added to update the original data set and second, another round of the PWL fitting using updated data set is performed. It has been found that added data points in this extra fitting is mostly less than 100 in our experiment. Therefore, the suggested model refinement does not incur any significant complexity penalty. Table VI shows the fitting errors on the extended data set. Compared with Table III, both mean and maximum error over extended data set do not change appreciably, implying that the model quality will not exhibit sharp degradation when used in real circuit optimizations.

V. Conclusion

This paper presented an alternative method for fitting a convex PWL function to a given set of data. The method iteratively solved linear optimization problems by judiciously increasing the number of planes in the PWL function at each iteration. The proposed heuristic is simple, but experimental results applied to the FET models in 90-nm CMOS technology indicate that the method works well in practice, and the computational complexity is not excessive. Therefore, the proposed technique can be used as an efficient modeling framework for circuit optimization via GP.

References

[1] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimiz. Eng.*, vol. 8, no. 1, pp. 67–127, Mar. 2007.

[2] G. G. E. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. IEEE*, vol. 88, no. 12, pp. 1825–1854, Dec. 2000.

[3] E. Hjalmarson, "Studies on design automation of analog circuits: The design flow," M.S. thesis, Linköping Stud. Sci. Technol., Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, Dec. 2003.

[4] M. Hershenson, S. Boyd, and T. Lee, "Optimal design of a CMOS opamp via geometric programming," *IEEE Trans. Comput.-Aided Design*, vol. 20, no. 1, pp. 1–21, Jan. 2001.

[5] D. M. Colleran, C. Portmann, A. Hassibi, C. Crusius, S. S. Mohan, S. Boyd, T. H. Lee, and M. del Mar Hershenson, "Optimization of phase-locked loop circuits via geometric programming," in *Proc. CICC*, Nov. 2003, pp. 377–380.

[6] Y. Xu, K.-L. Hsiung, X. Li, L. Pileggi, and S. Boyd, "Regular analog/RF integrated circuits design using optimization with recourse including ellipsoidal uncertainty," *IEEE Trans. Comput.-Aided Design Integr. Circuits Sys.*, vol. 28, no. 5, pp. 623–637, May 2009.

[7] S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz, "Digital circuit optimization via geometric programming," *Oper. Res.*, vol. 53, no. 6, pp. 899–932, 2005.

[8] J. Singh, Z.-Q. Luo, and S. Sapatnekar, "A geometric programming-based worst case gate sizing method incorporating spatial correlation," *IEEE Trans. Computer-Aided Design Integr. Circuits Sys.*, vol. 27, no. 2, pp. 295–308, Feb. 2008.

[9] M. Hershensons, "Design of pipeline analog-to-digital converters via geometric programming," in *Proc. ICCAD*, Nov. 2002, pp. 317–324.

[10] T. McConaghy and G. Gielen, "Template-free symbolic performance modeling of analog circuits via canonical-form functions and genetic programming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Sys.*, vol. 28, no. 8, pp. 1162–1175, Aug. 2009.

[11] J. Kim, J. Lee, L. Vandenberghe, and C.-K. K. Yang, "Error reduction methods in geometric-programming based analog circuit design optimization," in *Proc. ICCAD*, Nov. 2004, pp. 863–870.

[12] W. Daems, G. Gielen, and W. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Sys.*, vol. 22, no. 5, pp. 517–534, May 2003.

[13] T. Eeckelaert, W. Daems, G. Gielen, and W. Sansen, "Generalized posynomial performance modeling," in *Proc. Conf. DATE*, vol. 1. Nov. 2003, pp. 250–255.

[14] V. Aggarwal and U.-M. O'Reilly, "Simulation-based reusable posynomial models for MOS transistor parameters," in *Proc. Conf. DATE*, Apr. 2007, pp. 69–74.

[15] A. Magnani and S. Boyd, "Convex piecewise-linear fitting," *Optimiz. Eng.*, vol. 10, no. 1, pp. 1–17, Mar. 2009.

[16] S. Boyd and L. Vendenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2003.

[17] *MOSEK* [Online]. Available: http://www.mosek.com

[18] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Statist.*, vol. 19, no. 1, pp. 1–67, Mar. 1991.

[19] G. Jekabsons. *ARESLab: Adaptive Regression Splines Toolbox for MATLAB* [Online]. Available: http://www.cs.rtu.lv/jekabsons/

Fast Node Merging with Don't Cares Using Logic Implications

Yung-Chih Chen and Chun-Yao Wang, *Member, IEEE*

***Abstract*—Node merging is a popular and effective logic restructuring technique that has recently been applied to minimize logic circuits. However, in the previous satisfiability (SAT)-based methods, the search for node mergers required trial-and-error validity checking of a potentially large set of candidate mergers. Here, we propose a new method, which directly identifies node mergers using logic implications without any SAT solving calls. Although the efficiency benefits of the method come at the expense of quality, we further engage the redundancy removal and the wire replacement techniques to enhance its quality. The experimental results show that the proposed optimization method achieves approximately 46 times the speedup while possessing a competitive capability of circuit minimization compared to the state-of-the-art method.**

***Index Terms*—Algorithms, circuit optimization, node merging, observability don't cares (ODCs).**

I. Introduction

Node merging is a popular and efficient logic restructuring technique. It replaces a node with another node when the two are functionally equivalent or their functional differences are never observed at any primary output (PO). A major application of the technique is to minimize logic circuits. When two nodes are merged, one can be removed from the circuit, and this merger may cause other redundancies in the circuit such that the resultant circuit is minimized.

In previous works [4], [7], [11], [13], the methods for finding node mergers are satisfiability (SAT)-based methods. They perform random simulation, collect candidate mergers, and then use a SAT solver to check the validity of the candidate mergers. The work in [7] collects the candidate mergers that have the same simulation values and finds only functionally equivalent node mergers. Extending the work in [7], the method in [13] identifies additional node mergers whose functional differences are unobservable at any PO by considering observability don't cares (ODCs) in the process of collecting candidate mergers. However, because full observability computation is very time-consuming, the method sets

Manuscript received August 13, 2009; revised February 9, 2010 and May 5, 2010; accepted July 1, 2010. Date of current version October 20, 2010. This work was supported in part by the National Science Council of R.O.C., under Grants NSC 98-2220-E-007-015 and NSC 98-2220-E-007-023. This paper was recommended by Associate Editor S. Nowick.

The authors are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, R.O.C. (e-mail: ycchen@cs.nthu.edu.tw; wcyao@cs.nthu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2010.2058510

a k-bounded depth to extract local ODCs. With larger values of k, the method can find more node mergers but spends more central processing unit (CPU) time.

The work in [11] proposed a fast simulator that computes global but approximate ODCs from any depth. Although this method cannot identify the complete set of node mergers, it finds an average of 25% more node mergers compared to the local ODC-based method with $k = 5$ [13]. Furthermore, the work in [4] extended the node-merging technique for sequential circuit optimization by considering sequential ODCs.

Although both the works in [13] and [11] proposed methods to decrease the complexity of observability computation, they cannot avoid a potentially large number of SAT solving calls. Thus, in this paper, we propose a non-SAT-based method for detecting node mergers. Given a target node, we compute the mandatory assignments for the stuck-at 0 and 1 fault tests on it using logic implications. Then, we can derive its substitute nodes directly from the mandatory assignments without trial-and-error checking. To achieve high efficiency, we do not compute all mandatory assignments due to the exponential time complexity, and thus, spend some quality. For circuit size reduction, we extend the method with three techniques—wire replacement, redundancy removal, and mandatory assignment reuse—that can enhance its performance.

We conduct experiments on a set of IWLS 2005 benchmarks [14]. The experimental results show that the proposed method can quickly complement the local ODC-based method with $k = 5$ by finding additional node mergers. Additionally, for circuit size reduction, the proposed method has a speedup of 46 times for overall benchmarks while possessing a competitive capability compared to the state-of-the-art method [11].

The rest of this paper is organized as follows. Section II uses an example to demonstrate ODC-based node merging. It also reviews the related concepts in very-large-scale integration (VLSI) testing used in this paper. Sections III and IV present the proposed method for finding node mergers and its application for circuit minimization. Finally, the experimental results and conclusion are presented in Sections V and VI.

II. Preliminaries

A. Example

We use an example in Fig. 1 to demonstrate ODC-based node merging. The circuit in Fig. 1(a) is presented by using an and-inverter graph (AIG). Here, a, b, c, and d are primary inputs (PIs). v_1–v_5 are 2-input AND gates. Their connectivities are presented by directed edges. A dot marked on an edge indicates that an inverter INV is in between two nodes. In this circuit, v_1 and v_3 are not functionally equivalent. Merging them potentially affects the overall functionality. However, their values only differ when $d = 1$ and $b = c$. Since $b = c$ further implies $v_2 = 0$, which is an input-controlling value of v_5, the different values of v_3 with respect to v_1 are prevented from being observed. Thus, v_3 can be correctly replaced with v_1. The resultant circuit is shown in Fig. 1(b).

In this paper, a node to be replaced is referred to as a *target node* and a node that can correctly replace a target node is called a *substitute node* of the target node.

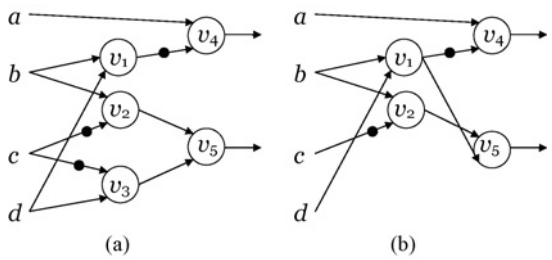

Fig. 1. Example of ODC-based node merging. (a) Original circuit. (b) Resultant circuit of replacing v_3 with v_1.

For ease of discussion, we only consider circuits presented as AIGs. Circuits having complex gates can also be handled by first transforming them into AIGs.

B. Background

An input of a gate g has an *input-controlling value* of g if this value determines the output value of g regardless of the other inputs. The inverse of the input-controlling value is called the *input-noncontrolling value*, e.g., the input-controlling value of an AND gate is 0 and its input-noncontrolling value is 1. A gate g is in the *transitive fanout cone* of a gate g_s if there exists a path from g_s to g.

The *dominators* [6] of a gate g (or a wire w) are a set of gates G such that all paths from g (or w) to any PO have to pass through all gates in G. Consider the dominators of a gate g (or a wire w): the *side inputs* of a dominator are its inputs that are not in the transitive fanout cone of g (or w).

In VLSI testing, a *stuck-at fault* is a fault model used to represent a manufacturing defect within a circuit. The effect of the fault is as if the faulty wire or gate were stuck at either 1 (stuck-at 1) or 0 (stuck-at 0). A stuck-at fault test is a process for finding a test that can generate different output values in the fault-free and the faulty circuits. Given a stuck-at fault f, if there exists such a test, f is said to be testable; otherwise, f is untestable. To make a stuck-at fault on a wire w testable, a test needs to activate and propagate the fault effect to a PO. In a combinational circuit, an untestable stuck-at fault on a wire indicates that the wire is redundant and can be replaced with a constant value 0 or 1. Similarly, if a stuck-at fault on a gate is untestable, all the wires led by the gate can be replaced with a constant value.

The mandatory assignments (MAs) are the unique value assignments to nodes necessary for a test to exist. Consider a stuck-at fault on a wire w; the assignments obtained by setting w to the fault-activating value and by setting the side inputs of dominators of w to the fault-propagating values are MAs. Then, these assignments can be propagated forward and backward to infer additional MAs using *logic implications*. Recursive learning [8], a learning technique in automatic test pattern generation (ATPG), whose complexity is exponential to the recursion depth can be used to infer more MAs. If the MAs of the fault are inconsistent, the fault is untestable, and therefore, w is redundant [12].

For convenience, in the rest of this paper, we use MAs($m = sav$) to denote the set of MAs for the stuck-at v fault test on m, where m can be a node or a wire and v is a logical value 0 or 1.

III. Substitute Node Identification

The method for identifying substitute nodes is similar to that in the work in [5], and thus, we briefly outline it here. For details of the method, the reader is referred to [5].

Replacing a node n_t with another node n_s can be considered setting a replacement error in a circuit, and it can be modeled as the misplaced wire error which is included in the typical design error models [1]. If no input pattern that can detect the error exists, the error is undetectable and n_t can be correctly replaced with n_s. Thus, finding substitute nodes of n_t is equivalent to finding the nodes such that the errors of using these nodes to replace n_t are undetectable.

To detect the error of replacing n_t with n_s, an input pattern must generate different values for n_t and n_s to activate the error effect of $n_t \neq n_s$, and propagate the value of n_t to a PO to make the error effect observable. When we can ensure that no input pattern can simultaneously activate and propagate the error effect, this replacement error is undetectable. Condition 1 below states a sufficient condition for n_s to be a substitute node of n_t based on the concept of the undetectable replacement error.

Condition 1*: If* $n_s = 1$ *and* $n_s = 0$ *are MAs for the stuck-at* 0 *and stuck-at* 1 *fault tests on* n_t*, respectively,* n_s *is a substitute node of* n_t*.*

Because $n_t = 1$ is an MA for the stuck-at 0 fault test on n_t, if $n_s = 1$ is also an MA, no input pattern can simultaneously generate $\{n_s = 0, n_t = 1\}$ and make $n_t = 1$ observable. Similarly, if $n_s = 0$ is also an MA for the stuck-at 1 fault test on n_t, no input pattern can simultaneously generate $\{n_s = 1, n_t = 0\}$ and make $n_t = 0$ observable. Thus, Condition 1 is established. Based on Condition 1, we can identify the substitute nodes of a target node n_t by computing MAs($n_t = sa0$) and MAs($n_t = sa1$).

In Fig. 1(a), e.g., consider finding the substitute nodes of v_3. First, we compute MAs($v_3 = sa0$). To activate the fault effect, v_3 is set to 1. To propagate the fault effect, v_2 is set to 1. We then propagate $\{v_3 = 1,\ v_2 = 1\}$ forward and backward to infer additional MAs using logic implications. The MAs are $\{v_3 = 1,\ v_2 = 1,\ d = 1,\ c = 0,\ b = 1,\ v_1 = 1,\ v_4 = 0,\ v_5 = 1\}$. Second, we use the same method to compute MAs($v_3 = sa1$). The MAs are $\{v_3 = 0,\ v_2 = 1,\ d = 0,\ c = 0,\ b = 1,\ v_1 = 0,\ v_5 = 0\}$. Finally, d and v_1 are determined to be the substitute nodes of v_3 due to the satisfaction of Condition 1. Although v_5 also satisfies Condition 1, it is excluded from being a substitute node of v_3. This is because v_5 is in the transitive fanout cone of v_3, replacing v_3 with v_5 will result in a cyclic combinational circuit.

This example shows that our method can simultaneously find more than one substitute node without trial-and-error checking. It is helpful if there are many substitute nodes that can be chosen to replace a target node when we consider various applications of the node-merging technique.

Furthermore, Condition 1 can be modified by reversing the value of n_s to find complemented substitute nodes that replace a target node together with an additional INV, i.e., if $n_s = 0$ and $n_s = 1$ are MAs for the stuck-at 0 and stuck-at 1 fault tests on n_t, respectively, n_t can be replaced by n_s together with an additional INV.

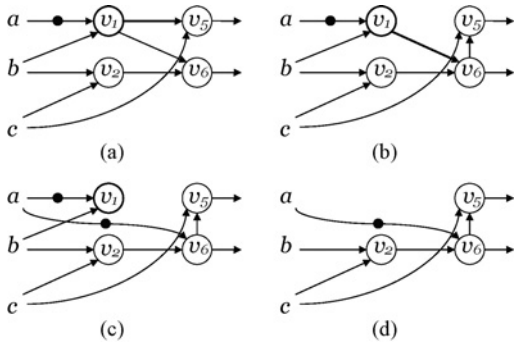

Fig. 2. Example of wire replacement. (a) Original circuit. (b) Resultant circuit of replacing $w(v_1 \rightarrow v_5)$ with $w(v_6 \rightarrow v_5)$. (c) Resultant circuit of replacing $w(v_1 \rightarrow v_6)$ with $w(a \rightarrow v_6)$ together with an INV. (d) Resultant circuit of removing v_1.

Thus, to identify substitute nodes for a target node n_t, we require only two MA computations: one is for the stuck-at 0 fault test on n_t and the other one is for the stuck-at 1 fault test on n_t. Then, nodes that have different values in the two MA sets and are not in the transitive fanout cone of n_t, are the substitute nodes.

The quality of the method depends on the completeness of the MA computations. Although computing more MAs could derive more substitute nodes, it has an exponential time complexity. Thus, instead of computing all MAs, we use the dominator-based MA computation method followed by the recursive learning technique [8] with the recursion depth 1 to balance quality and efficiency.

IV. Circuit Size Reduction

In this section, we present a circuit optimization algorithm based on the proposed node-merging method. Additionally, we introduce three techniques: wire replacement, redundancy removal, and MA reuse, which are engaged to enhance the performance of the algorithm.

A. Wire Replacement

Replacing a node n_t with another node n_s can be regarded as a rewiring process, where all the wires led by n_t are replaced by the added wires led by n_s. For the objective of removing a target node, regardless of whether the added wires are led by a single source node, the target node can be removed as well when all the wires it leads are replaced.

In Fig. 2(a), e.g., v_1 has no substitute node. However, the wire $w(v_1 \rightarrow v_5)$ can be replaced with the wire $w(v_6 \rightarrow v_5)$ as shown in Fig. 2(b), and then the wire $w(v_1 \rightarrow v_6)$ can be replaced with the wire $w(a \rightarrow v_6)$ together with an INV as shown in Fig. 2(c). Thus, we can remove v_1, and the resultant circuit is shown in Fig. 2(d).

For consistency, we name the wire to be replaced a *target wire* and the added wire that can replace the target wire a *substitute wire*. To find a substitute wire, we also propose a sufficient condition as presented in Condition 2.

Condition 2*: If* $n_s = 1$ *and* $n_s = 0$ *are MAs for the stuck-at* 0 *and stuck-at* 1 *fault tests on* $w(n_t \rightarrow n_i)$*, respectively,* $w(n_s \rightarrow n_i)$ *is a substitute wire of* $w(n_t \rightarrow n_i)$*.*

The reason why Condition 2 is established is similar to that of Condition 1. When Condition 2 is satisfied, no input

pattern can simultaneously generate different values for n_s and n_t, and make the value of n_t observable through $w(n_t \rightarrow n_i)$. Similarly, Condition 2 can be modified by reversing the value of n_s to find complemented substitute wires that replace a target wire together with an additional INV.

Let us review the example in Fig. 2(a). The MAs for the stuck-at 0 fault test on $w(v_1 \rightarrow v_5)$ are $\{v_1 = 1,\ a = 0,\ b = 1,\ c = 1,\ v_2 = 1,\ v_6 = 1\}$ and the MAs for the stuck-at 1 fault test on $w(v_1 \rightarrow v_5)$ are $\{v_1 = 0,\ v_6 = 0,\ c = 1\}$. Thus, $w(v_1 \rightarrow v_5)$ can be replaced with $w(v_6 \rightarrow v_5)$. Next, using the same method in Fig. 2(b), we can find that $w(v_1 \rightarrow v_6)$ can be replaced with $w(a \rightarrow v_6)$ together with an INV, as shown in Fig. 2(c). Finally, we can remove v_1 because it does not lead any wires.

B. Redundancy Removal

As mentioned in Section II-B, if the MAs of a stuck-at fault test on a node are inconsistent, the fault is untestable and the node is redundant. Our method for substitute node identification computes the MAs for the stuck-at 0 and stuck-at 1 fault tests on a target node n_t. Once we find that the MAs are inconsistent, we can replace n_t with a constant value 0 or 1 depending on the fault value. Thus, we can simultaneously detect if n_t is redundant without extra effort. Similarly, we use the same method to detect and remove redundant wires in the wire replacement technique.

C. MA Reuse

MA reuse is a method to reuse the computed MAs during circuit optimization. According to the concept of fault collapsing [2], which states that two equivalent stuck-at faults have the same test set, some stuck-at fault tests on different nodes can have the same MA set. Thus, we can reuse these MAs when optimizing a circuit.

Let us consider computing the MAs for the stuck-at fault tests on a node n with MA reuse. Suppose n_i is a fanout node of n and MAs($n_i = sa0$) has been computed. If there exists no INV between n and n_i, we can directly set MAs($w(n \rightarrow n_i) = sa0$) to MAs($n_i = sa0$) rather than re-compute the same MA set. Additionally, if n only leads n_i, we can set MAs($n = sa0$) to MAs($n_i = sa0$). However, if there exists an INV between n and n_i, we can set MAs($w(n \rightarrow n_i) = sa1$) to MAs($n_i = sa0$), and set MAs($n = sa1$) to MAs($n_i = sa0$) when n only leads n_i. For each node, only the MAs of its stuck-at 0 fault test can be reused.

D. Overall Algorithm

In circuit optimization, although the optimization orders of selecting a target node, a substitute node, and a substitute wire can affect the optimization results, it is difficult and might be time-consuming to evaluate which replacement is best. Thus, we use a heuristic optimization order in this paper. Each node in a circuit is selected as a target node n_t in the depth-first search (DFS) order from POs to PIs, and it is replaced with the substitute node that is closest to PIs.

However, if n_t is determined having no substitute node, we then perform wire replacement on the wires led by it. Each wire led by n_t is sequentially selected as a target wire and we replace it with its substitute wire that is closest to PIs once we find it. Since our objective is to remove n_t, if one target wire cannot be replaced, n_t is non-removable, and thus, we stop performing wire replacement on the remaining wires.

Circuit_Size_Reduction(Circuit C**)**
For each node n_t in C in the DFS order from POs to PIs
1. Compute MAs($n_t = sa0$) with MA reuse.
   1.1. If there exists a conflict in MAs($n_t = sa0$), replace n_t with 0 and **continue**.
   1.2. Store MAs($n_t = sa0$) for further reuse.
2. Compute MAs($n_t = sa1$) with MA reuse.
   2.1. If there exists a conflict in MAs($n_t = sa1$), replace n_t with 1 and **continue**.
3. $SubstituteNodes \longleftarrow$ nodes having different values in MAs($n_t = sa0$) and MAs($n_t = sa1$), and not in the transitive fanout cone of n_t.
4. If $SubstituteNodes \neq \emptyset$, replace n_t with the node that is in the set of $SubstituteNodes$ and closest to PIs, and then **continue**.
5. For each wire $w(n_t \rightarrow n_i)$ led by n_t
   5.1. Compute MAs($w(n_t \rightarrow n_i) = sa0$) with MA reuse.
      5.1.1. If there exists a conflict in MAs($w(n_t \rightarrow n_i) = sa0$), replace $w(n_t \rightarrow n_i)$ with 0 and **continue**.
   5.2. Compute MAs($w(n_t \rightarrow n_i) = sa1$) with MA reuse.
      5.2.1. If there exists a conflict in MAs($w(n_t \rightarrow n_i) = sa1$), replace $w(n_t \rightarrow n_i)$ with 1 and **continue**.
   5.3. $SubstituteWires \longleftarrow w(n_s \rightarrow n_i)$, where n_s has different values in MAs($w(n_t \rightarrow n_i) = sa0$) and MAs($w(n_t \rightarrow n_i) = sa1$), and is not in the transitive fanout cone of n_i.
   5.4. If $SubstituteWires \neq \emptyset$, replace $w(n_t \rightarrow n_i)$ with the wire that is in the set of $SubstituteWires$ and closest to PIs, and then **continue**; otherwise, **break**.

Fig. 3. Overall algorithm for circuit size reduction.

Fig. 3 shows the overall algorithm for circuit size reduction. Given a circuit C, the algorithm iteratively selects a target node n_t in the DFS order from POs to PIs. At each iteration, from steps 1–4, the algorithm finds the substitute nodes of n_t by computing MAs($n_t = sa0$) and MAs($n_t = sa1$), and replaces n_t with the substitute node that is closest to PIs. If n_t is replaced, the algorithm continues to consider the next target node.

However, if n_t has no substitute node, the algorithm starts performing wire replacement on the wires led by n_t in step 5. Similarly, for each target wire $w(n_t \rightarrow n_i)$, the algorithm finds its substitute wires by computing MAs($w(n_t \rightarrow n_i) = sa0$) and MAs($w(n_t \rightarrow n_i) = sa1$), and replaces it with the substitute wire that is closest to PIs. If $w(n_t \rightarrow n_i)$ is replaced, the algorithm continues to consider the next target wire. Otherwise, the algorithm stops performing wire replacement and returns to consider the next target node.

V. Experimental Results

We implemented our algorithms in C language within an ABC [3] environment. The experiments were conducted on a 3.0 GHz Linux platform (CentOS 4.6). The benchmarks are from the IWLS 2005 suite [14]. Each benchmark is initially transformed to an AIG format, and we only consider its combinational portion. Additionally, the recursive learning technique [8] is applied with the recursion depth 1 in our algorithms. The experiments consist of two parts. The first one is to show the efficiency and effectiveness of our method for finding substitute nodes. The second one is to show the capability of our method for circuit size reduction.

TABLE I
EXPERIMENTAL RESULTS OF SUBSTITUTE NODE IDENTIFICATION

Benchmark	N	N_{rep}	%	N_{sub}	Ratio	Time	N_{equ}	$N_{sub}^{k>5}$
i2c	1306	80	6.1	174	2.2	0.2	19	11
pci_spoci.	1451	170	11.7	890	5.2	0.6	46	93
systemcdes	3190	147	4.6	301	2.1	1.5	60	29
spi	4053	65	1.6	91	1.4	3.4	14	2
des_area	4857	80	1.6	152	1.9	5.6	5	16
tv80	9609	496	5.2	3864	7.8	17.2	146	2684
systemcaes	13 054	202	1.5	380	1.9	17.7	48	15
ac97_ctrl	14 496	98	0.7	242	2.5	3.2	33	1
mem_ctrl	15 641	1537	9.8	3588	2.3	98.8	1150	397
usb_funct	15 894	370	2.3	1271	3.4	6.3	108	77
aes_core	21 513	452	2.1	1742	3.9	15.2	29	910
pci_bridge32	24 369	309	1.3	621	2.0	21.7	53	43
wb_conmax	48 429	5608	11.6	41 996	7.5	28.2	188	11 385
des_perf	79 288	2505	3.2	6195	2.5	51.4	56	694
Average			4.5		3.3			
Total		12 119		61 507		271.0	1955	16 357

TABLE II
EXPERIMENTAL RESULTS OF OUR APPROACH AND [11] FOR CIRCUIT SIZE REDUCTION

Benchmark	# Nodes	Ours w/o RR		Ours w/o WR		Ours		[11]	
		%	Time	%	Time	%	Time	%	Time
pci_spoci.	878	10.9	0.3	10.9	0.2	11.8	0.3	9.2	6
i2c	941	2.0	0.2	1.9	0.1	2.2	0.1	3.2	3
dalu	1057	7.0	0.5	6.8	0.3	9.1	0.5	12.0	10
C5315	1310	0.5	0.2	0.5	0.1	0.5	0.1	0.7	2
s9234	1353	1.9	0.2	1.6	0.2	1.8	0.2	1.2	8
C7552	1410	3.3	0.4	2.8	0.3	3.3	0.5	3.4	8
i10	1852	5.3	0.9	5.2	0.6	5.8	0.9	1.3	12
s13207	2108	1.6	0.6	2.1	0.5	2.2	0.6	1.8	17
alu4	2471	13.8	7.4	21.4	5.3	22.6	6.8	22.9	64
systemcdes	2641	2.0	1.3	1.6	0.9	2.0	1.3	4.7	9
spi	3429	0.6	3.8	0.5	2.7	0.6	3.8	1.3	84
tv80	7233	3.9	15.2	3.8	10.6	4.6	14.8	7.1	1445
s38417	8185	0.6	1.7	0.6	1.2	0.7	1.8	1.0	275
mem_ctrl	8815	16.2	9.2	17.7	6.8	17.4	8.8	18.0	738
s38584	9990	1.1	14.4	1.4	11.4	1.7	14.3	0.8	223
ac97_ctrl	10 395	0.2	2.4	0.2	2.0	0.2	2.4	2.0	188
systemcaes	10 585	0.7	17.4	0.6	13.1	0.7	17.1	3.8	360
usb_funct	13 320	2.3	7.2	2.2	5.9	2.6	7.1	1.4	681
pci_bridge32	17 814	0.5	14.7	0.5	12.0	0.6	14.5	0.1	1134
aes_core	20 509	1.2	22.6	0.7	13.2	1.2	22.3	8.6	1620
b17	34 523	1.9	110.3	1.6	72.4	2.5	108.6	1.6	5000
wb_conmax	41 070	4.6	48.1	4.4	31.9	4.7	46.8	6.2	5000
des_perf	71 327	2.1	202.1	1.7	62.6	2.1	199.0	3.7	5000
Average		3.7		3.9		4.4		5.0	
Total			481.1		254.3		472.6		21 887
Ratio							1		46.3

A. Substitute Node Identification

Due to page limitations, we do not show the experimental results reported in the work in [13] and do not compare our method with it. That work has shown its efficiency at finding local ODC-based node mergers. Instead, we conducted the experiments to show that our method can complement it by finding additional node mergers.

In the experiments, each node in a benchmark is considered a target node and we find its substitute nodes without replacing it. Each identified node merger is further checked whether it could not be detected by the local ODC-based method [13] due to the bounded depth limit. This can be achieved by checking if a node merger we identify is not a candidate merger in the local ODC-based method. Thus, we re-implemented the local ODC-based algorithm with the bounded depth $k = 5$ to collect candidate mergers. For each benchmark, 2^5 random patterns are parallelly simulated for computing ODCs in the re-implemented algorithm.

Table I summarizes the experimental results. Column 1 lists the benchmarks. Column 2 lists the number of nodes in each benchmark represented by AIG N. Column 3 lists the number of target nodes that have substitute nodes, and thus, are replaceable N_{rep}. Column 4 lists the percentage of N_{rep} with respect to N. Column 5 lists the number of all substitute nodes N_{sub}. Column 6 lists the ratio of N_{sub} with respect to N_{rep}. It is also the average number of substitute nodes a replaceable node has. Column 7 lists the CPU time measured in seconds. Column 8 lists the number of target nodes having functionally equivalent substitute nodes. Column 9 lists the number of all substitute nodes that can be identified by our method but cannot by the local ODC-based method with $k = 5$.

The benchmark *i2c*, e.g., has 1306 nodes. Our method found that 80 nodes, or 6.1% of nodes, have substitute nodes and are replaceable. There are 174 substitute nodes in total and a replaceable node has an average of 2.2 substitute nodes. The CPU time is 0.2 s. Furthermore, there are 19 nodes having functionally equivalent substitute nodes. There are 11 substitute nodes that the local ODC-based method cannot detect, i.e., our method can find additional 11 substitute nodes when it is used together with the local ODC-based method with $k = 5$.

According to Table I, our method can identify substitute nodes for an average of 4.5% of nodes in a benchmark, with 3.3 substitute nodes for each on average. The overall CPU time for all benchmarks is only 271.0 s. Additionally, according to Columns 3 and 8, we can conclude that most node mergers our method finds are ODC-based node mergers. However, they also show that our method is not good at finding functionally equivalent node mergers. This is because proving two nodes to be functionally equivalent by checking if one of them logically implies the other one is difficult by using logic implications, especially when they have disjoint fanout cones. Fortunately, functionally equivalent node mergers can be easily detected by the SAT sweeping method [7].

Additionally, the last column shows that our method can find additional substitute nodes that the local ODC-based method with $k = 5$ cannot detect for all benchmarks. It can be expected that when our method is applied together with the local ODC-based method, we can obtain more substitute node choices and potentially more replaceable nodes with little CPU time overhead. Thus, our method can be considered another fast method that can complement the local ODC-based method.

B. Circuit Size Reduction

Next, we compare our method with the global ODC-based method [11] for circuit size reduction, which is superior to the local ODC-based method. To have a fair comparison, we initialize each benchmark by using the *resyn2* script in the ABC package as performed by [11], which performs local circuit rewriting optimization [9]. After the initialization, we optimize the benchmarks by using our overall method as shown in Fig. 3. For comparison, we also separately optimize the benchmarks by using our method without redundancy removal and the one without wire replacement. After the optimization, we apply an equivalence checking tool, *cec* [10], in the ABC package to verify the correctness of our optimization. Note that although we have each benchmark initialized as [11] did by applying the *resyn2* script, the initial number of nodes in each benchmark is still a little different from that reported in [11].

Table II summarizes the experimental results. Columns 1 and 2 list the benchmarks and the number of nodes in each benchmark represented by AIG, respectively. Columns 3, 5, and 7 list the percentage of circuit size reduction in terms of node count achieved by our method without redundancy removal, our method without wire replacement, and our overall method, respectively. Columns 4, 6, and 8 list the CPU time measured in seconds. Columns 9 and 10 list the corresponding results reported in [11]. The maximal CPU time in Column 10 is 5000 s, which is the CPU time limit set by the work.

Table II shows that although the efficiency benefits of our method come at the expense of the optimization quality, our method can work together with the redundancy removal and the wire replacement techniques to have a competitive capability with a speedup of 46.3 times, compared to the global ODC-based method.

VI. Conclusion

In this paper, we proposed an ATPG-based method for node merging by using logic implications. The method only requires two MA computations to find the substitute nodes of a target node. The process that previously required many SAT solving calls is thus reduced to be achievable in practically linear time. The experimental results showed that the proposed node-merging method can complement the local ODC-based method with the bounded depth $k = 5$ by finding additional node mergers. Moreover, we extend the node-merging method with three techniques, wire replacement, redundancy removal, and MA reuse, for circuit optimization. The experimental results showed that the proposed optimization algorithm has a competitive capability and expends much less CPU time compared to the state-of-the-art method.

References

[1] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic design verification via test generation," *IEEE Trans. Comput.-Aided Design*, vol. 7, no. 1, pp. 138–148, Jan. 1988.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Design for Testability*. Piscataway, NJ: IEEE Press, 1990.

[3] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification* [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[4] M. Case, V. Kravets, A. Mishchenko, and R. Brayton, "Merging nodes under sequential observability," in *Proc. Des. Autom. Conf.*, 2008, pp. 540–545.

[5] Y.-C. Chen and C.-Y. Wang, "Fast detection of node mergers using logic implications," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 785–788.

[6] T. Kirkland and M. R. Mercer, "A topological search algorithm for ATPG," in *Proc. Des. Autom. Conf.*, 1987, pp. 502–508.

[7] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proc. Int. Conf. Comput.-Aided Design*, 2004, pp. 50–57.

[8] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to CAD problems—test, verification, and optimization," *IEEE Trans. Comput.-Aided Design*, vol. 13, no. 9, pp. 1143–1158, Sep. 1994.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *Proc. Design Autom. Conf.*, 2006, pp. 532–536.

[10] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Proc. Int. Conf. Comput.-Aided Design*, 2006, pp. 836–843.

[11] S. M. Plaza, K.-H. Chang, I. L. Markov, and V. Bertacco, "Node mergers in the presence of don't cares," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2007, pp. 414–419.

[12] M. H. Schulz and E. Auth, "Advanced automatic test pattern generation and redundancy identification techniques," in *Proc. Int. Fault-Tolerant Comput. Symp.*, 1988, pp. 30–35.

[13] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't cares," in *Proc. Design Autom. Conf.*, 2006, pp. 229–234.

[14] *IWLS.* (2005) [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

On Undetectable Faults and Fault Diagnosis

Irith Pomeranz and Sudhakar M. Reddy

***Abstract*—The presence of an undetectable fault u_i may modify the response of a detectable fault d_j to a test set used for fault diagnosis. This may impact the accuracy of fault diagnosis based on the responses of single faults. Many state-of-the-art diagnosis processes are based on the responses of single stuck-at faults even though their goal is to diagnose defects (including multiple defects) that are different from stuck-at faults. Therefore, we study the effects of undetectable single stuck-at faults on the accuracy of fault diagnosis based on the responses of single stuck-at faults. For this purpose, we consider the cases where the response of a double stuck-at fault $u_i\&d_j$, which consists of an undetectable fault u_i and a detectable fault d_j, is different from the response of the single fault d_j. We show that there are significant, yet manageable, numbers of such faults in benchmark circuits under test sets used for fault diagnosis. In all these cases, a fault diagnosis process based on single stuck-at faults may not identify the locations of d_j and u_i as candidate defect sites if a defect affects the sites of d_j and u_i. We conclude that it is important to consider $u_i\&d_j$ during fault diagnosis in order not to preclude the sites of d_j and u_i as candidate defect sites.**

***Index Terms*—Diagnostic test generation, fault diagnosis, full-scan circuits, stuck-at faults.**

I. Introduction

Fault diagnosis is a process that identifies the likely locations of defects present in a chip after the chip produces a faulty output response to a test set. Conceptually, the fault diagnosis process compares the observed response of the chip to the responses expected in the presence of modeled faults. The defect is assumed to be present at one of the sites of the modeled faults that best match the observed response. These sites are referred to as candidate defect sites, and the faults are referred to as candidate faults. For the results to be considered accurate, the site of a candidate fault needs to point correctly to the site of the defect.

Several fault diagnosis procedures [1]–[9] use single stuck-at faults as a basis for diagnosis, noting that responses of common defects to test sets used for diagnosis can be represented as deviations from the responses of single stuck-at faults. For

Manuscript received February 7, 2010; revised April 17, 2010. Date of current version October 20, 2010. The work of I. Pomeranz and S. M. Reddy was supported in part by Semiconductor Research Corporation, under Grants 2007-TJ-1643 and 2007-TJ-1642. This paper was recommended by Associate Editor C.-W. Wu.

I. Pomeranz is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: pomeranz@ecn.purdue.edu).

S. M. Reddy is with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242 USA (e-mail: reddy@engineering.uiowa.edu).

Digital Object Identifier 10.1109/TCAD.2010.2053476