MULTI-NODE STATIC LOGIC IMPLICATIONS FOR REDUNDANCY IDENTIFICATION

Kabir Gulrajani[†] *and Michael S. Hsiao*[‡]
[†]Intel Corporation, Dupont, WA
[‡]Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ

Abstract

This paper presents a method for redundancy identification (RID) using multi-node logic implications. The algorithm discovers a large number of direct and indirect implications by extending single node implications [7] to multiple nodes. The large number of implications found by multi-node implication method introduces a new redundancy identification technique. Our approach uses an effective node-pair selection method which is $O(n)$ in the number of nodes to reduce execution time, and it can be used as an efficient preprocessing phase for test generation. Application of these multi-node static logic implications uncovered more redundancies in ISCAS85 combinational circuits than previous single-node methods without excessive computational effort.

1 Introduction

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 circuit. Direct implications can be easily learned during automatic test pattern generation (ATPG) process; indirect implications, on the other hand, involve extensive use of contrapositive law, transitive law, and the extended backward implication method [7]. Discovery of large numbers of indirect implications can have tremendous benefit in redundancy identification, ATPG, multi-level logic optimization and logic verification.

A number of previous approaches have dealt with implication procedures. A 16-value logic algebra and reduction list method were used in [6] to determine node assignments. [1] [2] used a transitive closure on direct implications to identify indirect implications. Due to the NP-hard nature of the problem for finding all the implications for a given set of nodes, the practicality of such complete algorithms is limited. Another complete learning algorithm is the recursive learning proposed in [5]. For this method, the depth of recursion must be kept low to keep the computation costs within reasonable bounds. [7] introduced the extended backward implication method, which captures some non-trivial implications between nodes.

In this work, we present a linear-order extension of single-node learning algorithm for logic implications. Besides using set algebra to find a very large number of direct and indirect implications within short time limits, we extend the algorithm to multi-nodes. We limit the number of nodes by considering only the most-likely candidate pairs that will produce useful indirect implications. In doing so, our complexity remains linear instead of quadratic in the number of nodes in the circuit. These multi-node implication sets are then used for identifying more redundancies. Our RID procedure is based on the FIRE algorithm [4, 3], except that it is modified for multiple nodes. By using this technique, we obtained a very large number of hard-to-find implications and more redundancies than those presented in previous implication-based RID's.

The remainder of the paper is organized as follows. Section 2 presents the basic concept for single-node implications and their use in redundancy identification. Section 3 describes how results for multi-node implications can be used to identify redundant faults in the circuit. Section 4 gives experimental results obtained on the ISCAS85 combinational benchmark circuits, and Section 5 concludes the paper.

2 Preliminaries

In deductive fault simulation, sets of faults are are propagated from inputs of a node to its outputs using set operations of intersection, union and difference. In an analogous manner, we propagate sets of implications from inputs to outputs of each node using the similar operations of set intersection, union, and set difference. Figure 1 shows the basic concept for direct

implications for the 2-input OR gate. In this figure, A_0

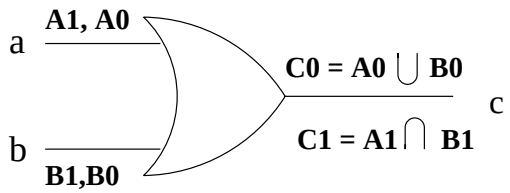

Figure 1: Single Node Implication

represents the set of implications obtained for setting node a to 0 and A_1 is the set of implications obtained by setting node a to 1. Similarly, B_0 and B_1 are sets of implications for setting input b to 0 and 1 respectively. Using direct implication, $c = 0$ implies that both a and b have to be zero; in other words, $C_0 = A_0 \bigcup B_0$. Likewise, $c = 1$ implies that at least one of a and b has to be zero; thus, $C_1 = A_1 \bigcap B_1$. Similar rules can be obtained for other primitive gates such as NOR, AND, NAND and NOT gates.

Before going further, we will use the notations $[a, v]$ to indicate logic value of node a to be v, a/v to indicate line a stuck-at v and, $impl[a, v]$ to denote the implication set of setting a to logic value v.

For indirect implications, all relevant implication sets are propagated, added to, and subtracted, using the following implication laws:

1. Transitive law
2. Contrapositive law
3. Extended Backward implication [7]

We will use Figure 2 to illustrate the transitive and contrapositive laws. In this figure, by direct implica-

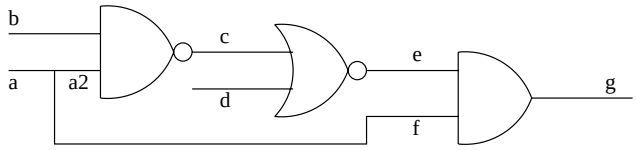

Figure 2: RID Using Implication Sets

tions and propagation due to transitive laws, we get $impl[a, 0] = \{[a, 0], [a2, 0], [c, 1], [e, 0], [f, 0], [g, 0]\}$. Using the contrapositive law we obtain:

1. $[a2, 1] \rightarrow [a, 1]$
2. $[c, 0] \rightarrow [a, 1]$
3. $[e, 1] \rightarrow [a, 1]$
4. $[f, 1] \rightarrow [a, 1]$
5. $[g, 1] \rightarrow [a, 1]$

Implications for other nodes can be computed in a similar manner. Extended Backward implications [7] further increases the number of implications by considering known values at the inputs of given nodes.

2.1 RID using single node implications

Redundant faults are those faults that are unexcitable, unpropagatable, or both. A method described in the FIRE algorithm [4] is used to identify these redundancies without performing ATPG. A fault is redundant if it requires conflicting values on the same line as a necessary condition to be detected.

For each node N in the circuit, let 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 redundant faults is simply $\{set_0 \bigcap set_1\}$.

We will use node a of Figure 2 again to explain how these sets are obtained. Observing the implication set $impl[a, 0] = \{[a, 0], [a2, 0], [c, 1], [e, 0], [f, 0], [g, 0]\}$, 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 consider the faults that require necessary assignments for propagation. All Faults on lines b, d, e, and f require lines a, c, e, and f to have non-controlling values. For instance, 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 assignment 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 $f/1$ redundant.

Our proposed multi-node algorithms (to be discussed in the next section), aside from using multi-node implications, the computation of single-node implications also differ from the one used in [7]. Rather than performing the extended backward implication iteratively, which is costly in time, our algorithms apply the extended backward implication on all nodes after all the other implication laws have been applied. As a result, significant speed up was obtained. However, a few implications may be missed as a consequence. Nevertheless, for most circuits, we were still able to identify all the redundancies.

3 Multi-node implications and RID

This method is a new technique aimed at increasing the implication set size in hope of finding more redun-

dancies that may have been missed by the previous single-node method. We limit the number of nodes in multi-node implication to 2 nodes. Let us consider two nodes, say a and b. We compute the following four implication sets: $impl\{[a,0]\bigcup[b,0]\}$, $impl\{[a,0]\bigcup[b,1]\}$, $impl\{[a,1]\bigcup[b,0]\}$, and $impl\{[a,1]\bigcup[b,1]\}$. If we define

1. set_0 = Untestable faults due to $impl\{[a,0]\bigcup[b,0]\}$,
2. set_1 = Untestable faults due to $impl\{[a,0]\bigcup[b,1]\}$,
3. set_2 = Untestable faults due to $impl\{[a,1]\bigcup[b,0]\}$,
4. set_3 = Untestable faults due to $impl\{[a,1]\bigcup[b,1]\}$,

then $\{set_0 \bigcap set_1 \bigcap set_2 \bigcap set_3\}$ would give us the set of redundant faults due to impossible value combinations on nodes a and b.

One important issue in the multi-node implication method is on selecting the right node-pairs, since performing this procedure on *all* node pairs will result in $O(n^2)$ complexity. We found that most nodes far apart in the circuit graph generally do not yield a large number of new implications. Therefore, we limited the size of node pairs by using the following two selection methods which give large implication sets at a linear cost.

1. $Method$ 1: In this method, we consider only 2-input nodes with at least one of the inputs having a fanout size greater than 1. We make two checks before we proceed for multi-node implications. These two checks are aimed at eliminating node pairs that have a low probability of yielding any new redundancies. Figure 3 explains the selection procedure for this method.

   (a) Case 1: Referring to Figure 3, if the 2-input node under consideration is node C, we see that one of the inputs, node A, reconverges in the circuit through a path that contains node C. Thus, including node B ensures that this path is always excited, which increases the possibility of finding a redundancy in this reconvergent path. Therefore, the two nodes selected for multi-node implications are A and B. In brief, the first check we make for each 2-input node N is to see if one of the input nodes reconverges with N through a path.

   (b) Case 2: If the above check returns no candidate node, we consider the second option where the 2-input node under consideration is a reconvergent node, or one of the nodes in a reconvergent path. These two possibilities are shown in Figure 4. In this figure, the 2-input node under consideration is C. This node C is a reconvergent node in part (a) of the figure, while it (node C) is on a reconvergent path on some other node in part (b) of Figure 4. For both situations, nodes A and B are considered for candidate multi-node implications, because keeping node C excited in either case may lead to obtaining more implications. The second check basically considers those 2-input nodes that lie on a reconvergent path or are reconvergent nodes themselves.

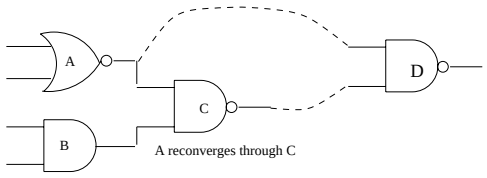

Figure 3: First Check Made For Selecting Node Pairs Using Method I

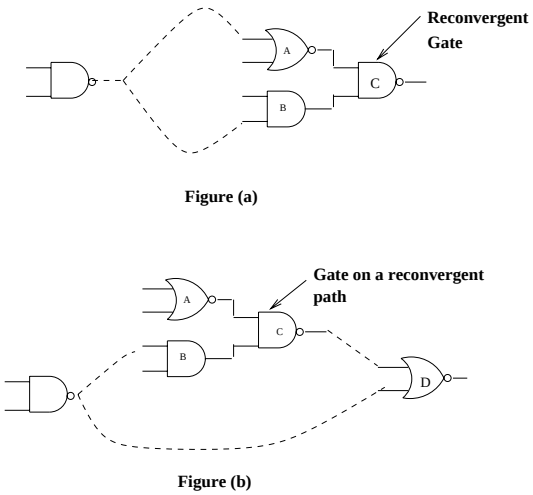

Figure 4: Second Check Made for Selecting Node Pairs Using Method I

In both cases of $Method$ 1, the number of node pairs we have to consider is limited to less than n, where n is the total number of nodes in the circuit graph.

2. $Method$ 2: For each level L in the circuit, we select the node with the maximum number of implications and denote it as node A. The remaining nodes in the same level are checked to see if they have a common predecessor with node A. If there exists a common predecessor, we mark this node to be a potential node partner for node A. Next, we

check all the potential node partners to see if they reconverge with node A in the circuit. If there exists a reconvergent path, these nodes are combined for multi-node RID.

The reason for selecting node A, which has the maximum number of implications, is that it increases the untestable fault list size. Therefore, the probability of set intersection increases when RID procedure is applied. For example, in Figure 5, node A is combined with node B as well as with node C for multi-node RID, since they each share a common predecessor with node A. The complexity of this method is also well below $O(n)$ for the entire circuit.

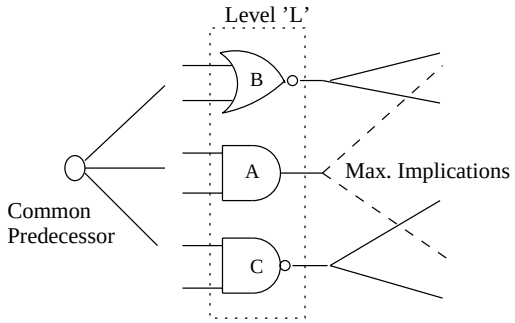

Figure 5: Using Method II for Selecting Node Pairs

Because we limit the number of node pairs during multi-node simulations to only $O(N)$ pairs, we do not incur the higher cost as in recursive learning [5], in which all pairs are considered. Another attribute of our algorithm that reduces the computation costs is the linear complexity in set operations (intersection, union and difference) used to compute implications.

4 Experimental Results

Both the implication and redundancy identification algorithms were computed for ISCAS85 combinational circuit on a UltraSPARC-1 station with 64MB RAM. Results for the number of implications using single-node implications are first shown in Table 1. For each circuit, the number of implications found is first reported, followed by the constant nodes in the circuit that can be deduced from the implications found, and the execution time given in seconds. For all circuits, large numbers of implications were obtained in a few seconds or minutes. For example, in circuit c3540, more than 300 thousand single-node implications were obtained in 329 seconds.

After single-node implications had been computed, they were then used to compute *multi-node* implications, which in turn were used to uncover redundancies in the circuit. Table 2 shows the comparison of the number of redundancies and CPU time for FIRE [4], Simprid [7], and our method. As indicated in Table 2, our multi-node technique almost always found more redundancies, and these redundancies were missed by both FIRE [4] and Simprid algorithm [7]. For instance, in circuit c3540, FIRE identified 93 redundant faults in 11.9 seconds, Simprid identified 105 in 14.7 seconds, and our multi-node method identified 115 redundant faults in 98 seconds. All the additional redundant faults discovered by our approach were *non-trivial* redundant faults, thus they required extra computation effort. However, because we select only $O(n)$ node pairs, we can keep the CPU time to a reasonable level. For circuits c499 and c880, there were no additional redundancies found by our multi-node method than the single-node methods, thus they were not included in the table.

Table 1: Number of Implications Found

Ckt.	Impl	#Const	Time(s)
c17	70	0	0.037
c432	2734	0	0.14
c499	7366	0	0.54
c880	6990	0	0.40
c1355	31718	0	4.16
c1908	47214	0	4.80
c2670	61560	11	18.60
c3540	309364	1	329.0
c5315	106758	1	24.80
c6288	30974	17	5.90
c7552	304814	3	259.80

Table 2: Redundancies Found

Ckt.	FIRE [4]		Simprid [7]		Multi-Node	
	Red	Time	Red	Time	Red	Time
c432	0	1.8	0	2.2	1	0.4
c2670	29	1.5	39	2.5	44	12.4
c3540	93	11.9	105	14.7	115	98.0
c6288	33	1.3	34	2.7	34	55.1
c7552	30	47.0	42	15.6	44	98.0

Finally, Table 3 shows the difference in the effectiveness between the two proposed multi-node methods described in the section 3. In this table, we report the number of redundant faults identified and the execution times by each multi-node method *without* first calling the single-node RID procedure. It is clear from Table 3 that $Method\ I$ is more effective than $Method\ II$, but requiring extra computation effort. For example, $Method\ I$ identified 101 redundant faults for c3540

while *Method II* identified only 73. This is because the number of node pairs selected using *Method II* is *fewer* than *Method I*. Furthermore, the sets of redundancies identified by the two methods are also different. Sometimes a method will uncover a redundancy missed by the other. However, in most cases, the redundancies from *Method I* covered all redundancies found by *Method II*.

Table 3: Comparison Between Method I and II

Ckt.	Method I		Method II	
	Time (s)	Red	Time (s)	Red
c432	0.3	1	0.08	0
c2670	29.0	38	19.0	20
c3540	53.0	101	32.0	73
c6288	42.0	34	8.0	17
c7552	56.0	15	26.0	9

5 Conclusion

A new method for finding combinational redundancies using multi-node implications has been presented. More implications as well as redundant faults were captured with this approach while keeping a linear complexity in the number of node-pairs to consider. Since this method of finding redundancies is based on static learning, it does not rely on any other procedure and can interface with existing applications (such as ATPG) readily. As true in other static learning applications, most of the execution time is spent on computing a small number of non-trivial indirect implications. Future work includes other methods for selecting node pairs, as well as application to sequential redundancies and state reachability.

References

[1] S. T. Chakradhar and V. D. Agrawal. A Transitive Closure Based Algorithm for Test Generation. In *Proc. of the Design Automation Conf.* ACM/IEEE, June 1991.

[2] S. T. Chakradhar and V. D. Agrawal. A Transitive Closure Algorithm for Test Generation. In *Trans. Computer-Aided Design*. IEEE, June 1993.

[3] M. A. Iyer and M. Abramovici. Low Cost Redundancy Identification for Combinational Circuits. *Proc. Int. Conference VLSI Design*, June 1994.

[4] M. A. Iyer and M. Abramovici. FIRE: A Fault independent Combinational Redundancy Identification Algorithm. *Trans. VLSI systems*, June 1996.

[5] W. Kunz and D. K. Pradhan. Recursive learning: An Attractive Alternative to the Decision Tree for Test Generation in Digital Circuits. *Trans. on CAD*, May 1993.

[6] J. Rajski and H. Cox. A Method to Calculate Necessary Assignments in ATPG. In *Proc. of the Int'l. Test Conf.* IEEE, September 1990.

[7] Zhao, M. Rudnick, and Janak Patel. Static Logic Implication with Application to Fast Redundancy Identification. *VLSI Test Symposium*, April 1997.