Efficient Diagnosis of Path Delay Faults in Digital Logic Circuits [*]

Pankaj Pant
Georgia Institute of Technology
pant@ee.gatech.edu

Abhijit Chatterjee
Georgia Institute of Technology
abhijit.chatterjee@ee.gatech.edu

Abstract

A new methodology involving effect-cause analysis has been demonstrated for the diagnosis of path delay faults. We seek to provide an improved understanding of the methods introduced in [6], with the goal of devising efficient representations and algorithms for the diagnosis of path delay faults. Results indicate that the diagnostic resolution obtained is very high and includes all possible causes of the observed delay faults.

1 Introduction

With rapidly shrinking feature sizes and high levels of integration in today's integrated circuits, the problem of testing manufactured digital parts for delay faults has become very essential [1, 2]. The first chips that are manufactured at the early stages of the development cycle of an IC are usually very susceptible to delay faults. This is because, during the design stage, it is usually not possible to accurately model many of second order effects that occur in deep-submicron designs. Most timing simulators use abstracted high level timing models to speed up the simulation times for large scale designs. As a result of this mismatches occur between the "expected" gate delays and the actual delays in the chip often leading to delay faults in the IC.

Along with the process of testing the manufactured chips for delay faults, there exists a need for a diagnosis capability that can identify the faulty locations. An efficient diagnosis tool can drastically reduce the redesign time by providing the designers with a small set of possible faults to investigate. Moreover, the diagnosis can also assist in improving and adjusting the high level timing models and simulation tools to reduce the extent of the mismatch between predicted and observed delays.

Earlier work on delay fault diagnosis [3, 4, 5] has focussed primarily on gate delay faults. The problem with the gate delay fault model is that it is unable to capture the situation where small delay variations in a number of gates accumulate to produce a delay fault. On the other hand, the obvious drawback of the path delay fault model is that the number of paths can grow exponentially with the circuit size. We cannot use the cause-effect diagnosis paradigm, where all possible fault combinations are simulated before the tests are applied and the results used to identify the faults. Under the path delay fault model the number of multiple faults can easily become intractable. In [6], the authors proposed a new scheme for path delay fault diagnosis. The central idea of their work was the use of cause-effect reasoning, ie. the results of the applied tests is studied to obtain all the possible reasons for the failing tests. This scheme allows the identification of all the possible causes of the observed faults.

Since the primary intention of the work in [6] was the demonstration of the concepts presented, the algorithms developed were preliminary and in many cases inefficient. In this paper, we take the ideas developed in [6] and augment them with novel and efficient algorithms and data-structures. The algorithms we present for extraction of the possible causes (hereby referred to as the *suspect set*) of the failing tests are much faster. A scheme has been demonstrated for trimming the suspect sets with the goal of improving the diagnostic resolution that is more complete and much faster than those proposed in [6]. Furthermore, we have developed efficient data-structures that enable the storage and retrieval of the suspect sets in a fast and memory efficient manner.

[*]This research was supported by NSF under Grant MIP-9502575 at the Georgia Institute of Technology.

2 Review of previous work

We will briefly review the basics of the circuit representations and the effect-cause paradigm that were presented in [6]. For a given test, there may be a number of paths in the circuit that are sensitized (robustly or non-robustly [7, 8]) to the primary outputs. If an erroneous value is observed at an output, then a delay fault is presumed to exist in one or more of the sensitized paths that terminate at the output. Hence, for the purpose of diagnosis only the paths in the input cone of the primary output concerned need to be analyzed. Without any loss of generality, we will henceforth assume that the circuit under test (CUT) has only one primary output.

To understand the methodology, consider the circuit in Fig. 1 which is excited by a two pattern test. Let us assume that the circuit failed the applied test. For the purpose of this discussion, we assume that a path delay fault with a rising input transition is equivalent to a path delay fault with a falling input transition. (This is generally not the case and in our implementations we have considered them to be distinct failures.) Table 1 enumerates the paths in the circuit.

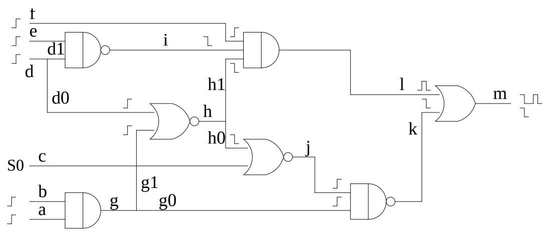

Figure 1: Example circuit

#	Path	#	Path
P_0	a-g-g0-k-m	P_6	c-j-k-m
P_1	a-g-g1-h-h0-j-k-m	P_7	d-d0-h-h0-j-k-m
P_2	a-g-g1-h-h1-l-m	P_8	d-d0-h-h1-l-m
P_3	b-g-g0-k-m	P_9	d-d1-i-l-m
P_4	b-g-g1-h-h0-j-k-m	P_{10}	e-i-l-m
P_5	b-g-g1-h-h1-l-m	P_{11}	f-l-m

Table 1: Paths of circuit in Fig. 1

Consider line h in the circuit. If the transition at line h is delayed, it implies that the transitions at both its inputs (lines $d0$ and $g1$) must also have been delayed since they are both going to the controlling value of the NOR gate. Hence, even though paths P_1 and P_7 (refer to Table 1) are not robustly or non-robustly sensitized by themselves, they could have produced a delay at the output *if both of them had delay faults*. Note that a delay fault in only one of them would not have been exposed by the given test. Thus, P_1P_7 is a multiple path delay fault (MPDF) [8] which is robustly sensitized to the output. We will refer to this situation by saying that the product term P_1P_7 is a suspect fault. The following definition generalizes this concept to cover all the possible suspects for a given failing test.

Definition 1 (Suspect set [6]) The *suspect set* is a set of product terms of paths. Each product term (or suspect) denotes a group of paths that could explain an error observed at the output provided that all the paths in the group are faulty.

It can be easily verified that the complete suspect set for the applied test is $\{P_0, P_3, P_1P_7, P_4P_7, P_2P_8P_9, P_2P_8P_{10}, P_5P_8P_9, P_5P_8P_{10}\}$. Any of these terms could have caused the test to fail.

Since the suspect set could become very large, it is desirable to find an alternate space efficient representation that is easier to work with. The suspect set can be represented very efficiently by an equivalent circuit [6] called the *suspect circuit* (C_S). The suspect circuit retains the path structure of the original circuit. Moreover, it also captures the relationship between the paths such that the suspect terms can be easily reconstructed.

The extraction of the suspect circuit is based on the following observation. Consider a gate which has one or more inputs with a controlling final transition (ie. the inputs change from the non-controlling value to the controlling value). All of those inputs must have delayed transition for the gate output to have a delayed transition. The suspect circuit captures this by an AND gate with only the inputs with the controlling transitions. On the other hand, if the gate has only non-controlling final transitions at its inputs, then a delayed transition at any of the changing inputs would cause the gate to have a delayed output transition. This is represented by an OR gate with those inputs that have the non-controlling transitions. A gate with only one input transition is represented by a BUF (buffer) gate. The suspect circuit can be extracted in a single breadth-first pass from the primary output to the primary inputs.

The suspect circuit for the test in Fig. 1 is shown in Fig. 2. Note that gate g is represented as an OR-gate since both its inputs have non-controlling transitions. On the other hand, gate h is represented as an AND-gate as both its inputs have controlling transitions. The suspect circuit does not include lines c and f since a delay in these lines cannot cause a delay at m.

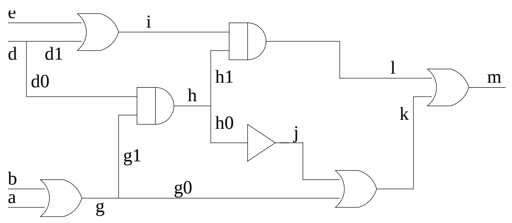

Figure 2: Suspect circuit for the test in Fig. 1

3 Extraction of the suspect set

There are several problems with the use of the suspect set as defined above. Firstly, even though a suspect set was defined in [6] for every line in the circuit, it was meaningful only at the output. Secondly, the number of terms in the suspect set of an internal line in the circuit could be exponentially larger than that of the output. This makes the enumeration of the suspect set very computationally intensive. Thirdly, there is no way to count the number of terms in the suspect set at the output other than by enumerating the entire set. In this section, we will describe an alternate representation that addressed all the above issues.

Definition 2 (Prefix path) Any path segment from a primary input to a line l is said to be a *prefix path* of l. For the sake of brevity, we will describe a prefix path from an input i to a line l as $(i\, f_1\, f_2 \ldots f_k l)$, where $f_1, f_2, \ldots, f_k$ are the fanout branches included in the prefix path.

Analogous to the suspect set, we ascribe to each line, l, in the suspect circuit a *local suspect set* (LSS(l)).

Definition 3 (Local suspect set) The *local suspect set* of l is a set of product terms consisting of prefix paths of l that could potentially be the cause of a delayed transition at l.

Unlike the suspect set, the local suspect sets are meaningful at every line in the circuit and provide information about all the possible causes of a delayed transition at a given line. We shall see that the two sets are equal at the output line of the suspect circuit. To demonstrate how the local suspect sets are constructed, we define the following set operations.

Definition 4 (Union [6]) The *union* of n sets $A_i, \ldots, A_n$ is defined as $\bigcup_{i=1\ldots n} A_i = \{a_i \mid a_i \in A_j, j = 1, \ldots, n\}$.

Definition 5 (AND-product [6]) The *AND-product* of n sets $A_i, \ldots, A_n$ is defined as $\prod_{i=1\ldots n} A_i = \{(a_1 \ldots a_n) \mid a_j \in A_j, j = 1, \ldots, n\}$.

Definition 6 (Extension) The *extension* of a prefix path $p = (l_1 \ldots l_k)$ by l is denoted as $(p \parallel l) = (l_1 \ldots l_k l)$, provided that $(l_1 \ldots l_k l)$ is also a valid prefix path in the circuit. The extension of a product of prefix paths, $a = (p_1 \ldots p_k)$, by l is defined as $(a \parallel l) = ((p_1 \parallel l) \ldots (p_k \parallel l))$. ie. all the paths in the term are extended. Finally, the extension of a set of product terms, A, by l is defined as $A \parallel l = \{a_i \parallel l \mid a_i \in A\}$.

We can construct the local suspect set of each line in the suspect circuit as follows.

```
GenerateLSS (suspect circuit, $C_S$)
Return: The local suspect set at each line of $C_S$
begin
    foreach line $l$ in forward topological order in $C$
        if $l$ is a primary input, LSS($l$) = $\{(l)\}$.
        if $l$ is the output of an AND gate, LSS($l$) =
            $\prod_{i \text{ an input of } l}$ (LSS($i$) $\parallel$ $l$). This is because the
            signals at all the input lines must be delayed for a
            delay fault to occur at $l$. Note that each prefix path
            is extended by $l$.
        if $l$ is the output of an OR gate, LSS($l$) =
            $\bigcup_{i \text{ an input of } l}$ (LSS($i$) $\parallel$ $l$), since any delayed input
            can cause a delay fault at $l$.
        if $l$ is the output of a BUF gate (with input $i$),
            LSS($l$) = (LSS($i$) $\parallel$ $l$).
        if $l$ is fanout branch of a fanout stem $s$, LSS($l$) =
            (LSS($s$) $\parallel$ $l$), since any fault which causes a delay
            at $s$ will also cause a delay at $l$.
end
```

Since all the prefix paths of the output, g, of the suspect circuit are complete input to output paths, LSS(g) contains all the product terms of paths in the suspect circuit that could potentially cause a delay fault at g. This is, by definition, the same as the suspect set of g. It is important to note that the local suspect set of a line can never have more terms than that of a line in its transitive fanout cone. On the other hand, the suspect set at an internal line could have exponentially more terms than that of the output, making its enumeration very difficult. Thus local suspect sets provide us with a more efficient way to compute the suspect set of the output. Table 2 shows

Line	Local Suspect Set
d_0	(dd_0)
d_1	(dd_1)
g	$(ag),(bg)$
g_0	$(ag_0),(bg_0)$
g_1	$(ag_1),(bg_1)$
h	$(ag_1h)(dd_0h),(bg_1h)(dd_0h)$
h_0	$(ag_1h_0)(dd_0h_0),(bg_1h_0)(dd_0h_0)$
h_1	$(ag_1h_1)(dd_0h_1),(bg_1h_1)(dd_0h_1)$
i	$(ei),(dd_1i)$
j	$(ag_1h_0j)(dd_0h_0j),(bg_1h_0j)(dd_0h_0j)$
k	$(ag_0k),(bg_0k),(ag_1h_0k)(dd_0h_0k),(bg_1h_0k)(dd_0h_0k)$
l	$(ag_1h_1l)(dd_0h_1l)(el),(bg_1h_1l)(dd_0h_1l)(el),$ $(ag_1h_1l)(dd_0h_1l)(dd_1l),(bg_1h_1l)(dd_0h_1l)(dd_1l)$
m	$P_0,P_3,P_1P_7,P_4P_7,$ $P_2P_8P_{10},P_5P_8P_{10},P_2P_8P_9,P_5P_8P_9$

Table 2: Local Suspect Sets

the application of these rules to compute the local suspect sets at each line in the suspect circuit of Fig. 2. The local suspect sets of the input lines (a, b, d and e) are trivial and have been omitted.

It is important to be able to efficiently count the number of suspect terms to get an estimate of the diagnostic resolution provided by a given test set. By mirroring the steps in the above procedure, we can non-enumeratively compute the number of terms in the local suspect sets of each line. The computation can be performed in linear-time (in the number of lines) by a single breadth first sweep of the suspect circuit from the primary inputs to the output. This method is much faster than enumerating the sets explicitly at each line to calculate their sizes.

Fig. 3 illustrates the calculation of the suspect set size at the output line m for the suspect circuit from Fig. 2. The sizes at each line are indicated beside the line names. We can see that the sizes of the local suspect sets are the same as indicated in Table 2.

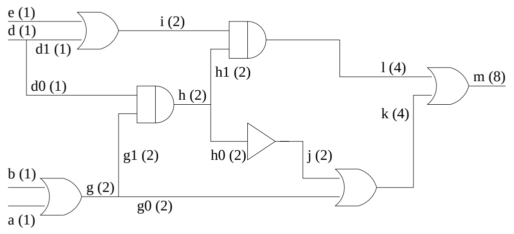

Figure 3: Computing the size of the suspect set

4 Elimination of robustly tested paths

In any given test set there will usually exist some passing tests. Any path that is robustly tested by such a test is proven to be fault-free by definition [7, 8]. These robustly tested path delay faults (RPDF) can be removed from the suspect sets to further improve diagnostic resolution. Let us assume that a suspect set contains the product term $P_iP_jP_k$. If path P_i has passed a robust test, then we can remove the term from consideration. This is because all the three paths P_i, P_j and P_k need to be faulty for the term P to have caused the delay at the output. Next we will show how this information can be used to update the suspect circuit (C_S) such that it does not contain any robustly tested path. The algorithms that we propose are more complete and much faster than those demonstrated in [6].

In [6], the authors conjecture that a line l in C_S can be removed if none of the paths passing though line l are "suspicious". The algorithm that was demonstrated by the authors marks a path to be non-suspicious only if it has been robustly tested. This definition is, however, too restrictive and leads to a number of paths being marked suspicious and hence not removable. A path P in C_S *ceases to be suspicious if (a) it has been robustly tested, or (b) it exists only in product terms with robustly tested paths*. The implication of the second clause is that path P could not have caused the test to fail by itself even if it had a delay fault, and hence should not be considered.

For the purpose of illustration, let us assume that paths P_0 and P_3 in Fig. 1 have passed robust tests. We can see from Fig. 2, that these are the only paths passing through line $g0$. Hence, line $g0$ can be removed from C_S.

Let us further assume that besides the paths P_0 and P_3, paths P_1 and P_4 have also been tested robustly. Given this information we cannot find any lines in C_S that have only robustly tested paths passing through it. Hence, it was concluded in [6] that C_S can not be reduced further. However, on further examination of Fig. 2 we find that there are three lines, $h0$, j and k, that have the paths P_1, P_4 and P_7 passing through them. The first two of these paths are fault free by assumption, while the third path occurs only in the product terms P_1P_7 and P_4P_7. Hence, all the three paths are non-suspicious by our new definition and the lines can be removed from C_S. The updated circuit is shown in Fig. 4. The suspect set is $LSS(m) = \{P_2P_8P_9, P_2P_8P_{10}, P_5P_8P_9, P_5P_8P_{10}\}$. Hence, by applying the new definition of non-suspicious paths we can further reduce the suspect circuit. This leads to a more memory efficient representation of the suspect set and better diagnostic resolution.

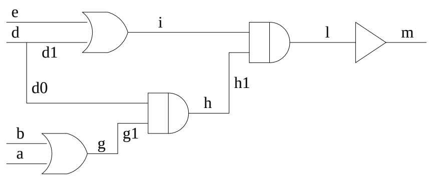

Figure 4: Updated suspect circuit

4.1 Removability of lines from C_S

It is not practical to explicitly check each line in C_S to see whether all the paths passing through it are no longer suspicious. We now present some simple implication rules that can significantly reduce the effort. These rules allow us to test for the removability of lines in C_S given that some neighboring lines are removable. The rules are divided into two categories: forward implications and backward implications. The forward implication rules are described below.

- If all the input lines of an OR gate in C_S are removable, then the output line is also removable. This follows from the observation that the paths passing through the output line are the union of the paths passing through the input lines.
- If any of the input lines of an AND gate in C_S is removable, then the output line is removable. Assume that line $h1$ in Fig. 2 is removable, i.e. all the paths passing through it are non-suspicious. Then, the paths passing through line l are either the paths through $h1$ or occur in a product term with some path passing through $h1$ and are hence non-suspicious. Hence, line l can be removed.
- If the input of a BUF gate in C_S is removable, then the output line is also removable.

The backward implication rules are listed next.

- If the output line of a gate if removable, then all the input lines are removable too. This is straightforward since all the paths passing through an output line must pass through one of the input lines.

- If all the fanout branches of a fanout point are removable, then the fanout stem is also removable. This is simply the reverse case of the forward implication of an OR gate and can be explained along the same lines.

To illustrate the process of implications let us revisit the reduced C_S in Fig. 2. Suppose that we are told that path P_8 has passed a robust test. (Note that P_8 was chosen for illustration purposes only. The path is not robustly testable by itself.) We can easily verify that line $h1$ is removable. Applying the forward implication rules, we see that line l is also removable since it is the output of an AND gate that has a removable input. Next we apply the backward rules and mark lines i, e and $d1$ removable. Thus by checking just one line ($h1$) we have reduced C_S to the circuit shown in Fig. 5.

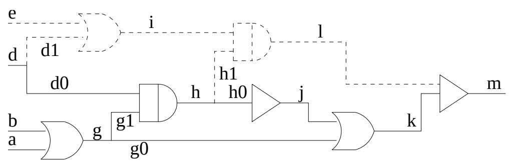

Figure 5: Illustration of implication rules

It can be shown that it is sufficient to test for the removability of the input lines and the fanout branches of C_S. By implication we can deduce the status of the rest of the lines. Next we present a recursive algorithm that eliminates all removable lines from C_S.

```
CheckRemovability (suspect circuit $C_S$, line $l$)
Return: TRUE if $l$ is removable from $C_S$
begin
    /* first check the fanout branches */
    while ($\exists$ unmarked fanout branch $f$ in $C_S$)
        $C^* \longleftarrow$ ExtractSubcircuit ($C_S, f$)
        if (CheckRemovability ($C^*, f$) returns TRUE)
            $C_S \longleftarrow$ ReduceByImplications ($C_S, f$)
        if ($l$ was marked removable)
            return TRUE
    end while
    /* next check the primary inputs */
    while ($\exists$ unmarked input line $i$ in $C_S$)
        $C^* \longleftarrow$ ExtractSubcircuit ($C_S$,$i$)
        if ($C^*$ consists of a single robustly tested path $P$)
            $C_S \longleftarrow$ ReduceByImplications ($C_S$,$i$)
        else if (CheckRemovability ($C^*$,$i$) returns TRUE)
            $C_S \longleftarrow$ ReduceByImplications ($C_S$,$i$)
        if ($l$ was marked removable)
            return TRUE
    end while
    return FALSE
end
```

The procedure ExtractSubcircuit extracts the sub-circuit of C_S that consists of only the paths passing through line l. ReduceByImplications deletes a removable line l from C_S and any other lines that are marked removable by applying the implication rules.

CheckRemovability is invoked with the initial suspect circuit and NULL as the inputs (i.e. we do not specify any line). The procedure then tests for the removability of each fanout branch and input lines, applying the implication rules after each test. For each fanout branch f, first the sub-circuit which consists of only the paths of C_S that pass through f is extracted (C^*). Observe that if line f is removable from C_S, it is also removable from C^* and vice-versa. Hence to check for the removability of f from C_S we recursively check for the removability of f in C^*, which is a simpler task since C^* is a smaller circuit. The recursion reduces the circuit till it becomes fanout free, at which point it becomes trivial to check for the removability of lines.

It can be seen that the recursion depth is bounded by the maximum number of nested reconvergent fanouts. The procedure is very fast and by using the implication rules can rapidly identify all the removable lines. This is much faster than the exhaustive routines for eliminating robustly tested paths in that were outlined in [9]. Moreover, as demonstrated, it is also more complete and can detect many more non-suspicious paths.

5 Suspect set storage and retrieval

In this section we present a suspect storage and retrieval technique which overcomes some of the complexity issues that the authors faced in [6]. The main problem was maintaining and using the information provided by robustly tested multiple path delay faults (RMPDF). A set of paths is said to be RMPDF tested [2] if they pass a test which satisfies the robustness conditions for all the off-path inputs. (Note that a gate may have more than one on-path inputs.) This test guarantees that at least one of the paths in the set is fault free. Similar to RPDFs, RMPDFs can be also be used to eliminate terms from the suspect set. Suppose that the term $P_iP_jP_k$ occurs in a given suspect set and the delay fault P_iP_k is RMPDF tested. Since this ensures that at least one of P_i and P_k is fault-free, the term $P_iP_jP_k$ can be removed from the suspect set. However, the possibility of a large number of RMPDFs coupled with the high complexity of performing RMPDF simulation on the suspect circuits makes this a daunting task.

Here we make an observations. Suppose that the terms P_iP_k and $P_iP_jP_k$ are both RMPDF tested. Then only the first term needs to be stored since it guarantees that at least one of P_i and P_k are fault-free which in turn implies that term $P_iP_jP_k$ is also fault-free. Thus, only the smallest RMPDFs need to be retained. In particular, if a path is robustly tested then all the RMPDFs which contain the path can be discarded. This greatly reduces the number of RMPDFs that need to be maintained for simulation purposes.

The process of detecting the RMPDFs is exactly the same as the detection of the suspect sets. Similar to suspect circuits, for each passing test, we extract the *robustly tested circuit* (R_C) which contains the RMPDFs. Next, procedure CheckRemovability is used to remove all the RPDFs from this circuit. The remaining RMPDFs can be extracted by applying the procedure described in Section 3. All the RMPDFs that can potentially be used to reduce the suspect sets are stored in a global database. The overall flow for creating the database of robustly tested and the construction of the final diagnosis set is summarized next.

```
DiagnoseCircuit (circuit $C$, test set $T$)
begin
    foreach passing test ($T_P$): identify the RPDFs.
        These paths are stored in the set $S_{TP}$.
    foreach passing test ($T_P$):
        Construct the robustly tested circuit ($R_C$).
        Reduce $R_C$ using the RPDFs in $S_{TP}$.
        foreach remaining RMPDF ($t_r$) in $R_C$:
            if a smaller RMPDF was previously added
                    Discard $t_r$.
            else    Delete any RMPDFs which contain $t_r$.
                    Add $t_r$.
    foreach failing test ($T_F$):
        Construct the suspect circuit ($S_C$).
        Reduce $S_C$ using the RPDFs in $S_{TP}$.
        foreach remaining suspect term ($t_s$) in ($S_C$):
            if a smaller RMPDF exists in the database
                    Discard $t_s$.
            else    Save $t_s$.
end
```

We have designed data-structures that make the process of RMPDF storage and simulation very efficient. Each RMPDF term is stored as a ring which can be accessed from each of the paths in the term as shown in Fig. 6 for the term $P_iP_jP_k$.

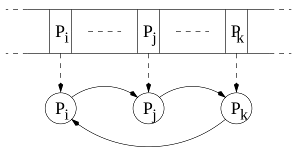

Figure 6: RMPDF storage

Efficient routines have been implemented for each of the steps described above. Addition and deletion of terms from this global structure is very straightforward. Moreover, since the rings representing the terms are indexed from each path in the term, it is very easy to detect all the existing subsets and supersets of a specified term.

Our proposed scheme obviates the need to keep the robustly tested circuits and the suspect circuits (as in [6]). More importantly, we do not need to perform any RMPDF simulations. Instead, we simply check for the existence of any sub-terms stored in the global database of RMPDFs. This can be done very efficiently and leads to a significant speedup in the algorithm execution.

6 Experiments and results

We have performed diagnostic experiments on the combinational parts of a number of ISCAS89 benchmark circuits. For each of the circuits, delay faults were injected by increasing the delay of five randomly chosen gates. The test sets that were used for the experiments included robust tests for the longest paths in the circuits and tests that excited a number of multiple path delay faults.

Table 3 indicates the results for RMPDF simulation and storage for the passing tests. (T and P indicate the number of RMPDF terms and the paths in them respectively.) As discussed in Section 5, we see that the number of RMPDFs that were actually recorded is much lesser than the total number of RMPDFs sensitized. Thus the memory penalty for saving the RMPDFs is minimal.

Circuit	Total Paths	Sensitized T	Sensitized P	Stored T	Stored P
s208	290	74	234	12	32
s298	462	72	162	0	0
s344	688	242	483	32	82
s349	708	164	383	65	140
s510	738	373	813	52	119
s526	820	169	365	11	28
s820	984	374	914	47	105
s832	1012	184	470	37	84

Table 3: RMPDFs storage results

Table 4 presents the results for the diagnosis experiments that were performed on the circuits. The total number of terms in the suspect sets from the failing tests are indicated in the first column. RPDF simulation was performed to reduce the size of the suspect circuits using the scheme demonstrated in [6] and our proposed methodology and the sizes of the reduced suspect circuits are presented for comparison. The final column shows the number of terms that were finally retained in the suspect set after comparing the terms against the RPDFs and RMPDFs. Note that this diagnosis set is the same for both the methods. The results indicate that our proposed scheme performs much better than the earlier technique in reducing the size of the suspect circuits. Moreover, a substantial amount of memory savings is achieved since we do not need to store any of the suspect circuits. Our technique saves only those terms that remain after RPDF and RMPDF simulation and discards the suspect circuits.

Ckt	Initial (T)	After reduction Hsu (T)	After reduction Our (T)	Final (T)	Final (P/P_F)
s208	115	83	72	47	41/31
s298	258	169	134	47	40/40
s344	94	68	58	25	25/12
s349	30	24	23	10	10/10
s510	85	62	60	20	16/15
s526	166	95	65	21	20/17
s820	44	33	31	14	12/8
s832	840	646	617	214	173/108

Table 4: Diagnostic results

Table 4 also shows the total number of paths in the final diagnosis set (P) and the number of them that pass through at least one of the faulty gates (P_F). This provides us with a measure of the diagnostic resolution provided by the algorithms. Note that the resolution is directly affected by the robust path delay fault coverage of the applied test set. We can see that in most of the cases the obtained resolution is very high.

7 Conclusions

We have presented new paradigms for effect-cause based analysis of path delay faults. Based on these, efficient representations of the suspect faults and fast diagnosis algorithms have been developed that assist in locating the sources of the failures. Results indicate that our proposed methodology performs significantly better than previously demonstrated techniques in terms of space and time.

References

[1] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, New York, N.Y., 1990.

[2] A. Krstic and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*. Kluwer Academic Publishers, Boston, MA, 1998.

[3] D. Dumas, P. Girard, C. Landrault, and S. Pravossoudovitch, "Effectiveness of a variable sampling time strategy for delay fault diagnosis," in *IEEE European Design and Test Conference*, pp. 518–523, 1994.

[4] P. Girard, C. Landrault, and S. Pravossoudovitch, "An advanced diagnostic method for delay faults in combinational faulty circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 6, pp. 277–294, 1995.

[5] K. Boateng, H. Takahashi, and Y. Takamatsu, "Multiple gate delay fault diagnosis using test-pairs for marginal delays," *IEICE Transactions on Information and Systems*, vol. E81-D, pp. 706–714, July 1998.

[6] Y.-C. Hsu and S. K. Gupta, "A new path-oriented effect-cause methodology to diagnose delay failures," in *International Test Conference*, pp. 758–767, 1998.

[7] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits," in *International Conference on Computer-Aided Design*, pp. 694–703, Nov 1985.

[8] K.-T. Cheng, A. Krstic, and H.-C. Chen, "Generation of high quality tests for robustly untestable path delay faults," *IEEE Transactions on Computers*, vol. 45, pp. 1379–1392, Dec 1996.

[9] Y.-C. Hsu and S. K. Gupta, "A new path-oriented effect-cause methodology to diagnose delay failures," Tech. Rep. CENG 98-07, University of Southern California, 1998.