Maximizing Impossibilities for Untestable Fault Identification

Michael S. Hsiao <*mhsiao@vt.edu*>
The Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA

Abstract

This paper presents a new fault-independent method for maximizing local conflicting value assignments for the purpose of untestable faults identification. The technique first computes a large number of logic implications across multiple time-frames and stores them in an implication graph. Then, by maximizing conflicting scenarios in the circuit, the algorithm identifies a large number of untestable faults that require such impossibilities. The proposed approach identifies impossible combinations locally around each Boolean gate in the circuit, and its complexity is thus linear in the number of nodes, resulting in short execution times. Experimental results for both combinational and sequential benchmark circuits showed that many more untestable faults can be identified with this approach efficiently.

1 Introduction

Current automatic test pattern generators (ATPGs) spend a lot of computational effort on identifying untestable faults in both combinational and sequential circuits. These faults are those that no vector sequence is able to detect; they are either unexcitable, unpropagatable, or both, due to reconvergent fanouts in the circuit. For such faults, the excitation/propagation criterion requires conflicting value assignments in the circuit. For sequential circuits, conflicting values may also include unreachable state assignments. Given enough time and backtrack limits, deterministic ATPGs can identify all untestable faults via exhaustive search. However, this may be prohibitively expensive.

To alleviate this problem, algorithms that do not rely on traditional branch-and-bound search technique has been proposed. FIRE [2] identifies a set of untestable faults that require conflicting value assignments in a circuit for detection. It considers only a subset of conflicting value assignments: conflicting values on the same line, since logic 0 and logic 1 assignments on the same signal is a special case of value conflict. It is extended in [3] for sequential circuits. Because the effectiveness of the FIRE algorithm depends on the completeness of the implications learned, it is critical to have a large set of implications. A number of approaches have been proposed for implication computation. A 16-value logic algebra and reduction list method were used in [5] to determine node assignments. Transitive closure procedure on implication graph was proposed in [1] to identify indirect implications. A complete learning algorithm is the recursive learning [4]. However, the depth of recursion must be kept low to keep the computation costs within reasonable bounds. Extended backward implication [6] was introduced that captures additional nontrivial implications in the circuit. More recently, multinode implications [7] have been proposed that extended the basic FIRE algorithm to consider node pairs and identified additional untestable faults. Finally, a compact implication graph has been proposed for sequential circuits that spans multiple time-frames without suffering from memory blow-up [8].

In this work, we aim at quickly identifying conflicting value assignments in the circuit based on implications. Then, faults that require such conflicting assignments for detection are identified. Note that no fault is specifically targeted, thus it is a fault-independent search process. We limit the conflicts by considering only local value assignments for each logic gate. In doing so, the computational complexity of the algorithm is linear with respect to the size of the circuit. Experimental results show that many more untestable faults for both combinational and sequential benchmark circuits can be identified with the proposed approach.

The rest of the paper is organized as follows. Section 2 gives the background for static implications and the FIRE algorithm for identifying untestable faults. Section 3 details the proposed algorithm for maximizing conflicting value assignments to compute untestable faults. Section 4 reports experimental results, and Section 5 concludes the paper.

2 Preliminaries

2.1 Static implications

Static logic implication (also called static learning) is a procedure which performs implications on both value assignments (0 and 1) for all nodes in the given cir-

*This research was supported in part by the National Science Foundation under contract 0196470

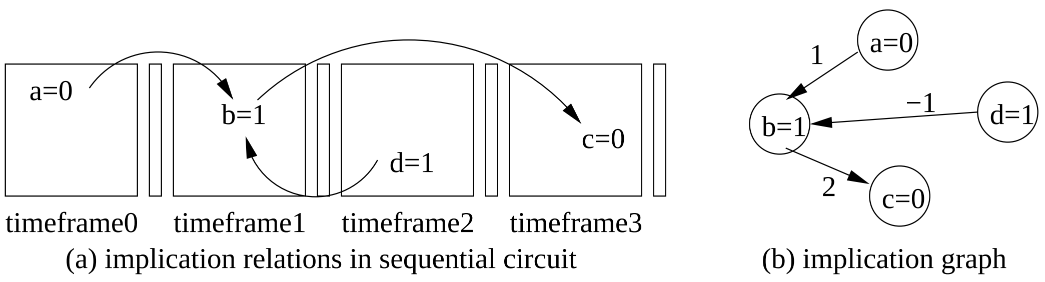

Figure 1: Implication Graph Example

cuit. Direct implications can be easily learned; indirect implications, on the other hand, involve extensive use of contrapositive law and transitive law. Discovery of large numbers of indirect implications can have tremendous benefit not only in untestable fault identification, but also in ATPG, multi-level logic optimization, logic verification, etc.

We will use the notation $[a, v]$ to indicate logic value v assigned to node a in the current time-frame 0; the '0' is implicit and is omitted in the notation. $[b, u, t]$ indicates value u assigned to node b in time-frame t (for sequential circuits), a/v to indicate the fault "line a stuck-at v", and $impl[a, v]$ to denote the implication set of assigning logic value v to node a in the current time-frame of the circuit.

The implications are stored in a directed graph (called implication graph), where each node corresponds to a circuit node assignment $[node, value]$, and each directed edge denotes an implication. Since we consider sequential circuits, some edge may have a non-zero integer weight associated with it, indicating the time frame to which the implication spans; this is similar to the implication graph described in [8]. Within the combinational portion of the circuit, all implication edges will have zero edge weights. The non-zero edge weights come in at the flip-flop boundaries only.

In general, implications can go beyond the current time frame, as shown in Figure 1. Part (a) of this figure illustrates an iterative logic array expansion of a sequential circuit, where four nodes have implication relations as shown (e.g., $[a, 0] \rightarrow [b, 1, 1]$, which is node $b = 1$ in the next time frame). Figure 1(b) shows the corresponding implication graph for the four nodes. The transitive law is also reflected by the implication graph. For instance, since $[a, 0] \rightarrow [b, 1, 1]$, and $[b, 1] \rightarrow [c, 0, 2]$, by transitive law, we obtain $[a, 0] \rightarrow [c, 0, 3]$. Note that during transitive propagation, the time-frame value is summed across the directed edges.

To illustrate where the non-edge edge weights arise around flip-flops, we will use a simple sequential circuit with its implication graph (Figure 2). In this example, we can see from the implication graph that $[b, 1]$ $\rightarrow$ $[c, 1, -1]$ directly, and that $[b, 1]$ indirectly implies

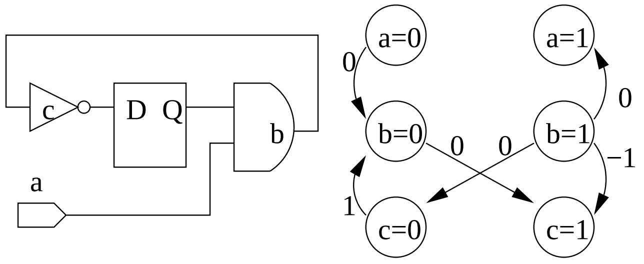

Figure 2: Sequential Implication Graph

$[c, 1, 1]$ via the nodes $c = 0$ and $b = 0$.

Note that there may exist *infinite* cycles, one such example is $[x, 1] \rightarrow [y, 1, 1]$ and $[y, 1] \rightarrow [x, 1, 1]$. In this scenario $[x, 1]$ will imply $\{[x, 1, 1], [x, 1, 2], [x, 1, 3], ...\}$. In order to prevent infinite loops while traversing the implication graph, a time-frame limit is used, to allow any implication to imply a maximum distance from the current time frame.

The contrapositive law states that if $[a, u] \rightarrow [b, v, t]$, then $[b, \bar{v}] \rightarrow [a, \bar{u}, -t]$. Using the example in Figure 1, since $impl[a, 0] = \{[b, 1, 1], [c, 0, 2]\}$, using the contrapositive law we obtain: $[b, 0] \rightarrow [a, 1, -1]$ and $[c, 1] \rightarrow [a, 1, -2]$. These contrapositive edges can also be added to the graph, if they do not already exist.

2.2 Untestable fault identification

Untestable faults are those faults that are unexcitable, unpropagatable, or both. A method described in the FIRE algorithm [2] is used to identify these untestable faults without performing ATPG. A fault is untestable if it requires conflicting values on some signals as a necessary condition to be detected. Specifically, FIRE considers only a special case of conflict value assignment: conflict on the same circuit signal.

In the single-line-conflict case, FIRE computes two sets of faults in the following manner with respect to a given signal n:
set_0 = the set of faults that require n *at* 0 as a necessary condition for excitation or propagation.
set_1 = the set of faults that require n *at* 1 as a necessary condition for excitation or propagation.

Then, the set of untestable faults is simply $\{set_0 \bigcap set_1\}$, which are the faults that require both $n = 0$ and $n = 1$ simultaneously for detection. To com-

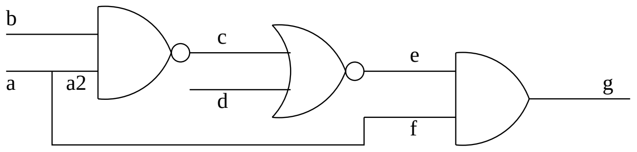

Figure 3: Identifying Unt. Faults With Implications

pute set_v, we first derive $impl[n, \bar{v}]$ from the implication graph and derive all unexcitable and unpropagatable faults due to the value assignments in $impl[n, \bar{v}]$.

We will use a simple combinational circuit fragment illustrated in Figure 3 to explain how these sets are obtained. (The same algorithm works for sequential circuits as well.) Computing the implication set for $(a = 0)$ using the implication graph, we get $impl[a, 0] = \{[a, 0], [a2, 0], [c, 1], [e, 0], [f, 0], [g, 0]\}$, it is clear that in order to excite any of the following faults, $a/0, a2/0, c/1, e/0, f/0, g/0$, we would require $a = 1$ as a necessary assignment. Now let us compute the faults that cannot be propagated due to $a = 0$. All Faults on lines b, d, e, and f require lines a, c, e, and f to have non-controlling values as necessary condition for propagation. In other words, setting a to 0 prevents the faults on lines d, e, c, and b to propagate to gate g. Following through this analysis, faults b/0, b/1, d/0, d/1, e/0, e/1, f/0, and f/1 require $a = 1$ as a necessary condition for propagation. Considering both excitation and propagation criteria, we obtain $set_1 = \{a/0, a2/0, b/0, b/1, c/1, d/0, d/1, e/0, e/1, f/0, f/1, g/0\}$. Likewise, we can compute set_0 for node a: $set_0 = \{a/1, a2/1, f/1\}$. The intersection of set_0 and set_1 is $\{f/1\}$, indicating that in order for fault $f/1$ to be detected, node a needs to take on both logic values 0 and 1; this is a conflicting/impossible assignment for node a, making fault $f/1$ untestable.

In general, the FIRE algorithm works for a general set of conflicting value assignments $\{v_1, v_2, ..., v_n\}$ on a set of signals $\{l_1, l_2, ..., l_n\}$. A set, set_i, that contains all faults requiring $l_i = v_i$ is computed. Then, the intersection of all sets set_i will contain the untestable faults that require this set of conflicting value assignments.

3 Maximizing Impossibilities

Finding trivial conflicting value assignments from the implication graph is easy, but it will not help to find more untestable faults because the single-line conflict approach has already taken these conflicts into account. For instance, if the implication set $impl[x, 0]$ includes $[y, 1]$, then the pair $\{[x, 0], [y, 0]\}$ naturally forms an conflicting value assignment. However, in the original FIRE algorithm, if we compute set_0 and set_1 to be the faults that require $x = 0$ and $x = 1$, respectively, then set_1 already contains all the faults that require $y = 0$ to be testable. This can be explained as follows: Since the set of faults that require $y = 0$ are obtained as those undetectable due to the value assignments in $impl[y, 1]$, and since $impl[y, 1] \subseteq impl[x, 0]$, the set of faults requiring $x = 1$, set_1, (i.e., undetectable computed from $impl[x, 0]$), must contain every fault that requires $y = 0$ as well.

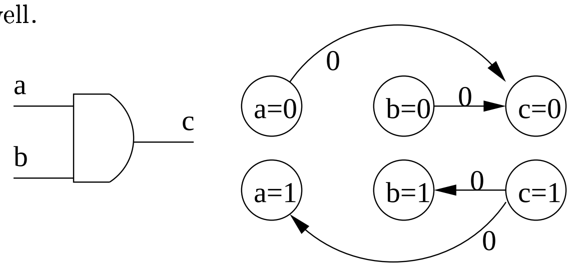

Figure 4: AND Gate Example

Our method is a new technique aimed at increasing the nontrivial impossible combinations, in hope of finding more untestable faults that require such conflicts. These untestable faults could not be identified by the original single-line-conflict method. Finding arbitrary value conflicts in the circuit can be computationally expensive, thus we limit the search for conflicting value assignments to those associated with a single Boolean gate, making our algorithm of $O(n)$ complexity, where n is the number of gates in the circuit.

Let us consider the AND gate and its implication graph, shown in Figure 4. When considering single-conflicting-line FIRE algorithm, there are three such cases for the AND gate: $\{a = 0, a = 1\}$, $\{b = 0, b = 1\}$, and $\{c = 0, c = 1\}$. By traversing the implication graph, the impossible value combination imposed by the conflicting line assignment $\{a = 0, a = 1\}$ includes the set $\{a = 0, a = 1, c = 0\}$. Similarly, we can obtain the sets of impossible value combination for conflicting line assignments $\{b = 0, b = 1\}$ and {c=0, c=1} to be $\{b = 0, b = 1, c = 0\}$ and $\{c = 0, c = 1, a = 1, b = 1\}$, respectively.

Note that there are other sets of impossible value combinations not covered by these three single-line conflicts. Not all remaining conflicting combinations are non-trivial. For example, consider the conflicting scenario $\{a = 0, c = 1\}$. This is a trivial value conflict because because $[a = 0] \rightarrow [c = 0]$ and $[c = 1] \rightarrow [a = 1]$. Therefore, $\{a = 0, c = 1\}$ is already covered by the single-line-conflicts $\{a = 0, a = 1\}$ and $\{c = 0, c = 1\}$.

There exists a conflicting assignment that is not covered by any single-line conflicts: $\{a = 1, b = 1, c = 0\}$. In order to compute the corresponding impossible value assignment set, we need to compute the following implications: $impl[a = 0]$, $impl[b = 0]$, and $impl[c = 1]$. By traversing the implication edges in the graph, we get the impossible value assignment set $\{a = 0, b = 0, c = 0, c = 1, a = 1, b = 1\}$. This set has not been covered in any of the previous impossible value assignment sets, and thus is new and may be used for obtaining additional untestable faults that require this conflict.

Impossible value combinations for other gate primitives and/or gates with different number of inputs can be derived in a similar manner.

3.1 Overall algorithm

Since our aim is to identify as many conflicting value assignments as possible, which leads to untestable faults, the new approach by maximizing impossibilities is performed on top of the single-conflict-line FIRE algorithm. Described below is the overall algorithm.

```
1.  Construct implication graph (learn any additional
        implications via extended backward impl, etc.)
2.  For each line $l$ in circuit
3.      Identify all untestable faults using the
            single-line-conflict FIRE algorithm
/* maximizing impossibilities algorithm */
4.  For each gate $g$ in circuit
5.      SIV = set of impossible value combinations
6.              not yet covered for gate $g$
7.      $i = 0$
8.      for each value assignment $(a = v)$ in SIV
9.          $set_i$ = faults requiring $a = \bar{v}$ to be detectable
10.         $i = i + 1$
11.     untestable_faults = untestable_faults $\bigcup(\bigcap_{\forall i} set_i)$
```

The implication graph is first constructed, with indirect implications computed and added to the graph. Then, a single-conflict-line FIRE algorithm is performed (line #3). Next, for each set of conflicting values not covered by the single-conflicting-line for each *gate*, we compute the set of faults untestable due to such conflicts. Because our algorithm on maximizing value impossibilities is performed once for each gate, the complexity is kept to be linear in the size of the circuit. For large circuits, the number of additional untestable faults can be significant.

4 Experimental Results

The proposed algorithm was written in C++ and experiments were conducted for ISCAS85 and ISCAS89 benchmark circuits on a 1.7 GHz Pentium-4 workstation with 256 MB of memory running the Linux operating system. The results are reported in Table 1. For each circuit, the number of time-frames used and the number of implications derived are first given, followed by the number of constants found and the time for constructing the implication graph. Next, untestable faults identified by the original single-line-conflict FIRE and corresponding execution time are reported. Then, the additional untestable faults captured by maximizing impossible value combinations and the extra execution time needed are given. Finally, the total number of untestable faults found and total execution time are listed. Note that the number of untestable faults identified by original single-line-conflict FIRE is similar to those reported in [8], which are the *highest* reported for many circuits until now.

From this table, many additional untestable faults were identified by the new maximizing impossibilities approach for the benchmark circuits. For example, in circuit c2670, original single-line-conflict FIRE algorithm identified 39 untestable faults, and the new approach identified 36 additional untestable faults, making a total of 75 faults identified as untestable, all with only 0.5 additional second. Likewise, in circuit c5315, 38 additional faults were identified by the new approach, in only 2.5 additional seconds. Similar results were obtained for sequential circuits. The number of additional untestable faults identified can be significant for large circuits, since these are the ones that ATPGs spend most of the time on. For example, in s5378, 88 more untestable faults were identified, making a total of 884 untestable faults, in only 120 seconds (including computation of implications). In s9234.1, the single-line-conflict FIRE method found 195 untestable faults, while the new approach identified 165 more, totally 360, all in less than 2 minutes. Likewise, in the other large sequential circuits, many additional untestable faults are identified by our new approach than by the single-line-conflict technique alone. In s35932, since the single-line-conflict method identified all untestable faults, no untestable faults remained that could be identified. Overall, although the computational effort for our method is more than the single-line-conflict approach, it scales linearly with circuit size, making this approach attractive.

The number of time frames allowed has an impact on execution time. More time frames require exponential cost in computing implications; thus, it is important to keep the number of time-frames small. Nevertheless, for most circuits, small number of time frames is usually enough for our technique to identify a large number of untestable faults, additional time-frames generally yielded no additional untestable faults for those reported in Table 1.

5 Conclusion

A new method for identifying untestable faults by maximizing impossible value combinations has been presented. Impossible value combinations around each gate in the circuit are discovered which were not considered by the single-line-conflict approach. Many more untestable faults for both combinational and sequential benchmark circuits were identified as a result. Because the impossible combinations are limited to individual gates, the complexity for the proposed algorithm

Table 1: Untestable Faults Identified

Ckt	TF	#	#	Impl	Single-Line-Confl		Max Imposs		Total	
		Impl	Const	Time	Unt	Time	Unt	Time	Unt	Time
c432	0	2,400	0	0.1	0	0.1	+2	0.1	2	0.3
c1908	0	45,564	0	0.6	4	0.2	+3	0.5	7	1.3
c2670	0	59,259	11	0.6	39	0.2	+36	0.5	75	1.3
c3540	0	310,102	1	2.3	105	2.3	+26	6.7	131	11.0
c5315	0	102,158	1	1.6	20	0.8	+38	2.5	58	4.9
c7552	0	298,050	4	12.8	42	1.5	+22	4.4	64	18.7
s298	4	7,326	3	0.6	3	0.1	+3	0.1	6	0.8
s344	2	11,089	5	0.1	3	0.1	+1	0.1	4	0.2
s349	2	11,502	6	0.1	5	0.1	+1	0.1	6	0.2
s386	1	26,790	3	0.3	60	0.1	+3	0.1	63	0.4
s400	1	13,629	1	0.5	8	0.1	+2	0.1	10	0.7
s444	1	16,383	2	0.1	16	0.1	+2	0.1	18	0.3
s526	4	30,811	1	5.4	2	0.3	+9	0.9	11	6.6
s526n	4	31,381	1	5.2	2	0.3	+8	0.9	10	6.5
s713	1	39,133	16	0.2	32	0.1	+6	0.1	38	0.3
s832	1	55,722	0	7.7	0	0.2	+4	0.6	4	8.4
s1196	1	71,744	0	0.6	0	0.5	+1	1.1	1	2.2
s1238	1	73,582	0	2.4	9	0.6	+12	1.3	21	4.2
s1423	1	59,442	0	0.3	9	0.3	+5	0.7	14	1.3
s5378	2	2,835,398	404	92.9	796	8.8	+88	18.2	884	120.0
s9234.1	1	1,808,662	22	27.6	195	24.5	+165	61.8	360	114.4
s13207.1	2	7,355,079	296	1655.4	376	35.7	+75	100.9	451	1795.2
s15850.1	2	5,327,682	76	141.4	317	157.0	+51	411.8	368	570.7
s35932	1	9,937,478	0	414.4	3984	81.9	+0*	709.8	3984	1210.0
s38417	2	19,522,595	102	506.4	328	1689.9	+138	4761.5	466	6456.2
s38584	1	34,227,650	630	4194.5	1460	597.7	+463	1749.8	1923	6553.5
s38584.1	1	34,206,109	403	5584.5	1490	516.4	+163	2150.1	1653	8265.6

TF: # of time frames used   Time measured in seconds

*: single-line-conflict already found all 3984 untestable faults in circuit

is maintained to be linear in terms of circuit size, keeping the overal computational effort low.

References

[1] S. T. Chakradhar and V. D. Agrawal, "A transitive closure algorithm for test generation," *IEEE Trans. Computer-Aided Design*, June 1993, pp. 1015 -1028.

[2] M. A. Iyer and M. Abramovici, "FIRE: a fault independent combinational redundancy identification algorithm," *IEEE Trans. VLSI systems*, June 1996, pp. 295-301.

[3] M. A. Iyer, D. Long, and M. Abramovici, "Identifying sequential redundancies without search," *Proc. Design Automation Conf.*, pp. 457-462, 1996.

[4] W. Kunz and D. K. Pradhan, "Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization," *IEEE Trans. on CAD*, Sept 1994, pp. 1149-1158.

[5] J. Rajski and H. Cox, "A method to calculate necessary assignments in ATPG," *Proc. Int'l. Test Conf.* 1990, pp. 25-34.

[6] J. Zhao, M. Rudnick, and J. Patel, "Static logic implication with application to fast redundancy identification," *Proc. VLSI Test Symp.*, 1997, pp. 288-293.

[7] K. Gulrajani and M. S. Hsiao, "Multi-node static logic implications for redundancy identification," *Proc. Design, Automation, and Test in Europe Conf.*, 2000, pp. 729-733.

[8] J. Zhao, J. A. Newquist, and J. Patel, "A graph traversal based framework for sequential logic implication with an application to c-cycle redundancy identification," *Proc. VLSI Design Conf.*, 2001, pp. 163-169.