PARALLEL PATTERN FAULT SIMULATION OF PATH DELAY FAULTS

Michael H. Schulz Franz Fink Karl Fuchs
Institute of Computer Aided Design, Department of Electrical Engineering
Technical University of Munich, D-8000 Munich 2, West Germany

Abstract

This paper presents an accelerated fault simulation approach for path delay faults. The distinct features of the proposed fault simulation method consist in the application of parallel processing of patterns at all stages of the calculation procedure, its versatility to account for both robust and non-robust detection of path delay faults, and its capability of efficiently maintaining large numbers of path faults to be simulated.

1 Introduction

In view of the steadily increasing quality requirements for VLSI chips and due to the fact that *statistical timing* rather than *worst-case timing* is frequently chosen as the basis for the design of high-speed circuits, delay testing has achieved great theoretical and practical importance. As is well-known, the purpose of delay testing is to ascertain that manufactured digital circuits meet their timing specifications and operate correctly at desired clock rates, which determine the maximum allowable path delay from primary inputs or latch outputs to primary outputs or latch inputs.

Two different delay fault models, called the *gate delay fault model* and the *path delay fault model*, respectively, have been proposed and frequently dealt with. The gate delay fault model has been introduced, in order to model those defects that cause an actual propagation delay through a distinct gate to exceed its worst-case specification [1]. Formerly, it was also referred to as *transition fault model*, which merely allows a *qualitative* consideration of delay faults and which is consequently only useful for the detection of delay faults of large size [2,3,4]. Since being restricted to large delay faults is neither sufficient nor satisfactory, manifold research activities have recently been undertaken, in order to introduce the indispensable *quantitative* point of view, i.e. to explicitly consider the actual size of the gate delay faults during automatic test pattern generation (ATG) and fault simulation [5,6,7,8].

On the other hand, in order to overcome the main deficiency associated with the gate delay fault model, which consists in its restriction to isolated failures, the path delay fault model has been developed [9]. Besides its capability to model distributed failures, which are typically caused by statistical variations in the manufacturing process, it is of particular use for circuits that have been designed on a statistical timing basis. Since those circuits are known to have a non-zero probability for the occurrence of a delay fault, even when all gate delays are within their specified worst-case ranges, delay testing does not only involve the detection of defects resulting in excessive delays, but also the identification of slow paths.

While a considerable number of ATG methods for path delay faults has been described during the last few years [10,11,12,13,14], the problem of fault simulation of path delay faults has only been addressed in [15]. Based upon the six-valued logic proposed in [15], this paper presents an accelerated fault simulation approach for path delay faults, which applies parallel processing of patterns at all stages of the calculation procedure, in order to gain the capability of simulating long pattern sequences for highly complex VLSI circuits within reasonable amounts of CPU-time. Moreover, many paths, that cannot be tested under the restrictive condition of robustness, may be well testable, if we refrain from that condition. This was our motivation for modifying the six-valued logic of [15] and devising a four-valued logic, in order to additionally account for non-robust detection of path delay faults. Finally, in view of the enormous number of paths that typically exist in today's circuits, we have developed a specific data structure, called the path tree, which has been designed to be highly economical in terms of memory requirements and which allows us to efficiently maintain hundreds of thousands of path faults to be simulated.

2 Background and Basic Definitions

2.1 Hardware Model

From an operational point of view, the global goal of delay testing is to guarantee that the propagation delays of all paths in a given circuit are less than the system clock interval. Except when using dynamic logic, it is well-known that testing a distinct delay fault requires a pair of two patterns rather than a single pattern as in the case of stuck-at testing. Fig. 1 illustrates the hardware model, which is frequently taken as the basis for delay testing [10,11,12,15,16]. At time T_0, the *initialization vector* V_1 is loaded into the input latches, which are assumed to be glitchless. After all signals of the circuit have been allowed to stabilize under V_1, the *propagation vector* V_2 is applied to the circuit by activating clock C_1. Finally,

the logic values at the primary outputs are sampled into the output latches at time $T_2 = T_1 + T_C$ by pulsing clock C_2, where T_C denotes the system clock interval at the desired functional clock rate.

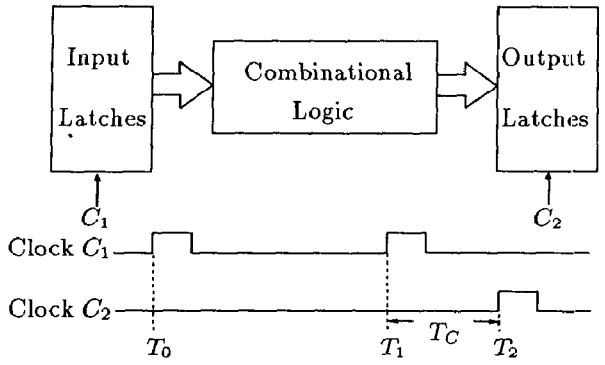

Fig. 1: Hardware Model and Clock Timings.

2.2 Circuit Structure and Path Graph

The Figs. 2 and 3 show how the structure of a combinational circuit C can be described by a directed graph $G_p = (V_p, E_p)$, which is called the *path graph* and which has already been used in [9] for the purpose of delay testing. However, while the approach discussed in [9] is restricted to circuits consisting of simple gates (AND-, NAND-, OR-, NOR-gates, buffers, inverters) only, we have extended it, in order to correctly model circuits containing XOR- and XNOR-gates in addition to simple gates. Our method for constructing G_p from the structure of a given combinational circuit C can be described as follows:

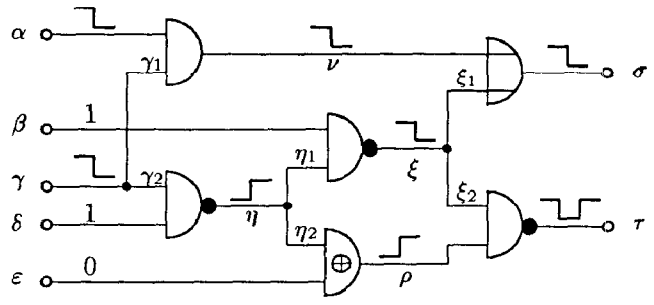

Fig. 2: Circuit C.

1) Identify all primary inputs (PIs) and all primary outputs (POs) of the combinational circuit as well as all fanout stems and all output signals of XOR- and XNOR-gates. For each of these signals, include two nodes, labeled rising and falling, respectively, into the node set V_p. Thus, for circuit C illustrated in Fig. 2, V_p is given by:

$$V_p = \{\alpha^r, \alpha^f, \beta^r, \beta^f, \gamma^r, \gamma^f, \delta^r, \delta^f, \epsilon^r, \epsilon^f, \eta^r, \eta^f, \xi^r, \xi^f, \rho^r, \rho^f, \sigma^r, \sigma^f, \tau^r, \tau^f\}. \quad (1)$$

2) Let (η^r, η^f) and (ξ^r, ξ^f) be the two node pairs that correspond to the signals η and ξ. Furthermore, let $S = (\eta, \kappa_1, ..., \kappa_n, \xi)$ be a *structural subpath* from signal η to

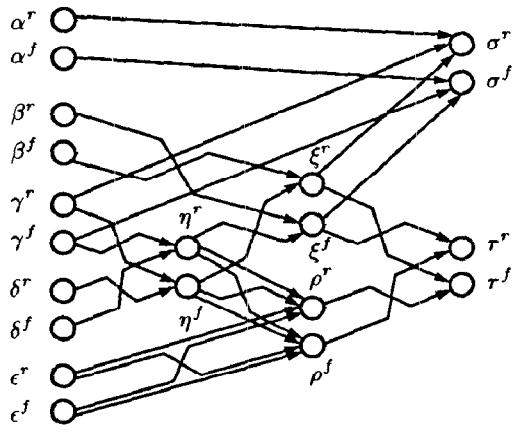

Fig. 3: Path Graph $G_p = (V_p, E_p)$.

signal ξ, which does not pass through a fanout stem or the output signal of a XOR- or a XNOR-gate, i.e.

$$\forall_{i=1,...,n} (\kappa_i^r \notin V_p \wedge \kappa_i^f \notin V_p). \quad (2)$$

If signal ξ does not represent the output signal of a XOR- or a XNOR-gate,

- include *two edges* (η^r, ξ^r) and (η^f, ξ^f) ((η^r, ξ^f) and (η^f, ξ^r)) into E_p, if the number of inverting gates on S is even (odd).

Otherwise, if signal ξ denotes the output signal of a XOR- or a XNOR-gate,

- include *four edges* (η^r, ξ^r), (η^r, ξ^f), (η^f, ξ^r), and (η^f, ξ^f) into E_p.

Note that, for each structural subpath from signal η to signal ξ that fulfills condition (2) stated above, we have to include the appropriate number of edges into E_p, in order to correctly model the structure of the circuit by the graph G_p.

2.3 Robust and Non-Robust Detection of Path Delay Faults

Considering Fig. 1 reveals that delay testing basically consists in testing the propagation delays of all paths through the combinational logic against the functional system clock interval T_C.

Definition 1: Let $P = (\kappa_1^{t_1}, ..., \kappa_n^{t_n})$ be a path in the path graph illustrated in Fig. 3, where κ_1 and κ_n correspond to a PI and a PO of the combinational logic, respectively, and $t_1, ..., t_n \in \{r, f\}$. Furthermore, let $APD(P)$ represent the *actual propagation delay* of transition t_1 along P. Then, P is said to have a path fault, if $APD(P)$ exceeds T_C, i.e.

$$APD(P) > T_C. \quad (3)$$

Definition 2: Let $\langle V_1, V_2 \rangle$ denote a two-pattern test applied to the combinational logic in the hardware environment shown in Fig. 1. At time T_1, when the circuit has stabilized under the initialization vector V_1, all signals are said to have their *initial values*. Conversely, the *final value* of a distinct signal is defined as the binary logic value (0 or 1) which that signal would assume under the propagation vector V_2 at time T_2, if the circuit would be free of delay faults.

Definition 3: A two-pattern test $\langle V_1, V_2 \rangle$ is called a *robust test* for a fault on path P, if it detects that fault *independently* of all *other delays* in the circuit and all *other delay faults* not located on the structural path corresponding to P.

Definition 4: A two-pattern test $\langle V_1, V_2 \rangle$ is called a *non-robust test* for a fault on path P, if it detects that fault under the assumption that the off-path sensitizing inputs [10,11] of *all* gates on the structural path corresponding to P stabilize at their *final values* prior to the time, at which the transition propagated along P arrives at them.

Theorem 1: A two-pattern test $\langle V_1, V_2 \rangle$ represents a robust test for a path fault on P, if, and only if,

- it provokes the transition t_1 at PI κ_1 and
- guarantees that all signals on the structural path corresponding to P cannot assume their final value according to V_2, unless the transition on the path under test has arrived at them.

Theorem 2: A two-pattern test $\langle V_1, V_2 \rangle$ represents a non-robust test for a fault on path P, if, and only if,

- it provokes the transition t_1 at PI κ_1 and
- V_2 causes all off-path sensitizing inputs along the structural path corresponding to P to assume those non-controlling values that allow the propagation of the transition t_1 from κ_1 to κ_n.

Proofs: Follow readily from Definitions 3 and 4.

Consider circuit C shown in Fig. 2. Obviously, the indicated two-pattern test consists of the initialization vector $V_1 = (\alpha, \beta, \gamma, \delta, \epsilon) = (1,1,1,1,0)$ and the propagation vector $V_2 = (0,1,0,1,0)$. According to Definition 3 and Theorem 1, this is a robust test for the fault on path $P_1 = (\gamma^f, \eta^r, \xi^f, \sigma^f)$, since all signals on the corresponding structural path $(\gamma, \gamma_2, \eta, \eta_1, \xi, \xi_1, \sigma)$ cannot assume their final values, unless the transition propagated along P_1 has arrived at them. Conversely, $\langle V_1, V_2 \rangle$ does not represent a robust test for the fault on path $P_2 = (\gamma^f, \eta^r, \xi^f, \tau^r)$, since an excessive delay in the rising transition on signal ρ may cause PO τ to have its expected final value 1 at time T_2, regardless of the delay along P_2. This would lead to declaring circuit C as fault-free, although there might be a path fault on P_2 in addition to the fault entailing the excessive delay in the transition on signal ρ. Actually, path P_2 can easily be shown to be not testable under the restrictive condition of robustness that does justice to a multiple fault assumption, which is frequently avoided even in the less complicated case of stuck-at testing. On the other hand, $\langle V_1, V_2 \rangle$ fulfills the conditions of a non-robust test for P_2, stated in Theorem 2. Assuming the transition on signal ρ to be "on time", the correct final value 1 can only be observed at PO τ, if there is no path fault on P_2. Finally, the interested reader may notice that the non-robust test $\langle V_1, V_2 \rangle$ for path P_2 can be validated to be robust by a robust test for path $P_3 = (\gamma^f, \eta^r, \rho^r, \tau^f)$, as e.g. $\langle (0,0,1,1,0), (0,0,0,1,0) \rangle$ [11]. However, even in the case that a non-robust test is not validatable by another robust one, it is our conviction that having at least information concerning the non-robust detection of a distinct path fault is clearly advantageous over lacking any information about that fault.

3 Robust and Non-Robust Path Delay Fault Simulation

3.1 Six-Valued and Four-Valued Simulation Logics

In order to perform simulation of robust detection of path delay faults, we basically make use of the six-valued logic proposed in [15], which comprises the values 0s, 0p, 0-, 1s, 1p, and 1-. Each of those six values can be viewed as an ordered pair (fv, pds). When assigned to a distinct signal κ of the circuit, the first component $fv(\kappa) \in \{0,1\}$ denotes the final value of signal κ under the two-pattern test $\langle V_1, V_2 \rangle$ according to Definition 2. The second component $pds(\kappa) \in \{s, p, \text{-}\}$, called the *path detectability status* at signal κ, is defined as follows:

Definition 5: A distinct signal κ of a combinational circuit C has $pds(\kappa) = s$, if, and only if, it is guaranteed to remain stable at $fv(\kappa)$ between T_1 and T_2, i.e. it is definitely free of any transition and any hazards between T_1 and T_2. Otherwise, $pds(\kappa) = p$, if, and only if, there is at least one path from a PI to signal κ whose corresponding path delay fault is robustly detectable at signal κ according to Definition 3 and Theorem 1. Finally, if $pds(\kappa) \neq s$ and $pds(\kappa) \neq p$, then $pds(\kappa) = \text{-}$.

Fig. 4 shows the propagation tables of the six-valued logic [15] for AND- and XOR-gates. Using an AND-gate with the two input signals κ_1 and κ_2 and the output signal η for the sake of illustration, note that $(\kappa_1, \kappa_2) = (0p, 0p), (\kappa_1, \kappa_2) = (0p, 1\text{-})$, and $(\kappa_1, \kappa_2) = (0p, 1p)$ result in $\eta = 0\text{-}$, indicating that the path faults, which are robustly detectable at κ_1, do not fulfill the condition for robust detection at η. The reason for this is that the falling transition, the possible hazard, or a delay in the rising transition on κ_2 may cause the gate output η to have its final value 0, although the transition on κ_1 has not yet arrived. This contradicts Theorem 1 and, as a consequence, $\langle V_1, V_2 \rangle$ cannot be a robust test for any path through signal η.

$\wedge$	0s	0p	0-	1s	1p	1-
0s	0s	0s	0s	0s	0s	0s
0p	0s	0-	0-	0p	0-	0-
0-	0s	0-	0-	0-	0-	0-
1s	0s	0p	0-	1s	1p	1-
1p	0s	0-	0-	1p	1p	1p
1-	0s	0-	0-	1-	1p	1-

$\oplus$	0s	0p	0-	1s	1p	1-
0s	0s	0p	0-	1s	1p	1-
0p	0p	0-	0-	1p	1-	1-
0-	0-	0-	0-	1-	1-	1-
1s	1s	1p	1-	0s	0p	0-
1p	1p	1-	1-	0p	0-	0-
1-	1-	1-	1-	0-	0-	0-

Fig. 4: Six-valued propagation tables for AND- and XOR-gates for robust path delay fault simulation.

Next, consider the case of non-robust path delay fault simulation. Clearly, all combinations of input values of the mentioned AND-gate, that lead to $pds(\eta) = p$ in robust delay fault simulation, will result in $pds(\eta) = p$ in non-robust delay fault simulation as well. In addition, since only the final values of the off-path sensitizing gate inputs are decisive for the non-robust detection of path delay faults (Theorem 2), the two input combinations $(\kappa_1, \kappa_2) = (0p, 1\text{-})$, and $(\kappa_1, \kappa_2) = (0p, 1p)$ do also allow the non-robust detection of faults on paths through κ_1 at the gate output η, what has to be taken into account by $\eta = 0p$. Thus, the propagation table for the AND-gate has to

be modified as illustrated in Fig. 5a, in order to perform non-robust path delay simulation. Now, if we recall that neither $pds(\eta) = s$ nor $pds(\eta) =$ - can be used for the detection of a path fault, the consideration of the six-valued propagation table in Fig. 5a reveals that the two columns corresponding to $\kappa_1 = 0s$ and $\kappa_1 = 0$- as well as the two columns corresponding to $\kappa_1 = 1s$ and $\kappa_1 = 1$- are entirely identical with regard to the resulting detectability of path faults at the output signal η. Since this holds true for OR- and XOR-gates as well, we combine the two values $0s$ and 0- ($1s$ and 1-) to a new one, denoted by $0\overline{p}(1\overline{p})$. As a consequence, we end up with a four-valued logic for non-robust path delay fault simulation, whose corresponding propagation table for AND-gates is shown in Fig. 5b.

∧	0s	0p	0-	1s	1p	1-
0s	0s	0s	0s	0s	0s	0s
0p	0s	0-	0-	0p	0p	0p
0-	0s	0-	0-	0-	0-	0-
1s	0s	0p	0-	1s	1p	1-
1p	0s	0p	0-	1p	1p	1p
1-	0s	0p	0-	1-	1p	1-

a) Six-Valued.

∧	0p	$0\overline{p}$	1p	$1\overline{p}$
0p	0p	$0\overline{p}$	0p	0p
$0\overline{p}$	$0\overline{p}$	$0\overline{p}$	$0\overline{p}$	$0\overline{p}$
1p	0p	$0\overline{p}$	1p	1p
$1\overline{p}$	0p	$0\overline{p}$	1p	$1\overline{p}$

b) Four-Valued.

Fig. 5: Modified and reduced propagation tables for non-robust path delay fault simulation (AND-gate).

3.2 Encoding of Logic Values

Since the final value of any signal κ, $fv(\kappa)$, is either 0 or 1 in both the six-valued as well as the four-valued simulation logic, its encoding is trivial and can be accomplished by a single bit. Conversely, the path detectability status $pds(\kappa)$ is three-valued in the six-valued simulation logic and, consequently, has to be encoded by an ordered pair of bits

$$pds(\kappa) = (s(\kappa), p(\kappa)), \tag{4}$$

where $s(\kappa)$ and $p(\kappa)$ are called the s-bit (stable bit) and the p-bit (path bit) of the logic value of signal κ, respectively. In the case of the four-valued simulation logic used for non-robust path fault simulation, however, there are only two possible values (p and $\overline{p}$) for $pds(\kappa)$. Thus, the p-bit is sufficient for encoding $pds(\kappa)$.

In order to transform a given signal value into its corresponding three-bit or two-bit representation, we need an appropriate coding technique. The Tables 1a and 1b illustrate the code we have chosen for representing the logic values of the six-valued and the four-valued simulation logic, respectively.

Logic Value of Signal κ	$fv(\kappa)$	$s(\kappa)$	$p(\kappa)$
0s/1s	0/1	1	0
0p/1p	0/1	0	1
0-/1-	0/1	0	0

a) Six-Valued Logic.

Logic Value of Signal κ	$fv(\kappa)$	$p(\kappa)$
0p/1p	0/1	1
$0\overline{p}/1\overline{p}$	0/1	0

b) Four-Valued Logic.

Table 1: Encoding of logic values.

Next, in order to support parallel processing of pattern pairs during fault simulation, we use three L-bit-wide machine words in the case of robust fault simulation and two L-bit-wide machine words in the case of non-robust fault simulation for representing a set of L logic values of signal κ, which usually correspond to a set of L pattern pairs $\langle V_1, V_2\rangle_1, ..., \langle V_1, V_2\rangle_L$. Thereby, all bits reflecting the final value of signal κ are stored in the first word, called the fv-word $vec[fv(\kappa)]$.

$$vec[fv(\kappa)] = (fv(\kappa)_1, \ldots, fv(\kappa)_L) \tag{5a}$$

Similarly,

$$vec[p(\kappa)] = (p(\kappa)_1, \ldots, p(\kappa)_L) \tag{5b}$$

denotes the p-word. Finally, in the case of robust simulation, we additionally need

$$vec[s(\kappa)] = (s(\kappa)_1, \ldots, s(\kappa)_L), \tag{5c}$$

which will be referred to as the s-word in the following. For example, choosing $L = 5$ and making use of the six-valued logic, the set $\{1p, 0s, 0\text{-}, 1s, 0p\}$ of logic values at signal κ is represented by $vec[fv(\kappa)] = (1, 0, 0, 1, 0)$, $vec[p(\kappa)] = (1, 0, 0, 0, 1)$, and $vec[s(\kappa)] = (0, 1, 0, 1, 0)$.

3.3 Parallel Pattern Path Delay Fault Simulation

First of all, the PIs of the combinational logic have to be initialized to their correct values according to the pattern pairs $\langle V_1, V_2\rangle_1, \ldots, \langle V_1, V_2\rangle_L$. Provided that the input latches (Fig. 1) are glitchless, in both robust as well as non-robust path delay fault simulation, the value $0p(1p)$ is assigned to PI α for the pattern pair i, if $\langle V_1, V_2\rangle_i$ causes PI α to assume the initial value 1(0) and the final value 0(1). Otherwise, if there is no difference between the initial value and the final value of PI α, i.e. if PI α remains stable at 0 or 1 between T_1 and T_2, PI α is initialized to $0s$ or $1s$ in the robust case and to $0\overline{p}$ or $1\overline{p}$ in the non-robust case.

Subsequently, those logic values have to be propagated towards the POs of the combinational logic. Obviously, this can be accomplished by performing the gate evaluations in levelized order accordingly to the corresponding propagation tables, as illustrated representatively for AND- and XOR-gates in Figs. 4 and 5. Considering an AND-gate with the input signals κ_1 and κ_2 and the output signal η, we have

$$fv(\eta) = fv(\kappa_1) \cdot fv(\kappa_2), \tag{6a}$$

$$s(\eta) = \overline{fv(\kappa_1)} \cdot s(\kappa_1) + \overline{fv(\kappa_2)} \cdot s(\kappa_2) + fv(\kappa_1) \cdot s(\kappa_1) \cdot fv(\kappa_2) \cdot s(\kappa_2), \quad \text{and} \tag{6b}$$

$$p(\eta) = fv(\kappa_1) \cdot fv(\kappa_2) \cdot \Big[p(\kappa_1) + p(\kappa_2)\Big] + \overline{fv(\kappa_1)} \cdot p(\kappa_1) \cdot fv(\kappa_2) \cdot s(\kappa_2) + \overline{fv(\kappa_2)} \cdot p(\kappa_2) \cdot fv(\kappa_1) \cdot s(\kappa_1) \tag{6c}$$

in the case of robust path delay fault simulation and

$$fv(\eta) = fv(\kappa_1) \cdot fv(\kappa_2) \quad \text{and} \tag{7a}$$

$$p(\eta) = fv(\kappa_1) \cdot p(\kappa_2) + fv(\kappa_2) \cdot p(\kappa_1) \tag{7b}$$

in the case of non-robust path delay fault simulation. Similar formulas can easily be derived for all other gate types. Applying the simulation procedure described so far to our circuit C

Gate Type	Binary Operations	
	Robust Path Delay FS	Non-Robust Path Delay FS
AND, NAND	$vec[s(\eta)] = vec[\overline{fv(\kappa_1)}] \cdot vec[s(\kappa_1)] + vec[\overline{fv(\kappa_2)}] \cdot vec[s(\kappa_2)] + vec[fv(\kappa_1)] \cdot vec[s(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot vec[s(\kappa_2)]$ $vec[p(\eta)] = vec[fv(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot \Big(vec[p(\kappa_1)] + vec[p(\kappa_2)]\Big) + vec[\overline{fv(\kappa_1)}] \cdot vec[p(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot vec[s(\kappa_2)] + vec[\overline{fv(\kappa_2)}] \cdot vec[p(\kappa_2)] \cdot vec[fv(\kappa_1)] \cdot vec[s(\kappa_1)]$	$vec[p(\eta)] = vec[fv(\kappa_1)] \cdot vec[p(\kappa_2)] + vec[fv(\kappa_2)] \cdot vec[p(\kappa_1)]$
OR, NOR	$vec[s(\eta)] = vec[fv(\kappa_1)] \cdot vec[s(\kappa_1)] + vec[fv(\kappa_2)] \cdot vec[s(\kappa_2)] + vec[\overline{fv(\kappa_1)}] \cdot vec[s(\kappa_1)] \cdot vec[\overline{fv(\kappa_2)}] \cdot vec[s(\kappa_2)]$ $vec[p(\eta)] = vec[\overline{fv(\kappa_1)}] \cdot vec[\overline{fv(\kappa_2)}] \cdot \Big(vec[p(\kappa_1)] + vec[p(\kappa_2)]\Big) + vec[fv(\kappa_1)] \cdot vec[p(\kappa_1)] \cdot vec[\overline{fv(\kappa_2)}] \cdot vec[s(\kappa_2)] + vec[fv(\kappa_2)] \cdot vec[p(\kappa_2)] \cdot vec[\overline{fv(\kappa_1)}] \cdot vec[s(\kappa_1)]$	$vec[p(\eta)] = vec[\overline{fv(\kappa_1)}] \cdot vec[p(\kappa_2)] + vec[\overline{fv(\kappa_2)}] \cdot vec[p(\kappa_1)]$
BUF, INV	$vec[s(\eta)] = vec[s(\kappa)]$ $vec[p(\eta)] = vec[p(\kappa)]$	$vec[p(\eta)] = vec[p(\kappa)]$
XOR, XNOR,	$vec[s(\eta)] = vec[s(\kappa_1)] \cdot vec[s(\kappa_2)]$ $vec[p(\eta)] = vec[p(\kappa_1)] \cdot vec[s(\kappa_2)] + vec[p(\kappa_2)] \cdot vec[s(\kappa_1)]$	$vec[p(\eta)] = vec[p(\kappa_1)] + vec[p(\kappa_2)]$

Table 2: Binary operations for robust and non-robust path delay fault simulation.

with the indicated two-pattern test (Fig. 2), we obtain

$$(\alpha, \beta, \gamma, \delta, \epsilon, \nu, \eta, \xi, \rho, \sigma, \tau) = (0p, 1s, 0p, 1s, 0s, 0\text{-}, 1p, 0p, 1p, 0p, 1\text{-}) \quad \text{and} \tag{8a}$$

$$(\alpha, \beta, \gamma, \delta, \epsilon, \nu, \eta, \xi, \rho, \sigma, \tau) = (0p, 1\overline{p}, 0p, 1\overline{p}, 0\overline{p}, 0\overline{p}, 1p, 0p, 1p, 0p, 1p) \tag{8b}$$

for robust and non-robust path delay fault simulation, respectively.

By extending (6) to

$$vec[fv(\eta)] = vec[fv(\kappa_1)] \cdot vec[fv(\kappa_2)], \tag{9a}$$

$$vec[s(\eta)] = vec[\overline{fv(\kappa_1)}] \cdot vec[s(\kappa_1)] + vec[\overline{fv(\kappa_2)}] \cdot vec[s(\kappa_2)] + vec[fv(\kappa_1)] \cdot vec[s(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot vec[s(\kappa_2)], \tag{9b}$$

$$vec[p(\eta)] = vec[fv(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot \Big(vec[p(\kappa_1)] + vec[p(\kappa_2)]\Big) + vec[\overline{fv(\kappa_1)}] \cdot vec[p(\kappa_1)] \cdot vec[fv(\kappa_2)] \cdot vec[s(\kappa_2)] + vec[\overline{fv(\kappa_2)}] \cdot vec[p(\kappa_2)] \cdot vec[fv(\kappa_1)] \cdot vec[s(\kappa_1)], \tag{9c}$$

and (7) to

$$vec[fv(\eta)] = vec[fv(\kappa_1)] \cdot vec[fv(\kappa_2)], \tag{10a}$$

$$vec[p(\eta)] = vec[fv(\kappa_1)] \cdot vec[p(\kappa_2)] + vec[fv(\kappa_2)] \cdot vec[p(\kappa_1)], \tag{10b}$$

we readily gain the capability of evaluating the AND-gate mentioned above in parallel for L pattern pairs, where $vec[fv(\eta)]$, $vec[s(\eta)]$, and $vec[p(\eta)]$ denote the fv-word, the s-word, and the p-word of signal η, respectively. Table 2 summarizes the binary operations, which are necessary for determining the path detectability status by parallel processing of pattern pairs in robust and non-robust path delay fault simulation for all gate types considered. Note that, by applying the law of associativity, these operations can easily be extended to gates with more than two inputs. The corresponding formulas for the parallel evaluation of the final value $vec[fv(\eta)]$ are identical with those used in parallel pattern stuck-at fault simulation [17]. Finally, it is worth noting that we have chosen the encoding of the logic values (Tables 1a and 1b) in such way that the number of binary operations, required for performing parallel pattern path delay fault simulation, is minimized.

3.4 Path Delay Fault Detection

After all gates of the combinational circuit have been evaluated by employing the binary operations listed in Table 2, all path delay faults, that are either robustly or non-robustly detectable by the L pattern pairs simulated, have to be identified. Just for the sake of explanation, let us first consider a single pattern pair $\langle V_1, V_2 \rangle$. In the case of robust path delay fault simulation, the identification of the path faults detected by $\langle V_1, V_2 \rangle$ is trivial. In fact, it can simply be accomplished by tracing all those structural paths, on which all signals have either the value 0p or 1p, in a depth-first manner from the POs to the PIs [15]. Considering Fig. 2 and Eq. (8a), the trace would proceed along $\sigma, \xi_1, \xi, \eta_1, \eta, \gamma_2$, and γ, representing the only path with a robustly detected path fault. Consequently, since $\gamma = 0p$, we would mark the path $P_1 = (\gamma^f, \eta^r, \xi^f, \sigma^f)$ as robustly tested.

Contrary to robust path fault simulation, the identification of detected path faults is a little bit more involved in non-robust path fault simulation. Assuming that the path trace has arrived at the output signal η of a distinct gate g with its inputs $\kappa_1, \ldots, \kappa_n$, we *cannot* simply extend our search for detected path faults over *all* inputs of g with the logic value 0p or 1p. In this case, we have to determine those gate inputs

κ_i, which have $pds(\kappa_i) = p$ *and* whose final value $fv(\kappa_i)$ is observable at the gate output η, i.e. for which the *local path sensitivity*

$$lps(\eta_{\kappa_i}) = p(\kappa_i) \cdot \eta_{\kappa_i}^{fv} = 1, \tag{11}$$

where

$$\eta_{\kappa_i}^{fv} = fv(\eta(\kappa_i)) \oplus fv(\eta(\overline{\kappa}_i)) \tag{12}$$

denotes the Boolean Difference η_{κ_i} with regard to the final values of the signals η and κ_i. Inspecting our circuit C (Fig.2) and Eq. (8b), for the NAND-gate with the output signal τ, we obtain $p(\xi_2) = 1$, $\tau_{\xi_2}^{fv} = 1$, $p(\rho) = 1$, and $\tau_{\rho}^{fv} = 0$. Thus, in non-robust path fault simulation, we additionally would have to trace along $\tau, \xi_2, \xi, \eta_1, \eta, \gamma_2$, and γ and, consequently, mark the fault on path $P_2 = (\gamma^f, \eta^r, \xi^f, \tau^r)$ as non-robustly detected. Note that, as a consequence from $\tau_{\rho}^{fv} = 0$, the trace must not proceed along $\tau, \rho, \eta_2, \eta, \gamma_2$, and γ, since the corresponding path fault on $P_3 = (\gamma^f, \eta^r, \rho^r, \tau^f)$ is neither robustly nor non-robustly detectable by the pattern pair indicated in Fig. 2.

Next, in order to support parallel processing of patterns also during the path trace, we introduce $vec[po(\eta)]$ as the *path observability mask* of signal η. Initializing the path trace at a distinct PO σ by

$$vec[po(\sigma)] = vec[p(\sigma)], \tag{13}$$

we recursively proceed in the mentioned depth-first manner from a gate output η to a gate input κ by evaluating

$$vec[po(\kappa)] = vec[po(\eta)] \cdot vec[lps(\eta_\kappa)], \tag{14}$$

where $vec[lps(\eta_\kappa)]$ is determined by parallel processing of patterns as expressed by

$$vec[lps(\eta_\kappa)] = vec[p(\kappa)] \cdot vec[\eta_\kappa^{fv}]. \tag{15}$$

Thus, when the path trace arrives at a PI, say α, $vec[po(\alpha)]$ indicates all those of the L pattern pairs that can be used for detecting path faults on the structural path traced along from PO σ to PI α.

4 The Path Tree

In order to effectively cope with the typically huge number of paths in today's VLSI circuits, we have developed a highly economical data structure, called the path tree. Its basic idea consists in storing parts of paths, that are common to many paths from a distinct PI to the POs, only once rather than explicitly carrying them along for each path separately. Moreover, the structural subpath $S = (\eta, \kappa_1, \ldots, \kappa_n, \xi)$ with $\eta \in V_p$ and $\xi \in V_p$ is referred to only by that signal or that fanout branch, which uniquely identifies S. Consider the paths $P_1 = (\gamma^f, \eta^r, \xi^f, \sigma^f)$, $P_2 = (\gamma^f, \eta^r, \xi^f, \tau^r)$, $P_3 = (\gamma^f, \eta^r, \rho^r, \tau^f)$ $P_4 = (\gamma^r, \eta^f, \xi^r, \sigma^r)$, $P_5 = (\gamma^r, \eta^f, \xi^r, \tau^f)$, and $P_6 = (\gamma^r, \eta^f, \rho^r, \tau^f)$. Recalling that, for example, the structural subpath $S = (\gamma, \gamma_2, \eta)$ is uniquely identified by the fanout branch γ_2, the two paths P_1 and P_4 (P_2 and P_5) can be represented by the same three path tree nodes γ_2, η_1, and ξ_1 (γ_2, η_1, and ξ_2). Since P_1 and P_4 (P_2 and P_5) correspond to the same structural path and differ in the transition propagated along them only, this can be taken into account by storing the respective direction of the transition at the PI, from which the paths emanate, in the common leaf node $\xi_1(\xi_2)$. In Fig. 6, the entry 'f' in node $\xi_1(\xi_2)$ refers to path $P_1(P_2)$, while the entry 'r' denotes path $P_4(P_5)$.

Contrary to P_1, P_2, P_4, and P_5, the paths P_3 and P_6 pass through a XOR-gate. As is well-known, in the case of XOR- or XNOR-gates, the direction of the transition on the gate output signal is dependent of both the direction of the transition on the gate input signal located on the path under consideration as well as the logic value of the off-path-sensitizing gate input. Thus, in addition to P_3 and P_6, two functional paths $P_7 = (\gamma^f, \eta^r, \rho^f, \tau^r)$ and $P_8 = (\gamma^r, \eta^f, \rho^f, \tau^r)$ exist, which correspond to the same structural path as P_3 and P_6 do. In order to distinguish P_3 and P_6, which both produce a rising transition at the output signal ρ of the XOR-gate, from P_7 and P_8, we have to include the node ρ^r in the path tree (Fig. 6). Again, storing the falling and the rising transition at PI γ in the leaf node ρ^r allows us to distinguish between P_3 and P_6 and to uniquely refer to either of those two paths during fault simulation or ATG [13,14].

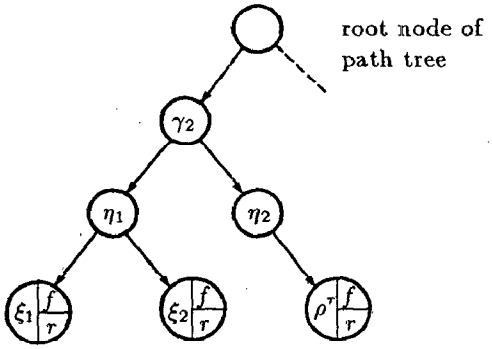

Fig. 6: Path Tree for representing P_1, P_2, P_3, P_4, P_5, and P_6.

In this context, it should explicitly be mentioned that a distinct structural path S in general corresponds to 2^{k+1} functional paths in the path graph, where k denotes the sum of the numbers of XOR- and XNOR-gates located on S. Thus, a given set of L pattern pairs may result in the robust or non-robust detection of delay faults on several of those functional paths. Denoting the number of delay faults on S that are detectable by a set of L pattern pairs by NDF_S^L, we have

$$NDF_S^L \leq min(2^{k+1}, L). \tag{16}$$

Naturally, in the special case where S does not pass through any XOR- or XNOR-gate, i.e. $k = 0$, S corresponds to exactly two functional paths and, consequently,

$$NDF_S^L \leq min(2, L). \tag{17}$$

5 Experimental Benchmark Results

The proposed fault simulation method for robust and non-robust detection of path delay faults has been implemented in the programming language C on a Micro-VAX, which has a machine word length L of 32 bits. In order to demonstrate the efficiency of our fault simulation approach, we have performed robust and non-robust path delay fault simulations of 10000 random pattern pairs for the ten well-known ISCAS benchmarks [18]. Thereby, we did not impose any restriction on the number of path faults to be considered, but have simulated

the path delay faults on *all* functional paths, ranging in number between 17284 for circuit c880 and $1.98 * 10^{20}$ for circuit c6288.

The achieved results are summarized in Table 3, where PPFS and DPFP denote abbreviations for parallel pattern fault simulation and detected path fault processing, respectively. DPFP comprises the path trace, performed in parallel for 32 pattern pairs, and the maintenance of the path tree which, in this experiment, is used for recording all robustly and non-robustly detected path delay faults. As Table 3 substantiates, our fault simulation approach is capable of both simulating long pattern sequences within reasonable amounts of CPU-time as well as maintaining large numbers of path faults considered. For example, in the case of circuit c3540, the corresponding path tree for storing the 356681 detected path delay faults requires a total memory resource of 16.2 megabytes, implying that on the average only 45.5 bytes are necessary for the representation of one path through the entire circuit. Moreover, as we would have expected from [16], only a few faults are detected under the restrictive condition of robustness. On the other hand, the simulated 10000 random pattern pairs succeeded in detecting considerable numbers of path delay faults non-robustly in most circuits. Since many of those faults can be proved to be robustly undetectable (robustly redundant) with the aid of a complete deterministic ATG approach, as e.g. [13,14], our fault simulator's capability of accounting for non-robust detection of path delay faults may become extremely important, especially in the case of designs based upon statistical timing.

Circuit Name	Det. Path Faults		CPU-Times [sec.]			Memory Requirements [kB]
	Robust	Non-Rob.	PPFS	DPFP	Total	
c432	476	9151	49.3	242.1	291.4	237.1
c499	39	120216	53.9	570.6	624.5	2879.4
c880	1020	5720	111.6	302.1	413.7	106.2
c1355	23	326551	165.2	1511.3	1676.5	10726.1
c1908	695	38310	235.5	650.7	886.2	1275.6
c2670	1486	50071	329.6	1871.1	2200.7	2080.1
c3540	1546	355135	456.9	5440.9	5897.8	16241.5
c5315	4928	144697	674.3	4238.7	4913.0	3996.9
c6288	53	> 1000000	509.3	n.a.	n.a.	> 40000
c7552	3257	154968	971.6	6772.0	7743.6	4070.6

Table 3: Results of robust and non-robust path delay fault simulation of 10000 random pattern pairs.

6 Conclusions

In this paper we have described an accelerated fault simulation method for path delay faults. Based upon a six-valued logic and a four-valued logic, our fault simulation approach has been shown to be capable of simulating both robust as well as non-robust detection of path delay faults. Thereby, parallel processing of patterns is applied at all stages of the fault simulation procedure. Moreover, we have introduced a specifically developed data structure, called the path tree, which has been designed to be highly economical in terms of memory requirements and which allows us to efficiently maintain hundreds of thousands of path faults to be simulated.

Acknowledgement

The authors are very grateful to Prof. K. J. Antreich of Technical University of Munich for his valuable suggestions and his helpful advice.

References

[1] Z. Barzilai and B. K. Rosen. "Comparison of AC Self - Testing Procedures". *IEEE Int. Test Conf.*, 89 – 91, Oct. 1983.

[2] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar. "Transition Fault Simulation". *IEEE Design and Test*, 32 – 38, April 1987.

[3] M. H. Schulz and F. Brglez. "Accelerated Transition Fault Simulation". *Proc. 24th DAC*, 237 – 243, June 1987.

[4] S. Koeppe. "Modeling and Simulation of Delay Faults in CMOS Logic Circuits". *IEEE Int. Test Conf.*, 530 – 536, Sept. 1986.

[5] J. L. Carter, V. S. Iyengar, and B. K. Rosen. "Efficient Test Coverage Determination for Delay Faults". *IEEE Int. Test Conf.*, 418 – 427, Sept. 1987.

[6] V. S. Iyengar, B. K. Rosen, and I. Spillinger. "Delay Test Generation 1 – Concepts and Coverage Metrics". *IEEE Int. Test Conf.*, 857 – 866, Sept. 1988.

[7] V. S. Iyengar, B. K. Rosen, and I. Spillinger. "Delay Test Generation 2 – Algebra and Algorithms". *IEEE Int. Test Conf.*, 867 – 876, Sept. 1988.

[8] A. K. Pramanick and S. M. Reddy. "On the Detection of Delay Faults". *IEEE Int. Test Conf.*, 845 – 856, Sept. 1988.

[9] J. J. Shedletsky and J. D. Lesser. "An Experimental Delay Test Generator for LSI Logic". *IEEE Trans. On Comp.*, Vol. C-29, No. 3, 235 – 248, March 1980.

[10] C. J. Lin and S. M. Reddy. "On Delay Fault Testing in Logic Circuits". *IEEE Trans. On CAD*, Vol. 6, No. 5, 694 – 703, Sept. 1987.

[11] S. M. Reddy, C. J. Lin, and S. Patil. "An Automatic Test Pattern Generator for the Detection of Path Delay Faults". *IEEE Int. Conf. On CAD*, 284 – 287, Nov. 1987.

[12] E. S. Park and M. R. Mercer. "Robust and Nonrobust Tests for Path Delay Faults in a Combinational Circuit". *IEEE Int. Test Conf.*, 1027 – 1034, Sept. 1987.

[13] M. H. Schulz, K. Fuchs, and F. Fink. "Advanced Automatic Test Pattern Generation Techniques for Path Delay Faults". *19th FTCS*, June 1989.

[14] M. H. Schulz, K. Fuchs, and F. Fink. "An Improved and Versatile Automatic Test Pattern Generation Algorithm for Path Delay Faults". *To be published.*

[15] G. L. Smith. "Model for Delay Faults Based Upon Paths". *IEEE Int. Test Conf.*, 342 – 349, Sept. 1985.

[16] J. Savir and W. H. McAnney. "Random Pattern Testability of Delay Faults". *IEEE Trans. On Comp.*, Vol. 37, No. 3, 291 – 300, March 1988.

[17] K. J. Antreich and M. H. Schulz. "Accelerated Fault Simulation and Fault Grading in Combinational Circuits". *IEEE Trans. On CAD*, Vol. CAD-6, No. 5, 704 – 712, Sept. 1987.

[18] F. Brglez and H. Fujiwara. "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran". *Proc. IEEE Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation*, 663 – 698, June 1985.