \in \mathbf{B}^n$ we ask for a subcircuit of $C$ which can stabilize the Po on $f(v)$ independent of the remaining circuitry of $C$. Such a subcircuit will be called a *stabilizing system* of $C$ for input $v$. (Note, that a stabilizing system normally only constitutes a small portion of the overall circuit. For example to stabilize the output of an Or gate to logic 1 it is sufficient to stabilize one of its inputs to 1.) There can exist many different *stabilizing systems* to $v$. Such a system can be computed as follows:

**Algorithm 1**
/* computes stabilizing system $S$ to $v \in \mathbf{B}^n$ */
Include the Po of $C$ and the lead connecting to it in $S$.
While there exists a gate of $C$ which is not included in $S$ and drives a lead already belonging to $S$:
(1) If $g$ is a Not gate:
Include $g$ and its input lead in $S$.
(2) If $g$ is a And, Or, Nand, Nor gate:
(a) If the stable input values of $g$ under $v$ are all non-controlling[1], then include all of the input leads of $g$ in $S$.
(b) If a set $L = \{l_1, \ldots, l_k\}$, $k \geq 1$, of the input leads of $g$ has controlling stable values under v, then include an arbitrary lead from $L$ in $S$.
(3) If $g$ is a Pi: Include $g$ in $S$.
□

**Definition 2** *A* stabilizing system $S$ *of circuit* $C$ *for input* $v$ *is a subcircuit of* $C$ *computed by Algorithm 1.*

*The set of logical paths of stabilizing system* $S$ *for input* $v$ *is given by:*

$$\mathcal{LP}(v,S) := \{(P, \overline{x} \rightarrow x) | (P \text{ path from } \mathrm{PI} \text{ to } \mathrm{PO} \text{ in } S) \text{ and } (x \text{ value of } \mathrm{PI}(P) \text{ under } v)\}$$

Note, that a stabilizing system $S$ computed by Algorithm 1 is *minimum* in the following sense: If any lead is removed from $S$ then the property of stabilizing the output of $C$ independent of the values on leads not included in $S$ is not guaranteed any more. This is due to the fact that only one input of set $L$ is picked in Step 2(b) of Algorithm 1.

Usually, there are many possible stabilizing systems to $v$ depending on which input of set $L$ is selected in Step 2(b). This fact is illustrated by the following example.

**Example 1** *Figure 1 shows three possible stabilizing systems (indicated by the lines drawn in bold) for input* $v = 111$ *in an example circuit taken from [LSBSV93].*

**Definition 3** *A* complete stabilizing assignment $\sigma$ *is a mapping which assigns to each input vector* $v \in \mathbf{B}^n$ *a stabilizing system to* $v$*, denoted by* $\sigma(v)$*.*

*The set of all logical paths corresponding to the stabilizing systems selected by* $\sigma$ *is given by:*

$$\mathcal{LP}(\sigma) := \bigcup_{v \in \mathbf{B}^n} \mathcal{LP}(v, \sigma(v))$$

[1]The non-controlling value for an And, Nand (Or, Nor) gate is 1 (0). The controlling value is the complement of the non-controlling one.

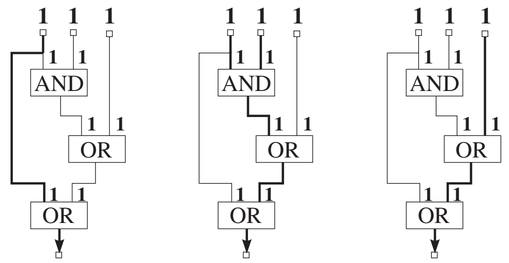


Figure 1: Stabilizing systems for input 111

From Example 1 it becomes clear that there is usually a huge number of possible complete stabilizing assignments for a given circuit.

The following theorem shows that it is sufficient to only test the logical paths from $\mathcal{LP}(\sigma)$ robustly.

**Theorem 1** *Let* $\mathcal{LP}(C)$ *denote the set of all logical paths in* $C$*, and consider an arbitrary complete stabilizing assigment* $\sigma$ *for* $C$*. Then:*

*Every subset from* $\mathcal{RD}(\sigma) := \mathcal{LP}(C) \setminus \mathcal{LP}(\sigma)$ *is an RD-set.*

**Proof**:
It follows directly from Definition 1 that any subset of an RD-set is again an RD-set. Thus, it suffices to show that $\mathcal{RD}(\sigma)$ is an RD-set, i.e. we have to prove that for all implementations $C_m$ of $C$ and all clock periods $\tau$ it holds that: The absence of delay faults $> \tau$ on the logical paths from $\mathcal{LP}(\sigma)$ implies the delay of $C_m$ is $\leq \tau$.

Let $v$ be an arbitrary input vector to $C_m$. Consider the stabilizing system $\sigma(v)$ which is assigned to $v$ by $\sigma$. From the definition of stabilizing system it directly follows that it is sufficient to switch the lines included in $\sigma(v)$ in order to stabilize the output of $C_m$ on $f(v)$ for input vector $v$. The delay for switching $\sigma(v)$ is bounded by the maximum of the delays of all logical paths in $\mathcal{LP}(v, \sigma(v))$. Since by assumption the delay of all these logical paths is $\leq \tau$ in $C_m$, it follows that the output of $C_m$ stabilizes in time $\leq \tau$ for input $v$. ■

**Example 2** *Figure 2 gives a possible choice of* $\sigma$ *for the circuit of Example 1. The circuit leads belonging to a stabilizing system are drawn in bold. The two leftmost stabilizing systems are assigned to more than one input combination, i.e. all input combinations which set the leftmost* Pi *to* 1 *(resp. the leftmost* Pi *to* 0 *and the rightmost* Pi *to* 1*).*

*From Theorem 1 it follows that it is not necessary to test all* 8 *logical paths of the example circuit. Instead it*

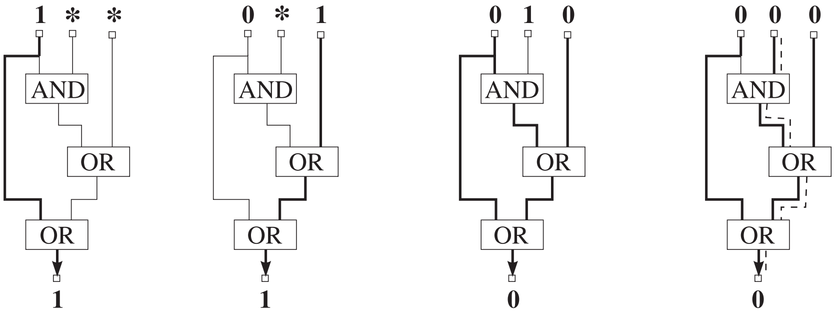


Figure 2: Complete stabilizing assigment

*is sufficient to only test the* 6 *logical paths from* $\mathcal{LP}(\sigma)$ *robustly in order to verify the temporal correctness of the circuit. (Note, that the logical path indicated by the dashed line in the rightmost stabilizing system is the only element from* $\mathcal{LP}(\sigma)$ *which can not be tested robustly.)*

Before considering how to compute a complete stabilizing assignment $\sigma$ and its associated RD-set $\mathcal{RD}(\sigma)$, let us first compare the above theory with the results of [CC93, LSBSV93].

In [CC93] the notion of functional sensitizable paths has been introduced. It has been shown, that functional unsensitizable paths are redundant, and need not be considered for delay testing.

**Definition 4 ([CC93])** *A logical path* $(P, \overline{x} \to x)$ *is called* functional sensitizable. $\Longleftrightarrow$ *There exists an input vector* $v$ *such that:*

(FU1) *$v$ sets* PI$(P)$ *to* $x$*, and*

(FU2) *for each gate* $g$ *on* $P$ *with its on-path input having a non-controlling stable value under* $v$*, all the side-inputs of* $g$ *have non-controlling stable values for* $v$*.*

It is interesting to compare the above defintion to the criterion for non-robust testability:

**Lemma 1 ([SFF89])** *A logical path* $(P, \overline{x} \to x)$ *is* non-robustly testable. $\Longleftrightarrow$ *There exists an input vector* $v$ *such that:*

(NR1) *$v$ sets* PI$(P)$ *to* $x$*, and*

(NR2) *for each gate* $g$ *on* $P$ *all the side-inputs of* $g$ *have non-controlling stable values for* $v$*.*

**Proof**:
For the original criterion of [SFF89] the existence of a two pattern sequence $< v_1, v_2 >$ is claimed. Since the existence of $v_1$ is always guaranteed for a circuit without input space restrictions, this condition has been omitted. ■

**Definition 5** *Let* $C$ *be a combinational circuit.*

*(a) The set of logical paths from* $\mathcal{LP}(C)$ *which are functional sensitizable is denoted by* $\mathcal{FS}(C)$*.*

*(b) The set of logical paths from* $\mathcal{LP}(C)$ *which are non-robustly testable is denoted by* $\mathcal{T}(C)$*.*

Since the conditions for non-robust testability are more restrictive than those for functional sensitizability, it follows that $\mathcal{T}(C) \subset \mathcal{FS}(C)$. The following lemma classifies the logical path sets selected by choosing a complete stabilizing assignment with respect to this hierarchy.

**Lemma 2** *Let* $\sigma$ *be an arbitrary complete stabilizing assignment for circuit* $C$*. Then:*

$$\mathcal{T}(C) \subset \mathcal{LP}(\sigma) \subset \mathcal{FS}(C)$$

**Proof**:
$\mathcal{T}(C) \subset \mathcal{LP}(\sigma)$:
Let $(P, \overline{x} \to x)$ be a non-robustly testable logical path in circuit $C$. By Lemma 1 there exists an input vector $v$ for $C$ fulfilling conditions (NR1) and (NR2). We will show that $P$ is included in any stabilizing system $S$ to $v$ computed by Algorithm 1.
The proof is done inductively: Clearly, the Po of $P$ is included in $S$. Let $g$ be the next gate of $P$, and $l$ its on-path input. If $g$ is a Not then $g$ and $l$ will be included in $S$. If $g \in \{$And, Or, Nand, Nor$\}$ two cases have to be distinguished: If the *stable value of $l$ under $v$ is non-controlling*, then by condition (NR2) all side-inputs of $l$ have non-controlling stable values too, and thus, in Step 2(a) of Algorithm 1 all input leads of $g$ will be included in $S$. If the *stable value of $l$ under $v$ is controlling*, it follows from condition

(NR2) that $l$ is the only element of set $L$ in Step 2(b) of Algorithm 1, and thus, will be selected for $S$.

$\mathcal{LP}(\sigma) \subset \mathcal{FS}(C)$:
Consider an arbitrary logical path $(P, \overline{x} \to x) \in \mathcal{LP}(v, \sigma(v))$. We will show that $(P, \overline{x} \to x)$ is functional sensitized by $v$.
From the definition of $\mathcal{LP}(v, \sigma(v))$ it follows that $v$ sets PI($P$) to $x$, i.e. condition (FU1) is fullfilled. For a gate $g$, Algorithm 1 only includes an input lead of $g$ with non-controlling stable value in $\sigma(v)$, if all other input leads of $g$ have non-controlling stable values under $v$ (see Step 2(a)). Thus, all paths from $\sigma(v)$ also fulfill condition (FU2) from the definition of functional sensitizability for input $v$. ■

Thus, by choosing a complete stabilizing assignment $\sigma$ we are selecting a subset $\mathcal{LP}(\sigma)$ of the functional sensitizable logical paths $\mathcal{FS}(C)$ which includes all robustly and non-robustly testable logical paths $\mathcal{T}(C)$. The corresponding situation is illustrated in Figure 3.

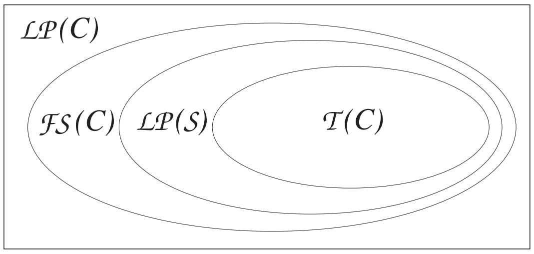


Figure 3: Hierarchy of logical path sets

Comparing Theorem 1 to the methodology developed in [LSBSV93] for identifying RD-sets, it can be shown that both approaches are equivalent:

**Lemma 3** *The RD-sets characterized by Theorems 2.1 and 2.2 of [LSBSV93] and Theorem 1 are the same.*

**Proof**: (Sketch)
Since the RD-sets characterized by Theorem 2.1 of [LSBSV93] include those given by Theorem 2.2 as a special case, it suffices to compare Theorem 2.1 and Theorem 1.

A mapping, which assigns a stabilizing system to each input vector $v$ with $f(v) = x$, is called a *complete $x$-stabilizing assignment $\sigma_x$*.

By transforming the circuit to a leaf-dag in [LSBSV93] each path in $C$ is 'personalized'. In addition, inverters are moved to the PIs to make the paths internally non-inverting. As representative for each path an I-edge is chosen, which is either the input lead of the first gate on the path (non-inverting path), or the output lead of the paths inverter (inverting path).

If a multiple stuck-at-$\overline{x}$ fault at the I-edges corresponding to a set of paths $\mathcal{P}$ is redundant, this is equivalent to the fact that there exists a complete $x$-stabilizing assignment $\sigma_x$ such that none of its stabilizing systems contains a path from $\mathcal{P}$. (The I-edges of the paths from $\mathcal{P}$ need not be able to switch to $x$ since the paths from $\mathcal{P}$ are not necessary for stabilizing output value $x$.) ■

After having compared our approach to previous results from the literature, let us now concentrate on the question of how to choose a complete stabilizing assignment $\sigma$. Here, we arrive at the following optimization goal: "Choose $\sigma$ such that the size of $\mathcal{LP}(\sigma)$ is minimum." Choosing $\mathcal{LP}(\sigma)$ as small as possible, minimizes the number of paths which have to be considered for test generation. In addition, it maximizes the fault coverage, which by Theorem 1 is given as the number of testable logical paths divided by $|\mathcal{LP}(\sigma)|$. Finally, it helps to reduce design for testability overhead, since the logical paths from $\mathcal{LP}(\sigma)$ which are not testable have to be considered for design for testability modifications. The following example illustrates the above facts.

**Example 3** *In Example 2 a possible complete stabilizing assignment $\sigma$ has been given for our example circuit (see Figure 2). $\mathcal{LP}(\sigma)$ consists of 6 logical paths from which only 5 are robustly testable. The 6-th logical path (indicated by the dashed line in Figure 2) is neither robustly nor non-robustly testable. Thus, the fault coverage for this example would only be $\frac{5}{6} \cdot 100\%$, and the dashed path would have to be considered for design for testability modifications.*

*Figure 4 shows a different choice for the stabilizing system to input $(0,0,0)$. The set of logical paths for the resulting complete stabilizing assignment $\sigma'$ consists only of the five logical paths of $C$ which are robustly testable. Thus, the actual fault coverage when testing these five paths is 100%, and no design for testability modifications are necessary.*

## 4 RD-set identification without circuit unfolding

As already mentioned, the heuristic of [LSBSV93] for computing a minimum RD-set gradually unfolds the circuit and searches for stuck-at redundancies (corresponding to sets of RD-paths). In this section a new

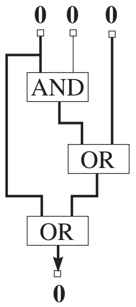


Figure 4: Stabilizing system to input 000

methodology is developed which avoids unfolding of the circuit.

**Definition 6** *An* input sort $\pi$ *of a circuit $C$ is a mapping which orders the inputs of each gate $g$ of $C$ completely, i.e. for each gate $g$ its input leads are numbered from 1 to fan-in$(g)$.*
*For an input lead $l$ of gate $g$, $\pi(g,l)$ denotes the position of $l$ with respect to the input sort of $g$.*

Given an input sort $\pi$ we can fix a complete stabilizing assignment $\sigma^\pi$ as follows:

> For an input vector $v$ the corresponding stabilizing system $\sigma^\pi(v)$ is obtained by applying Algorithm 1 with the following restriction: In Step 2(b) always choose the input lead $l \in L$ with minimum $\pi(g,l)$.

Now instead of considering all possible choices for complete stabilizing assignments only the assignments from $\{\sigma^\pi | \pi \text{ arbitrary input sort}\}$ will be considered. This restricts the search space but will allow us to work more efficiently.

**Example 4** *Figure 5 gives a possible input sort $\pi$ for our example circuit. The corresponding complete stabilizing assignment $\sigma^\pi$ is the optimum assignment of Example 3. Thus, for this circuit the restricted search space still contains the optimum solution.*

Let us assume that a good input sort $\pi$ is given. (The problem of how to best choose such a sort will be discussed later.) The logical path set to $\sigma^\pi$ (the corresponding RD-set $\mathcal{RD}(\sigma^\pi)$) can be computed based on the following lemma:

**Lemma 4** *For a circuit $C$, let $\pi$ be an input sort, and $(P, \overline{x} \to x)$ a logical path in $C$.*
*$(P, \overline{x} \to x) \in \mathcal{LP}(\sigma^\pi) \iff$ There exists an input vector $v \in \mathbf{B}^n$ such that:*

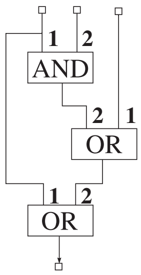


Figure 5: Optimum input sort

($\pi$1) *$v$ sets $\mathrm{PI}(P)$ to $x$,*

($\pi$2) *for each gate $g$ of $P$ such that its on-path input has a non-controlling stable value under $v$, all the side-inputs of $g$ have non-controlling stable values under $v$,*

($\pi$3) *for each gate $g$ of $P$ such that its on-path input $l$ has a controlling stable value under $v$, all low-order[2] side-inputs of $l$ have non-controlling stable values under $v$.*

**Proof**:
It suffices to show that: $v$ satisfies conditions ($\pi$1)-($\pi$3) for path $P \iff (P, \overline{x} \to x) \in \mathcal{LP}(v, \sigma^\pi(v))$.

"$\Rightarrow$": Let $v$ be an input vector fulfilling conditions ($\pi$1)-($\pi$3) for path $P$. We inductively show that $P$ will be included in $\sigma^\pi(v)$:
Clearly, the PO of $P$ is included in $\sigma^\pi(v)$. Let $g$ be the next gate of $P$, and $l$ its on-path input. If $g$ is a NOT then $g$ and $l$ will be included in $\sigma^\pi(v)$. If $g \in \{\text{AND}, \text{OR}, \text{NAND}, \text{NOR}\}$ two cases have to be distinguished: If the *stable value of $l$ under $v$ is non-controlling*, then by condition ($\pi$2) all side-inputs of $l$ have non-controlling stable values too, and thus, all input leads of $g$ are included in $\sigma^\pi(v)$ by Algorithm 1. If the *stable value of $l$ under $v$ is controlling*, then by condition ($\pi$3) all low-order side-inputs of $l$ have non-controlling stable values. Thus, in Step 2(b) of Algorithm 1 $l$ is the element of set $L$ with lowest sort number and will be selected for $\sigma^\pi(v)$.

"$\Leftarrow$": Since this direction is proven analogously, it is omitted here for brevity. ■

**Remark 1** *If condition ($\pi$3) of Lemma 4 is omitted, we obtain the conditions of Definition 4 for functional sensitizable paths.*

In [CC93] an algorithm has been given to determine which logical paths are functional sensitizable.

[2] Let $g$ be a gate and $l$ an input to $g$. The *low-order side-inputs of $l$* are all inputs $l'$ to $g$ with $\pi(g,l') < \pi(g,l)$.

In order to save running time, this algorithm does not compute the actual set $\mathcal{FS}(C)$ but a small superset $\mathcal{FS}^{sup}(C)$which gives a very good approximation of $\mathcal{FS}(C)$. Based on the similarity of their characterizations, the same methodology can also be applied for approximating $\mathcal{LP}(\sigma^{\pi})$ for a given input sort $\pi$:

**Algorithm 2** All logical paths of the circuit are implicitly[3] enumerated. For each logical path $(P, \overline{x} \rightarrow x)$ we check whether there exists an input vector $v$ satisfying conditions ($\pi$1)-($\pi$3). If such an input vector exists, then $(P, \overline{x} \rightarrow x) \in \mathcal{LP}(\sigma^{\pi})$. In order to speed-up the computation process, the above check is not performed exactly. As suggested in [CC93], only the local implications induced by conditions ($\pi$1)-($\pi$3) are performed to see whether they result in a contradiction. If there is no contradiction, the logical path is assumed to be an element of $\mathcal{LP}(\sigma^{\pi})$. Thus, the algorithm actually computes a superset $\mathcal{LP}^{sup}(\sigma^{\pi})$ of $\mathcal{LP}(\sigma^{\pi})$ (subset $\mathcal{RD}^{sub}(\sigma^{\pi}) := \mathcal{LP}(C) \setminus \mathcal{LP}^{sup}(\sigma^{\pi})$ of $\mathcal{RD}(\sigma^{\pi})$). As will be seen in the experimental results of Section 6, the quality of the approximation $\mathcal{LP}^{sup}(\sigma^{\pi})$ is very good.

Since the generalizations of the procedure given in [CC93] are straight forward, we omit details here for brevity. □

**Remark 2** *The procedure of [CC93] can also be applied to compute a superset $\mathcal{T}^{sup}(C)$ approximating the set $\mathcal{T}(C)$ of all non-robustly testable logical paths. This is done by (implicitly) enumerating all logical paths, and checking for each logical path whether there exists an input vector $v$ fulfilling the conditions of Lemma 1.*

## 5 How to choose the input sort?

Let us now turn to the optimization problem of choosing the input sort $\pi$ such that $\mathcal{LP}(\sigma^{\pi})$ is minimized. Since solving this problem exactly is too time consuming, we will focus on fast heuristics computing a near optimum solution.

**Definition 7** *Let $l$ be an input lead to a gate $g$ in circuit $C$.*

*(a) The set of all physical (logical) paths of $C$ going through $l$ is denoted by $\mathcal{P}(l)$ ($\mathcal{LP}(l)$).*

*(b) The set of logical paths from $\mathcal{LP}(l)$ such that the transition at $l$ has as final value the controlling value of $g$ is denoted by $\mathcal{LP}_c(l)$.*

**Remark 3** *Clearly, $|\mathcal{LP}_c(l)| = \frac{1}{2}|\mathcal{LP}(l)| = |\mathcal{P}(l)|$.*

Consider a gate $g$ with two inputs $l_1$ and $l_2$ as an example. By chosing the input sort for $g$ we can guide Algorithm 1 which input to select in Step (2)(b) if both inputs have controlling stable values. As a consequence, we can control whether paths from $\mathcal{LP}_c(l_1)$ or $\mathcal{LP}_c(l_2)$ are prefered for inclusion in $\mathcal{LP}(\sigma^{\pi})$. Now let us assume that $|\mathcal{LP}_c(l_1)| < |\mathcal{LP}_c(l_2)|$. Since we want to minimize the size of $\mathcal{LP}(\sigma^{\pi})$, it is a good idea to preferably include paths from $\mathcal{LP}_c(l_1)$ in $\mathcal{LP}(\sigma^{\pi})$. Our first heuristic for choosing the input sort is based on this observation.

**Heuristic 1** *Choose $\pi$ such that for any two input leads $l$ and $l'$ of a gate $g$ in $C$:*

$$|\mathcal{LP}_c(l)| < |\mathcal{LP}_c(l')| \Longrightarrow \pi(g,l) < \pi(g,l')$$

*If $|\mathcal{LP}_c(l)| = |\mathcal{LP}_c(l')|$, the corresponding inputs can be ordered arbitrarily.*

Since $|\mathcal{LP}_c(l)| = |\mathcal{P}(l)|$, computation of such an input sort simply corresponds to path counting and thus, can be done in linear time with respect to circuit size.

The strategy of Heuristic 1 can be improved based on the following observation:
From Lemma 2 it follows that logical paths which are non-robustly testable (not functional sensitizable) will always (never) be included in $\mathcal{LP}(\sigma^{\pi})$. Thus, $\mathcal{LP}_c(l) \cap (\mathcal{FS}(C) \setminus \mathcal{T}(C))$ gives a better measure for input sorting than $\mathcal{LP}_c(l)$.

**Heuristic 2** *Let $\mathcal{T}_c(l)$ ($\mathcal{FS}_c(l)$) denote the set of all logical paths from $\mathcal{LP}_c(l)$ which are non-robustly testable (functional sensitizable). Choose $\pi$ such that for any two input leads $l$ and $l'$ of a gate $g$ in $C$:*

$$|\mathcal{FS}_c(l) \setminus \mathcal{T}_c(l)| < |\mathcal{FS}_c(l') \setminus \mathcal{T}_c(l')| \Longrightarrow \pi(g,l) < \pi(g,l')$$

*Again, if $|\mathcal{FS}_c(l) \setminus \mathcal{T}_c(l)| = |\mathcal{FS}_c(l') \setminus \mathcal{T}_c(l')|$, the corresponding inputs are ordered randomly.*

An exact computation of the cost measure $|\mathcal{FS}_c(l) \setminus \mathcal{T}_c(l)|$, applied in Heuristic 2 for input sorting, would be too time consuming. Based on the algorithm of [CC93] and Remark 2, the following method can be used for approximating the cost measure:

[3] By applying the concept of 'prime segments' [CC93] the number of paths which have to be considered can be reduced drastically, i.e. we take advantage of the fact that all extensions of a logical path segment which does not fulfill the conditions of Lemma 4 are elements of $\mathcal{RD}(\sigma^{\pi})$.

**Algorithm 3**

/* approximate computation of cost measure for Heuristic 2 */

(1) Run the algorithm of [CC93] for the criterion of Definition 4 to determine the approximation $\mathcal{FS}_c^{sup}(l) \supset \mathcal{FS}_c(l)$ for all lines $l$ of $C$.

(2) Run the algorithm of [CC93] for the criterion of Lemma 1 to determine the approximation $\mathcal{T}_c^{sup}(l) \supset \mathcal{T}_c(l)$ for all lines $l$ of $C$.

(3) Choose the input sort $\pi$ such that:
$|\mathcal{FS}_c^{sup}(l) \setminus \mathcal{T}_c^{sup}(l)| < |\mathcal{FS}_c^{sup}(l') \setminus \mathcal{T}_c^{sup}(l')|$
$\Longrightarrow \quad \pi(g,l) < \pi(g,l')$ □

Clearly, the above method for input sorting is much more time consuming than Heuristic 1. But, as will be seen in the experimental results, it also increases the quality of the sort function (i.e. the size of the computed RD-set) considerably.

## 6 Experimental Results

This section evaluates the effectiveness of the proposed approach with respect to quality of the result, i.e. size of the identified RD-set, and CPU-time (measured on a SUN SPARC 10 workstation).

Table 1 shows results for the ISCAS85 benchmark set. The run for c6288 could not be completed, since this circuit has more than $1.9 \cdot 10^{20}$ logical paths [PR94]. The percentage of logical paths which can be identified as functional unsensitizable [CC93] is given in column FUS. From Lemma 2 it follows that any RD-set obtained by applying Theorem 1 contains these paths as a subset. Columns Heu1 (Heu2) give the percentage of logical paths which were identified RD by Heuristic 1 (resp. Heuristic 2). For most circuits these numbers are considerably larger than the percentage of functional unsensitizable paths. The increase varies from 2.26% for circuit c880 up to 42.3% for circuit c1908. Comparing the results of columns Heu1 and Heu2, it can be seen that Heuristic 2 gives always better results than Heuristic 1. The average improvement in percentage of RD-paths is 2.51%. Column $\overline{\text{Heu2}}$ of Table 1 gives RD-set sizes obtained by choosing the input sort inversely to Heuristic 2. The fact that the percentage of paths which can be classified RD reduces dramatically indicates that our heuristics for choosing the input sort aim in the right direction.

Table 2 gives the number of logical paths and the running times of Heuristics 1 and 2 for each of the ISCAS85 benchmarks. As can be seen, our approach even handles circuit c3540 which has over 57 million logical paths in less than 15 hours. The enormous improvements in running time compared to the approach of [LSBSV93] are illustrated by the fact that for circuit c499 this algorithm had not finished after a running time of 69 hours, i.e. even Heuristic 2 is more than 1000 times faster for this circuit. Comparing the CPU-times of Heuristics 1 and 2, an increase by factor 3 or more can be observed for most of the circuits. This is due to the fact that the procedure of [CC93] has to be executed three times for Heuristic 2, two times for computing the priority function (see Algorithm 3) and once for the actual RD-set computation.

| circuit | FUS | Heu1 | Heu2 | $\overline{\text{Heu2}}$ |
|---|---|---|---|---|
| c432 | 64.25 % | 90.12 % | 91.12 % | 84.29 % |
| c499 | 30.05 % | 39.50 % | 53.79 % | 30.05 % |
| c880 | 0.94 % | 1.81 % | 3.20 % | 0.94 % |
| c1355 | 81.19 % | 83.27 % | 86.70 % | 81.19 % |
| c1908 | 32.79 % | 74.95 % | 75.09 % | 33.34 % |
| c2670 | 77.26 % | 81.27 % | 82.42 % | 77.79 % |
| c3540 | 72.16 % | 94.89 % | 94.99 % | 83.33 % |
| c5315 | 78.05 % | 83.79 % | 83.80 % | 81.74 % |
| c7552 | 68.78 % | 75.63 % | 76.70 % | 72.18 % |

Table 1: Results for ISCAS85 benchmarks

| circuit | total no. of logical paths | CPU-time for Heu1 | CPU-time for Heu2 |
|---|---|---|---|
| c432 | 583,652 | 0:25 | 1:27 |
| c499 | 795,776 | 1:12 | 3:22 |
| c880 | 17,284 | 0:07 | 0:14 |
| c1355 | 8,346,432 | 3:03 | 9:17 |
| c1908 | 1,458,114 | 2:22 | 12:10 |
| c2670 | 1,359,920 | 3:01 | 9:53 |
| c3540 | 57,353,342 | 2:24:06 | 14:29:38 |
| c5315 | 2,682,610 | 3:13 | 10:31 |
| c7552 | 1,452,988 | 4:37 | 15:07 |

Table 2: Running times for Heuristics 1 and 2

Our improvements in running time compared to the approach of [LSBSV93] come from two facts: First, we consider a restricted search space (complete stabilizing assignments $\sigma^\pi$ which are given by an input sort $\pi$), and second, $\mathcal{LP}(\sigma^\pi)$ is not computed exactly but approximated by an upper bound. To estimate the average loss in quality due to the above restrictions we did the following experiment:
We synthesized multi-level implementations for all of

| circuit | no. of logical paths | approach of [LSBSV93] % of RD-paths | CPU-time | new approach (Heuristic 2) % of RD-paths | CPU-time |
|---|---|---|---|---|---|
| apex1 | 13,756 | 8.52 % | 46:39 | 7.89 % | 0:30 |
| Z5xp1 | 20,102 | 94.75 % | 3:44 | 94.14 % | 0:05 |
| apex5 | 23,836 | 60.63 % | 16:15 | 59.43 % | 0:18 |
| bw | 24,380 | 91.37 % | 8:01 | 89.68 % | 0:09 |
| apex3 | 35,270 | 71.53 % | 1:02:54 | 70.95 % | 0:38 |
| misex3 | 40,578 | 67.25 % | 1:39:40 | 63.78 % | 0:31 |
| seq | 52,886 | 63.35 % | 3:59:35 | 57.81 % | 0:42 |
| misex3c | 1,856,452 | 99.53 % | 7:54:22 | 99.29 % | 4:13 |

Table 3: Comparision of Heuristic 2 to the approach of [LSBSV93]

the two-level MCNC benchmarks[4]. We then ran the algorithm of [LSBSV93] and Heuristic 2 for all circuits which could be handled by the approach of [LSBSV93] and which have a non-empty RD-set. On the average the percentage of RD-paths which were identified by our approach was only 2.05% less than for the solution of [LSBSV93]. Table 3 gives the exact results for the largest of these benchmarks. It shows that, while the quality of our approximation is very good for most of the circuits, there are still few designs (misex3 and seq) with a considerable difference.

## 7 Conclusion

In this paper we have studied the problem of which paths actually need to be tested in order to check that a manufactured circuit meets its timing spezificatons. A theory has been developed which puts previous results [CC93, LSBSV93] in a common framework, thus allowing for a better understanding of their relation. Since the approach of [LSBSV93] is too time consuming to be run on large circuits, we have proposed a new methodology for RD-set computation which trades running time against quality of the result. Experimental results have shown that with only a small loss in accuracy huge improvements in running time can be achieved.


### Acknowledgement

The authors want to thank G. Hotz, B. Becker, and A. Saldanha for their support and helpful discussions. In addition, we thank R.K. Brayton, W.K. Lam, A. Saldanha, and A.L. Sangiovanni-Vincentelli for providing their RD-set identification program.


[4]Each circuit was synthesized by applying script.rugged in the SIS system [SSM+92].

## References


[BAA92] D. Bhattacharya, P. Agrawal, and V.D. Agrawal. Delay fault test generation for scan/hold circuits using boolean expressions. In *Proceedings of the 29th Design Automation Conference*, pages 159–164, June 1992.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proceedings IEEE International Symposium on Circuits and Systems*, 1985.

[BR83] Z. Barzilai and B.K. Rosen. Comparision of AC self-testing procedures. In *Proceedings of the 1983 International Test Conference*, pages 89–91, 1983.

[CC93] K.-T. Cheng and H.-C. Chen. Delay testing for non-robust untestable circuits. In *Proceedings of the 1993 International Test Conference*, pages 954–961, 1993.

[CC94] K.-T. Cheng and H.-C. Chen. Generation of high quality non-robust tests for path delay faults. In *31st Design Automation Conference*, pages 365–369, 1994.

[FFS91] K. Fuchs, F. Fink, and M.H. Schulz. DYNAMITE: an efficient automatic test pattern generation system for path delay faults. *IEEE Transactions on CAD*, 1991.

[HWA94] M. Henftling, H.C. Wittmann, and K.J. Antreich. Path hashing to accelerate delay fault simulation. In *31st Design Automation Conference*, pages 522–526, 1994.

[LR87] C.J. Lin and S.M. Reddy. On delay fault testing in logic circuits. *IEEE Transactions on CAD*, pages 694–703, September 1987.

[LSBSV93] W.K. Lam, A. Saldanha, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Delay fault coverage and performance tradeoffs. In *30th Design Automation Conference*, pages 446–451, 1993.

[MSS+91] P.C. McGeer, A. Saldanha, P.R. Stephan, R.K. Brayton, and A. Sangiovanni-Vincentelli. Timing analysis and delay fault test generation using path-recursive functions. In *Proceedings of the International Conference on CAD-91*, pages 180–183, 1991.

[PR94] I. Pomeranz and S.M. Reddy. An efficient non-enumerative method to estimate the path delay fault coverage in combinational circuits. *IEEE Transactions on CAD*, pages 240–250, 1994.

[PRU93] I. Pomeranz, S.M. Reddy, and P. Uppaluri. NEST: A non-enumerative test generation method for path delay faults in combinational circuits. In *Proceedings of the 30th Design Automation Conference*, pages 439–445, 1993.

[RLP87] S.M. Reddy, C.J. Lin, and S. Patil. An automatic test pattern generator for the detection of path delay faults. In *International Conference on CAD*, pages 284–287, 1987.

[SFF89] M.H. Schulz, F. Fink, and K. Fuchs. Parallel pattern fault simulation for path delay faults. In *Proceedings of the 26th Design Automation Conference*, pages 357–363, 1989.

[Smi85] G.L. Smith. Model for delay faults based upon paths. In *Proceedings of the 1985 International Test Conference*, pages 342–349, 1985.

[SSM+92] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proceedings of the International Conference on Computer Design*, pages 328–333, 1992.

[ULKK94] B. Underwood, W. Law, S. Kang, and H. Konuk. Fastpath: A path-delay test generator for standard scan designs. In *International Test Conference*, pages 154–163, 1994.

[Var94] P. Varma. On path-delay testing in a standard scan environment. In *International Test Conference*, pages 164–173, 1994.