Using Simulation and Satisfiability to Compute Flexibilities in Boolean Networks

Alan Mishchenko, *Member, IEEE*, Jin S. Zhang, *Member, IEEE*, Subarna Sinha, Jerry R. Burch, *Member, IEEE*, Robert Brayton, *Fellow, IEEE*, and Malgorzata Chrzanowska-Jeske, *Senior Member, IEEE*

***Abstract*—Simulation and Boolean satisfiability (SAT) checking are common techniques used in logic verification. This paper shows how simulation and satisfiability (S&S) can be tightly integrated to efficiently compute flexibilities in a multilevel Boolean network, including the following: 1) complete "don't cares" (CDCs); 2) sets of pairs of functions to be distinguished (SPFDs); and 3) sets of candidate nodes for resubstitution. These flexibilities can be used in network optimization to change the network structure while preserving its functionality. In the first two applications, simulation quickly enumerates most of the solutions while SAT detects the remaining solutions. In the last application, simulation efficiently filters out most of the infeasible solutions while SAT checks the remaining candidates. The experimental results confirm that the combination of simulation and SAT offers a computation engine that outperforms binary decision diagrams, which are traditionally used in such applications.**

***Index Terms*—Boolean network, logic synthesis, satisfiability, simulation.**

I. Introduction

Logic synthesis transforms a Boolean network with the goal of improving its area, delay, power, testability, etc. While maintaining the required input–output global functionality of the network, it is possible to change the local function and local fanins of some nodes. The environment of a node determines the extent to which the node can be changed, resulting in the so-called flexibility of the node in the network. Several formalisms for computing these flexibilities have been proposed over the years, offering powerful analysis techniques to discover opportunities for network optimizations. The power of each formalism correlates with the complexity of the computation it requires. In this paper, we focus on three such formalisms, namely: 1) complete "don't cares" (CDCs) [22]; 2) sets of pairs of functions to be distinguished (SPFDs) [31]; and 3) node resubstitutions [14]. Traditionally, algorithms for these formalisms have been implemented using binary decision diagrams (BDDs) [1], [5] and are computationally quite expensive in practice. We reformulated these problems using simulation and Boolean satisfiability (SAT), which led to improvements in the speed and practicality of the algorithms.

Simulation has always played an important role in testing and verification of digital designs. Its main advantage is that it catches many bugs quickly. However, it is practically an incomplete technique, because it is infeasible to simulate all possible patterns on large circuits. In contrast, SAT is able to prove, by efficient exploration of the search space, that a certain property always holds. Because of its exhaustive nature, SAT can be slow in solving large problems or many instances of small problems. Recently, remarkable progress in SAT algorithms [9], [19], [25] has made it possible to extend its range of applications.

The two methods, simulation and SAT (S&S), make a powerful combination, taking advantage of the strengths of each method. Simulation is fast at finding satisfying assignments or disproving the properties, thereby saving the runtime that SAT would need for exhaustive search. However, random simulation quickly saturates (stops turning out new assignments), at which point SAT can search for any remaining assignments. When the computational resources are intelligently divided among the methods, S&S becomes a formidable competitor to BDD-based approaches.

One reason why SAT outperforms BDDs in these logic synthesis applications is that, for many problems, construction of both BDDs and SAT requires, in the worst case, exponential time in the problem size. The BDD-based approach starts by constructing the canonical representation for the given functions. SAT starts by searching the solution space immediately while relying on the available circuit to represent the functions. Thus, SAT avoids the overhead of constructing the canonical representation. This overhead is too costly in many practical instances when the size of intermediate or final BDD is prohibitively large and dynamic variable reordering is inefficient or very slow. Similar arguments for the development of SAT-only combinational equivalence checkers are stated in [10].

The main contribution of this paper is in formulating and presenting three algorithms, based on S&S, to compute flexibilities in multilevel Boolean networks. The flexibilities computed are CDCs, SPFDs, and resubstitution candidates. Experimental results show that S&S offers much more affordable runtimes than the earlier methods. This extends the scope and applicability of algorithms for network optimization.

Manuscript received July 14, 2005. This work was supported in part by the National Science Foundation under grants CCR 9988402 and CCR 0312676, by the MARCO Focus Center for Circuit System Solution under Contract 2003-CT-888, and by the California Micro Program with industrial sponsors, Fujitsu, Intel, Magma, and Synplicity. This paper was recommended by Guest Editor R. I. Bahar.

A. Mishchenko and R. Brayton are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: alanmi@eecs.berkeley.edu; brayton@eecs.berkeley.edu).

J. S. Zhang and M. Chrzanowska-Jeske are with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207 USA (e-mail: jinsong@ece.pdx.edu; jeske@ece.pdx.edu).

S. Sinha is with Synopsys, Inc., Mountain View, CA 94043 USA (e-mail: Subarna.Sinha@synopsys.com).

J. R. Burch is with Synopsys, Inc., Hillsboro, OR 97124 USA (e-mail: jrb@synopsys.com).

Digital Object Identifier 10.1109/TCAD.2005.860955

The paper is organized as follows. Section II gives a brief background on Boolean networks, simulation, and SAT. The background on CDCs, SPFDs, and resubstitution, together with corresponding S&S-based algorithms to compute each of these flexibilities and the experimental results, are presented in Sections III–V, respectively. Section VI presents some relevant implementation details. Section VII concludes the paper and suggests possible directions for future work.

II. Background

A. Boolean Network

A Boolean network N is a directed acyclic graph (DAG) such that for each node i in N, there is a Boolean function f_i and a Boolean variable y_i, where $y_i = f_i$. A node i is a fanin of a node j if there is a directed edge $\{i, j\}$ and a fanout if there is a directed edge $\{j, i\}$. A node i is a transitive fanin (TFI) of a node j if there is a directed path from i to j and a transitive fanout (TFO) if there is a directed path from j to i. The sources of the graph are the primary inputs (PIs) of the network; the sinks are the primary outputs (POs). The functionality of a node in terms of its immediate fanins is its local function. The functionality of a node in terms of the PIs of the network is its global function.

As with other logic optimization problems, when computing flexibilities of network nodes, we consider only acyclic combinational Boolean networks.

The following notation is used throughout the paper: X is the set of PIs, x is a particular PI variable; Y is the set of local inputs of a particular node, y is a particular local variable; and Z the set of POs, z a particular PO variable. $g(X)$ and $f(Y)$ are, respectively, the global and local functions of a node.

B. Simulation

Simulation computes the values of the internal signals and POs of a network, given the values of the PIs. One round of simulation involves propagating one particular set of values through the network; its complexity is linear in the network size. Since we consider only combinational networks, there are no state variables.

We contrast two forms of simulation, namely: 1) random simulation, when values of the PIs are assigned randomly and 2) guided simulation, when PIs are assigned based on certain information, such as that provided by a SAT solver about assignments that prove or disprove a property. Only random simulation is used in the applications discussed in this paper.

As detailed below, we use two ways to control the amount of simulation performed, namely: 1) the static approach, where a fixed number of rounds of simulation is performed and 2) the dynamic approach, where simulation stops when no new solution is discovered after several rounds.

C. Satisfiability

Boolean SAT [19], [25] is a process of proving that a given Boolean formula has a satisfying assignment. Although solving a general SAT problem is NP-complete in the problem size, many practical problems have properties, which dramatically reduce the complexity. For example, if a SAT problem is formulated for a circuit, using the circuit structure can reduce the problem complexity [27].

The performance of SAT solvers has improved greatly in the last few years. State-of-the-art SAT solvers, such as those found in [9], [10], and [25], are based on techniques that dramatically speed up exploration of the search space. The following are the most successful.

1) Nonchronological backtracking [19]: This is a way of exhaustively exploring the search tree more efficiently by skipping some branches.
2) Dynamic variable ordering [25]: The branching variable is determined by the dynamically updated "activity" of variables. Each time a variable participates in a conflict analysis, its activity is increased by a fixed amount. The activity counters are periodically divided by a constant to help focus the SAT search on recent conflicts.
3) Two-literal watching [25]: The clause database is organized in such a way that fewer clauses are visited when literals are assigned, and no clauses are visited when literals are unassigned.

In our applications, we are solving "bounded ALL-SAT" problems. The form of these problems is to determine all satisfying assignments of a formula $\exists_x P(X, Y)$, where X is the PI space, which grows with network size, and Y is the local input space, which has a fixed size. These are called ALL-SAT problems, because a SAT solver needs to enumerate the whole space of satisfying assignments. This increases the computational complexity compared to the classical SAT problem, which only requires one satisfying assignment. However, since the set Y of free variables is bounded, the asymptotic complexity is the same as in the classical SAT problem.

When used together with SAT, simulation filters out a significant number of satisfying assignments (infeasible candidates), leaving SAT to work only on the hardest ones. Therefore, solution enumeration based on S&S works well in practice, resulting in affordable runtimes for many benchmarks. The approach presented in this paper can be improved further by incorporating recently published efficient methods for solving the ALL-SAT problem [13].

III. Computing CDCs

Optimization of Boolean networks using don't cares plays an important role in technology independent logic synthesis and incremental resynthesis of mapped netlists. Traditionally, only satisfiability don't cares (SDCs) and compatible observability don't cares (CODCs) have been used [28]. CODCs form a subset of the CDCs (or complete flexibility [23]) projected onto a node by its context in a multilevel network. It was shown experimentally [21], [22] that the computation of CDCs is comparable in runtime and memory requirements to the computation of SDCs and CODCs, while the amount of flexibility is larger. This additional freedom offered by CDCs leads to an increase in optimization quality.

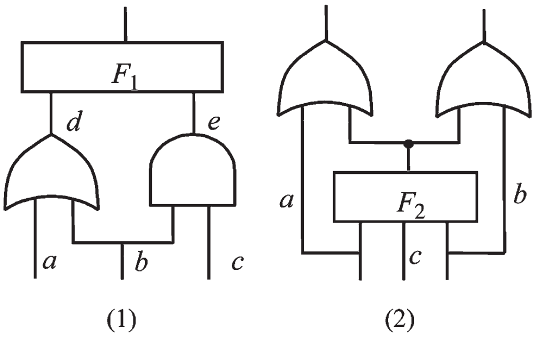


Fig. 1. Examples: (1) SDCs and (2) ODCs.

A. Background

Definition 3.1: A completely specified Boolean function (CSF) is a many-to-one mapping from n-dimensional $(n \geq 0)$ Boolean space into a one-dimensional space: $\{0,1\}^n \rightarrow \{0,1\}$. An assignment of n Boolean variables is called a minterm. A CSF has negative (positive) minterms, which correspond to the assignments, for which it takes a value of 0 (1). The union of positive and negative minterms is called the care minterms.

A don't care for a logic function is a minterm for which the function can take either 0 or 1 as a possible value. If there exists at least one such input combinations, the function is called an incompletely specified Boolean function (ISF). Such minterms are called don't care minterms. One ISF is said to be larger than another if it has more don't care minterms.

A CSF is compatible with an ISF (implements the ISF), if the CSF can be derived from the ISF by assigning either 0 or 1 to each don't care minterm.

Definition 3.2: In Boolean logic, conjunctive normal form (CNF) is a method of standardizing and normalizing logical formulas. A logical formula is considered to be in CNF if and only if it is a single conjunction of one or more disjunctions of one or more literals. For example, the following formulas are in CNF: $A \wedge B$; $\neg A \wedge (B \vee C)$; and $(A \vee B) \wedge (\neg B \vee C \vee D)$.

Definition 3.3: The CDCs, or CF, of a node in the binary network, is the largest ISF (as a function of the node's fanins), whose don't care minterms represent conditions under which the output of the node does not influence the values produced by any of the POs of the network.

The CDCs include the SDCs, which arise because some combinations are not produced as the inputs of the node, and the observability don't cares (ODCs), which arise because under some conditions, the output of the node does not matter. Fig. 1 shows a situation when node F_1 has SDCs in the local space ($d = 0$ and $e = 1$) due to limited controllability, while node F_2 has ODCs ($a = 1$ and $b = 1$) due to limited observability.

CDCs are important for network optimization, because replacing the node's function by any CSF compatible with the ISF representing a node's CDCs does not change the functionality of the POs of the network. A key observation about CDCs is that they are not compatible, i.e., the POs of the network may produce incorrect values if CDCs derived for several nodes are used independently. In this sense, CDCs differ from CODCs [29]. However, if CDCs are computed and used immediately to optimize a node before continuing to another node, compatibility is not required. In that case, whenever a CDC is computed for a node, it reflects all prior changes to other nodes.

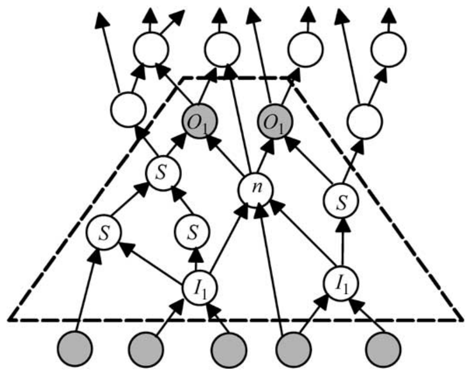


Fig. 2. Example of 1×1 window.

B. Compute CDCs Using S&S

Don't care computations are traditionally performed in the context of the entire Boolean network, as exemplified by SIS [30]. In the case of CDCs, this approach is optimal in that it guarantees that the don't cares computed are the largest don't cares possible for a node in the network. However, the network may be too large, making the computation too expensive. In such cases, the computation can be restricted to a relatively small neighborhood subnetwork. The don't cares computed for the node in this subnetwork are complete for the subnetwork, but not for the entire network; thus, considering a larger subnetwork leads to a longer runtime but may result in more don't cares.

A windowing method has been developed to limit the subnetwork used for don't care computation to only a few levels of logic surrounding the node. A detailed discussion of this windowing method can be found in [24]. An important observation is that reconvergence is responsible for don't cares; hence, along with the TFIs and TFOs of the node, a window should contain all reconvergent paths that begin and terminate in the window. Of course, if the window extends all the way to include PIs and POs, the window CDC is equal to the CDC.

We refer to the window of a node constructed by including l_i TFI logic levels and l_o TFO logic levels as an $l_\mathrm{i} \times l_\mathrm{o}$ window. For example, Fig. 2 shows a 1×1 window for node n. The window's roots (top) and leaves (bottom) are shaded.

To compute CDCs using the whole network or a subnetwork derived by windowing, we construct a miter network, as shown in Fig. 3. A miter [2] refers to a circuit whose output evaluates to 1 if and only if some property does not hold. The first network N represents the original network, while the second network N' has an additional inverter added at the output of the given node. Comparing the POs of these networks detects when the change in the node's behavior influences the POs' functionalities. Thus

$$C(X) = \sum_i [g_i(X) \oplus g'_i(X)] \tag{1}$$

represents the care set in the global space. The ODCs of the node in the global space is the complement of the care set,

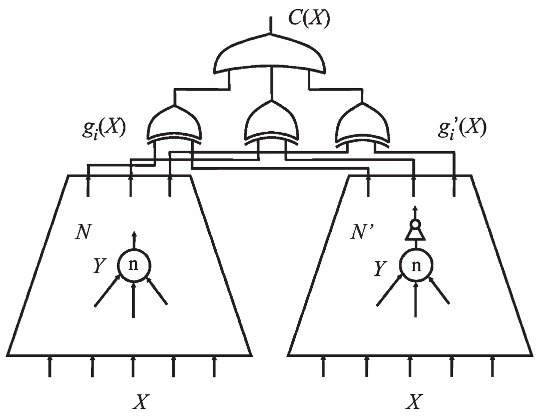


Fig. 3. Illustration of CDC computation.

```
function CompleteDC( node $n$ , context $S$ )
{
    aig $M$ = ConstructMiter($S$, $n$);
    function $C_1$ = RandomSimulation( $M$ );
    cnf $P$ = CircuitToCNF( $M$ ) $\wedge$ FunctionToCNF( $\overline{C_1}$ );
    function $C_2$ = SatSolutions( $P$ );
    return $\overline{C_1 + C_2}$;
}
```

Fig. 4. Pseudocode of S&S-based CDC computation.

given as

$$\text{ODC}(X) = \overline{C(X)} = \prod_i [g_i(X) \equiv g_i'(X)]. \qquad (2)$$

The local CDCs are computed by imaging the global ODCs into the local space. To this end, mapping $M(X,Y)$ is used, which uses the TFI of the node to relate the global and local spaces

$$\begin{aligned}\text{CDC}(Y) &= \forall_X \left[M(X,Y) \Rightarrow ODC(X)\right] \\ &= \forall_X \left[\overline{M(X,Y)} + ODC(X)\right]. \qquad (3)\end{aligned}$$

This computation adds the SDC, $\overline{M(X,Y)}$, to the already computed ODC, resulting in the CDC. Thus, (3) can be interpreted as follows: "A don't care minterm Y belongs to the CDC if it is either an SDC or an ODC for all assignments of the PI variables X." If external don't cares are available, they are added to the ODCs.

CDCs can be computed by a straightforward application of BDD operations to (1)–(3), first deriving the global functions of the POs of the two networks, $\{g_i(X)\}$ and $\{g_i'(X)\}$, where the index i varies over the POs.

Fig. 4 shows the pseudocode of the S&S-based CDC computation.

The top-level procedure CompleteDC takes node n and its context S given by the network (or by a window constructed for node n). Procedure ConstructMiter applies structural hashing [15] to the miter shown in Fig. 3. The AND-INV graph (AIG) M is constructed in one depth-first search (DFS) traversal of the nodes in S, without duplication of the window. Random simulation is applied to the miter network. Each assignment of the PIs variables X, such that the output of the miter is 1, detects a care minterm of the node in terms of variables Y. These are put in the set C_1. Only unique care minterms are collected. Bit-parallel simulation is performed until saturation, which is reached when m successive rounds of simulation (each consisting of 32 random patterns) do not identify any new local care minterms. In our implementation, we set m to 10. In practice, for most windows with 10–30 inputs, this approach requires about 50–100 rounds of simulation.

TABLE I
RUNTIME COMPARISON FOR BDD VERSUS S&S FOR CDC COMPUTATION

Name	Window 1 x 1		Window 2 x 2		Window 4 x 4	
	BDDs	S&S	BDDs	S&S	BDDs	S&S
b14	1.47	0.67	3.50	0.84	12.29	1.24
b15	0.84	0.99	3.11	1.20	26.70	5.30
b17	2.97	1.33	6.69	3.21	48.59	4.37
b20	2.98	2.18	6.19	2.19	20.18	2.23
b21	3.42	2.13	6.48	2.79	18.34	2.42
b22	4.50	3.18	9.62	4.86	27.80	3.24
s15850	0.17	0.26	0.39	0.28	4.14	0.30
s35932	0.28	0.20	0.44	0.28	1.10	0.53
s38417	1.16	0.50	3.40	0.55	18.78	1.15
pj1	1.69	1.58	5.75	1.38	15.26	2.35
pj2	0.20	0.20	0.28	0.26	3. 66	0.28
Ave	**1.00**	**0.80**	**1.00**	**0.46**	**1. 00**	**0.14**

The CNF P is the conjunction of clauses derived from M and the complement of C_1, the part of the care set already derived by random simulation. The CNF of M is derived using a well-known technique, which adds three CNF clauses for each AND gates, e.g., the clauses added for gate $ab = c$ are as follows: $\overline{c} + a$; $\overline{c} + b$; and $\overline{a} + \overline{b} + c$. The only other clause added to the CNF is the one asserting that the output of the miter is equal to one. Better techniques for translating AIG into CNF can be found in [37], which will further reduce the runtime.

The SAT solver enumerates through the satisfying solutions C_2 of the resulting problem representing the remaining part of the care set. In practice, often, the SAT problem has no solutions ($C_2 = 0$). In such cases, SAT is only useful to prove the completeness of the care set derived by simulation.

Simulation and SAT effectively enumerate the minterms of the care set in the local space of the node. Therefore, it should be limited to nodes with roughly ten inputs or less, which is typically the case for most Boolean networks. If larger nodes exist, this limit can be enforced by decomposing the large nodes. The implementation can also be extended to return incomplete satisfying assignments, which correspond to cubes rather than minterms of the care set, similar to [13].

C. Experimental Results

Both BDD-based and S&S-based methods for computing CDCs of a node in the context of both a window and the whole network were implemented in the package MVSIS [26]. The following experiments were done on a Windows XP computer with a 1.6-GHz central processing unit (CPU) and 1 GB of random-access memory (RAM), although less than 256 MB of RAM was needed for the largest benchmarks in Table I. The resulting networks were verified using a SAT-based verifier in MVSIS designed along the lines of [10], [15], and [18].

TABLE II
NETWORK OPTIMIZATION USING CDCS, WINDOWING, AND SAT

Name	In/Out/Latch	Literals in factored forms			Runtime, s	
		sweep	mfsw22	script22	mfsw22	script22
b14	32 / 54 / 245	17388	10664	7911	3.9	18.0
b15	36 / 70 / 449	16244	15056	10948	6.1	22.9
b17	37 / 97 / 1415	57311	49067	37877	35.7	104.8
b20	32 / 22 / 490	35149	21826	16813	7.6	55.0
b21	32 / 22 / 490	35908	22312	16932	9.3	51.1
b22	32 / 22 / 735	52276	33017	25174	13.5	59.8
s15850	14 / 87 / 597	7303	6350	4033	1.2	4.0
s35932	35 / 320	24408	20248	10986	4.2	16.7
s38417	28 / 106	18699	17327	13640	4.5	15.5
pj1	1769 / 1063/0	34828	30547	18076	9.5	37.0
pj2	690 / 429/0	7422	6464	3457	1.1	4.0
Ave		**1.00**	**0.79**	**0.54**	**1.00**	**4.36**

Table I compares the runtime of the don't care computations, using BDDs and S&S for windows of different sizes. The benchmarks used were the largest ITC'99 benchmarks [11] (b-files), the largest sequential circuits from the ISCAS benchmarks [40] (s-files), and the combinational logic extracted from cores of the PicoJava microprocessor [36] (pj-files). The second column of Table II gives the numbers of inputs, outputs, and latches in the selected benchmarks.

Three window sizes were considered (1×1, 2×2, and 4×4). In each case, the runtimes (in seconds) of the BDD-based computation ("BDDs") and the S&S-based computation ("S&S") are reported. It was formally verified that the CDCs computed in each case by BDDs and S&S using the same window were identical. The last row in Table I shows the average of the ratios S&S/BDD runtimes in all cases.

The measurements in Table I are not exactly comparable due to different pruning criteria employed by the two methods. One pruning technique uses window rescaling, which reduces the scope of a window if its size exceeds a predefined limit. For example, if a 4×4 window turns out to be too large, it is automatically replaced by a 3×3 window. For BDDs, the window is rescaled if it has more than 30 leaves or more than 15 roots, while for SAT, the window is rescaled if it contains more than 500 AND gates after structural hashing.

Table I indicates that the S&S-based computations are faster and scale better than the BDD-based ones. For 1×1 windows, S&S is, on average, 20% faster, for 2×2 windows, over two times faster, while for 4×4 windows, over seven times faster. This ratio increases further with the window size. The larger the window, the more CDC flexibility for the node. Thus, window sizing provides a tradeoff between runtime and optimization potential.

It is worth noting that the combination of simulation and BDD does not benefit CDC computation. To compute the remaining care minterms not detected using simulation, the general BDD image computation is still needed, which is just as hard as computing all the care minterms.

Table II shows the results of network optimization using the S&S-based flow for the benchmarks from Table I. These benchmarks are relatively large. As a result, BDD-based methods, full_simplify in SIS and mfs in MVSIS without windowing, cannot be applied.

The first column of Table II lists the benchmark names; the second column shows the numbers of inputs, outputs, and latches; the next three columns contain the numbers of literals in the factored forms in 1) the original benchmark after sweeping ("sweep"), 2) after applying mfs with 2×2 windowing ("mfsw22"), and 3) as part of a script ("script22"); the last two columns show the runtimes in seconds for the two optimization options. The script used in this experiment is script22. This is similar to script.rugged in SIS, with full_simplify (CODC computation implemented using BDDs) replaced by mfs with 2×2 windows (mfs–w 22, CDC computation using S&S).

Table II demonstrates that the proposed don't-care-based optimization flow can be applied to large circuits. This is because the don't care computation is performed in a window, and is, therefore, local and does not depend on the circuit size. The overall runtime scales well with the problem size and is predictable; a rule of thumb is for mfs–w 22, the computation takes about 1 s per 3000 literals in the original netlist.

These two experiments demonstrate that, compared to the BDD-based approach, the proposed method enhances optimization quality, reduces runtime, and provides robustness and scalability for large problems. Thus, the computation of internal don't cares becomes more affordable and applicable to large industrial networks.

IV. COMPUTING SPFDS

SPFDs [31], [39] express a different type of flexibility of nodes in a multilevel network. They offer greater flexibility than CDCs by allowing the node function and functions in its TFO to change. SPFDs contain CDCs as a special case but are not comparable with the notion of multioutput Boolean relations [38]. It has been proven [35] that "minimum" SPFDs provide more optimization power than CDCs [24], rewiring [6], and ATPG techniques [16]. Some interesting applications of the minimum SPFDs in logic synthesis include node optimization using the windowing concept proposed in Section III [34], and combined rewiring and mapping during resynthesis optimizations [32]. This section proposes an efficient way of computing SPFDs using S&S applied to the circuit representation of Boolean functions.

A. Background

Consider two networks N and N' with identical structure. Let X and X' be their respective PIs, and Z and Z' be their respective POs. For corresponding nodes n and n' in network N and N', denote their output variables to be y and y' and fanins Y and Y'. Let the local and global functions at n and n' be $y = f(Y)$, $y = g(X)$ and $y' = f(Y')$, $y' = g(X')$, respectively.

Definition 4.1: The global SPFD of node n, $\text{SPFD}^n(X, X')$, specifies that the minterms in the care ON set of n have to be distinguished from the minterms in the care OFF set of n. An SPFD is a Boolean function over the product of PI spaces $X \times X'$ and is computed as

$$\text{SPFD}^n(X, X') = g(X) \oplus g(X'). \qquad (4)$$

Similarly, the local SPFD of node n is

$$\mathrm{SPFD}(Y, Y') = f(Y) \oplus f(Y'). \tag{5}$$

For example, the local SPFD of a two-input OR gate is {(00,01), (00,10), (00,11)}, i.e., the OFF set minterm (00) has to be distinguished from all the ON set minterms (01, 10, 11). Intuitively, a pair of minterms (x, x') belonging to the SPFD of a node can be thought of as an elementary unit of information distinguished by the node, while the total SPFD of a node (the set of pairs of input minterms that it can distinguish) represents the information processing capability of the node.

Definition 4.2: A cut C of network N is a subset of nodes, such that every path from the PIs to the POs passes through at least one node in C.

Definition 4.3: A cut C is redundant if there exists $n \in C$ such that $C \setminus n$ is a cut. Otherwise, the cut is irredundant.

Definition 4.4: Let C be an irredundant cut containing node n. $C \setminus n$ is called a separator of n, denoted $\mathrm{Sep}(n)$.

Intuitively, information needs to propagate from PIs to POs, and a separator is a set of nodes such that the information (SPFDs) passing through them, combined with the information (SPFD) passing through node n, subsumes the information required at the POs.

Definition 4.5: $\mathrm{Sep}_2(n)$ dominates $\mathrm{Sep}_1(n)$ if the union of SPFDs of nodes in $\mathrm{Sep}_2(n)$ contains the union of SPFDs of nodes in $\mathrm{Sep}_1(n)$

$$\sum_{\alpha \in \mathrm{Sep}_1(n)} \mathrm{SPFD}^{\alpha}(X) \subseteq \sum_{\beta \in \mathrm{Sep}_2(n)} \mathrm{SPFD}^{\beta}(X). \tag{6}$$

$\mathrm{Sep}_1(n)$ is equivalent to $\mathrm{Sep}_2(n)$ if the union of SPFDs of nodes in $\mathrm{Sep}_1(n)$ is the same as the union of SPFDs of nodes in $\mathrm{Sep}_2(n)$.

Definition 4.6: The largest (smallest) separator of n, $\mathrm{Sep}_{\max(\min)}(n)$, is the separator that dominates (is dominated by) all other separators of n.

The largest separator of n is composed of all PIs not in $\mathrm{TFI}(n)$, plus the nodes in $\mathrm{TFI}(n)$ that have a fanout outside $\mathrm{TFI}(n)$. The smallest separator is composed of all POs not in $\mathrm{TFO}(n)$, plus any node with a fanout to $\mathrm{TFO}(n) \setminus n$. Intuitively, nodes located closer to the PIs have more information.

Definition 4.7: The minimum global SPFD of n with respect to a separator $\sigma = \mathrm{Sep}(n)$, $\mathrm{SPFD}^n_\sigma(X, X')$, is the set of pairs of minterms of the global SPFDs of the POs not contained in the SPFDs of nodes in the separator

$$\mathrm{SPFD}^n_\sigma(X, X') = \sum_{\alpha \in \mathrm{PO}} \mathrm{SPFD}^{\alpha}(X, X') \wedge \overline{\sum_{\beta \in \sigma} \mathrm{SPFD}^{\beta}(X, X')}. \tag{7}$$

Thus, minterm pair (x, x') belongs to the minimum global SPFD of node n with respect to some separator σ if x and x' are distinguished by at least one PO, but not by any of the nodes in σ. Intuitively, σ is a barrier through which information must flow to get to the $\mathrm{PO} \in \mathrm{TFO}(n)$, and the minimum global SPFD is the information necessarily provided by n, not available at any node of σ. Note that the minimum SPFD is always defined with respect to a particular separator σ, however, use of the largest separator will guarantee the smallest minimum SPFD at a node.

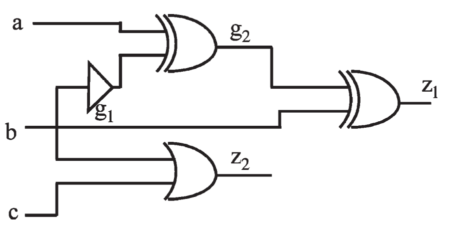


Fig. 5. Example circuit.

Definition 4.8: The minimum local SPFD of n with respect to a separator $\sigma = \mathrm{Sep}(n)$, $\mathrm{SPFD}^n_\sigma(Y, Y')$, is the image of $\mathrm{SPFD}^n_\sigma(X, X')$ in the local space of node n

$$\mathrm{SPFD}^n_\sigma(Y, Y') = \exists_{X,X'} \mathrm{SPFD}^n_\sigma(X, X') \wedge M(X, Y) \wedge M(X', Y') \tag{8}$$

where $M(X, Y)$ and $M(X', Y')$ are mappings from the PI spaces X and X' into the fanin spaces Y and Y' of nodes n and n' in networks N and N'.

Example 4.1: Consider the circuit shown in Fig. 5. The largest separator of node g_1 is $\{a, b, c\}$. The global SPFD of z_1 is equal to $\{(1{-}{-}, 0{-}{-})\}$ (the minterms are in the form of abc). The global SPFDs of a, b, and c are $\{(1{-}{-}, 0{-}{-})\}$, $\{({-}1{-}, {-}0{-})\}$ and $\{({-}{-}1, {-}{-}0)\}$, respectively. All the minterm pairs of the global SPFD of z_1 are contained in the global SPFD of a. Hence, the minimum SPFD of g_1 with respect to the largest separator is empty. However, the node is neither $s{-}a{-}0$ nor $s{-}a{-}1$ redundant. This is because nodes in the TFO of g_1 depend on the specific information flow through node g_1.

The usefulness of a minimum SPFD is that the current function at a node n can be replaced by any function that contains the node's minimum SPFD. However, after the replacement, in order to preserve the functions of the POs, the functions on the output side of σ may need to be changed. The property of minimum SPFDs guarantees that the new functions of these nodes can always be derived. Note that the use of the largest separator has the maximum number of nodes that might have to be changed.

If a minimum SPFD of n is empty, then n does not provide any "useful" information that is not already supplied by other nodes in σ. Therefore, n can be removed, while other nodes can be resynthesized, keeping the network's behavior unchanged. The procedures for resynthesizing the nodes using their minimum SPFDs are described in [31].

B. Computing SPFDs Using S&S

BDD-based SPFD computations limit the applicability of SPFDs to medium-sized circuits [31]. In [33], an improvement was proposed, which computes SPFD of a node by putting together two copies of the logic cone of the node. The circuit representation of these cones is translated into CNF and given to the SAT solver enhanced with the capability to exhaustively enumerate the satisfying assignments. The set of all satisfying assignments produced by the SAT solver is the SPFD of the

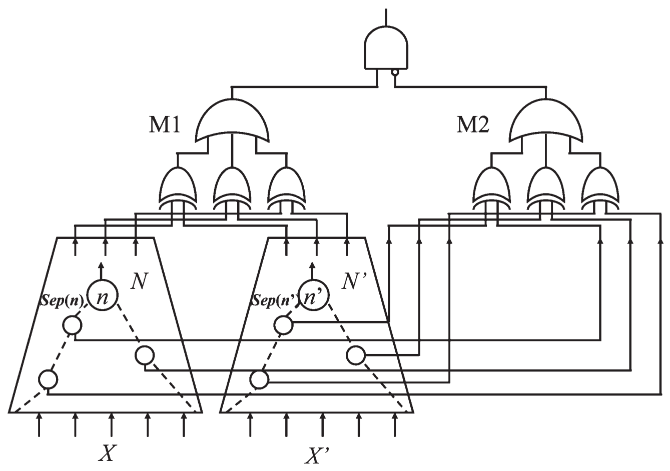


Fig. 6. Specialized miter for computing SPFDs.

TABLE III
RUNTIME STATISTICS OF S&S-BASED VERSUS BDD-BASED MINIMUM SPFD COMPUTATION

Circuit	#Nodes	BDD(s)	S&S(s)	Gain	Stopped (BDD)	Stopped (SAT)
dalu	1131	>4000	291.4	>13.7	737	661
frg2	522	1769.2	119.4	14.9	75	135
pair	824	323.9	97.8	3.3	289	119
C499	202	44.9	14.9	3.0	202	133
C880	357	3515.1	20.1	174.9	245	87
C1355	514	1813.6	64.2	28.2	514	314
C1908	880	3980.4	177.7	22.4	638	492
C3540	1667	>4000.0	554.4	>7.2	1663	1192
C6288	2416	>4000.0	2446.7	>1.6	2416	2312
C7552	3266	>4000.0	1651.4	>2.4	3256	1363
C5315	2288	>4000.0	798.7	>5.1	2247	0
i10	2488	>4000.0	1070.3	>3.7	2337	1117
Ave.				> 23.0		

given node. Simulation was not used and the problem formulation was not tuned for SAT. As a result, although the method of [33] can handle larger circuits than previous approaches based entirely on BDDs, its runtime is also larger. In this paper, we propose a much more efficient algorithm to compute SPFDs using S&S applied to the circuit representation of Boolean functions, offering better runtimes than previous approaches and with application to larger circuits.

The minimum SPFD computation for a node begins by building a miter for the node. The following steps describe how the miter is constructed and used in the S&S-based minimum SPFD computations.

1) Construct the miter cone as shown in Fig. 6. The miter cone feeds PIs X into the first network N and PIs X' into the second network N'. Identical separators σ of the nodes n and n' are introduced in both networks. The POs are connected pairwise to form miter cone M1. The outputs of the nodes in the two separators are connected pairwise to form the miter cone M2. The output of M2 is one if and only if the separator can distinguish a PI minterm pair (x, x'). Finally, the M1 output is ANDed with the complement of the M2 output. As a result, the final output is one if and only if the POs of the network can distinguish (x, x'), but the separator cannot distinguish it. By construction, the final output is one only for the PI minterm pairs that belong to the minimum SPFD of n with respect to separator σ.
2) Perform simulation on the miter cone network. Assign random simulation vectors at the PIs and propagate them to the POs. Each pair of simulated minterms x and x' in the PI space has a corresponding pair $(y,\, y')$ in the local space Y and Y'. If the output of the network is one, then (y, y') belongs to $\text{SPFD}_{\sigma}^{n}(Y, Y')$. A static simulation model is used where a fixed number of patterns are simulated. We found that simulating 512 random patterns works well for most benchmarks.
3) Convert the miter cone network into a SAT instance. The local SPFD minterms (in Y, Y') already computed (which belong to the minimum SPFD) are complemented and added to the SAT instance as blocking clauses, and SAT is used to enumerate through the remaining satisfying (Y, Y') assignments. When SAT returns "unsatisfiable," the complete minimum SPFD of the node is obtained.

Some practical applications require checking the equivalence of the SPFDs of two separators, or of two multioutput networks with the same PIs but possibly different POs. To check the equivalence of SPFDs of the networks, it is enough to construct for each network a miter similar to miter M1 in Fig. 6. The outputs of the miters are combined using an additional EXOR gate. The resulting "miter of miters" is constant 0 if and only if the multioutput networks have the same SPFDs. The problem can be solved using an SAT solver.

C. Experiment Results

Both the S&S-based and BDD-based computations of the minimum SPFDs for the smallest separator were implemented in SIS [30]. The smallest separator is more useful for network optimization, because it allows for a more efficient control of changes performed on the network after resynthesis of the node. These changes are limited to the TFO of the node resynthesized. A detailed discussion about limiting the area of change and modifications to the fanouts even further can be found in [31].

For S&S, the SPFD miter was generated for each node in the circuit and the minimum local SPFD computed. The BDD-based computation used BDDs exclusively, including building the global SPFDs of the PO nodes and the nodes of the separator, constructing the minimum global SPFD of the node, and the image computation.

In most examples, the limiting factor using BDDs was the construction of the minimum global SPFD of the node. This implies that a combination of simulation and BDDs is not suitable for computing the minimum SPFDs. Even if all the minterm pairs in a minimum SPFD can be enumerated using simulation, the BDD-based computation for ensuring the completeness of the minimum SPFD computation could still encounter the same blowup problems.

The experiments were performed on a 400-MHz UltraSparc II with 4.0 GB of RAM. Table III compares the S&S-based

and the BDD-based computations of the minimum SPFDs for each node in the network. Columns 1 and 2 report the name of the benchmark and the number of nodes, respectively. Since the BDD-based computation cannot be applied to very large circuits, we used a randomly selected subset of medium-sized MCNC benchmarks. Columns 3 and 4 report the runtimes in seconds for the BDD-based and the S&S-based implementations, respectively. The overall time limit was set to 4000 s for each circuit. The gain in runtime ration (BDD/S&S) is reported in Column 5. Thus, S&S demonstrates an average performance improvement of at least 23× compared to BDDs.

In general, the improvement due to S&S is even larger, since the runtime of the BDD-based implementation for some of the larger circuits exceeds the timeout limit. The BDD-based implementation is very slow for large circuits, and, hence, limitations on the size of the intermediate BDDs had to be introduced. The largest size to which an intermediate BDD was allowed to grow was set to 50 000 nodes. To attempt a similar limitation on the SAT solver, the maximum number of backtracks was set to 200. If the resource limit (size of BDD or number of SAT backtracks) is reached in computing the SPFDs of a node, the program abandons its computation and moves on to a different node until the total 4000-s runtime limit is reached. The number of network nodes for which these resource limits were reached is reported in Columns 6 and 7 for BDDs and S&S, respectively. In addition, Column 6 includes the number of nodes that could not be processed when the total runtime of the BDD-based computation exceeded the 4000-s timeout. These numbers clearly demonstrate that the S&S-based computation can process a larger number of nodes than the BDD-based implementation for the same benchmark, even though we put a limit on the number of SAT backtracks.

It is worth pointing out that the current implementation of the algorithm cannot be applied to sequential circuits, which tend to have more shorter paths with smaller support set. As a result, the BDD-based method may not suffer as much when applied to sequential circuits. However, we feel that S&S will still perform better than BDD. We hope to collect supporting results in our future work.

The results in Table IV summarize the contribution of simulation to the S&S-based minimum SPFD computation. Column 1 reports the name of each circuit. Column 2 reports the total number of minterm pairs in the minimum SPFDs of the nodes in the circuit. The number of minterm pairs identified by simulation is reported in Column 3. The runtimes of simulation and SAT in the S&S-based implementation are presented in Columns 4 and 5, respectively. The results show that simulation can identify about 40% of all minterm pairs in a minimum SPFD and contributes to a good fraction of the total runtime. Thus, simulation plays a significant role in the S&S-based minimum SPFD computation.

The two experiments demonstrate that S&S is better suited than a pure BDD-based algorithm for the efficient computation of minimum SPFDs, especially for large circuits. In addition, we repeat that a combined simulation and BDD-based algorithm would suffer from a lot of the disadvantages of a pure-BDD-based algorithm, because the primary bottleneck in the minimum SPFD computation is in computing the global SPFDs of the separator nodes and the PO nodes. Another advantage of the proposed method is that the efficient use of minimum SPFDs opens up the possibility of their use in an optimization framework for realistic designs. In previous work, only subsets of the minimum SPFD could be used for optimization. Future work might investigate the impact of the added flexibility provided by the minimum SPFDs in optimization framework.

TABLE IV
CONTRIBUTION OF SIMULATION TO S&S-BASED MINIMUM SPFD COMPUTATION

Circuit	#Total minterm pairs	#Minterm pairs using SIM	SIM Runtime (s)	SAT Runtime (s)
dalu	3088	1329	5.8	288.0
frg2	8982	2867	35.4	84.6
pair	8450	3519	20.2	78.5
C499	458	178	2.4	12.2
C880	1584	697	4.7	15.2
C1355	596	238	5.9	57.9
C1908	2542	664	10.3	167.3
C3540	1278	615	10.2	540.4
C6288	324	162	20.3	2256.2
C7552	8758	3771	31.1	1593.9
C5315	5600	2522	22.6	764.3
i10	9896	2935	36.7	1025.7

V. Computing Resubstitutions

Resubstitution plays an important role both in technology-independent [30] as well as technology-dependent [17] logic synthesis. This section describes an S&S-based algorithm to compute sets of nodes for Boolean resubstitution. In particular, we focus on speeding up resubstitution used in the resynthesis flow of [14].

A. Background

Definition 5.1: Resubstitution of a node in a network replaces the node's local function by a new local function, which depends on a different set of fanins, but does not alter the global functionality of the node.

Resubstitution can be used to restructure the network to minimize delay, area, routeability, etc. Delay can be improved if the new fanin(s) arrive earlier and routeability can be improved if they are closer than those replaced. Area can be improved if after resubstitution, some of the old fanins of the node have no fanouts and, therefore, can be removed from the network.

The existence of a resubstitution is closely related to the concept of functional dependency [12]. Resubstitution functions can also be computed using interpolation [20]. The definitions and the theorems below are taken from [12]. We give a new computation procedure that relies on S&S rather than BDDs, as in [12].

Definition 5.2: Given node n with global function $g(X)$ and nodes $m_1, m_2, \ldots, m_k$ with global functions $y_{m_1}(X)$,

TABLE V
TRUTH TABLE OF G AND CANDIDATE SETS

$a\ b\ c$	g	Set 1		Set 2	
		$y_1 = \overline{a}b$	$y_2 = a\overline{b}c$	$y_3 = (a+b)$	$y_4 = bc$
000	0	0	0	0	0
001	0	0	0	0	0
010	1	1	0	1	0
011	1	1	0	1	1
100	0	0	0	1	0
101	1	0	1	1	0
110	0	0	0	1	0
111	0	0	0	1	1

$y_{m_2}(X), \ldots, y_{m_k}(X)$, nodes $m_1, m_2, \ldots, m_k$ can resubstitute node n if the global function of n can be expressed as

$$g(X) = F\left(y_{m1}(X), y_{m2}(X), \ldots, y_{m_k}(X)\right) \tag{9}$$

where $F(y_1, y_2, \ldots, y_k)$ is a Boolean function called a resubstitution function.

Theorem 5.1: Nodes $m_1, m_2, \ldots, m_k$ with global functions $y_{m1}(X), y_{m2}(X), \ldots, y_{m_k}(X)$ can resubstitute node n with global function $g(X)$ if and only if there is no minterm pair (x_1, x_2), such that $g(x_1) \neq g(x_2)$ but $y_{mj}(x_1) = y_{mj}(x_2)$, for all j, $1 \leq j \leq k$.

Thus, resubstitution is possible if and only if the distinguishing power of g (output) is not greater than the union of the distinguishing powers of all the functions y_{mj} (inputs).

Example 5.1: Suppose the global function of node n is $g = (a \oplus b)(b \vee c)$, where a, b, and c are the PIs. Consider two sets of resubstitution candidates with global functions: ($y_1 = \overline{a}b$, $y_2 = a\overline{b}c$) and ($y_3 = a \vee b$, $y_4 = bc$). Table V shows the truth tables of all the global functions. The set (y_3, y_4) is not a valid resubstitution candidate for g, because minterm pair (101, 110) is distinguished by g but not distinguished by y_3 and y_4. On the other hand, the set (y_1, y_2) satisfies Theorem 5.1, because all the minterm pairs distinguished by g are also distinguished by at least one function in the set.

Theorem 5.2: Let $Y = \{y_1, y_2, \ldots, y_k\}$. Node n with global function $g(X)$ can be resubstituted using nodes $m_1, m_2, \ldots, m_k$, each with global functions $y_{m1}(X), y_{m2}(X), \ldots, y_{mk}(X)$, if and only if $F^{\mathrm{ON}}(Y)F^{\mathrm{OFF}}(Y) = 0$, where

$$F^{\mathrm{OFF}}(Y) = \exists_X \left[\overline{g}(X) \wedge \prod_{j=1}^{k} y_j \equiv y_{mj}(X)\right] \tag{10}$$

$$F^{\mathrm{ON}}(Y) = \exists_X \left[g(X) \wedge \prod_{j=1}^{k} y_j \equiv y_{mj}(X)\right]. \tag{11}$$

If this property holds, then any function $F(Y)$ that agrees $(F^{\mathrm{ON}}(Y) \subseteq F(Y) \subseteq \overline{F^{\mathrm{OFF}}(Y)})$ with the above incompletely specified function (represented by its ON set and OFF set) can be a new resubstitution function for node n.

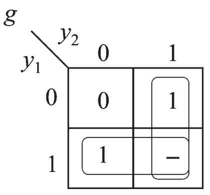


Fig. 7. Resubstitution function for g.

Example 5.2: Using the same functions as in the above example, we compute the ON set and the OFF set based on Theorem 5.2 with respect to local nodes y_1 and y_2: $F^{\mathrm{OFF}}(Y) = \overline{y}_1\overline{y}_2$ and $F^{\mathrm{ON}}(Y) = \overline{y}_1 y_2 \vee y_1\overline{y}_2$, as shown in Fig. 7. Minimizing this ISF, we derive the simplest resubstitution function for node n using candidates y_1 and y_2: $g = y_1 \vee y_2 = \overline{a}b \vee a\overline{b}c$.

B. Computing Resubstitution Using S&S

Given a node and a set of resubstitution candidate sets derived using the structural support of $g(X)$, the goal is to determine those candidate sets that can be used for resubstitution. The following steps describe the computation of feasible candidate sets.

1) Assign pairs of random vectors at the PIs corresponding to minterms x_1 and x_2. Perform simulation of the network and compare the outputs for functions g and resubstitution candidate sets $\{y_1, y_2, \ldots, y_k\}$. Using Theorem 5.1, eliminate the candidate sets for which resubstitution does not exist. Repeat random simulation for a predetermined number of rounds. Experimentally, we found that 64 simulation rounds (32-bit patterns each) works well in practice.
2) Apply SAT to each of the remaining candidate sets. SAT is used to compute the ON set and OFF set of F (the images) in Theorem 5.2 by enumerating the satisfying assignments not found by simulation. (The combinations appearing during simulation at the root node and the candidate nodes are added to the solver as blocking clauses.) If the images computed by SAT satisfy Theorem 5.2, resubstitution exists, and the resubstitution function is found by minimizing the incompletely specified function represented by the ON set and the OFF set.

C. Experimental Results

S&S-based and BDD-based resubstitution algorithms were implemented in the resynthesis package used to improve the quality of standard-cell technology mapping in MVSIS [26]. The experiments were run on a Pentium 4 computer with a 1.8-GHz CPU and 512 MB of RAM. Several randomly selected large benchmarks from MCNC [40], ITC [11], ISCAS [4], and PicoJava [36] benchmark suites demonstrate the applicability of the proposed algorithm to a large variety of circuits. The benchmarks were read into MVSIS followed by technology mapping [7], using a gain-based standard-cell library derived from mcnc.genlib [30], by ignoring the load-dependent part of the delay. Next, the netlist was resynthesized to reduce area,

TABLE VI
RUNTIME COMPARISON OF S&S-BASED VERSUS BDD-BASED RESUBSTITUTION COMPUTATION

Name	In/Out/ Latch	BDD (s)	Simulation & Sat (s)			Gain
			SIM	SAT	Total	
dalu	75/16/0	78.74	1.38	2.12	3.50	22.5
des	256/245/0	104.66	3.48	12.75	16.23	6.5
frg2	143/139/0	30.78	1.57	2.42	3.99	7.7
i10	257/224/0	172.99	2.24	2.34	4.58	37.8
k2	45/45/0	53.41	1.99	9.38	11.37	4.7
pair	173/137/0	75.44	1.52	1.27	2.79	27.0
C432	36/7/0	83.00	0.36	0.29	0.65	127.7
C2670	233/140/0	48.04	0.69	1.28	1.97	24.4
C5315	178/123/0	109.95	1.13	1.86	2.99	36.8
C7552	207/108/0	145.04	2.68	6.11	8.79	16.5
s15850	14/87/597	170.60	2.80	3.45	6.25	27.3
s35932	35/320/1728	40.45	1.40	1.28	2.68	15.1
pj1	1769/1063/0	163.90	3.98	4.82	8.80	18.6
b14	32/54/245	173.46	1.72	2.86	4.58	37.9
b17	37/97/1414	274.01	2.46	3.93	6.39	42.9
b20	32/22/490	166.99	2.46	6.17	8.63	19.4
b22	32/22/703	169.49	1.81	3.98	5.79	29.3
Ave.						**29.5**

TABLE VII
RESYNTHESIS STATISTICS

Name	Run Time (s)	Area Imp. (%)	Total cand. sets	DSIM (%)	DSAT (%)	PSAT (%)	USAT (%)
dalu	4.82	13.96	417,162	94.68	1.12	4.20	0.00
des	17.74	2.94	731,665	89.32	9.82	0.83	0.02
frg2	4.53	15.41	341,118	92.61	0.04	7.33	0.01
i10	7.61	7.58	825,859	97.35	1.31	1.33	0.01
k2	11.12	8.09	313,343	83.21	14.14	2.65	0.00
pair	3.23	12.35	399,039	96.06	0.00	3.94	0.00
C432	0.84	7.68	97,375	96.89	1.79	1.32	0.00
C2670	2.18	20.98	150,979	93.44	2.62	3.86	0.08
C5315	3.86	13.15	369,656	96.21	0.25	3.62	0.01
C7552	9.87	15.83	578,142	92.12	3.73	4.10	0.05
s15850	9.75	11.45	825,209	95.16	2.64	2.20	0.01
s35932	5.71	3.21	451,714	96.79	0.00	3.21	0.00
pj1	66.79	10.29	3,988,250	95.00	2.99	2.00	0.01
b14	25.56	5.17	2,367,501	96.64	2.36	0.97	0.02
b17	180.67	8.44	9,050,793	93.20	5.73	1.05	0.02
b20	61.93	7.52	5,672,596	96.95	1.64	1.38	0.03
b22	101.04	7.14	8,130,898	96.69	2.00	1.26	0.05
Ave.		**10.07**		**94.25**	**3.07**	**2.66**	**0.02**

with a total runtime limit of 3 min taken for all nodes in the network.

During the resynthesis step, the runtime of S&S-based and BDD-based resubstitution was compared. The resubstitution runtimes include only the time to derive the final valid candidate sets, but not the time to select the initial candidate sets or to perform the actual resubstitution after a valid set is chosen. The results of both computations are verified against each other. Table VI contains the experimental results. Columns 1 and 2 list the names and the characteristics of each benchmark. Column 3 lists the runtime of the BDD-based approach. Columns 4, 5, and 6 list the runtimes of the S&S-based approach (simulation, SAT, and total, respectively). Table VI shows an average 29× performance improvement of the S&S-based approach compared to the BDD-based approach.

To give more insight into the S&S-based resubstitution in these experiments, additional statistics are given in Table VII. Column 2 gives the total resynthesis runtime. Since the runs were limited to 3 min, resynthesis did not complete for benchmark b17. However, the area improvements and the total number of resubstitutions are valid. Column 3 is the percentage of area improvement due to resynthesis. The total number of sets of candidates considered (over all nodes) is reported in Column 4. Column 5 (DSIM) and Column 6 (DSAT) give the percentages of the candidate sets that were filtered out by simulation and SAT, respectively. Column 7 (PSAT) reports the percentages of the candidate sets that SAT proved useful for substitution. The last column (USAT) is the percentage of candidate sets not proved by SAT due to a resource limit on the number of backtracks. Table VII shows that an average 94% of the candidate sets are filtered out by simulation, indicating that simulation plays a significant role in this application of S&S.

Tables VI and VII together demonstrate the power of S&S for solving a computationally hard problem in logic synthesis.

TABLE VIII
RUNTIME COMPARISON BETWEEN SIM + SAT VERSUS SIM + BDD

Name	SIM (sec)	SAT (sec)	BDD (sec)	Gain SAT/BDD
dalu	1.31	2.62	2.57	1.02
des	3.13	13.4	9.02	1.49
frg2	1.72	1.93	1.76	1.10
i10	2.43	3.27	7.53	0.43
k2	1.78	8.55	7.63	1.12
pair	1.21	1.06	1.53	0.69
C432	0.23	0.12	0.82	0.15
C2670	0.52	1.27	1.91	0.66
C5315	1.03	1.84	2.18	0.84
C7552	2.11	5.82	14.42	0.40
s15850	2.87	5.02	6.39	0.79
s35932	1.04	1.30	0.67	1.94
pj1	13.21	28.61	41.09	0.70
b14	7.40	12.37	21.59	0.57
b17	17.54	48.02	87.11	0.55
b20	16.26	24.88	52.87	0.47
b22	23.26	41.29	78.98	0.52
Ave				0.79

We also performed an experiment to compute resubstitutions using simulation and BDDs (S&B), instead of SAT. BDDs are used to check the validity of the remaining 6% candidate sets after simulation, in contrast to Table VI, where BDDs are used to check all candidate sets. Table VIII contains the runtime information. Column 2 is the common simulation runtime, used in S&S and S&B. SAT and BDD runtimes are given in Columns 3 and 4, respectively. Column 5 shows the performance improvement between SAT and BDDs. On the set of 17 benchmarks, SAT wins in 12 cases, mostly in larger circuits. The average SAT/BDD runtime ratio is 0.79, demonstrating that SAT wins over BDDs in performance alone, without the advantages of simulation.

VI. Implementation Details

A. Simulation

The more efficient simulation is, the larger is the part of the search space it covers in a short time, leaving SAT to work on the remaining part. To ensure efficient simulation, in our implementation, we employed the following techniques.

1) Using AIGs [15] as our network representation. AIGs are compact and homogeneous. Simulating an AIG node involves bitwise operations on the simulation information of the fanins.
2) Performing simulation in a bit-parallel fashion, that is, simulating 32 or 64 minterms simultaneously.
3) Allocating memory for storing the simulation information in one large memory array. The nodes as well as the chunks of memory associated with the nodes are ordered topologically in a DFS order. This reduces the number of cache misses and improves the speed of simulation-intensive applications.
4) In CDC computation, simulation stopped upon saturation, i.e., no care minterm turns up after a fixed number of simulation rounds. In computing SPFD and node resubstitution, however, a static simulation scheme was used. This was done because it is simple and works quite well in practice.

Both bit-parallel simulation and AIGs lead to a significant speedup in computation. The impact of controlling the amount of simulation depends on each problem instance, but the above general principles work for a variety of applications and benchmarks.

Simulation is an efficient technique, because, in most applications, it has a linear complexity in the size of the circuit and in the number of patterns simulated. In one of the applications, computation of SPFDs, simulation has a worst case quadratic complexity in the number of patterns, because pairs of simulation patterns must be considered. In practice, the worst case complexity is reduced by computing equivalence classes of simulation patterns. Two patterns belong to the same class if they produce the same values at all POs. When we compute the SPFDs, we considered only pairs of patterns inside each equivalence class. This reduced the number of pairs considered, but the complexity is still higher than linear. This observation explains why the impact of simulation on the runtime of S&S is less pronounced for SPFDs, compared to other applications.

B. Satisfiability

The SAT solver used in these applications was implemented using an "extensible SAT solver," MiniSat [8]. MiniSat is very efficient despite its small size [600 lines of C++ code written without a standard template library (STL)]. In our experiments, it outperformed several popular SAT solvers. Moreover, the implementation of MiniSat is easy to understand and modify, in complete agreement with the intention of its developers.

For the applications described in this paper, MiniSat was modified to treat satisfying assignments similar to how conflicts are treated. In this case, when a new assignment is found, it is added to the list of satisfying assignments, and a blocking clause is generated, which prevents it from appearing again. After this, a nonchronological backtracking is performed to the lowest level allowed by the blocking clauses. This technique makes enumeration of satisfying solutions more efficient, compared to an implementation that restarts the solver at the topmost level each time a new satisfying assignment is found.

A new SAT instance is constructed for each new problem, such as computing the don't cares for a node or checking resubstitution using a subset of nodes. The variables and clauses are added to the solver in the order corresponding to a DFS order of nodes in the network or the window considered. No static variable ordering was used, because MinSat [8] dynamically determines a good variable ordering by applying the variable state independent decaying sum (VSIDS) heuristic [25]. This heuristic focuses the solver on the relevant parts of the search space and is relatively cheap to realize. In our experiments, updating variable activities and computing next branching variable took about 20% of the solver runtime.

C. Integration of Simulation and SAT

The information from the simulation to the SAT solver is passed using additional SAT clauses. In the naïve implementation, each pattern found by simulation is transformed into a clause and added to the solver. In this case, the number of additional clauses can be close to the number of minterms in the input space. If the input space of a problem is large (say, ten inputs), it leads to a large number of clauses, which consume memory and slow down the solver. This motivates the need to compact the additional clauses.

In our implementation of CDC computation, the clauses are compacted by deriving the Boolean function of all satisfying assignments found by simulation, transforming this function into an irredundant sum of product (ISOP) [8], [21], and deriving the clauses from the ISOP using DeMorgan's rule. This led to a substantial speedup in SAT. In the other two applications (SPFDs and resubstitution), the improved implementation did not give an advantage over the naïve one.

Depending on the application, the role of simulation and SAT in solving the problem is different. Typically, a problem can be solved with SAT alone without simulation at the cost of increases in runtime. Among the three applications considered, SPFD computation is the least sensitive to using simulation. Disabling simulation for SPFDs increases runtime roughly by 50% on average. In the other applications, CDCs and resubstitution, the runtime of SAT-only solution typically increases an order of magnitude or more compared to the runtime of S&S.

VII. Conclusion and Future Work

Flexibilities in a Boolean network can be computed and used to transform logic functions of nodes without changing the PO functions. This process leads to more optimization, compared to that based on faster but suboptimal algebraic methods [3]. Although several formalisms for modeling flexibilities are known, the computational cost is often very high, which makes the computations impractical for mainstream logic synthesis. In this paper, we revisited three algorithms for computing flexibilities, CDCs, SPFDs, and resubstitutions, and proposed more efficient formulations and implementations based on simulation and Boolean SAT. Experimental results show that the new implementations are significantly better than the previous ones based on BDDs. The following observations can be made.

1) In computing CDCs, S&S is, on average, 20% faster than BDDs for 1×1 windows, over two times faster for 2×2 windows, and over seven times faster for 4×4 windows. This ratio increases further with the window size.
2) In computing SPFDs, S&S is, on average, at least 23 times faster than BDDs.
3) In computing resubstitutions, S&S is, on average, 29 times faster than BDDs. Even if simulation is coupled with the BDD-based computation, SAT still outperforms BDDs.

Future work may include developing new applications of CDCs and SPFDs in technology-independent optimization and technology mapping. Several improvements to the windowing algorithm will be investigated, which will allow for more flexibilities to be computed with smaller computational effort. Another interesting application is to develop an efficient S&S-based algorithm for negation and permutation of inputs, negation of outputs (NPN) matching of large Boolean functions for which BDD cannot be constructed.

References

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. DAC*, Orlando, FL, 1990, pp. 40–45.

[2] D. Brand, "Verification of large synthesized designs," in *Proc. ICCAD*, Santa Clara, CA, 1993, pp. 534–537.

[3] R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," in *Proc. ISCAS*, Rome, Italy, 1982, pp. 29–54.

[4] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. ISCAS*, Portland, OR, 1989, pp. 1929–1934.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[6] S.-C. Chang, L. P. P. P. van Ginneken, and M. Marek-Sadowska, "Circuit optimization by rewiring," *IEEE Trans. Comput.*, vol. 48, no. 9, pp. 962–970, Sep. 1999.

[7] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *Proc. ICCAD*, San Jose, CA, 2005, pp. 519–526.

[8] O. Coudert, J. C. Madre, H. Fraisse, and H. Touati, "Implicit prime cover com putation: An overview," in *Proc. SASIMI*, Nara, Japan, 1993, pp. 413–422.

[9] N. Eén, N. Sörensson, "An extensible SAT-solver," in *Proc. Theory Applications Satisfiability Testing (SAT)*, Santa Margherita, Ligure, Italy, 2003, pp. 502–518. [Online]. Available: http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/

[10] E. Goldberg, M. Prasad, and R. K. Brayton, "Using SAT for combinational equivalence checking," in *Proc. DATE*, Munich, Germany, 2001, pp. 114–121. [Online]. Available: http://eigold.tripod.com/

[11] F. Corno, M. Sonza Reorda, and G. Squillero, "RT-level ITC 99 benchmarks and first ATPG results," *IEEE Des. Test. Comput.*, pp. 44–53, Jul./Aug. 2000. [Online] Available: http://www.cad.polito.it/tools/itc99.html

[12] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *Proc. CAV*, Boston, MA, 2004, pp. 268–280.

[13] H.-S. Jin, H.-J. Han, and F. Somenzi, "Efficient conflict analysis for finding all satisfying assignment of a Boolean circuit," in *Proc. TACAS* , N. Halbwachs, L. Zuck, Eds., Edinburgh, U.K., 2005, vol. 3440, pp. 287–300.

[14] V. N. Kravets and P. Kudva, "Implicit enumeration of structural changes in circuit optimization," in *Proc. DAC*, San Diego, CA, 2004, pp. 438–441.

[15] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, Dec. 2002.

[16] W. Kunz, D. Stoffel, and P. Menon, "Logic optimization and equivalence checking by implication analysis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. CAD-16, no. 3, pp. 266–281, Mar. 1997.

[17] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 16, no. 8, pp. 813–833, Aug. 1997.

[18] F. Lu, L. Wang, K. Cheng, and R. Huang, "A circuit SAT solver with signal correlation guided learning," in *Proc. DATE*, Munich, Germany, 2003, pp. 892–897.

[19] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999.

[20] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. CAV*, Boulder, CO. Berlin, Germany: Springer-Verlag, 2003, vol. 2725, pp. 1–13.

[21] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Trans. Fundam.*, vol. E76-A, no. 6, pp. 973–976, Jun. 1993.

[22] A. Mishchenko and R. K. Brayton, "Simplification of non-deterministic multi-valued networks," in *Proc. ICCAD*, San Jose, CA, 2002, pp. 557–562.

[23] ——, "A theory of non-deterministic networks," in *Proc. ICCAD*, San Jose, CA, 2003, pp. 709–716.

[24] ——, "SAT-based complete don't-care computation for network optimization," in *Proc. DATE*, Munich, Germany, 2005, pp. 418–423.

[25] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. DAC*, Las Vegas, NV, 2001, pp. 530–535.

[26] MVSIS Group, *MVSIS: Multi-Valued Logic Synthesis System*, Berkeley: Univ. California. [Online]. Available: http://www-cad.eecs.berkeley.edu/mvsis/

[27] M. Prasad, P. Chong, and K. Keutzer, "Why is ATPG easy?," in *Proc. DAC*, New Orleans, LA, 1999, pp. 22–28.

[28] H. Savoj and R. K. Brayton, "The use of observability and external don't-cares for the simplification of multi-level networks," in *Proc. DAC*, Orlando, FL, 1990, pp. 297–301.

[29] H. Savoj, "Don't cares in multi-level network optimization," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, May 1992.

[30] E. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Tech. Rep. UCB/ERL M92/41, 1992. ERL.

[31] S. Sinha and R. K. Brayton, "Implementation and use of SPFDs in optimizing Boolean networks," in *Proc. ICCAD*, San Jose, CA, 1998, pp. 103–110.

[32] S. Sinha, S. Khatri, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Binary and multi-valued SPFD-based wire removal in PLA networks," in *Proc. ICCD*, Austin, TX, 2000, pp. 494–503.

[33] S. Sinha and R. K. Brayton, "Improved robust SPFD computations," in *Proc. IWLS*, Tahoe City, CA, 2001, pp. 156–161.

[34] S. Sinha, A. Mishchenko, and R. K. Brayton, "Topologically constrained logic synthesis," in *Proc. ICCAD*, San Jose, CA, 2002, pp. 679–686.

[35] S. Sinha, X. Wang, and R. K. Brayton, "Comparing two rewiring models," in *Proc. IWLS*, Temecula, CA, 2004, pp. 438–445.

[36] SUN Microelectronics, *PicoJava Microprocessor Cores*. [Online]. Available: http://www.sun.com/microelectronics/picoJava/

[37] M. N. Velev, "Efficient translation of Boolean formulas to CNF in formal verification of microprocessors," Tech. Rep. Jan. 2005. [Online]. Available: http://www.ece.cmu.edu/~mvelev/Velev_TechReport_R1.pdf

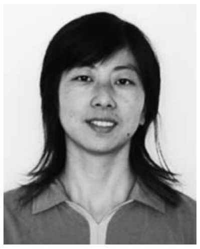
[38] Y. Watanabe, L. Guerra, and R. K. Brayton, "Logic optimization with multi-output gates," in *Proc. ICCD*, Cambridge, MA, 1993, pp. 416–420.

[39] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *Proc. ICCAD*, San Jose, CA, 1996, pp. 254–261.

[40] S. Yang, "Logic synthesis and optimization benchmarks," Version 3.0. Tech. Rep., Microelectron. Center North Carolina, Research Triangle Park, NC, 1991.

Alan Mishchenko (M'99) received the M.S. degree in applied mathematics and information technology from the Moscow Institute of Physics and Technology, Moscow, Russia, in 1993, and the Ph.D. degree in computer science from the Glushkov Institute of Cybernetics, Kiev, Ukraine, in 1997.

Since 1998, he has been a Research Scientist in the U.S. From 1998 to 2002, he was with the Portland State University, Portland, OR. Since 2002, he has been with the University of California, Berkeley. His research interests include developing computationally efficient methods for logic synthesis and verification.



Jin S. Zhang (M'93) received the B.S. degree in electrical engineering, the B.A. degree in English for science and technology, and the M.S. degree in electrical engineering from Tianjin University, Tianjin, China, in 1991, 1991, and 1994, respectively, and the M.S. degree in electrical and computer engineering from Portland State University, Portland, Oregon, in 2001. She is currently working toward the Ph.D. degree in electrical and computer engineering at Portland State University.

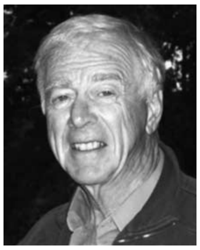
From 1995 to 2002, she was with Lattice Semiconductor Corp. and Cadence Design Systems and Real Intent, Inc., working on logic and layout verification.

Subarna Sinha received the B.Tech. (Hons.) degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India in 1996, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 2002.

She is currently working with the Advanced Technology Group, Synopsys, Inc., Mountain View, CA. Prior to that, she was a member of the Strategic CAD Laboratories at Intel. Her research interests include logic synthesis, physical design, and design for manufacturability.

Jerry R. Burch (M'92) received the B.S. and M.S. degrees in computer science from the California Institute of Technology, Pasadena, CA, in 1984 and 1985, respectively, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1992.

From 1992 to 1994, he was a Postdoctoral Scholar in the Computer Science Department, Stanford University. From 1994 to 2002, he was with Cadence Berkeley Laboratories. He is currently with the Advanced Technology Group, Synopsys, Inc., Hillsboro, OR.



Robert Brayton (M'75–SM'78–F'81) received the B.S. degree in electrical engineering from Iowa State University, Ames, in 1956, and the Ph.D. degree in mathematics from MIT, Cambridge, in 1961.

He was a member of the Mathematical Sciences Department, IBM T. J. Watson Research Center until he joined the Electrical Engineering and Computer Science Department, University of California, Berkeley, in 1987. He held the Edgar L. and Harold H. Buttner Endowed Chair and is currently the Cadence Distinguished Professor of Electrical Engineering. He has authored over 400 technical papers and ten books in the areas of the analysis of nonlinear networks, simulation, and optimization of electrical circuits, logic synthesis, and formal design verification.

Dr. Brayton is a member of the National Academy of Engineering and a Fellow of the AAAS. He received the 1991 IEEE CAS Technical Achievement Award, the 1971 IEEE Guilleman–Cauer award, and the 1987 ISCAS Darlington Award. He received the 2000 CAS Golden Jubilee and the IEEE Millennium Medals, the 2002 Iowa State University Marston Medal, and the 2006 IEEE Piore Award.

Malgorzata Chrzanowska-Jeske (S'86–M'88–SM'98) received the M.S. degree in electrical engineering from Politechnika Warszawska (the Technical University of Warsaw), Warsaw, Poland, in 1972, and the Ph.D. degree in electrical engineering from Auburn University, Auburn, AL, in 1988.

She has served on the faculty of the Technical University of Warsaw and as a Design Automation Specialist at the Research and Production Center of Semiconductor Devices, Warsaw, Poland. Since 1989, she has been with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR, currently as a Professor and the department's Chair. Her research interests include vertically integrated computer-aided design (CAD) for very large scale integration (VLSI) integrated circuit (IC) and mixed signal system-on-chip (MS-SOC), three-dimensional (3-D) chip architectures, field programmable gate array (FPGA) synthesis and architecture, design for manufacturability and testability in deep submicron, and nano/bioelectronics. She has published more than 100 technical papers.

Dr. Chrzanowska-Jeske is a member of the ACM and Eta Kappa Nu. She was a recipient of the 1990 Best Paper Award from the Alabama Section of IEEE for a paper on the simulation of a bipolar transistor at low temperature published in the IEEE TRANSACTIONS ON ELECTRON DEVICES. She is a member of the VLSI Systems and Applications, and Nanoelectronics and Gigascale Systems Technical Committees of the IEEE Circuits and Systems Society. She has served on the Technical, Steering, and Organizing Committees of many international conferences and was the Technical Chair of the 2002 International Conference on Electronics, Circuits and Systems. In 2004, she was a Guest Editor of the *International Journal on Analog Integrated Circuits and Signal Processing*. She serves as a panelist for the National Science Foundation (NSF) and as a reviewer for National Research Council Canada (NRC) and many international journals and conferences.