Fast Identification of Untestable Delay Faults Using Implications*

Keerthi Heragu
Janak H. Patel
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign, Urbana, IL 61801
heragu@crhc.uiuc.edu, patel@crhc.uiuc.edu

Vishwani D. Agrawal
Bell Labs, Lucent Technologies
700 Mountain Avenue
Murray Hill, NJ 07974
va@research.bell-labs.com

Abstract

We propose a novel algorithm to rapidly identify untestable delay faults using pre-computed static logic implications. Our fault-independent analysis identifies large sets of untestable faults, if any, without enumerating them. The cardinalities of these sets are obtained by using a counting algorithm that has quadratic complexity in the number of lines. Since our method is based on an incomplete set of logic implications, it gives only a lower bound on the number of untestable faults. A post-processing step can list the untestable faults, if desired. Targeting untestable delay faults for test generation by an automatic test pattern generation (ATPG) tool can be avoided. The method works for the segment delay fault model and its special case, the path delay fault model, and identifies robustly untestable, non-robustly untestable, and functionally unsensitizable delay faults. Results on benchmark circuits show that many delay faults are identified as untestable in a very short time. For the benchmark circuit c6288, our algorithm identified 1.978×10^{20} *functionally unsensitizable path faults in* 3 *CPU seconds.*

1 Introduction

Physical defects that cause timing violations are modeled by delay faults at the logic level. *Path delay faults* [1, 2] model defects that cause cumulative propagation delays along paths to exceed specified limits. Segment delay faults [3, 4] model defects on segments, whose length L can be as small as 1 or as large as the maximum logic depth.

It is known that ATPG algorithms spend most of their time in generating tests for hard-to-detect faults or in trying to prove that a fault is untestable. This makes it very attractive to explore techniques that identify untestable faults quickly so that they need not be targeted by an ATPG tool. An estimate of the number of untestable faults can be used during the design phase to obtain a quick and approximate idea of the testability of a circuit. A lot of research has been done in the area of identifying untestable stuck-at-faults. While most of the algorithms are ATPG-based [5, 6], some use fault-independent techniques [7, 8], based on logic implications [9, 10], to identify untestable faults. Previous algorithms for identifying untestable path delay faults are ATPG-based and they involve identifying unsensitizable segments [11, 12]. Explicit enumeration of all path faults that pass through arbitrary unsensitizable segments is prohibitively complex for circuits with a large number of delay faults [11]. The runtime becomes lower when specific segments are used [12].

We propose a novel algorithm to identify untestable delay faults using pre-computed static logic implications. Using logic implications of both value assignments (0 and 1), at each line g, we perform a fault-independent analysis to identify a set of lines S, such that every fault passing through g and $\{m : m \in S\}$ is untestable with respect to some combination of signal values. Information about stuck-fault redundancies and constant logic values, generated from static logic implications, can also be used in our analysis. We propose a non-enumerative counting technique, based on previously published algorithms [13, 3], that uses information gathered from our implication analysis and computes a lower bound on the number of untestable delay faults. The counting procedure has a quadratic complexity in the number of lines.

Our algorithm is applicable to the segment delay fault model and its special case, the path delay fault model. In this paper, we only illustrate our algorithm for identifying untestable path delay faults. Results are presented for both models and they include robustly untestable [1, 2], non-robustly untestable [14] and functionally unsensitizable [11] delay faults.

2 Notation and Definitions

The following discussion is limited to combinational circuits composed of simple gates (AND, OR, NOT,

*This research was supported in part by Semiconductor Research Corporation under contract SRC 96-DP-109, in part by ARPA under contract DABT63-96-C-0069, and in part by Hewlett-Packard under an equipment grant.

NAND, and NOR). Let *line* denote an interconnection between two gates, $gt(l)$ denote the gate for which line l is an input, and $in(g)$ denote the set of inputs of gate g. Let $n(V)$ represent the stable signal value of n under a vector V, where n is either a gate or a line.

A physical path P is a sequence $(g_0, l_0, ..., l_{n-1}, g_n)$, where $(g_1, ..., g_{n-1})$ are gates, $(l_0, ..., l_{n-1})$ are lines, g_0 is a primary input (PI), and g_n is a primary output (PO). Lines $(l_0, ..., l_{n-1})$ are called *on-path inputs* of P. An input $\{i : i \in in(g_k), i \neq l_{k-1}, 1 < k < n\}$ is called a *side-input* of P. There are two *logical paths*, P^1 and P^0, associated with P, corresponding to a rising and a falling transition respectively on g_0. A faulty logical path is called a *path delay fault*. In this paper, we use the terms *path delay faults*, *path faults* and *logical paths* interchangeably. A test for a path delay fault consists of a vector pair $<V_1, V_2>$.

Definition 2.1 ([11]) *A vector pair $<V_1, V_2>$ is said to functionally sensitize a path fault $P^\psi = (g_0, l_0, ..., g_n)$, where $\psi \in \{0, 1\}$, iff:*
(1) $g_0(V_1) = \overline{\psi}$, $g_0(V_2) = \psi$, and
(2) if an on-path input of P^ψ has a non-controlling value under V_2, the corresponding side-inputs have non-controlling values under V_2.

Definition 2.2 ([14]) *A vector pair $<V_1, V_2>$ is said to be a non-robust test for a path delay fault $P^\psi = (g_0, l_0, ..., g_n)$, where $\psi \in \{0, 1\}$, iff:*
(1) $g_0(V_1) = \overline{\psi}$, $g_0(V_2) = \psi$, and
(2) all side-inputs of P^ψ have non-controlling values under V_2.

Definition 2.3 ([1, 2]) *A vector pair $< V_1, V_2 >$ is said to be a robust test for a path delay fault $P^\psi = (g_0, l_0, ..., g_n)$, where $\psi \in \{0, 1\}$, if it guarantees detection of the fault irrespective of the delays of all other signals in the circuit. This condition is satisfied iff:*
(1) $<V_1, V_2>$ is a non-robust test for P^ψ, and
(2) if an on-path input of P^ψ has a controlling value under V_2, the corresponding side-inputs have steady non-controlling values on both vectors.

Remark 2.1 *The existence of a vector V_1, that satisfies the requirement on it given by Definitions 2.1 and 2.2, can be guaranteed for a circuit without any restrictions on its input values.*

Remark 2.2 *The set of functionally sensitizable path faults is a superset of the set of non-robustly testable path faults, which in turn is a superset of the set of robustly testable path faults.*

Remark 2.3 *Only the robustness criterion imposes the requirement $\forall_{1<k<n}\{g_k(V_1) \neq g_k(V_2)\}$ which in turn implies $\forall_{1<k<n-1}\{l_k(V_1) \neq l_k(V_2)\}$*

We say that a path fault P^ψ is *sensitized* on a vector pair $<V_1, V_2>$ iff P^ψ is either functionally sensitized, robustly tested, or non-robustly tested by $<V_1, V_2>$.

Lemma 2.1 *If a path fault $P^\psi = (g_0, l_0, ..., g_n)$, where $\psi \in \{0, 1\}$, is sensitized on a vector pair $< V_1, V_2 >$, $\forall_{0<k<n}\{g_k(V_2) = \psi$ if the number of inversions between g_0 and the output of g_k is even; $g_k(V_2) = \overline{\psi}$ otherwise$\}$.*

Proof: We will prove the lemma by induction.
Induction Basis: Since P^ψ is sensitized by $<V_1, V_2>$, from the above definitions, we know that $g_0(V_2) = \psi$.
Induction Hypothesis: Suppose the claim is true for $\{g_m : 0 < m < n\}$.
Induction Step: Let $g_m(V_2) = \xi$ and g_{m+1} be non-inverting. Since g_{m+1} is non-inverting, the number of inversions between g_0 and the output of g_m is equal to the number of inversions between g_0 and the output of g_{m+1}. If ξ is the controlling value of g_{m+1}, $g_{m+1}(V_2) = \xi$. If ξ is the non-controlling value of g_{m+1}, by the above definitions, the side-inputs corresponding to g_{m+1} *have to be non-controlling* for P^ψ to be sensitized and hence, $g_{m+1}(V_2) = \xi$. Since $g_m(V_2) = g_{m+1}(V_2)$ and the number of inversions remains unchanged, by the induction hypothesis, the claim holds for g_{m+1}. A similar argument can be made for the case when g_{m+1} is inverting. □

3 Implication-based Analysis

We assume that logic implications of both value assignments (0 and 1) for every line in the circuit are available. We present lemmas that use logic implications on a line g to help identify a set of lines S, such that every path fault passing through g and $\{m : m \in S\}$ is untestable with respect to some combination of signal values. Let $S_P(x^\alpha, y^\beta)$, where $\alpha, \beta \in \{0, 1\}$, denote the set of path faults passing through lines x and y such that for every path fault $\{P^\psi : P^\psi \in S_P(x^\alpha, y^\beta)\}$, $[x(V_2) = \alpha]$ and $[y(V_2) = \beta]$ are necessary conditions for P^ψ to be sensitized on a vector pair $<V_1, V_2>$ (refer to Lemma 2.1).

3.1 Robust Untestability Analysis

We say that $[S_P(x^\alpha, y^\beta) = RU]$ if all path faults in the set $S_P(x^\alpha, y^\beta)$ are robustly untestable.

Lemma 3.1 *For lines x and y, if $(x = \alpha) \Rightarrow (y = \beta)$, where $\alpha, \beta \in \{0, 1\}$ and β is the controlling value of $g = gt(y)$, then $\forall_{z \in in(g), z \neq y} \forall_{\xi \in \{0,1\}} [S_P(x^\alpha, z^\xi) = RU]$.*

Proof: Consider a path fault $\{P^\psi : P^\psi \in S_P(x^\alpha, z^\xi)\}$, where $\psi \in \{0, 1\}$. Line y is a side-input of P^ψ. For a vector pair $<V_1, V_2>$ to be a robust test for P^ψ, the following conditions are necessary: (1) $x(V_2) = \alpha$ since $P^\psi \in S_P(x^\alpha, z^\xi)$, and (2) $y(V_2) = \overline{\beta}$, the non-controlling value of g (by Definition 2.3). However, since $(x = \alpha) \Rightarrow (y = \beta)$, a vector V_2 that satisfies conditions (1) and (2) cannot exist and hence, P^ψ is robustly untestable. □

Lemma 3.2 *For lines x and y, if $(x=\alpha)\Rightarrow(y=\beta)$, where α is the non-controlling value of $f=gt(x)$, and β is the controlling value of $g=gt(y)$, then $\forall_{a\in in(f),a\neq x}\forall_{b\in in(g),b\neq y}\forall_{\zeta\in\{0,1\}}\forall_{\xi\in\{0,1\}}[S_P(a^\zeta,b^\xi)=RU]$*

Proof: Consider a path fault $\{P^\psi : P^\psi\in P\}$, where $\psi\in\{0,1\}$ and P is the set of all path faults that pass though $\{a : a\in in(f), a\neq x\}$ and $\{b : b\in in(g), b\neq y\}$. Lines x and y are side-inputs of P^ψ. By Definition 2.3, for a vector pair $<V_1,V_2>$ to be a robust test for P^ψ, the following conditions are necessary: (1) $x(V_2)=\alpha$, the non-controlling value of f, and (2) $y(V_2)=\overline{\beta}$, the non-controlling value of g. However, since $(x=\alpha)\Rightarrow(y=\beta)$, a vector V_2 that satisfies conditions (1) and (2) cannot exist and hence, P^ψ is robustly untestable. □

Lemma 3.3 *For lines x and y, if $(x=\alpha)\Rightarrow(y=\beta)$, where $\alpha,\beta\in\{0,1\}$ and β is the controlling value of $g=gt(y)$, then $\forall_{z\in in(g),z\neq y}[S_P(x^{\overline{\alpha}},z^\beta)=RU]$.*

Proof: Consider a path fault $\{P^\psi : P^\psi\in S_P(x^{\overline{\alpha}},z^\beta)\}$, where $\psi\in\{0,1\}$. Line y is a side-input of P^ψ. For a vector pair $<V_1,V_2>$ to be a robust test for P^ψ, the following conditions are necessary: (1) $x(V_2)=\overline{\alpha}$ since $P^\psi\in S_P(x^{\overline{\alpha}},z^\beta)$ and hence $x(V_1)=\alpha$ (by Remark 2.3), and (2) $y(V_1)=\overline{\beta}$, the non-controlling value of g (by Definition 2.3(2)). However, since $(x=\alpha)\Rightarrow(y=\beta)$, a vector V_1 that satisfies conditions (1) and (2) cannot exist and hence, P^ψ is robustly untestable. □

Lemma 3.4 *For lines x and y, if $(x=\alpha)\Rightarrow(y=\beta)$, where $\alpha,\beta\in\{0,1\}$, then $[S_P(x^{\overline{\alpha}},y^\beta)=RU]$ and $[S_P(x^\alpha,y^{\overline{\beta}})=RU]$.*

Proof: By definition, to sensitize a path fault $\{P^\psi : P^\psi\in S_P(x^{\overline{\alpha}},y^\beta)\}$ on a vector pair $<V_1,V_2>$, $[x(V_2)=\overline{\alpha}]$ and $[y(V_2)=\beta]$ are necessary conditions. Hence, by Remark 2.3, for $<V_1,V_2>$ to be a robust test for P^ψ, the following conditions are necessary: (1) $[x(V_1)=\alpha]$, and (2) $[y(V_1)=\overline{\beta}]$. However, since $(x=\alpha)\Rightarrow(y=\beta)$, a vector V_1 that satisfies conditions (1) and (2) cannot exist and hence, P^ψ is robustly untestable. Similarly, to robustly test a path fault $\{P^\psi : P^\psi\in S_P(x^\alpha,y^{\overline{\beta}})\}$, (1) $[x(V_2)=\alpha]$, and (2) $[y(V_2)=\overline{\beta}]$ are necessary conditions. Since a vector V_2 that satisfies conditions (1) and (2) cannot exist, P^ψ is robustly untestable. □

Lemma 3.5 *If a line x is identified as having a constant value assignment $\{\alpha : \alpha\in\{0,1\}\}$, then $\forall_{\zeta\in\{0,1\}}[S_P(x^\zeta,x^\zeta)=RU]$.*

Proof: Since a vector pair $<V_1,V_2>$ that satisfies $\{x(V_1)\neq x(V_2)\}$ cannot exist, by Remark 2.3, all path faults that pass through x are robustly untestable. □

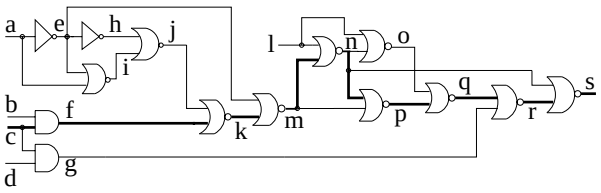

Figure 1: Portion of ISCAS-85 benchmark $c6288$

3.2 Functional Unsensitizability Analysis

Functionally unsensitizable path faults can be ignored during delay fault testing and timing analysis [11]. We say that $[S_P(x^\alpha,y^\beta)=FU]$ if all path faults in the set $S_P(x^\alpha,y^\beta)$ are functionally unsensitizable.

Lemma 3.6 *For lines x and y, if $(x=\alpha)\Rightarrow(y=\beta)$, where $\alpha,\beta\in\{0,1\}$, then $[S_P(x^\alpha,y^{\overline{\beta}})=FU]$.*

Proof: Similar to the proof of Lemma 3.4. □

Lemma 3.7 *If a line x is identified as having a constant value assignment $\{\alpha : \alpha\in\{0,1\}$, then $[S_P(x^{\overline{\alpha}},x^{\overline{\alpha}})=FU]$.*

Proof: Similar to the proof of Lemma 3.5. □

Other lemmas similar to those presented in Section 3.1 can be derived for identifying functionally unsensitizable path faults. However, in our experiments, we found that they do not help identify any additional functionally unsensitizable path faults.

Figure 1 illustrates an example of identifying a functionally unsensitizable path fault using Lemma 3.6. Consider the path fault $P^0=(c,f,k,m,n,p,q,r,s)$. We make two observations:

- For a vector pair $<V_1,V_2>$ to functionally sensitize P^0, the following conditions are necessary: (1) $[c(V_2)=0]$ (from Definition 2.1), and (2) $[s(V_2)=1]$ since the number of inversions between c and s is seven (from Lemma 2.1). Hence $[P^0\in S_P(c^0,s^1)]$.
- $(c=0)\Rightarrow(s=0)$ from static implication learning [10].

From Lemma 3.6, we can conclude that P^0 is functionally unsensitizable. Similarly $Q^0=(c,f,k,m,n,o,q,r,s)$ is functionally unsensitizable. We explicitly enumerate untestable path faults in this example only for illustration. Our algorithm obtains the number of untestable path faults without enumerating them.

3.3 Non-robust Untestability Analysis

We say that $[S_P(x^\alpha,y^\beta)=NU]$ if all path faults in the set $S_P(x^\alpha,y^\beta)$ are non-robustly untestable. To identify non-robustly untestable path faults, Lemmas 3.1 and 3.2 can be used by replacing RU by NU in them and Lemmas 3.6 and 3.7 can be used by replacing FU by NU in them.

4 Using Pre-computed Implications

We assume that some set of implications of both value assignments for every line in the circuit are available.

4.1 Maintaining Untestability Information

Using our fault-independent implication analysis at each line g, we construct four sets of lines associated with g: $O^1E^1(g)$, $O^1E^0(g)$, $O^0E^1(g)$, and $O^0E^0(g)$.

Definition 4.1 *A set $O^\alpha E^\beta(g)$ associated with a line g, where $\alpha, \beta \in \{0,1\}$, consists of lines $\{x : [S_P(g^\alpha, x^\beta) = Untestable]\}$*

Depending on the specific testability criterion used, rules that help construct the OE sets can be derived from the lemmas presented in Section 3. For conciseness, we only present an example involving the functional sensitization criterion. If we know that $(x=\alpha) \Rightarrow (y=\beta)$, where $\alpha, \beta \in \{0,1\}$, and y is in the fanout cone of x, from Lemma 3.6, y is added to $O^\alpha E^{\overline{\beta}}(x)$.

4.2 Counting Untestable Path Faults

Pomeranz and Reddy propose a linear-time counting algorithm [13] that can compute the number of paths in a single pass from POs to PIs. Based on their idea, we propose a counting algorithm that uses the OE sets on every line to compute a lower bound on the number of untestable path faults.

A *partial path fault* is a path fault without the restriction that its origin should be a PI. In this section, we use the terms *path faults* and *partial path faults* interchangeably. Consider a circuit with T lines. Let $B(x)$ denote the set of all branches of a fanout stem x. For a line y that is not a stem, let S^y denote the output of $gt(y)$. We associate a few variables with each line m:

- N_1^m (N_0^m) : number of testable path faults, as determined by our procedure, that originate at m and require a value of 1 (0) on m to be *sensitized*
- T_α^m : temporary values of N_α^m on a specific iteration
- $eval^m$: indicates whether T_1^m and T_0^m have been computed on a specific iteration
- $P(m)$: set of all immediate predecessor lines of m
- red_α^m : indicates if $\{S_P(m^\alpha, m^\alpha) = Untestable\}$, where $\alpha \in \{0,1\}$, on a specific iteration

To avoid reseting flags such as $eval^m$ and red_α^m, we use unique identifiers on each iteration. With no implications available, all path faults originating at a line $\{m : m \in$ fanout cone of line $a\}$ contribute towards the computation of N^a. Suppose $O^0E^1(a) = \{m\}$. By definition, N_1^m can now be ignored when computing N_0^a. However, it is possible that N_1^m may contribute towards the computation of N values for other lines in the fanin cone of m. Assuming that the OE sets associated with every line are available, Algorithm 4.1 processes lines in a reverse topological order (POs to PIs) to compute a lower bound on the number of untestable path faults.

Algorithm 4.1

```
testable-path-count() {
  ∀1<m<T {eval^m = red_1^m = red_0^m = 0}
  ∀1<m<T {N_0^m = N_1^m = ∞}
  for x = T to 1   // reverse topological order
    foreach(a ∈ O^1E^1(x)) {
      insert(a);   red_1^a = x }
    foreach(a ∈ O^1E^0(x)) {
      insert(a);   red_0^a = x }
    label(x, x)
    N_1^x = T_1^x
    foreach(a ∈ O^0E^1(x)) {
      insert(a);   red_1^a = x+T }
    foreach(a ∈ O^0E^0(x)) {
      insert(a);   red_0^a = x+T }
    label(x, x+T)
    N_0^x = T_0^x
  # of testable faults (T_P) = Σ_{i is a PI} {N_1^i + N_0^i }
  # of untestable faults = Total # of path faults - T_P
}
label(x, id) {
  insert(x)
  while(n = dequeue())
    eval^n = id; T_0^n = T_1^n = 0
    if(red_1^n ≠ id)
      if(n is connected to a PO){T_1^n = 1} else
      if(n is a stem)T_1^n = Σ_{b∈B(n)} choose(N_1^b, T_1^b, b, id)
      else {T_1^n = choose(N_α^{S^n}, T_α^{S^n}, S^n, id)
        // α=1 if gt(n) is non-inverting; 0 otherwise
    T_1^n = lesser(T_1^n, N_1^n) // lesser of the two values
    if(red_0^n ≠ id)
      if(n is connected to a PO){T_0^n = 1} else
      if(n is a stem)T_0^n = Σ_{b∈B(n)} choose(N_0^b, T_0^b, b, id)
      else {T_0^n = choose(N_α^{S^n}, T_α^{S^n}, S^n, id)
        // α=0 if gt(n) is non-inverting; 1 otherwise
    T_0^n = lesser(T_0^n, N_0^n)
    if(T_1^n ≠ N_1^n) or if(T_0^n ≠ N_0^n)
      ∀a∈P(n){(if a is closer to a PO than x) insert(a)}
}
choose(x, y, n, id) {
  if(eval^n ≠ id){return(x)} else {return(y)} }
insert(x) {insert x into an event list EL}
dequeue() {
  if EL is non-empty, dequeue and return element
  of EL that is closest to a PO; return 0 otherwise }
```

4.3 Counting Untestable Segment Faults

The discussion in Sections 3 and 4.1 is also applicable to the segment delay fault model. We modify the segment counting algorithm presented in the literature [3] to use implications to determine a lower bound on the number of untestable segment faults.

Table 1: A lower bound on the number of untestable delay faults

Ckt. Name	Segment faults				Path faults							
	Robustly untestable				Robustly untestable			Non-robustly untestable		Functionally unsensitizable		
	$L=3$		$L=5$									
	#	%	#	%	#	%	cpu(s)	#	%	#	%	cpu(s)
c880	0	0.0	0	0.0	326	1.9	0	163	0.9	163	0.9	0
c1355	576	8.8	7,840	33.9	8,005,696	95.9	2	7,150,240	85.7	6,745,120	80.8	1
c1908	8	0.1	1,260	6.0	1,070,307	73.4	4	1,067,159	73.2	442,048	30.3	1
c2670	253	2.6	1,307	6.7	1,321,906	97.2	2	1,317,795	96.9	1,314,962	96.7	1
c3540	673	4.7	4,129	12.7	53,610,698	93.5	18	52,488,315	91.5	34,300,319	59.8	6
c5315	238	1.1	1,486	3.6	2,013,498	75.1	14	1,865,548	69.5	1,129,995	42.1	4
c6288	2,442	8.1	23,577	26.1	1.978×10^{20}	99.9	3	1.978×10^{20}	99.9	1.978×10^{20}	99.9	3
c7552	563	1.5	3,709	4.1	981,720	67.6	24	910,926	62.7	555,050	38.2	7
s5378	284	1.8	815	3.0	6,396	23.6	9	4,869	18.0	3,718	13.7	4
s9234	1,225	4.3	4,275	8.7	442,526	90.4	58	413,785	84.5	282,149	57.6	21
s13207	1,817	5.0	5,091	8.8	2,300,812	85.5	50	1,870,582	69.5	1,722,492	64.0	24
s15850	3,349	7.1	12,321	15.3	322,581,591	97.9	101	303,523,949	92.1	274,843,560	83.4	40
s35932	11,866	12.5	35,372	27.4	354,324	89.9	519	265,863	67.4	248,567	63.0	241
s38417	3,229	2.9	15,149	7.6	1,675,008	60.2	86	1,377,425	49.5	796,701	28.6	42
s38584	1,608	1.3	6,938	3.6	1,623,570	75.1	26	1,169,090	54.1	1,123,814	52.0	15

†Only a subset of direct implications was available for s38584

5 Results

We implemented our algorithm in C++ and ran experiments using a HP 9000/735 workstation with 256MB of memory. We use the implications generated by Zhao et. al. [10] for our work. Direct implications, determined by forward and backward propagation starting at the node under consideration, and indirect implications, found by applying the contrapositive law [9], transitive law and backward implications [10] were available for most circuits. Only a subset of direct implications was available for $s38584$. Information on constant value assignments that is generated as a by-product of their implication procedure is also used. For all the ISCAS-85 circuits, their program took less than 150 seconds to generate the relevant information [10]. Times taken by their program for the ISCAS-89 circuits were not available.

Table 1 shows results of the algorithm for a subset of all values that segment length L, an input parameter to the program, can take. For path delay faults, we consider the robust, non-robust and functional sensitization criteria. For segment faults with $(L = 3)$ and $(L = 5)$, we only consider the robust testability criteria. Columns with the heading # show the number of untestable faults determined by our procedure. Columns with the heading % indicate the percentage of untestable faults with respect to the total number of faults. The runtimes in CPU seconds are shown only for identifying robustly untestable and functionally unsensitizable path faults. Runtimes for identifying robustly untestable segment faults and non-robustly untestable path faults were similar to those for identifying robustly untestable path faults. However, there is a marginal increase in the runtimes as L is increased. In all cases, the runtime of our method is very small and is independent of the number of faults. While our method identifies more untestable path faults in some circuits, previously published methods [11, 12] do better for some circuits.

Since we identify only a subset of all untestable faults, firm conclusions cannot be drawn. However, for circuits such as $c1908$, it appears that many non-robustly untestable path faults do not belong to the set of functionally unsensitizable faults. Such path faults are testable only as a multiple fault [16]. Dealing with multiple path faults may be computationally intractable for large circuits. In such cases, using the segment delay fault model, with a small value of L, may be a feasible alternative. For circuits like $c2670$, most of the path faults are functionally unsensitizable. These path faults can be ignored for the purposes of delay testing and the

Table 2: Our results versus ATPG results

Ckt. Name	Non-robustly untestable path faults	
	Exact [15]	Our procedure
s5378	19.0%	18.0%
s9234	87.8%	84.5%
s13207	82.3%	69.5%
s15850	96.7%	92.1%
s35932	85.2%	67.4%
s38417	59.1%	49.5%
s38584	84.5%	54.1%

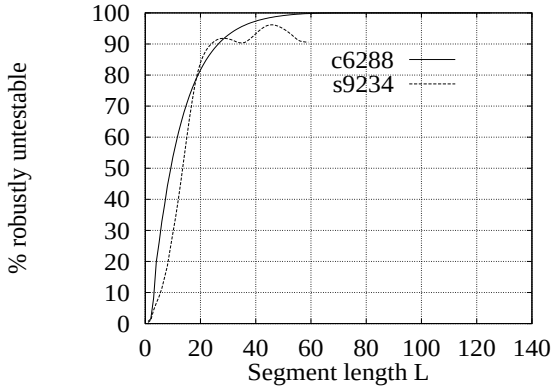

Figure 2: % of untestable faults versus segment length

path fault model may be practical in such cases.

Table 2 compares our results with the exact results obtained from a test generator [15] for the ISCAS-89 circuits. The exact number of untestable faults is unknown for many of the ISCAS-85 circuits. Our method gives only a lower bound on the number of untestable faults since it is based on an incomplete set of implications. During an ATPG run, a lower bound on the number of untestable faults may be useful as a stopping criterion if the required fault efficiency is reached.

In our experiments, the percentage of untestable faults increases monotonically with L and then decreases as L approaches the maximum logic depth. Figure 2 shows this trend for robustly untestable faults in $c6288$ (maximum logic depth: 125) and $s9234$ (maximum logic depth: 59). For $c6288$, we identified $1.9788271{\times}10^{20}$ robustly untestable path faults, of which 1.9778345×10^{20} were also functionally unsensitizable.

6 Concluding Remarks

Our algorithm uses static logic implications and rapidly computes a lower bound on the number of robustly untestable, non-robustly untestable, and functionally unsensitizable delay faults by using a non-enumerative counting procedure. Untestable faults can also be listed if desired and targeting them for test generation by an ATPG tool can be avoided. One of the main features of the algorithm is that its complexity of computation does not grow with the number of delay faults. This is especially important when considering the path delay fault model since circuits typically have a large number of path faults. The implication analysis presented in this paper considers one implication at a time. It may be possible to obtain better results by considering multiple implications simultaneously.

References

[1] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. International Test Conf.*, pp. 342–349, Nov. 1985.

[2] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. on CAD*, vol. 6, pp. 694–703, Sept. 1987.

[3] K. Heragu, J. H. Patel, and V. D. Agrawal, "Segment Delay Faults: A New Fault Model," in *Proc. VLSI Test Symp.*, pp. 32–39, Apr. 1996.

[4] K. Heragu, J. H. Patel, and V. D. Agrawal, "SIGMA: A Simulator for Segment Delay Faults," in *Proc. International Conf. CAD*, pp. 502–508, Nov. 1996.

[5] I. Pomeranz and S. M. Reddy, "On Achieving Complete Testability of Synchronous Sequential Circuits with Synchronizing Sequences," in *Proc. International Test Conf.*, pp. 1007–1016, Oct. 1994.

[6] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits," *IEEE Trans. on CAD*, vol. 14, pp. 1155–1160, Sep. 1995.

[7] M. A. Iyer and M. Abramovici, "FIRE: A Fault-Independent Combinational Redundancy Identification Algorithm," *IEEE Trans. on VLSI Systems*, vol. 4, pp. 295–301, Jun. 1996.

[8] M. A. Iyer, D. E. Long, and M. Abramovici, "Identifying Sequential Redundancies Without Search," in *Proc. 33rd Design Automation Conf.*, Jun. 1996.

[9] W. Kunz and D. K. Pradhan, "Accelerated Dynamic Learning for Test Pattern Generation in Combinational Circuits," *IEEE Trans. on CAD*, vol. 12, pp. 684–694, May 1993.

[10] J. Zhao, E. M. Rudnick, and J. H. Patel, "Static Logic Implication with Application to Redundancy Identification," in *Proc. VLSI Test Symp.*, pp. 288–293, Apr. 1997.

[11] K. T. Cheng and H. C. Chen, "Delay Testing for Non-Robust Untestable Circuits," in *Proc. International Test Conf.*, pp. 954–961, Oct. 1993.

[12] S. Kajihara, K. Kinoshita, I. Pomeranz, and S. Reddy, "A Method for Identifying Robust Dependent and Functionally Unsensitizable Paths," in *Proc. 10th International Conf. on VLSI Design*, pp. 82–87, Jan. 1996.

[13] I. Pomeranz and S. M. Reddy, "An Efficient Non-Enumerative Method to Estimate the Path Delay Fault Coverage in Combinational Circuits," *IEEE Trans. CAD*, vol. 13, pp. 240–250, Feb. 1994.

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

[15] K. Fuchs, M. Pabst, and T. Rossel, "RESIST: A Recursive Test Pattern Generation Algorithm for Path Delay Faults Considering Various Test Classes," *IEEE Trans. on CAD*, vol. 13, pp. 1550–1561, Dec. 1994.

[16] W. Ke and P. R. Menon, "Synthesis of Delay-verifiable Combinational Circuits," *IEEE Trans. on CAD*, vol. 44, pp. 213–222, Feb. 1995.