Fault Diagnosis and Logic Debugging Using Boolean Satisfiability

Alexander Smith, *Student Member, IEEE*, Andreas Veneris, *Senior Member, IEEE*, Moayad Fahim Ali, *Student Member, IEEE*, and Anastasios Viglas, *Member, IEEE*

***Abstract*—Recent advances in Boolean satisfiability have made it an attractive engine for solving many digital very-large-scale-integration design problems. Although useful in many stages of the design cycle, fault diagnosis and logic debugging have not been addressed within a satisfiability-based framework. This work proposes a novel Boolean satisfiability-based method for multiple-fault diagnosis and multiple-design-error diagnosis in combinational and sequential circuits. A number of heuristics are presented that keep the method memory and run-time efficient. An extensive suite of experiments on large circuits corrupted with different types of faults and errors confirm its robustness and practicality. They also suggest that satisfiability captures significant characteristics of the problem of diagnosis and encourage novel research in satisfiability-based diagnosis as a complementary process to design verification.**

***Index Terms*—Boolean satisfiability debugging, design errors, diagnosis, faults, verification, VLSI.**

I. Introduction

RECENT years have seen an increased use of Boolean satisfiability (SAT)-based tools in the design cycle for very-large-scale-integration (VLSI) circuits. Design verification and model checking [1]–[6], test generation [7], logic optimization [8], and physical design [9], among other problems, have been successfully tackled with SAT-based solutions. This trend is due to recent advances in SAT solvers [2], [10], [11] that make them efficient solution platforms for theoretically intractable problems previously difficult to solve with other traditional methods [4]. The use of SAT in the VLSI design cycle is strengthened by the amount of ongoing research into SAT solvers. Any improvement to the state-of-the-art in SAT solving immediately benefits all SAT-based solutions.

Although SAT-based solutions have been used for many circuit-design problems, no SAT-based solution for logic diagnosis has yet been proposed in existing literature. Given an erroneous design, an implementation of its specification, and a set of input test vectors, logic diagnosis examines correct and erroneous test-vector responses to identify circuit locations that are potential sources of failure. Depending on the stage of the design cycle, shown in Fig. 1, and the type of malfunction

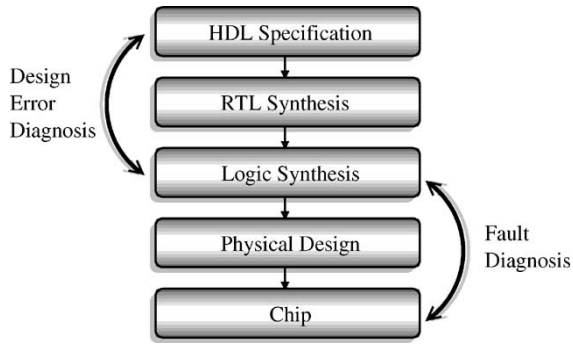

Fig. 1. Digital VLSI design flow.

(“soft” or “hard”), logic diagnosis is used in design-error diagnosis or fault diagnosis.

Fault diagnosis occurs when a fabricated chip fails testing due to the presence of one or more defects [12], [13]. Physical defects are commonly modeled using fault models at the logic level [13]. Given a faulty chip and a correct logic netlist, fault diagnosis is performed in order to identify locations in the netlist corresponding to chip lines that potentially carry defects. This aids the test engineer who later probes these candidate locations in order to identify the type of the defect. Design-error diagnosis and correction (or logic debugging) occurs in the early stages of the design cycle, when the specification is coded in some hardware-description-language (HDL) (or register-transfer-level [RTL]) description and the design is given in the form of a logic netlist [14]. Design errors are usually caused by specification changes, bugs in automated tools, and the human factor [15], [16]. As VLSI designs increase in size and complexity, errors become more frequent and harder to track. Given an erroneous design, design-error diagnosis identifies lines in the netlist that are potentially erroneous.

It is notable that the logic diagnosis of combinational circuits is an inherently difficult problem. The solution space grows exponentially with the number of circuit lines and the number of injected faults [16]. This is because the implementation of the specification (HDL or the failing chip) is treated as a “black box,” controllable at the primary inputs and observable at the primary outputs—a situation depicted in Fig. 2. This complexity increases when diagnosing sequential finite-state machines, in which state equivalence is lost due to reshuffling of memory elements [15], [17]. For these reasons, development of efficient diagnosis tools for combinational and sequential circuits remains a challenging task.

Manuscript received May 24, 2004; revised September 10, 2004. This paper was recommended by Associate Editor K. Chakrabarty.

A. Smith, A. Veneris, and M. Fahim Ali are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: alexander.smith@utoronto.ca; veneris@eecg.toronto.edu; moayad@eecg.toronto.edu).

A. Viglas is with the School of Information Technologies, University of Sydney, NSW 2006, Australia (e-mail: tasos@it.usyd.edu.au).

Digital Object Identifier 10.1109/TCAD.2005.852031

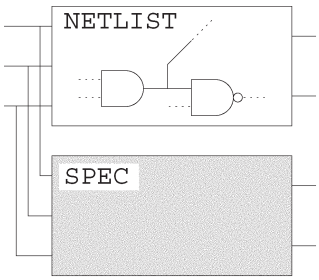

Fig. 2. Fault diagnosis and logic debugging.

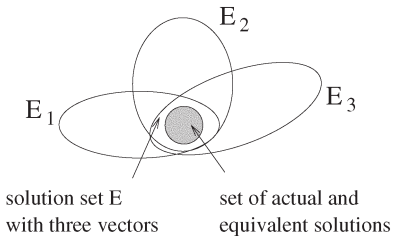

Fig. 3. Space intersection in traditional diagnosis.

This paper presents a novel SAT-based solution for logic diagnosis of multiple faults or design errors in combinational and sequential circuits [18], [19]. This work does not build a new SAT solver, but proposes an SAT-based formulation of diagnosis in which existing solvers [2], [10], [11] can be utilized to find a solution. The types of faults and errors treated here are ones that change the logic functionality of the design at the primary output, irrespective of any timing considerations. The proposed formulation presents a radically new framework for performing diagnosis. It may be used as a stand-alone diagnosis tool, or it may be used to complement traditional diagnosis approaches.

The proposed formulation is intuitive and easy to implement. It can decouple diagnosis from fault modeling, if necessary, in order to perform model-free diagnosis [12], [20]. Model-free diagnosis does not make any assumption on the fault types present in the circuit. This gives it the advantage that it can capture faults with a nondeterministic (unmodeled) behavior [12], [20]. Using the classic stuck-at-fault model as an example, we also show that the proposed method can easily be extended to perform model-based diagnosis. We tailor the model-based version of the proposed algorithm around the stuck-at-fault model because it can model other types of faults and design errors [13], [14].

A number of implementation tradeoffs and heuristics are presented, which improve run-time performance, reduce memory requirements, and take advantage of circuit structural information. Experiments on large combinational and sequential designs corrupted with multiple faults confirm the practicality of the approach. They also confirm that the learning/conflict-analysis procedures in modern SAT solvers allow them to efficiently enumerate the solution space during diagnosis. The theory and experiments in this paper suggest that SAT embraces essential characteristics of logic diagnosis. Since SAT captures a VLSI design at various degrees of abstraction [3], [4], [21] and because of recent advancements in SAT solvers [2], [10], [11] that tackle previously intractable problems efficiently, the proposed research provides opportunities for the development of new SAT-based diagnosis tools and novel diagnosis-specific SAT algorithms.

Since both fault diagnosis and logic debugging have similar goals, this paper is presented in terms of diagnosis for stuck-at faults, unless otherwise stated. Section II contains background information and the problem definition. Sections III and IV give SAT-based formulations of model-free logic diagnosis for combinational and sequential circuits, respectively. Section V analyzes space requirements and performance heuristics. Section VI tailors the method for model-based diagnosis using the stuck-at fault. Section VII contains experiments, and Section VIII concludes the paper.

II. Preliminaries

A. Background

Traditionally, diagnosis techniques are classified as cause–effect or effect–cause techniques [13]. Cause-effect analysis usually simulates all faults in order to compile fault dictionaries. These dictionaries contain entries of faults and respective failing primary-output values. Given a failing chip and a set of k vectors from the tester, the chip responses are matched with those in the dictionary to return a set of potential faults for each vector. Effect–cause analyses do not use fault dictionaries. They simulate the input vectors and apply structural circuit-traversal techniques to identify candidate fault locations.

In both cases, sets of candidate faults $E_1, E_2, \ldots, E_k$ are returned. When any member of each E_i is injected in the netlist, it explains the (faulty or nonfaulty) behavior of the ith test vector alone. These sets are later intersected $(E = E_1 \cap E_2 \cap \cdots \cap E_k)$ to return the final set E of faults that explain the chip behavior for all input vectors. This process is shown in Fig. 3 for three test vectors.

The quality of diagnosis relates to its resolution, that is, its ability to return in E the lines where defects reside. Due to fault equivalence [13], [16], a solution is not always unique. Therefore, the faults returned by a diagnosis algorithm are classified as either actual or equivalent faults. In order to reduce the work of the test engineer, E ideally should contain only these two types of faults (Fig. 3). In theory, this is achievable if the algorithm bases its results on the complete input test-vector space and enumerates the solution space exhaustively [16]—a computationally infeasible task. Fortunately, in practice, a small set of input vectors with high stuck-at-fault coverage provides good resolution (more than 90% on average) for fault diagnosis and logic debugging [14], [16], [22].

Traditional effect–cause diagnosis methods are classified as either symbolic or simulation based. Symbolic methods [23], [24] operate by building an error equation that encodes all corrections. Simulation-based algorithms [20], [25], [26] typically use a backtrace procedure to identify potential fault locations, and then perform simulation to verify that a candidate location is capable of correcting the design or explaining the fault. For some types of faults with high fanout, such as open faults, the amount of simulation required can be excessive [20]. Symbolic methods can be used to help mitigate this problem

[24]. Since the solution space increases exponentially with the number of faults, incremental methods have been proposed to explore this search space efficiently [20], [27], [28]. Such methods examine one location at a time and rely on heuristics to find the locations at which a fault may apply.

The SAT-based method we present here performs diagnosis of multiple faults. It encodes the cardinality and the location of candidate faults in the SAT formula and it lets the SAT solver handle the computationally intensive task of exploring the search space. Thus, we avoid the need to rank candidate locations or explicitly enumerate fault tuples. The solver performs these operations implicitly using the learning and conflict-analysis procedures built into modern SAT engines.

B. Problem Definition

The proposed algorithms work on circuits with the primitive gate types AND, OR, NOT, NAND, NOR, XOR, and XNOR, and with fault-free memory elements (D flip − flops). They also assume that memory elements can reliably be initialized to their reset states.

Our algorithm starts after testing (or verification) has failed. The specification is given as a logic netlist, and the faulty behavior is given as a set of failing test-vector responses. The goal of diagnosis is to identify faults in the netlist that explain the observed test-vector responses. Since the number of faults present is not known ahead of time, the algorithm starts by searching for single-fault solutions. If none exist, it then searches for double-fault solutions, and so on. Each run of diagnosis is performed by generating a conjunctive normal form (CNF) formula Φ and solving it with an SAT solver. Sections III and IV show how this is done for combinational and sequential circuits, respectively.

The input to the problem is an implementation of a circuit specification, given as a netlist $\mathcal{C}$, and a set of input/output test-vector responses. The outputs of these test-vector responses do not match the expected behavior of the specification. When dealing with combinational circuits, this set of vectors contains k distinct elements $V_{\mathcal{C}} = \{v_1, v_2, \ldots, v_k\}$. In sequential diagnosis, the specification is given as a set of k test sequences $V_{\mathcal{S}} = V^{1,m_1}, V^{2,m_2}, \ldots, V^{k,m_k}$. Each test sequence V^{j,m_j} contains input vectors $v^{j,1}, v^{j,2}, \ldots, v^{j,m_j}$ simulated in m_j consecutive cycles. We obtain sets $V_{\mathcal{C}}$ and $V_{\mathcal{S}}$ with random simulation. Test-vector generation for faults and errors is not the topic of this work [13], [17], [29], [30].

The output of the method is a set of lines at which some fault model can be applied to rectify the design for the set of input test vectors ($V_{\mathcal{C}}$ or $V_{\mathcal{S}}$). The method also returns information useful for identifying the types of faults on these lines.

The algorithms for combinational and sequential diagnosis are described on circuits with r primary inputs $X = (x_1, x_2, \ldots, x_r)$ and t primary outputs $Y = (y_1, y_2, \ldots, y_t) = f(X)$. If the circuit is sequential and some of its latches are fully scannable, then these scannable latches are treated as pseudoprimary inputs and outputs in the sets X and Y. In sequential circuits, the initial state is $Q_{\mathrm{I}} = q_1, q_2, \ldots, q_u$ and the primary outputs are defined as $Y = f(X, Q_{\mathrm{I}})$. We use $L = \{l_1, l_2, \ldots, l_n\}$ to represent internal circuit lines including stems and branches. The methods add new hardware to the original circuit. This hardware requires two extra lines per original circuit line. We use the notation $S = \{s_1, s_2, \ldots, s_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ to label these lines.

When diagnosing combinational circuits, variables for all circuit lines x_i, l_i, w_i, and y_i are duplicated to model circuit constraints under simulation of each vector v_j. To avoid confusion, we use the notation x_i^j, l_i^j, w_i^j, and y_i^j for these variables and X^j, L^j, W^j, and Y^j for the respective sets (vectors) of variables. Superscript j corresponds to the index of the simulated test vector v_j.

In the diagnosis of sequential circuits, variables for all circuit lines are also needed for each vector $v^{j,m}$, $m = 1, \ldots, m_j$ in every sequence V^{j,m_j}. We write $x_i^{j,m}$, $l_i^{j,m}$, $w_i^{j,m}$, and $y_i^{j,m}$ to represent these variables (circuit lines), and $X^{j,m}$, $L^{j,m}$, $W^{j,m}$, and $Y^{j,m}$ to represent sets of variables. Superscripts j and m match the indices of test vector $v^{j,m}$ in cycle m. In both types of circuits, $S = \{s_1, s_2, \ldots, s_n\}$ is used to indicate both variable and line names. The variables for lines S are common to all test vectors and sequences. The reason for this is given in Section III-B.

The algorithms presented here turn a diagnosis problem into an instance of a Boolean SAT problem. We do not present a new SAT solving algorithm here. Instead, the SAT instance we generate can be solved with any standard SAT solver [10], [11]. SAT solvers normally operate on Boolean formulas in CNF. This means that the formula is expressed as the product of a set of clauses, where each clause is the sum of a set of literals. A literal is a either variable or its negation. We use the same procedure for expressing logic netlists in CNF form as the one described in [7]. The functionality of each logic gate is represented by a conjunction of clauses. For example, the logic gate $z = x$ AND y is represented by the clauses $(\bar{z} + x) \cdot (\bar{z} + y) \cdot (z + \bar{x} + \bar{y})$. A CNF formula representing the entire circuit is formed by taking the product of the clauses for all gates.

Sections III and IV present the SAT-based formulations for combinational and sequential circuits, respectively. Because the one for combinational circuits is simpler, we present it first. In fact, combinational diagnosis using SAT under this formulation is a special case of sequential diagnosis where $m_1 = m_2 = \ldots = m_k = 1$ for the vectors in the test sequence $V_{\mathcal{S}}$; that is, each test sequence is exactly one cycle long.

III. SAT-Based Diagnosis of Combinational Designs

Given a logic netlist and a set of vectors $V_{\mathcal{C}}$, the algorithm introduces new hardware into the circuit and translates the modified circuit into a CNF formula $\Phi_{\mathcal{C}}$. This formula has two components. Together, they enforce the constraints of the test vectors on candidate fault sites (lines), and they require that fault sets be returned with a specific cardinality. These requirements are satisfied in $\Phi_{\mathcal{C}}$ if fault effects can be injected at some lines in the netlist so that the netlist emulates the specification for all test vectors $V_{\mathcal{C}}$.

The first component of $\Phi_{\mathcal{C}}$ is the conjunction of k CNF formulas C^j (L^j, W^j, X^j, Y^j, S), $1 \leq j \leq k$. Each C^j encodes

constraints from test vector v_j on the logic netlist. Potential fault locations are indicated by adding additional hardware to the circuit, as explained in Section III-A. We later show that $C^j\ (L^j, W^j, X^j, Y^j, S)$ is satisfied if and only if there is a set of fault values that can be injected in the circuit so as to replicate the behavior of the faulty chip at the primary outputs Y for all vectors v_j.

The second component $E_N(S)$, described in detail in Section III-B, encodes constraints on the cardinality of injected faults. These constraints are also coded into the circuit with new hardware, which is later translated into CNF. The number of faults for which diagnosis is to be performed is a user-specified parameter N. The algorithm usually starts with $N = 1$ and it increases its value if it fails to return with a solution.

The complete formula $\Phi_{\mathcal{C}}$ is written as

$$\Phi_{\mathcal{C}} = E_N(S) \cdot \prod_{j=1}^{k} C^j(L^j, W^j, X^j, Y^j, S).$$

Intuitively, the conjunction $\prod_{j=1}^{k} C^j\ (L^j, W^j, X^j, Y^j, S)$ requires that every candidate set of faults satisfies all C^j constraints for all vectors v_j. In other words, fault sets for each vector v_j are intersected as in traditional diagnosis (Fig. 3).

By construction, a satisfying assignment for $\Phi_{\mathcal{C}}$ is one that returns exact information about the locations of the lines the test engineer needs to probe. If no such assignment exists, then no set of N is sufficient to explain the observed behavior. In the following sections, we describe how to compile both components of $\Phi_{\mathcal{C}}$ to perform model-free diagnosis.

As presented below, solving Φ performs diagnosis for N faults, where the value of N must be supplied by the user. Since the value of N is not known before diagnosis begins, the algorithm starts with $N = 1$ and increments its value if the solver fails to return any locations.

A. Test Vector Constraints

This component of $\Phi_{\mathcal{C}}$ is comprised of k CNF formulas C^j that model circuit and fault constraints for each vector v_j, $1 \leq j \leq k$. To simplify the presentation, we first show how to compile this component for a single fault location. At the end of this section, we generalize the construction to cover all possible fault locations.

To represent potential fault sites, extra hardware is added to the circuit and later translated into CNF. To model the potential presence of a fault on line l, a multiplexer is inserted on this line with select line s. The original line l is attached to the multiplexer's 0-input and the multiplexer's output is connected to the former fanout of line l. A new input line w is added and attached to the 1-input of the multiplexer. This multiplexer is later translated into CNF with the rest of the circuit.

Consider the circuit in Fig. 4(a). The potential presence of a fault on line l_1 can be represented by a multiplexer as shown in Fig. 4(b). The first input of the multiplexer is connected to the output of gate l_1 and the second input of the multiplexer is connected to a new line w_1. The output of the multiplexer is connected to the original output of l_1. Observe that the

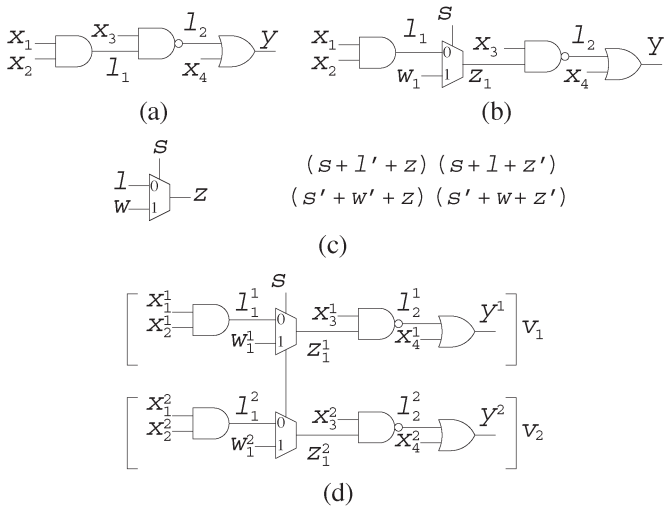

Fig. 4. Modeling candidate fault locations. (a) Original circuit. (b) Modeling a potentially faulty line. (c) CNF representation of a multiplexer. (d) Circuit construction for two test vectors.

functionality of the original (modified) circuit is selected when the value of the select line s is set to 0 (1) [22].

The CNF of the multiplexer logic is given in Fig. 4(c). It can be seen that only four clauses are required to represent it. Hence, the CNF formula representing the new circuit in Fig. 4(b) is $C = (x_1 + \overline{l}_1) \cdot (x_2 + \overline{l}_1) \cdot \overline{x}_1 + \overline{x}_2 + l_1) \cdot (s + \overline{l}_1 + z_1) \cdot (s + l_1 + \overline{z}_1) \cdot (\overline{s} + \overline{w}_1 + z_1) \cdot (\overline{s} + w_1 + \overline{z}_1) \cdot (x_3 + l_2) \cdot (z_1 + l_2) \cdot (\overline{x}_3 + \overline{z}_1 + \overline{l}_2) \cdot (\overline{l}_2 + y) \cdot (\overline{x}_4 + y) \cdot (l_2 + x_4 + \overline{y})$.

To generate the final form of C^j, we need to insert additional clauses that represent the input/output behavioral constraints of test vector v_j. This is done with a set of unit clauses for the set of primary inputs $X = \{x_1, x_2, \ldots, x_r\}$ and the primary outputs $Y = \{y_1, y_2, \ldots, y_t\}$. The literals in these unit clauses have the same phases as their respective logic values in test vector v_j; if v_j assigns the value 1 (0) to input x_i, then $x_i^j(\overline{x}_i^j)$ appears in the formula.

Example 1: Recall the circuit in Fig. 4(a) and assume that there is a single stuck-at-1 fault on line l_1. The input test vector $v = (x_1\ x_2\ x_3\ x_4) = (1\ 0\ 1\ 0)$ detects the fault, as a logic 1 appears at the output of the good circuit while a logic 0 appears at the output of the faulty one. The construction requires unit-literal clauses x_1, $\overline{x}_2$, x_3, $\overline{x}_4$ and $\overline{y}$ to be added to C. Hence, the final formula for vector v is $C^v = C \cdot x_1 \cdot \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 \cdot \overline{y}$. This CNF formula represents the circuit constraints and faulty circuit response for test vector v.

This process is repeated for every test vector v_j to generate formulas $C^j\ (L^j, W^j, X^j, Y^j, S)$ for $j = 1 \ldots k$. Note that each formula requires a new set of variables for the primary inputs X^j, primary outputs Y^j, internal circuit lines L^j, and fault sites W^j. This is because each input test vector will result in a different set of constraints on the circuit netlist. However, only one set of select line variables S is used for all k instances of the circuit.

Example 2: As an illustration of the above process, Fig. 4(d) shows the diagnosis representation of the circuit in Fig. 4(a) for one potentially faulty line l_1 and two test vectors. Two multiplexers are injected into two identical

copies of the circuit with a common select line s. This select line indicates the presence of a fault at the same location in both circuits. Suppose the two input vectors are $v_1 = (1, 0, 1, 0)$ and $v_2 = (0, 1, 1, 0)$, with corresponding (faulty) output values $y^1 = 0$ and $y^2 = 0$. Then the unit clauses $x_1^1 \cdot \overline{x}_2^1 \cdot x_3^1 \cdot \overline{x}_4^1 \cdot \overline{y}^1$ express the constraints of vector v_1, and clauses $\overline{x}_1^2 \cdot x_2^2 \cdot x_3^2 \cdot \overline{x}_4^2 \cdot \overline{y}^2$ represent vector v_2. These ten clauses are added to the CNF of the circuit in Fig. 4(d) to model the faulty input and output test-vector constraints. Observe that, since select line variable s is common to both copies of the circuit, if a SAT solver sets $s = 1$, it effectively forces variables w_1^1 and w_1^2 to assume values such that the new circuit emulates the failing primary-output responses of the test vectors.

The preceding discussion shows how to compile C^j (L^j, W^j, X^j, Y^j, S) for only one potential-fault location. In the final formulation, all internal circuit lines may be suspect locations. Therefore, n multiplexers with distinct select lines $s_1, s_2, \ldots, s_n$ are inserted, one on each line and fanout branch of the circuit. This completes the first component of $\Phi_{\mathcal{C}}$.

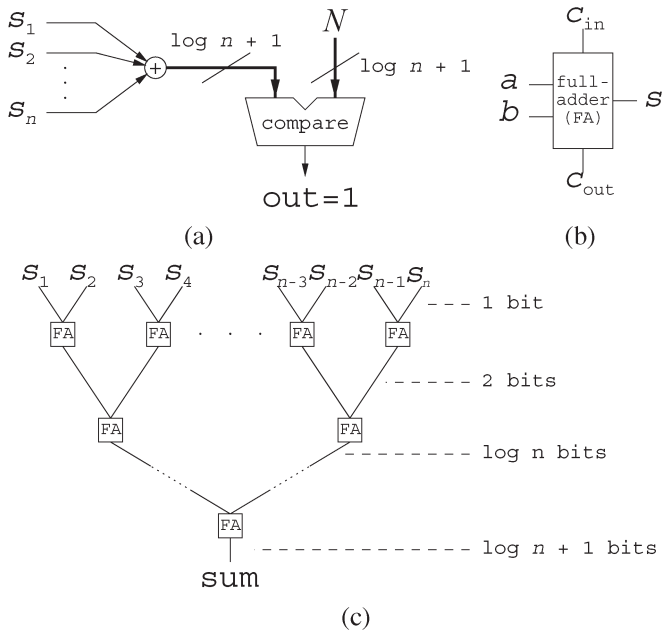

Fig. 5. Counter for multiple faults.

B. Fault Cardinality Constraints

The second component of $\Phi_{\mathcal{C}}$ encodes the constraint that solutions must have exactly N excited fault sites. It is generated by attaching additional hardware to the circuit and then converting this hardware to CNF as part of $\Phi_{\mathcal{C}}$. We first show a straightforward means of generating these constraints for the single $(E_1(S))$ and double $(E_2(S))$ fault cases. This initial method does not yield a practical implementation, but it illustrates the intended effect of $E_N(S)$. We then describe a practical construction of $E_N(S)$.

Example 3: Consider the formula C^v from Example 1. This formula models the circuit in Fig. 4(b) under test vector $v = (1, 0, 1, 0)$. Assume that s is introduced as an additional unit-literal clause, so that the formula becomes $C^v = C \cdot x_1 \cdot \overline{x}_2 \cdot x_3 \cdot \overline{x}_4 \cdot \overline{y} \cdot s$. The addition of this clause in C^v forces the select line to be set to a constant 1. This has the effect of always selecting line w_1 instead of the original circuit line l_1 in Fig. 4. Given this new C^v, a SAT solver will attempt to find a satisfying variable assignment for the circuit lines and the variable w_1 so that the circuit emulates the faulty-chip behavior for vector v. The SAT solver will be forced to set $w_1 = 1$, which correctly indicates a stuck-at-1 fault on line l_1.

The role of $E_N(S)$ in $\Phi_{\mathcal{C}}$ is an extension of the above example. Formula $\Phi_{\mathcal{C}}$ can be updated with constraints that enumerate exhaustively all possible sets of N fault sites. These constraints will enumerate subsets of N select lines $s_{i_1}, s_{i_2}, \ldots, s_{i_N}$ that may simultaneously be activated. Each set of active select lines indicates N active fault locations.

One way to achieve this behavior is to explicitly express the fault locations of interest. For instance, $E_1(S)$ can be written in CNF form as follows:

$$E_1(S) = (s_1 + s_2 + \cdots + s_n) \cdot \prod_{\substack{i=1\ldots n-1 \\ j=i+1\ldots n}} (\overline{s}_i + \overline{s}_j).$$

The first clause requires that at least one select line be set to 1, and that the remaining clauses cause $E_1(S)$ to become unsatisfied if more than one select line is set to 1. Clearly, the set of new clauses introduced by $E_1(S)$ is $O(n^2)$. This idea can be extended to multiple errors. For example, it can be shown that

$$E_2(S) = (s_1 + s_2 + \cdots + s_n) \cdot \prod_{\substack{i=1\ldots n-2 \\ j=i+1\ldots n-1 \\ p=j+1\ldots n}} (\overline{s}_i + \overline{s}_j + \overline{s}_p)$$

causes the SAT solver to search for solutions with one or two active faults, and requires $O(n^3)$ clauses.

Although this representation for $E_N(S)$ is intuitive, in practice it requires an exponential number of clauses $O(n^{N+1})$ to be added to the formula. Clearly, memory requirements for this representation become prohibitive quickly.

To overcome a memory explosion with increasing values of N, we follow a different approach. We encode constraints that enumerate the same solutions space, but we do so implicitly by using the hardware construction shown in Fig. 5(a), which is converted into CNF and appended to $\Phi_{\mathcal{C}}$. This hardware acts as a counter forcing the SAT solver to enumerate sets of N fault sites. It performs a bitwise addition of the multiplexer select lines $S = \{s_1, s_2, \ldots, s_n\}$ and compares the result to the user-defined number of faults N. The output of the comparator is forced to logic 1 with a unit-literal clause so that the bitwise addition of the members of S (that is, the set of fault sites enumerated) is always equal to N.

One may decide to build the comparator in such a way as to enforce a "less than or equal to N" condition rather than "strictly equal to N." Although theoretically sound, this scheme may in practice degrade the performance of the algorithm. This would happen with user-specified values of N that are larger than the minimum number of fault locations required to replicate the faulty behavior. In these cases, the solver would output many solutions by enumerating fault-redundant sets of locations.

As with the select lines themselves, the variables introduced with the hardware in Fig. 5(a) are common to all test vectors in $V_{\mathcal{C}}$. Intuitively, this implicit hardware representation for $E_N(S)$ provides a tradeoff between time and space. Experiments show that modern SAT solvers take advantage of this tradeoff; they avoid an exponential explosion in the time domain while their memory requirements remain low. In the remainder of this section, we show how to construct the counter hardware in CNF with $O(n)$ clauses.

As seen in Fig. 5(a), the counter contains an adder for the select lines and a comparator. Assume that the binary representation of the integer passed from the adder to the comparator is $b_{\log n} \ldots b_1 b_0$. A comparator for SAT-based debugging of two faults is formed by adding CNF $\bar{b}_{\log n} \cdot \bar{b}_{\log n-1} \cdot \ldots \cdot \bar{b}_2 \cdot b_1 \cdot \bar{b}_0$ to $E_N(S)$. This ensures that exactly two select lines are always 1 and all others are forced to 0. Otherwise $\Phi_{\mathcal{C}}$ would not be satisfied. In a similar manner, we can form a comparator in CNF for any value of N with $\log n + 1$ unit-literal clauses.

An implementation for the adder with $O(n)$ clauses is shown in Fig. 5(c). The 1-bit values of the select lines are added progressively in a binary-tree fashion to compute the $(\log n + 1)$-bit sum. The binary tree has $\log n + 1$ levels, with the select lines at level 0. At each level $i = 1 \ldots \log n$, $2^{\log n - i}$ integer sums are produced by adding the integers from the previous level pairwise. Each sum is $i + 1$ bits long, and these bits are produced with a sequence of full-adders as shown in Fig. 5(b).

A full-adder can be encoded in CNF using 14 clauses (or six clauses if the carry-in is omitted). Thus, the size of the adder in CNF is proportional to the number of CNF variables (bits) used to hold the values of the select lines and all intermediate results of the adder tree. Hence, the total number of these CNF variables is

$$\begin{aligned} \#\, CNF\ variables &\le \sum_{i=0}^{\log n} 2^{\log n - i}(i+1) \\ &= 2^{\log n}\left[\sum_{i=0}^{\log n} i\left(\frac{1}{2}\right)^i + \sum_{i=0}^{\log n}\left(\frac{1}{2}\right)^i\right] \\ &\le 2^{\log n}\left[\sum_{i=0}^{\infty} i\left(\frac{1}{2}\right)^i + \sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i\right] \\ &= 4 \cdot 2^{\log n} \\ &= O(n). \end{aligned}$$

This calculation uses the fact that $\sum_{i=0}^{\infty} x^i = (1/(1-x))$ and $\sum_{i=0}^{\infty} i x^i = (x/(1-x)^2)$ when $|x| < 1$. Since creating the CNF clauses contributes a constant multiplicative factor of 14, the size of the CNF formula for the counter is $O(n)$.

For multiple-fault diagnosis, the search space is exponential in the number of circuit lines. For example, for double-fault diagnosis, there are initially n^2 pairs of candidate lines to be examined. In the worst case, simulation-based approaches such as [25] and [20] must enumerate and simulate $O(n^2)$ pairs of lines explicitly. With the proposed algorithm, this exploration of the search space is done implicitly by the SAT solver. The space requirements of the SAT formula do not change with the value of N. Instead, we rely on learning and backtracking techniques in the SAT solver to explore this space efficiently.

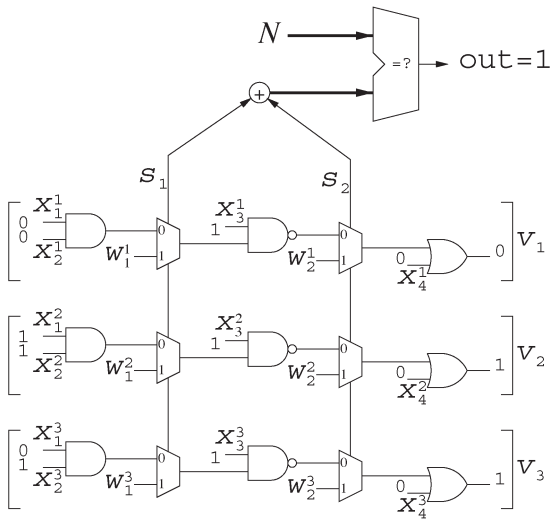

Fig. 6. Complete construction for Φ.

Example 4: Fig. 6 shows the complete construction for both the test-vector and fault-cardinality constraints of $\Phi_{\mathcal{C}}$ for the example circuit from Fig. 4(a) for three test vectors v_1, v_2, and v_3 with failing primary outputs. A multiplexer has been inserted on each circuit line. For the sake of clarity, multiplexers have been omitted from inputs and outputs in this figure. In other words, in this example, we assume that the primary input/output of the circuit are fault free; otherwise, multiplexers should be added in a similar manner. The select lines at the top of the circuit have been added together bitwise using the hardware from Fig. 5(a). The input and output values of the three failing test vectors have been shown. For this example, the SAT solver looks for single solutions since N is required to be 1 in the counter circuitry.

IV. SAT-Based Diagnosis of Sequential Designs

This section describes the SAT-based diagnosis method for sequential designs. This method reduces to the one for combinational circuits when every input test sequence contains only one vector.

Given a sequential netlist and a set of vectors V_S as defined in Section II-B, the algorithm builds a CNF formula $\Phi_{\mathcal{S}}$

$$\Phi_{\mathcal{S}} = E_N(S) \cdot \prod_{j=1}^{k} \prod_{m=1}^{m_j} C^{j,m}(L^{j,m}, W^{j,m}, X^{j,m}, Q_I, Y^{j,m}, S).$$

We observe that $\Phi_{\mathcal{S}}$ is quite similar to $\Phi_{\mathcal{C}}$. It also has two components: The first component $E_N(S)$ enumerates fault-cardinality constraints. Its implementation is identical to the one found in Section III-B. The second component is the conjunction of CNF formulas $C^{j,m}$ $(L^{j,m}, W^{j,m}, X^{j,m}, Q_I, Y^{j,m}, S)$ for all input test vector sequences $j = 1, \ldots, k$ and all simulation cycles $m = 1, \ldots, m_j$. Intuitively, each group of CNF formulas $C^{j,m}$, $m = 1 \ldots m_j$ enforces the

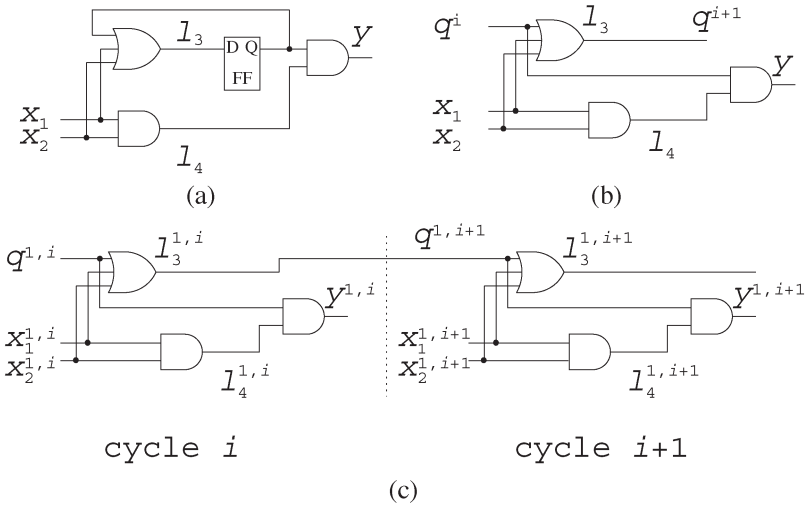

Fig. 7. Example circuit.

constraints of test sequence V^{j,m_j} on the logic netlist by applying values to the inputs $(X^{j,m})$ and the outputs $(Y^{j,m})$ at all time frames. This component of $\Phi_{\mathcal{S}}$ is slightly different from the corresponding one in $\Phi_{\mathcal{C}}$. The remainder of this section explains how to generate it.

In order to simplify the presentation, we develop the theoretical problem formulation around an example that assumes a single-input sequence V^{1,m_1} with two cycles; that is, $k = 1$ and $m_1 = 2$. At the end of this section, the results are generalized for multiple input sequences $(k > 1)$ with an arbitrary number of simulation cycles.

When $k = 1$, this component contains m_1 copies of the CNF formula $C^{j,m}$ $(L^{j,m}, W^{j,m}, X^{j,m}, Q_I, Y^{j,m}, S)$. Each copy enforces different constraints on potential fault locations by specifying the input/output behavior of the (single) test vector for both cycles. This representation resembles the iterative logic array (ILA) modeling of a sequential netlist [13], [17].

In the ILA representation, a sequential circuit is "unrolled" in time. This is performed using identical copies of its combinational circuitry at different simulation cycles, where the inputs of the memory elements from cycle i are connected to the appropriate gates in cycle $i + 1$. For example, the ILA representation of the sequential circuit in Fig. 7(a) is shown in Fig. 7(c) for an input test sequence that is two cycles long. The equivalence between these two representations becomes evident if we redraw the circuit of Fig. 7(a) as shown in Fig. 7(b), with inputs and outputs of the memory elements depicted as pseudooutputs and pseudoinputs of the design.

The circuit is first transformed to its ILA representation. Error locations are then modeled by attaching extra hardware in a manner similar to the one described in Section III. This hardware reflects the potential presence of some fault on a line of the circuit in all m_1 simulation cycles, but it does not require that the fault be excited in all cycles. It merely indicates that the fault exists, and that it may or may not be excited in every cycle. Once again, to model the presence of a fault on line $l_i^{1,m}$, a multiplexer with select line s is attached to every instance $m = 1, \ldots, m_1$ of this line for all m_1 cycles. All of these m_1 multiplexers are later translated into CNF as part of $\Phi_{\mathcal{S}}$. The

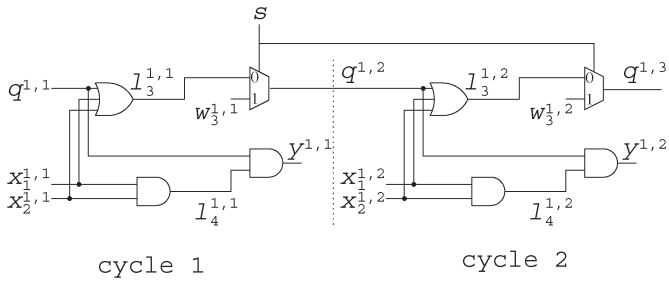

Fig. 8. Diagnosis in two cycles.

first input of each multiplexer is attached on the line $l_i^{1,m}$ and the second input is attached to a new line $w_i^{1,m}$. The output of each multiplexer is connected to the former fanout of gate $l_i^{1,m}$. All m_1 multiplexer copies at different cycles share the same select line s.

Consider again the circuit in Fig. 7(a) and assume the true defect to be a stuck-at-0 fault on line l_3. Since the gate is an input only to a memory element, any input test sequence needs at least two cycles to detect it at the primary output [13]. Test vector sequence $V^{1,2} = \{v^{1,1}, v^{1,2}\} = \{x_1^{1,1}x_2^{1,1}, x_1^{1,2}x_2^{1,2}\} = \{10, 11\}$ produces the erroneous primary output value 0. The presence of a fault on line l_3 can be represented by two multiplexers with common select line s on lines $l_3^{1,1}$ and $l_3^{1,2}$ of the ILA representation of this circuit, as shown in Fig. 8. Observe that when $s = 1$, a new circuit with free inputs $w_3^{1,1}$ and $w_3^{1,2}$ is selected.

Using the four clauses from Fig. 4(c) to encode each multiplexer, the SAT formula for the ILA circuit implementation for cycle i is $\mathcal{F}^i = (\overline{q}^{1,i} + l_3^{1,i}) \cdot (\overline{x}_1^{1,i} + l_3^{1,i}) \cdot (\overline{x}_2^{1,i} + l_3^{1,i}) \cdot (q^{1,i} + x_1^{1,i} + x_2^{1,i} + \overline{l}_3^{1,i}) \cdot (x_1^{1,i} + \overline{l}_4^{1,i}) \cdot (x_2^{1,i} + \overline{l}_4^{1,i}) \cdot (\overline{x}_1^{1,i} + \overline{x}_2^{1,i} + l_4^{1,i}) \cdot (s + \overline{l}_3^{1,i} + q^{1,i+1}) \cdot (s + l_3^{1,i} + \overline{q}^{1,i+1}) \cdot (\overline{s} + w_3^{1,i} + \overline{q}^{1,i+1}) \cdot (\overline{s} + \overline{w}_3^{1,i} + q^{1,i+1}) \cdot (q^{1,i} + \overline{y}^{1,i}) \cdot (l_4^{1,i} + \overline{y}^{1,i}) \cdot (\overline{q}^{1,i} + \overline{l}_4^{1,i} + y^{1,i})$. Hence, the CNF for the circuit in Fig. 8 is $\mathcal{F} = \mathcal{F}^1 \cdot \mathcal{F}^2$.

Once multiplexers are introduced, the updated ILA circuit representation is translated into CNF. To get the final $C^{j,m}$,

```
1  diagnose(Set of Lines L, Set of Lines G, Set of TestSequences V_S, int N)
2  begin
3    // Transform the circuit.
4    foreach (Line g_i ∈ G)
5      insert a mux on g_i with free input w_i and select line s_i
6    foreach (TestSequence V^{j,m_j} ∈ V_S)
7      for integer x := 1 to m_j
8          if (j > 1 or x > 1)
9            duplicate all lines in L except select lines
10         if (x > 1)
11           connect cycle m_{x-1} state inputs to cycle m_x state outputs
12   generate hardware for binary addition of select lines
13
14   associate a CNF variable with each line in L
15   let S : set of Lines := added select lines // |S| = |G|
16   let R : set of Lines := output of addition hardware // |R| = ⌈log |G|⌉ + 1
17
18   // Generate the CNF formula.
19   generate CNF formula Φ from Set of Lines L
20   foreach (TestSequence V^{j,m_j} ∈ V_S)
21     generate unit clauses for input, output and initial state values of V^{j,m_j}
22   foreach (bit b ∈ N)
23     if (b = 1) then Φ := Φ · (r_b)
24                 else Φ := Φ · (r̄_b)
25
26   // Find solutions.
27   let T : Set of (Set of Lines) := ∅ // T is a set of solutions.
28   while (sat_solve(Φ) = SATISFIABLE)
29         let t : Set of Lines := active select lines // t is a single solution.
30         let c : Clause := ∅
31         foreach (Line s_i ∈ t)
32           c := c + s̄_i
33         Φ := Φ · c // Disable t so it will not be found again.
34         T := T ∪ t // Add t to the set of solutions T.
35   return T
36 end
```

Fig. 9. Sequential SAT-based diagnosis.

we need to insert clauses to represent the input and output constraints for the erroneous circuit and *all* m_1 cycles of test sequence V^{1,m_1}. This is easily done with a set of unit-literal clauses for primary input variables $x_1^{1,m}, x_2^{1,m}, \ldots, x_r^{1,m}$, erroneous primary output variable $y^{1,m}$, and initial state variables Q_{I} for every cycle.

Example 5: Recall the circuit from Fig. 8. The stuck-at-0 fault on l_3 is detected in the second cycle with test sequence $V^{1,2} = \{10, 11\}$ because $y_{\mathrm{err}}^{1,2} = 0$ and $y_{\mathrm{corr}}^{1,2} = 1$. To enforce the correct input/output-vector constraints from $V^{1,2}$ on the ILA representation, we add unit-literal clauses $\overline{q}^{1,1}$, $x_1^{1,1}$, $\overline{x}_2^{1,1}$, $\overline{y}^{1,1}x_1^{1,2}$, $x_2^{1,2}$, and $\overline{y}^{1,2}$. Unit-literal clause $\overline{q}^{1,1}$ is added because we assume that the memory elements of the circuit can be correctly initialized to their reset states. Therefore, the final CNF formula for $V^{1,2}$ is $\mathcal{F}' = \mathcal{F} \cdot \overline{q}^{1,1} \cdot x_1^{1,1} \cdot \overline{x}_2^{1,1} \cdot \overline{y}^{1,1} \cdot x_1^{1,2} \cdot x_2^{1,2} \cdot \overline{y}^{1,2}$. Observe that if $\mathcal{F}'$ is passed to an SAT solver, the engine will necessarily assign $s = 1$. The assignment $s = 0$ will cause the solver to backtrack with a conflict, as the erroneous circuit would then produce a correct primary-output behavior, which does not match the test sequence.

This is repeated for every test sequence V^{j,m_j}, $j = 1 \ldots k$ to get formulas $C^{j,m}(L^{j,m}, W^{j,m}, X^{j,m}, Q_I, Y^{j,m}, S)$, the product of which forms the second component of $\Phi_{\mathcal{S}}$. Finally, as in $\Phi_{\mathcal{C}}$, multiplexers are inserted at every line of the netlist, but only one set of select line variables $S = s_1, s_2, \ldots, s_n$ is used. This is because the error locations of a solution must satisfy all vector constraints simultaneously.

V. Implementation

In this section, we discuss implementation details, memory requirements, and performance heuristics for the combinational and sequential SAT-based diagnosis algorithms presented earlier.

Pseudocode for the sequential-diagnosis algorithm is found in Fig. 9. Since, as noted earlier, combinational SAT-based diagnosis is a special case of sequential diagnosis, the same pseudocode works for both. For this reason, all heuristics are described in terms of sequential diagnosis, unless otherwise stated. As shown in that figure, the input to the diagnosis procedure is the complete set of circuit lines L, the current set of suspect faulty lines G, and the set of test-vector sequences $V_{\mathcal{S}}$. With the current description of the algorithm, we have $G = L$. When we present the performance-improving heuristics, we run the algorithm in multiple passes and rounds in which $G \subseteq L$.

The algorithm first attaches a multiplexer to each line in G to model the potential presence of a fault on that line

(Fig. 9, lines 4–5). The circuit is then duplicated for each cycle m_x in sequence V^{j,m_j}, where $x = 1, \ldots, m_j$, $j = 1, \ldots, k$, and $k = |V_S|$ (lines 7–9). If cycle m_x is not the first in the sequence, then the state inputs of cycle m_{x-1} are connected to the state outputs of cycle m_x (lines 10–11). Note that if the circuit is combinational, $m_j = 1$ and lines 10–11 are never executed. This process is repeated for every sequence in V_S. Next, the select line-addition hardware is generated (line 12), the test-sequence constraints are enforced (lines 20–21), and the constraint on the number N of activated faults is encoded (lines 22–24).

The solving process for the formula begins in line 28. If a satisfying assignment is found, the algorithm identifies the active select lines for this solution, adds them to the solution set T, and removes them from future consideration (lines 29–34). This also forces the solver to backtrack and continue exploring the remaining part of the solution space, as explained in Heuristic 2 later in this section. When no more solutions exist, the complete set of solutions found is returned.

We see that one multiplexer is introduced for every circuit line. The resulting netlist can be turned into a CNF formula with $O(n)$ clauses [7], since each multiplexer can be translated to CNF using four clauses. Since the countercircuitry adds an additional $O(n)$ clauses, the space requirements for Φ_S are equal to $O(n) + \sum_{i=1}^{k} \sum_{j=1}^{m_i} O(n) \leq O(n) + km_{\max}O(n) = O(nkm_{\max})$, where $m_{\max}$ is the maximum number of cycles in any test sequence in V_S. This shows that space requirements are linear in the number of circuit lines n, the number of test sequences k, and the length of each test sequence in V_S. These requirements reduce to $O(nk)$ for Φ_C for combinational circuits, since $m_{\max} = 1$.

A. Implementation Heuristics

In this section, we present a set of heuristics that reduce memory requirements and improve performance. These heuristics can be used independently or together by enriching the CNF and the SAT-solving process. Performance is improved by taking advantage of backtracking and clause-learning [2], [10], [11] techniques in modern SAT solvers. These heuristics also show that structural properties of the circuit can provide useful information to SAT-based diagnosis algorithms.

Heuristic 1—Reducing Space Requirements: Although linear in the number of circuit lines, the CNF formula may grow quickly with the number of test vectors. To reduce space requirements while preserving efficiency, Φ_S may be broken into a set of formulas $\Phi_S^1, \Phi_S^2, \ldots, \Phi_S^{\lceil k/p \rceil}$. Each formula encodes constraints for only p of the k test sequences, reducing the space requirements for each accordingly.

The original formula is equivalent to the conjunction $\prod_{i=1}^{\lceil k/p \rceil} \Phi_S^i$. The set of solutions to Φ_S is equal to the intersection of the sets of solutions to each Φ_S^i, as shown in Fig. 3. In other words, instead of running the SAT solver on the original formula Φ_S, we can run it in consecutive passes on formulas Φ_S^i, $1 \leq i \leq \lceil k/p \rceil$. When creating Φ_S^i, it is only necessary to place multiplexers on circuit lines that are activated in one or more solutions to Φ_S^{i-1}.

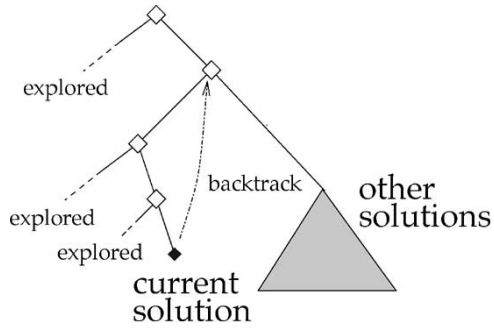

Fig. 10. Implementation heuristics.

This heuristic is a time–space tradeoff. The rationale behind it comes from the fact that, in diagnosis, a small number of vectors usually screens out the majority of the invalid candidates [13], [14], [16]. The experiments in Section VII confirm this result and show that, with $p = 5$ sequences, the first couple of passes are usually sufficient to eliminate more than 90% of the candidates. This means that the remaining formulas are easier to solve, and the later passes run much more quickly.

This idea can be further refined for sequential diagnosis as follows. Since the CNF of the circuit presented to the SAT solver is replicated for a number of cycles for each input/output-vector sequence, the SAT instance may become large. To ease the task of the SAT solver, test sequences can be sorted in increasing order of size $m_{i_1} \leq m_{i_2} \leq \cdots \leq m_{i_k}$, and presented in this order to the SAT solver. This ensures that the first few SAT instances—which tend to be the hardest—have a relatively small size, and so present an easier task to the solver. Larger sequences are solved later, when the process has already reduced the set of candidate error locations.

Heuristic 2—All-Solution Logic Diagnosis: This heuristic is useful when all solutions to an instance of the diagnosis problem are desired. This is often the case in fault diagnosis and logic rewiring [22]. In logic debugging, the designer is usually interested in some solution that rectifies the design.

Suppose a solution is given as some set of fault sites $s_{i_1}, s_{i_2}, \ldots, s_{i_N}$. When it is found, the SAT solver is instructed to search for additional solutions by adding the clause $(\overline{s}_{i_1} + \overline{s}_{i_2} + \ldots + \overline{s}_{i_N})$ to the formula on-the-fly as a learned clause (as is done in lines 30–33 of Fig. 9). This makes the current solution invalid, forcing the SAT solver to backtrack and search for additional solutions.

This process is illustrated in Fig. 10. Dashed lines indicate previously-explored portions of the solution space. If the solver were to be restarted from scratch once a solution was found, it would reexplore much of the already-explored search space. By disabling the current solution and forcing a backtrack, the SAT solver will not reexamine search space it has already discarded. It will also retain any learned clauses it has accumulated up to this point, which can help speed up the search for the remaining solutions. This heuristic is not specific to diagnosis. It can be applied to any SAT problem for which multiple solutions are to be found.

Heuristic 3—Disabling Unnecessary Branching: This heuristic prevents the SAT solver from branching needlessly on free input variables at inactive fault locations. Consider a fault

```
1  diagnose_two_pass(Set of Lines L, Set of TestSequences V_S, int N)
2  begin
3    // Find solutions at dominators.
4    let D : Set of Lines := a dominator cover for L
5          // D is a set of lines such that every line l ∈ L
               is dominated by some line d ∈ D.
6    let T_D : Set of (Set of Lines) := ∅
7    for (n = 1 to N)
8        T_D := T_D ∪ diagnose(L, D, V_S, n)
9
10   // Find solutions within the dominator cones.
11   let S : Set of Lines := ∅
12   foreach (t ∈ T_D)
13     foreach (l ∈ t)
14       S := S ∪ (all lines dominated by l)
15   return diagnose(L, S, V_S, N)
16 end
```

Fig. 11. Two-pass diagnosis.

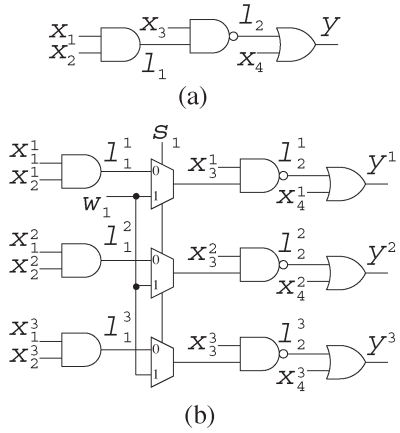

Fig. 12. Stuck-at fault model.

location at line l_i, represented by multiplexers with shared select line s_i and free inputs $w_i^{j,m}$. If $s_i = 0$, then it does not matter what values are assigned to the $w_i^{j,m}$ lines. This is obvious from the circuit structure, but this information is lost when the circuit is translated into CNF. We can incorporate this information into the formula by adding clauses of the form $(s_i + \overline{w}_i^{j,m})$ for each $w_i^{j,m}$. This clause is equivalent to the logic implication $(\overline{s}_i \rightarrow \overline{w}_i^{j,m})$. As soon as s_i is set to 0 (disallowing a fault at this location), all of the $w_i^{j,m}$ lines will be set to 0 immediately by Boolean constraint propagation [10], effectively removing them from the SAT solver's set of free variables.

Heuristic 4—Using Structural Information: As a further performance enhancement, the algorithm can be modified to run in two rounds to take advantage of structural circuit information as in traditional diagnosis [16], [25]. In the first round, multiplexers are only inserted at structural dominators of the circuit. Recall that a line l is a dominator of line l' if all paths from l' to any primary output go through line l [13]. Therefore, any fault effect at l' that is observable at a primary output must propagate through l.

In the first round, the solver will look for faults only at dominator lines. Once a set of dominator solutions has been identified, a second round is run to search for solutions on the lines that they dominate. The number of potential fault locations in the first round is typically about one-fifth of the total number of lines in benchmark circuits [16]. The search space that the SAT solver must explore is reduced accordingly. Fig. 11 gives the pseudocode for this implementation, which is applicable to both single- and multiple-fault/error diagnosis. The code for the procedure diagnose() is found in Fig. 9.

VI. Model-Based Diagnosis

The method, as described so far, performs model-free diagnosis. For a given potential fault site l_i, each of the k test vectors has an independent free input w_i^j. There are no restrictions placed on the values of these inputs. When select line s_i is activated, the lines w_i^j can be assigned whatever values are necessary to justify the faulty output behavior for all test vectors. Since the method does not impose any restriction on these variables, it performs model-free diagnosis [12].

The ability to perform diagnosis with no assumptions on the fault model is often desirable. For example, a fault with nondeterministic behavior may produce different results for each test vector [12]. A model-free diagnosis method can capture such faults by allowing a different value to be assigned to each free input. The price of this flexibility is usually a larger number of equivalent solutions, as a larger number of locations may be able to cause the observed fault effects. Resolution is improved with specific fault models [12], [13]. This tradeoff is not unique to this method; it applies equally to any comparison between a model-free and a model-based diagnosis method.

The proposed method can be extended to model-based diagnosis using the stuck-at-fault model. We have chosen this fault model because of its simplicity and because it can be used to model other faults and design errors [13], [14]. Experiments with stuck-at-fault diagnosis in Section VII show that model-based diagnosis for stuck-at faults often performs better than model-free diagnosis.

A fault model constrains the behavior of candidate fault lines. For example, a stuck-at-fault model imposes the restriction that the faulty line must assume the same value (a constant 1 or 0) under all vectors. A constant 1 represents a stuck-at-1 fault, while a constant 0 represents a stuck-at-0 fault.

In the model-free formulation of SAT-based diagnosis, we create k distinct copies of the circuit, with a separate free input variable w_i^j for each vector j. This is shown in Fig. 4(d) for $k = 2$. For a stuck-at-v fault $(v \in \{0, 1\})$, w_i^j must assume the same logic value v for all k copies of the circuit. We can replicate this effect in Φ_C simply by generating a single w_i input for line l_i and sharing it among all multiplexed copies of the circuit.

This construction is illustrated in Fig. 12. The same value is injected for each of the k test vectors. In other words, if a satisfying assignment (for a failing input test vector) sets $s_1 = 1$ and $w_1 = 0$, then the faulty behavior of the circuit is explained by a stuck-at-0 fault on line l_1, etc.

VII. Experiments

The automated diagnosis tool for faults and errors in combinational and sequential circuits described in the previous sections was implemented in C++ using zChaff [10] as the underlying SAT engine. Experiments are conducted on a

TABLE I
COMBINATIONAL MODEL-FREE FAULT DIAGNOSIS

Circuit Name	Number of Gates	Single Faults				Double Faults			
		Number of Solutions		CPU (second)		Number of Solutions		CPU (second)	
		Dominator	All	Dominator	All	Dominator	All	Dominator	All
c432	376	2	5	0.03	0.02	3	48	0.05	< 0.01
c499	534	8	55	0.05	0.02	2	111	0.21	0.03
c880	699	1	5	0.07	0.04	3	33	0.13	0.05
c1355	527	9	64	0.04	0.02	7	91	0.28	0.02
c1908	811	5	16	0.06	0.02	17	260	0.13	0.02
c2670	1268	2	13	0.12	0.04	1	111	0.31	0.05
c3540	2126	2	13	0.45	0.08	4	95	1.97	0.18
c5315	3203	1	7	0.48	0.21	2	79	2.21	0.22
c6288	5322	1	6	3.86	0.84	2	33	43.23	2.49
c7552	4214	5	20	0.69	0.27	17	225	1.77	0.20

TABLE II
COMBINATIONAL MODEL-BASED FAULT DIAGNOSIS

Circuit Name	Number of Clauses	Single Faults				Double Faults			
		Number of Solutions		CPU (second)		Number of Solutions		CPU (second)	
		Dominator	All	Dominator	All	Dominator	All	Dominator	All
c432	7895	1	5	0.03	0.02	3	33	0.06	0.02
c499	12050	9	14	0.04	0.04	1	2	0.18	0.17
c880	13975	1	6	0.07	0.04	3	6	0.15	0.12
c1355	12125	5	7	0.04	0.04	4	12	0.16	0.11
c1908	16372	5	14	0.07	0.07	11	69	0.16	0.10
c2670	27497	2	18	0.14	0.06	3	9	0.27	0.48
c3540	39569	1	8	0.57	0.17	2	19	3.47	0.52
c5315	62581	1	4	0.73	0.59	2	18	3.28	0.79
c6288	95381	1	6	3.37	0.92	8	10	20.82	14.60
c7552	77046	3	8	0.65	0.29	30	41	1.04	1.13

Pentium IV 2.8 GHz Linux platform with 2 GB of memory using combinational ISCAS'85, large sequential ISCAS'89, and ITC'99 benchmark circuits optimized for area using SIS (`script.rugged`) [31]. For each circuit, we report three types of experiments:

1) model-free diagnosis for single and double stuck-at faults;
2) model-based diagnosis for single and double stuck-at faults;
3) model-free diagnosis for single and double-gate replacement and missing/extra-wire design errors.

The types and locations of faults/errors injected in the circuits are selected at random. In all experiments, the faults/errors inserted are not redundant, and they change the functionality of the design at the primary outputs. Each line in a table or point on a graph is the result of averaging ten experiments. Average values for these experiments and discussion on parameters important to the performance of the algorithms are reported in the following sections. All run-times are in seconds. Heuristic 2 (Section V-A) was built into the prototype diagnosis tool, and is therefore used for all experiments.

A. Diagnosis of Combinational Circuits

In diagnosis of combinational circuits, we use a total of 20 erroneous input test vectors ($|V_{\mathcal{C}}| = 20$) and the algorithm runs in four passes of five vectors each (Heuristic 1, Section V-A). The two-pass heuristic (Heuristic 4) is also used. The branching heuristic (Heuristic 3) is used for model-free diagnosis. It is not applicable to model-free diagnosis.

Table I contains results for model-free diagnosis of single and double stuck-at faults. In this case, Heuristic 3 is also applied on top of the other heuristics. The first column contains the circuit name, and the column that follows gives the number of gates for each circuit. Results for single (double) faults are found in columns 3...6 (7...10). Columns 3 and 4 (7 and 8) show the number of fault sites returned after each round from Heuristic 4. Column 3 (7) shows the number of fault sites at structural dominators during the first round. Column 4 (8) shows the number of equivalent fault sites returned at the end of the second round. Columns 5 and 9 contain the central-processing-unit (CPU) times per fault site for the first round. Columns 6 and 10 contain the overall average CPU time per fault site. Thus, the total run-time for the first round can be determined by multiplying the numbers in columns 3 and 5 (7 and 9), while the total run-time for the entire diagnosis procedure can be found by multiplying the numbers in columns 4 and 6 (8 and 10). All in all, the data in Table I indicate that SAT is both run-time efficient and accurate when used in diagnosis.

Table II contains results on model-based diagnosis for stuck-at faults. Average values in this table are reported in a similar manner to the one for Table I. The only difference is that the second column shows the number of clauses in $\Phi_{\mathcal{C}}$ at the beginning of the first round (Heuristic 4), before any learned clauses are added. These values confirm that memory requirements are indeed linear in circuit size and in the number of vectors. For example, c3540 requires approximately half the number

TABLE III
COMBINATIONAL MODEL-FREE LOGIC DEBUGGING

Circuit Name	Number of Clauses	Single Faults				Double Faults			
		Number of Solutions		CPU (second)		Number of Solutions		CPU (second)	
		Dominator	All	Dominator	All	Dominator	All	Dominator	All
c432	6175	1	5	0.02	0.01	3	16	0.02	0.01
c499	9270	1	5	0.04	0.01	2	25	0.08	0.01
c880	10375	1	3	0.04	0.02	2	25	0.06	0.01
c1355	9405	3	15	0.02	0.01	2	23	0.09	0.01
c1908	11922	1	4	0.05	0.02	7	70	0.04	0.01
c2670	22937	3	25	0.05	0.01	4	59	0.09	0.01
c3540	27829	1	5	0.24	0.07	3	25	0.83	0.10
c5315	46871	1	9	0.44	0.07	3	74	0.72	0.03
c6288	67801	1	4	1.67	0.60	3	30	15.10	1.38
c7552	57896	2	10	0.43	0.10	10	291	0.37	0.02

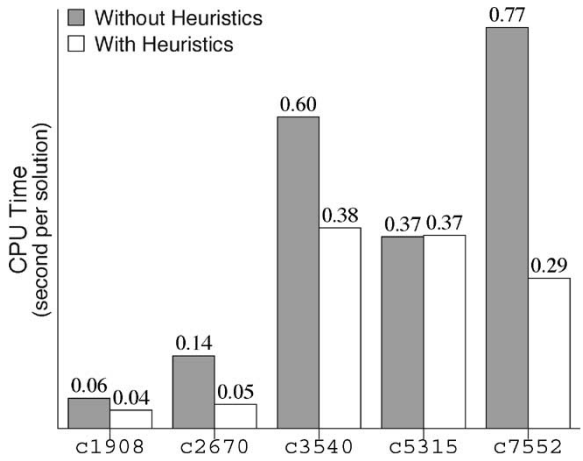

Fig. 13. Heuristic speed-up.

of clauses of c7552 because it has about half the number of lines.

A closer look at the number of fault sites returned by the model-free and model-based stuck-at fault algorithms (Tables I and II) suggests that model-based diagnosis outperforms model-free diagnosis in terms of its resolution. This result encourages further work in fault/error modeling using Boolean SAT to improve performance and increase diagnostic accuracy. It should be noted that, as with any diagnosis method, when a fault model is used, some faults that do not conform to the applied model may not be detected, resulting in reduced fault coverage.

Table III shows results for a model-free diagnosis of gate-replacement design errors [14]. In this experiment, we consider erroneous replacements between gates of types AND, OR, NAND, and NOR. Unlike stuck-at faults, these design errors will not occur on fanout branches. Thus we have fewer lines at which to insert multiplexers, which results in CNF formulas with smaller numbers of clauses than those shown in Table II. Once again, as shown by the values in this table, both the resolution and run-time of the method confirm the effectiveness of a SAT-based approach to logic debugging.

To further examine the benefit of the SAT-based Heuristics 2 and 3 from Section V-A, Fig. 13 depicts the performance of the algorithm for stuck-at fault-model-free diagnosis when these heuristics are present and when they are not. Recall that Heuristic 2 backtracks once a solution is found, in order to reuse previous computation and return the remaining solutions. Heuristic 3 requires variable w_i^j on line l_i immediately to assume a logic value of 0 once s_i is not selected for vector v_j, preventing the SAT solver from branching on this variable.

The bars labeled "without heuristics" show the average run-times with these two heuristics disabled. This figure confirms that these heuristics improve performance in almost all cases. Most notable are circuits c2670 and c7552 for which the average speed-up is over 250%. In the future, we plan to develop additional heuristics to improve performance of an SAT solver when used as an underlying engine for logic diagnosis.

B. Diagnosis of Sequential Circuits

This section presents results for sequential diagnosis of circuits with no state-equivalence information between the implementation of the specification and the netlist. Again, we use 20 erroneous-input test sequences, and the algorithm runs in four passes of five test sequences each ($p = 5$).

Table IV reports average values for model-free stuck-at diagnosis of sequential designs. The format is similar to those in the previous section. The main difference between this table and the ones from Section VII-A is the addition of columns 3, 4, 9, and 10. These columns contain the minimum and maximum number of cycles required to observe the errors. Strictly speaking, these numbers represent the average range of the m_j values in the set of test sequences V_S.

As with combinational-circuit diagnosis, the results reported here show that the method exhibits excellent resolution. The number of locations is small enough to aid the task of the VLSI engineer who will physically probe the faulty chip. Moreover, CPU times confirm that it offers good resolution with low computational overhead. For example, it diagnoses single stuck-at faults in a large circuit such as s38417 in less than 100 s on the average.

As discussed in Section V, the space requirements of the proposed method are $O(npm_{\max})$, where n is the number of circuit lines, p is the number of test sequences in Φ_S^i, and $m_{\max}$ is the maximum length of these test sequences. To gain further insight into its behavior, we examine the effect of changing one of the parameters n, p, or $m_{\max}$ while the other two remain the same.

TABLE IV
SEQUENTIAL MODEL-FREE FAULT DIAGNOSIS

Circuit Name	Number of Gates	Single Faults						Double Faults					
		Number of Cycles		Number of Solutions		CPU (second)		Number of Cycles		Number of Solutions		CPU (second)	
		Minimum	Maximum	Dominator	All	Dominator	All	Minimum	Maximum	Dominator	All	Dominator	All
s1196	1068	1	2	1	4	0.28	0.32	1	3	4	35	1.04	0.35
s1488	1221	2	3	6	24	0.28	0.13	2	3	6	87	7.05	0.62
s1494	1236	2	2	4	15	0.29	0.26	2	3	4	30	2.08	0.42
s3384	3008	4	4	5	15	0.85	0.47	6	7	18	127	31.80	4.01
s4863	3599	2	5	4	15	2.88	1.20	3	5	4	51	36.91	6.81
s5378	2674	2	2	6	12	0.52	0.51	1	2	11	45	6.46	4.46
s6669	5634	4	5	5	18	10.4	1.48	3	4	18	128	17.70	2.26
s15850	7887	2	2	4	14	1.65	0.99	1	2	11	41	16.10	19.4
s35932	17588	3	5	7	16	4.20	3.89	4	6	64	329	410.30	25.86
s38417	26427	3	3	2	11	9.05	8.29	3	5	26	153	100.22	26.19
b14	14498	3	4	7	19	5.72	2.44	3	3	48	300	48.7	6.85
b15	19297	6	6	14	28	5.37	3.51	7	7	20	117	11.26	4.74
b21	32366	4	5	14	28	62.4	20.5	3	3	61	398	110.78	16.57

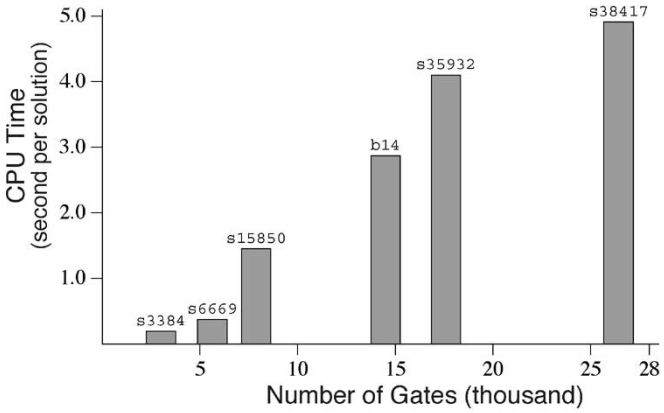

Fig. 14. Circuit size n versus run-time.

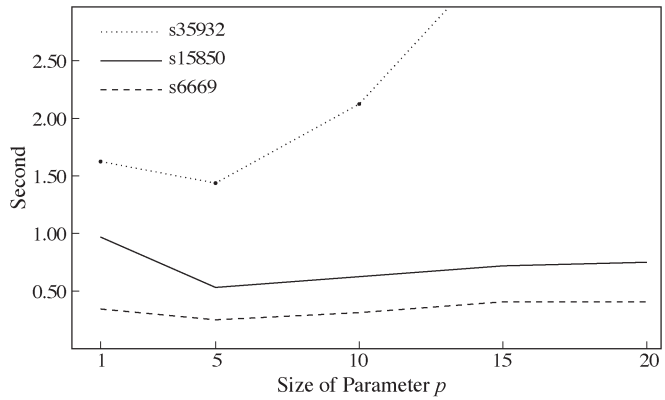

Fig. 15. $\Phi^i_{\mathcal{S}}$ size p versus run-time.

Fig. 14 illustrates the relationship between the circuit size n and the overall run-time per solution for single-error experiments. This graph verifies that the method scales linearly with the circuit size, which is in line with traditional diagnosis results. This indicates that SAT can provide an efficient platform for sequential-logic debugging of large real-life industrial designs.

Fig. 15 illustrates the relationship between the parameter p and the overall CPU time when $m_{\max} = 2$. Three sample circuits of different size suggest that the best value for p performance-wise is 5. This is because, when p grows, so does the size of the CNF formula, which makes the SAT instance harder to solve. Smaller values of p enforce less tight constraints and increase the number of potential locations the SAT solver returns. The efficiency achieved with $p = 5$ seems to balance these two parameters.

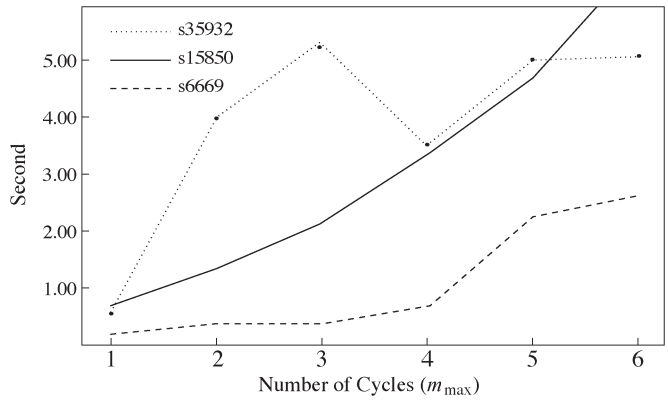

Fig. 16. Number of cycles $m_{\max}$ versus run-time.

The analysis for varying values of $m_{\max}$ with $p = 5$ is shown in Fig. 16. As with the data in Fig. 14, the CPU time scales well with an increasing number of cycles. This similarity between the two behaviors is partly due to the fact that both $m_{\max}$ and n are directly associated with the size of the CNF formula $\Phi^i_{\mathcal{S}}$. As the CNF formula increases, so does the time required to solve the overall problem.

More results on model-based diagnosis of stuck-at faults, and model-free diagnosis of design errors for some of the larger sequential circuits in the ISCAS'89 and ITC'99 family of benchmarks are presented in Tables V and VI, respectively. In both cases, the method returns with good resolution in an efficient manner. Comparison of the resolution for both single and double stuck-at faults between Tables IV and V reveals a similar trend to the one observed in combinational SAT-based fault diagnosis in that model-based diagnosis using SAT outperforms model-free diagnosis.

Table VII provides insight into the behavior of the underlying SAT solver during sequential SAT-based debugging. This table

TABLE V
SEQUENTIAL MODEL-BASED FAULT DIAGNOSIS

Circuit Name	Number of Clauses	Single Faults						Double Faults					
		Number of Cycles		Number of Solutions		CPU (second)		Number of Cycles		Number of Solutions		CPU (second)	
		Minimum	Maximum	Dominator	All	Dominator	All	Minimum	Maximum	Dominator	All	Dominator	All
s1196	26895	1	3	2	4	0.32	0.43	1	2	7	17	1.07	0.85
s1488	39200	2	3	3	12	0.52	0.60	1	3	14	130	1.22	0.26
s3384	259878	5	6	3	6	3.13	2.17	4	6	21	121	45.00	22.00
s9234	137445	3	5	8	22	0.78	0.34	2	5	74	145	3.92	2.5
s15850	230656	2	2	5	11	1.63	1.61	1	2	6	48	4.50	1.03
s35932	1353992	4	6	7	10	4.23	4.58	3	5	63	176	105.40	31.40
b14	775592	3	3	5	7	8.05	6.11	3	4	27	69	10.51	4.53
b15	1908085	6	6	5	8	10.52	9.26	7	7	20	39	11.68	13.97
b21	1821904	3	4	6	14	58.6	22.8	3	3	80	191	113.40	186.46

TABLE VI
SEQUENTIAL MODEL-FREE LOGIC DEBUGGING

Circuit Name	Number of Clauses	Single Faults						Double Faults					
		Number of Cycles		Number of Solutions		CPU (second)		Number of Cycles		Number of Solutions		CPU (second)	
		Minimum	Maximum	Dominator	All	Dominator	All	Minimum	Maximum	Dominator	All	Dominator	All
s1196	20558	1	3	2	5	0.19	0.06	1	3	5	44	0.35	0.03
s1488	31591	2	5	2	10	0.27	0.08	1	2	4	31	0.61	0.08
s3384	162770	2	6	6	12	1.79	0.82	6	6	12	121	16.1	3.48
s9234	91980	3	4	8	13	0.56	0.40	2	4	8	41	4.09	0.77
s15850	296131	2	4	15	39	0.68	0.31	2	4	75	161	19.3	5.01
s35932	797942	2	6	6	10	2.72	2.27	2	3	18	45	79.9	32.1
b14	561916	2	5	4	8	5.35	3.72	2	2	9	37	0.96	0.29
b15	895512	4	6	22	32	3.61	0.91	2	2	6	30	1.89	0.45
b21	1215548	3	6	4	10	36.5	14.9	1	1	10	49	4.75	0.81

TABLE VII
CONFLICT CLAUSES ADDED DURING DIAGNOSIS

Circuit	Number of Conflict Clauses	Number of Locations	Number of Levels	Circuit	Number of Conflict Clauses	Number of Locations	Number of Levels
c432	75	2	40	c1355	83	9	27
c2670	203	2	46	c7552	441	5	68
s1196	25	1	31	s3384	79	5	141
s5378	48	6	29	s15850	75	4	69
s35932	96	7	12	s38417	79	2	50

shows the number of added clauses (excluding those added in Heuristic 2) for the first round of diagnosis. The parameters used for these experiments are the same as those for single-fault diagnosis in Tables I and IV.

It is interesting to note that the number of conflict clauses added by the solver for each circuit is relatively small. This number seems to relate to the number of structural levels of the circuit [13]. Combinational circuits have deeper structures and create more conflicts than sequential circuits, which have their structure repeated in consecutive cycles in the ILA. In both cases, this indicates that the SAT solver makes few "wrong" decisions leading to conflicts and backtracks. We believe that this is because the sequential-diagnosis SAT-based instances, as formulated herein, are SAT problems in which solution constraints are tightly specified in terms of the circuit structure and input test sequences. Therefore, the majority of the circuit lines acquire their "correct" values through Boolean constraint propagation [10]. This leads to the conclusion that the solver is given a relatively easy problem to solve irrespective of the circuit size.

TABLE VIII
MEMORY USAGE FOR SEQUENTIAL MODEL-BASED STUCK-AT-FAULT DIAGNOSIS

Circuit	Gates	Max Sequence	Clauses	Memory (megabyte)
b14	14498	3	729090	121.73
b15	19297	4	1284420	206.53
b21	32366	6	3147527	478.84

Finally, memory requirements for sequential stuck-at-fault model-based diagnosis for some large circuits are shown in Table VIII. These tests are run for 20 failing patterns and for a band size of five vectors. The memory usage reported here is the total amount of memory used by zChaff just before SAT solving begins. No learned clauses have been accumulated at this point, but learned clauses typically comprise less than 1% of the total formula size (Table VII).

The numbers in this table confirm the analysis from Section V. Memory requirements scale linearly with the number of gates, vectors, and test sequences. This makes it applicable to large industrial circuits. For example, from Table VIII,

one may project that a circuit with one million gates (typical for an industrial circuit) would require approximately 97 million clauses and 14.5 GB of memory for fault diagnosis and test-vector sequences with an average length of five. This amount of memory is common in an industrial setting.

VIII. Conclusion

A SAT-based formulation of multiple-fault diagnosis and logic debugging for combinational and sequential-logic circuits was presented. The method is practical in an industrial environment, and it automatically benefits from advances in modern Boolean SAT solvers. Theoretical and experimental results on large circuits with multiple faults and multiple-design errors confirm that Boolean SAT provides an efficient and effective solution to design diagnosis. This offers new opportunities for SAT-based diagnosis tools and diagnosis-specific SAT algorithms.

Acknowledgment

The authors would like to thank Dr. M. S. Abadir and Prof. R. Drechsler for valuable insights arising from technical discussions on performing sequential diagnosis using Boolean satisfiability.

References

[1] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using SAT for combinational equivalence checking," in *Proc. ACM/IEEE Design, Automation and Test in Europe*, Munich, Germany, Mar. 2001, pp. 114–121.

[2] F. Lu, L.-C. Wang, K.-T. Cheng, and R. C.-Y. Huang, "A circuit SAT solver with signal correlation guided learning," in *Proc. ACM/IEEE Design, Automation and Test in Europe*, Munich, Germany, 2003, pp. 892–897.

[3] G. Parthasarathy, M. K. Iyer, K.-T. Cheng, and L.-C. Wang, "Safety property verification using sequential SAT and bounded model checking," *IEEE Des. Test. Comput.*, vol. 21, no. 2, pp. 132–143, Mar.–Apr. 2004.

[4] M. N. Velev and R. E. Bryant, "Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors," in *Proc. Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 226–231.

[5] V. Paruthi and A. Kuehlmann, "Equivalence checking combining a structural SAT-solver, BDDs, and simulation," in *Proc. IEEE Int. Conf. Computer Design*, Austin, TX, Sep. 2000, pp. 459–464.

[6] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *Proc. Int. Conf. Computer-Aided Verification*, Copenhagen, Denmark, 2002, pp. 250–264.

[7] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[8] P. Tafertshofer, A. Ganz, and M. Henftling, "A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1997, pp. 648–655.

[9] R. G. Wood and R. A. Rutenbar, "FPGA routing and routability estimation via Boolean satisfiability," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 1, pp. 222–231, Jun. 1998.

[10] M. H. Moskewicz, C. F. Madigan, Y. Zhao, and L. Zhang, "Chaff: Engineering an efficient SAT solver," in *Proc. Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 530–535.

[11] 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.

[12] R. C. Aitken, "Modeling the unmodelable: Algorithmic fault diagnosis," *IEEE Des. Test. Comput.*, vol. 14, no. 3, pp. 98–103, Jul.–Sep. 1997.

[13] N. Jha and S. Gupta, *Testing of Digital Systems*. New York: Cambridge Univ. Press, 2003.

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

[15] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*. Boston, MA: Kluwer, 1998.

[16] A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1803–1816, Dec. 1999.

[17] T. Kropf, *Introduction to Formal Hardware Verification*. New York: Springer-Verlag, 1999.

[18] M. Y. Fahim Ali, A. Veneris, S. A. Safarpour, R. Drechsler, A. Smith, and M. Abadir, "Debugging sequential circuits using Boolean satisfiability," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 2004, pp. 204–209.

[19] A. Smith, A. Veneris, and A. Viglas, "Design diagnosis using Boolean satisfiability," in *Proc. IEEE Asia South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2004, pp. 218–223.

[20] J. B. Liu and A. Veneris, "Incremental fault diagnosis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 2, pp. 240–251, Feb. 2005.

[21] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *Proc. Design Automation Conf.*, Los Angeles, CA, Jun. 2000, pp. 675–680.

[22] A. Veneris and M. S. Abadir, "Design rewiring using ATPG," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1469–1479, Dec. 2002.

[23] P.-Y. Chung and I. N. Hajj, "Diagnosis and correction of multiple logic design errors in digital circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 6, pp. 233–237, Jun. 1997.

[24] S.-Y. Huang, "Speeding up the Byzantine fault diagnosis using symbolic simulation," in *Proc. IEEE VLSI Test Symp.*, Monterey, CA, May 2002, pp. 193–198.

[25] S.-Y. Huang and K.-T. Cheng, "ErrorTracer: Design error diagnosis based on fault simulation techniques," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1341–1352, Sep. 1999.

[26] D. B. Lavo, B. Chess, T. Larrabee, and F. J. Ferguson, "Diagnosing realistic bridging faults with single stuck-at information," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 3, pp. 255–268, Mar. 1998.

[27] L. M. Huisman, "Diagnosing arbitrary defects in logic designs using single location at a time (SLAT)," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 1, pp. 91–101, Jan. 2004.

[28] D. B. Lavo, I. Hartanto, and T. Larrabee, "Multiplets, models, and the search for meaning: Improving per-test fault diagnosis," in *Proc. Int. Test Conf.*, Baltimore, MD, Oct. 2002, pp. 250–259.

[29] G. Fey and R. Drechsler, "Finding good counter-examples to aid design verification," in *Proc. ACM/IEEE Int. Conf. Formal Methods and Models Co-Design*, Mont Saint-Michel, France, 2003, pp. 51–52.

[30] I. Hamzaoglu and J. H. Patel, "New techniques for deterministic test pattern generation," in *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, Apr. 1998, pp. 446–452.

[31] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, and R. K. Brayton, "Sequential circuit design using synthesis and optimization," in *Proc. IEEE Int. Conf. Computer Design*, Baltimore, MD, Oct. 1992, pp. 328–333.

Alexander Smith (S'98) received the B.A.Sc. degree with honors in engineering science and the M.A.Sc. degree in computer engineering, both from the University of Toronto, Toronto, ON, Canada, in 2002 and 2004, respectively. He is currently pursuing the Ph.D. degree in aerospace engineering.

He has research interests in Boolean satisfiability, constraint satisfaction, and task-planning algorithms.

Andreas Veneris (S'96–M'99–SM'05) was born in Athens, Greece. He received the diploma in computer engineering and informatics degree from the University of Patras, Patras, Greece, in 1991, the M.S. degree in computer science from the University of Southern California, Los Angeles, in 1992, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1998.

He was a Visiting Faculty of the University of Illinois at Urbana-Champaign in 1998–1999. In 1999, he joined the University of Toronto, Toronto, Canada. He is currently an Associate Professor cross-appointed with the Department of Electrical and Computer Engineering and the Department of Computer Science. His research interests include CAD for synthesis, diagnosis, and verification of digital circuits and systems, data structures, and combinatorics. He has coauthored a book.

Dr. Veneris is a member of ACM, AAAS, Technical Chamber of Greece, and the Planetary Society. He is a corecipient of a Best Paper award in ASP-DAC'01.

Moayad Fahim Ali (S'01) was born in Alexandria, Egypt. He received the B.Eng. degree with distinction from the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada, in 2003. He is currently pursuing the M.A.Sc. degree at the University of Toronto, Toronto, ON, Canada.He has research interests in CAD for verification and diagnosis of VLSI circuits, as well as Boolean satisfiability.

Anastasios Viglas (S'96–M'02) received the diploma in electrical and computer engineering degree from the National Technical University of Athens, Athens, Greece, in 1996, and the Ph.D. degree in computer science from Princeton University, Princeton, NJ, in 2002.

He is currently a Lecturer at the University of Sydney, Sydney, Australia, in the School of Information Technologies. He held a postdoctoral research fellowship at the University of Toronto, Toronto, ON, Canada, from 2001 to 2003. His research interests include the design and analysis of algorithms, complexity theory, combinatorial optimization, as well as game theory.

Dr. Viglas is a member of ACM.