Improved Design Debugging using Maximum Satisfiability

Sean Safarpour, Hratch Mangassarian, Andreas Veneris
Department of Elec. and Comp. Eng.
University of Toronto, Toronto, Canada
{sean, hratch, veneris}@eecg.toronto.edu

Mark H. Liffiton, Karem A. Sakallah
Department of Elec. Eng. and Comp. Sci.
University of Michigan, Ann Arbor, USA
{liffiton, karem}@eecs.umich.edu

***Abstract*— In today's SoC design cycles, debugging is one of the most time consuming manual tasks. CAD solutions strive to reduce the inefficiency of debugging by identifying error sources in designs automatically. Unfortunately, the capacity and performance of such automated techniques must be considerably extended for industrial applicability. This work aims to improve the performance of current state-of-the-art debugging techniques, thus making them more practical. More specifically, this work proposes a novel design debugging formulation based on maximum satisfiability (max-sat) and approximate max-sat. The developed technique can quickly discard many potential error sources in designs, thus drastically reducing the size of the problem passed to an existing debugger. The max-sat formulation is used as a pre-processing step to construct a highly optimized debugging framework. Empirical results demonstrate the effectiveness of the proposed framework as run-time improvements of orders of magnitude are consistently realized over a state-of-the-art debugger.**

I. Introduction

Functional verification tasks dominate the effort of contemporary VLSI and SoC design cycles. A major step of functional verification is design debugging, which determines the root cause of failed verification tasks such as simulation or equivalence checking. For example, when a simulation run fails because a design's behavior is inconsistent with its specification, debugging identifies the components responsible for the discrepancy.

Hardware debugging is overwhelmingly performed manually in the industry today. Designers and verification engineers must analyze the failed verification instances, the design and the specification to realize which design components or blocks are the root cause of the failure. Due to the "guess-and-check" nature of the problem, this task is accepted as one of the most time-consuming processes of the VLSI and SoC design cycles. As design complexities nearly double with every generation, so does the daunting debugging effort. Clearly, automated debugging solutions are needed to increase a designer's debugging and verification efficiency.

Automated debugging is a computationally intensive problem since its complexity increases dramatically with the size of the design, the length of the error traces and with the number of errors present in the design. There is a rich history of debugging techniques and algorithms developed over the last decades which seek to tackle this problem [1], [2], [6]. Although efficient for relatively small design blocks and particular design types, these solutions have not been extended to industrial problems. More recently, several debugging techniques based on formal techniques such as Boolean Satisfiability (SAT) [10] and Quantified Boolean Formula (QBF) [5] have demonstrated great promise and encouraged further research in formal techniques for debugging. Despite these successes, the capacity and performance of both traditional and newer debugging techniques must be greatly improved to make debugging practical for industrial problems.

This work proposes a novel framework with the aim of greatly reducing the run-time of state-of-the-art debuggers. This technique presents the first maximum satisfiability (max-sat) formulation for design debugging. The formulation is constructed using the constraints corresponding to the erroneous design, the input stimulus, and the expected correct response. The formulation is unsatisfiable, since the incorrect design cannot produce the correct response, and it can only be satisfied if some of the constraints are removed. An all-solution max-sat solver can iteratively find maximal satisfiable subsets of the constraints. The complement of any of these subsets is a set of constraints whose removal makes the problem satisfiable. These constraints will be shown to correspond directly to the erroneous gates or components in the design. The proposed max-sat technique is developed for combinational and sequential circuits as well as for problems with single or multiple input stimuli and expected responses.

The proposed technique is an alternative approach to hardware debugging which can be easily enhanced to over-approximate solutions. The over-approximation allows for a trade-off between the tool's performance and the resolution of the solutions. More specifically, approximation can reduce the problem complexity and thus require less run-time at the cost of finding larger, less precise solutions. Although not exact, this approach can be employed as a pre-processing step that filters solutions for a second stage exact debugger. The second debugger benefits from having fewer suspect error sources which translates into faster run-times. The combined two-step debugging framework reduces the complexity of both stages, resulting in an efficient debugging solution.

A suite of experiments on combinational and sequential circuits for single and multiple vectors are conducted to demonstrate the benefit of the proposed framework. On average, the over-approximation technique quickly eliminates 92% of the suspects. The second stage debugger uses the filtered suspects to find the exact error sources in a fraction of the time it would take otherwise. Overall, performance improvements of 200 times or two orders of magnitude over a state-of-the-art debugger are observed consistently.

In the next section, background is provided on the max-sat approach used as well as on design debugging. Sec. III presents the proposed max-sat approach for combinational circuits and Sec. IV extends this for sequential circuits and for multiple vectors. Sec. V present the over-approximation technique and the overall framework developed for optimal performance, respectively. Experiments are presented in Sec. VI followed by the conclusion in Sec. VII.

II. Background

A. Maximum Satisfiability

The algorithm from [8] is used to solve a generalization of the max-sat problem. While max-sat is concerned with finding a satisfiable set of clauses with maximum cardinality, this can be generalized to find Maximal Satisfiable Subsets (MSSes). An MSS is a satisfiable subset of a formula's clauses that is maximal in the sense that adding any one of the remaining clauses would make it UNSAT. Any max-sat solution is of course an MSS, but MSSes can be different (smaller) sizes as well. In this work, the *complements* of MSSes, sets of clauses whose removal makes the instance satisfiable, are of interest. Just as an MSS is maximal, its complement is minimal, and we refer to such a set as a Minimal Correction Set (MCS). This work makes use of two following techniques developed as extensions to the algorithm from [8]:

- Finding all MCSes up to size k
- Grouping clauses to produce "approximate" MCSes

Finding all MCSes up to size k is performed by the algorithm AllMCSes from [8], which was developed as the first phase of an approach for finding all Minimal Unsatisfiable Subsets (MUSes). This procedure solves consecutive optimization problems, finding MCSes in order of increasing size (equivalent to finding their complementary MSSes in order of decreasing size). MCSes are returned as they are found, and execution can be stopped when a size limit is reached.

The second ability, of grouping clauses, depends on the way the algorithm uses clause-selector variables. Every clause C_i is augmented with a new variable y_i, producing $C'_i = (\overline{y_i} + C_i) = (y_i \rightarrow C_i)$. When y_i is assigned TRUE, the original clause C_i must be satisfied, while when y_i is FALSE, C'_i is satisfied, essentially disabling the original clause. This gives a standard SAT solver the ability to enable and disable constraints implicitly within the normal backtracking search. By assigning the same y variable to *multiple* clauses, a set of clauses can be treated as a single higher-level constraint (the conjunction of all clauses given the same y variable) that can be enabled and disabled at once. Using this approach, each MCS is a minimal set of *groups* of constraints whose removal makes the instance satisfiable. This leads to an over-approximation of an MCS of the original clauses, because extra clauses will be included in groups even though they may not be necessary. The benefit of the over-approximation is that it can greatly increase the performance of the algorithm as the search space is reduced exponentially.

This work uses the MCS techniques outlined above for debugging. Although not precise in the general case, the term max-sat is used throughout to refer collectively to the techniques above for simplicity.

B. Automated Design Debugging

The problems of *design debugging* and *fault diagnosis*, which occur at different stages of the VLSI design cycle, have strong similarities. The latter occurs when a fabricated chip fails during the testing phase due to the presence of manufacturing defects [10], while design debugging occurs at the early stages of the design cycle, when the implemented design does not meet its functional specifications. In this paper the terminology and assumptions are those of design debugging, however, the proposed techniques can apply to fault diagnosis as well.

The input of the design debugging problem is an erroneous circuit C, a set of input stimuli I for which the design fails verification, and the corresponding correct output responses O. The components I and O, also called input/output vectors, can be obtained from simulation-based verification tools or formal tools such as equivalence checkers and model checkers.

An error source at the circuit-level exhibits an erroneous response at the primary outputs for at least one of the provided vectors. In this paper, a model-free diagnosis strategy is used, which can "detect" any type of gate/module error. The *error cardinality* N_g is the maximum number of simultaneous error sources the debugger assumes exist in the circuit. The complexity of design debugging increases exponentially with the error cardinality [11]. A design debugging tool must return all possible solutions, *i.e.* all potential error tuples up to the size of the error cardinality.

Traditionally, methods based on simulation, path-tracing and binary decision diagrams have been used to tackle the design debugging problem [1], [2], [6]. Recently, SAT-based strategies [10] have been proven to be effective as their performance increases with that of the underlying SAT solvers. This approach formulates the design debugging problem by constructing circuit constraints, translating it to a Boolean formula in *Conjunctive Normal Form* (CNF), and giving it to an all-solution SAT solver. Note that deriving a CNF from a circuit is a simple linear time algorithm as there is a one-to-one correspondence between circuit gates and CNF formulas. Table I shows five basic gates along with their CNF representations.

Gate	**CNF**
$y = \text{NOT}(x)$	$(x + y) \cdot (\overline{x} + \overline{y})$
$y = \text{AND}(x_1, x_2, \ldots, x_n)$	$(x_1 + \overline{y}) \cdot (x_2 + \overline{y}) \cdot \ldots \cdot (x_n + \overline{y}) \cdot$ $(\overline{x_1} + \overline{x_2} + \cdots + \overline{x_n} + y)$
$y = \text{OR}(x_1, x_2, \ldots, x_n)$	$(\overline{x_1} + y) \cdot (\overline{x_2} + y) \cdot \ldots \cdot (\overline{x_n} + y) \cdot$ $(x_1 + x_2 + \cdots + x_n + \overline{y})$
$y = \text{XOR}(x_1, x_2)$	$(\overline{x_1} + x_2 + y) \cdot (x_1 + \overline{x_2} + y) \cdot$ $(x_1 + x_2 + \overline{y}) \cdot (\overline{x_1} + \overline{x_2} + \overline{y})$
$y = \text{MUX}(s, x_1, x_2)$	$(x_1 + \overline{y} + s) \cdot (\overline{x_1} + y + s) \cdot$ $(x_2 + \overline{y} + \overline{s}) \cdot (\overline{x_2} + y + \overline{s})$

TABLE I

Gate to CNF translation

III. Debugging Combinational Circuits with Max-sat

Given an erroneous circuit C, an input stimulus I, and the corresponding correct output response O a CNF formula can be produced as follows.

$$\Phi = I \cdot O \cdot \mathrm{CNF}(C)$$

This CNF problem is naturally unsatisfiable because the erroneous circuit cannot produce the correct output response under the given input vector. Since the inconsistency between a circuit's actual and correct response is due to some gate-level error sources, the unsatisfiability of the problem is due to the clauses derived from these error sources. In other words, the clauses that are at conflict in the CNF correspond to the circuit-level error sources from which they are derived. Therefore, the circuit-level errors can be identified by finding the CNF-level *error clauses*.

The max-sat approach in Sec. II-A can identify Maximal Satisfiable Subsets (MSSes) whose complements are Minimal Correction Sets (MCSes). These MCSes represent sets of clauses whose removal from the CNF make the problem satisfiable. In the formula Φ constructed using the constraints I, O, and $\mathrm{CNF}(C)$, the MCSes map directly to error clauses. Once the error clauses are identified through MCSes, the gate-level suspects are found by mapping each clause to the gate it is originally derived from as described in Sec. II.

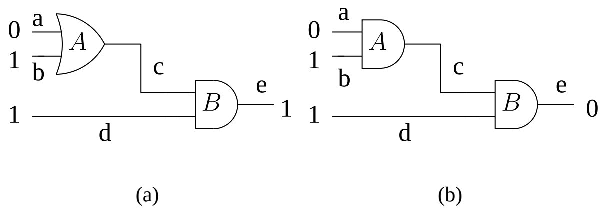


Fig. 1. Correct and erroneous circuit

For example, consider the correct and erroneous circuit in Fig. 1 (a) and (b) where gate A is mistakenly implemented as an AND gate instead of an OR gate. Under the input stimulus $\{a = 0, b = 1, d = 1\}$ the circuit has a response of $\{e = 0\}$ instead of the correct response of $\{e = 1\}$. The corresponding erroneous CNF for the circuit and the input/output vectors are shown below.

$$\begin{gathered}(\overline{a}) \cdot (b) \cdot (d) \cdot (e)\\ (a+\overline{c}) \cdot (b+\overline{c}) \cdot (\overline{a}+\overline{b}+c)\\ (c+\overline{e}) \cdot (d+\overline{e}) \cdot (\overline{c}+\overline{d}+e).\end{gathered}$$

Here, the max-sat approach described in Sec. II-A can return the MCS $(a + \overline{c})$ as a solution because removing this clause from the CNF makes the formula satisfiable. Notice that this clause is derived from the erroneous gate A.

The above example illustrates how the removal of an error clause can help identify the error source. Further analysis of the example demonstrates that there are other clauses such as $(c + \overline{e})$ whose removal can satisfy the problem. Indeed, more than one error clause may exist in a given problem corresponding to the many potential error sources at the gate-level. These are more commonly known as equivalent errors or faults in the diagnosis literature [1]. Note that the removal of the clause $(\overline{a})$ also satisfies the problem, however since this constraint is not part of the circuit component of the CNF (*i.e.* C), it is not considered as an error clause.

For the debugging technique to be complete, all equivalent errors must be found. Each of these is known as a suspect error source because it may fix the problem such that erroneous circuit produces the correct response for the given input vector. As a result, the AllMCSes algorithm of Sec. II-A is used to find all error clauses and consequently all gate-level error suspects.

A. Error Clause Cardinality

Since the solution space for the AllMCSes algorithm is exponential, an explicit limit for the maximum cardinality of the MCSes is advised to prevent memory explosion. In practice, this limit, called the *error clause cardinality*, must be relatively small due to memory and performance considerations. The error clause cardinality determines the completeness and efficiency of the proposed technique.

Since this work is primarily concerned with gate-level debugging the limit used must correspond with the gate-level cardinality of conventional debuggers. In Sec. II the error cardinality N_g is defined as the maximum gate tuples that may be responsible for the erroneous behavior. At the level of the CNF encoding, the error clause cardinality N_c must be set to a value such that all the gate-level errors at N_g can be found using the proposed max-sat approach. Thus completeness in this context is with respect to the gate-level debuggers such as [5]. The following theorem proves that the proposed approach is complete for a given value of N_g.

Theorem: The algorithm AllMCSes called on the problem $\Phi = I \cdot O \cdot \mathrm{CNF}(C)$ with a limit of N_c is complete if N_c is equal to [the maximum number of clauses derived for any single gate in the CNF] $\times N_g$.

Proof: Proof by contradiction. Suppose there is a gate-level error not identified by the proposed approach using the error cardinality limit N_c. Since AllMCSes iteratively finds sets of clauses with cardinality 1 up to N_c, the gate-level error must be caused by more than N_c clauses. However, N_c is equal to the maximum number of clauses derived from any one gate times N_g, so the error must be caused by more than N_g gate-level sources. Therefore the error is not found using conventional debuggers with N_g either. $\square$

In many circuit-based SAT problems, the circuit is first converted to a 2-input AND-INVERTER graph and then translated into CNF [4], [7]. In such a CNF formula, the maximum number of clauses from any gate is 3, thus $N_c = 3 \times N_g$. Using this value for N_c results in finding all the solutions found using conventional debuggers with N_g. In CNF formulas derived from arbitrary circuits where the number of clauses generated can greatly vary from one gate to another, the proposed max-sat debugging technique may return more solutions than the gate-level debugger for a given N_g. As discussed further in Sec. V this scenario does not pose a problem under the proposed framework.

B. Error Group Cardinality

The previous section presented a limit for the error clause cardinality to guarantee completeness for the proposed approach. Although complete, increasing the error clause cardinality is not always desired as the complexity of the debugging problem is exponentially related to the error cardinality. Here, the grouping ability described in Sec. II-A is used to reduce the complexity of the problem while maintaining completeness.

Grouping all clauses derived from the same gate together allows the max-sat solver to "enable" or "disable" all of those clauses simultaneously. In effect, this gives the solver the ability to treat each gate as a single high-level constraint, leading to solutions (MCSes) found directly in terms of the gates. Under this problem restriction, the *error clause-group cardinality*, N_{cg} required to find gate-level errors can be effectively N_g.

Theorem: By grouping all clauses derived from the same gate together, the proposed technique is complete if the error clause-group cardinality $N_{cg} = N_g$.

Proof: Since each group has a one-to-one correspondence with a circuit gate, when a group is found as part of an MCS, all clauses corresponding to the original gate are "disabled" by the AllMCSes algorithm. Thus every solution found by AllMCSes maps to a set of the original gates. Hence, limiting the group cardinality is equivalent to limiting the gate cardinality. □

Re-visiting the example of Fig. 1, grouping the clauses of gate A together with the clause-selector variable y_A and the clauses of gate B together with the clauses-selector variable y_B, results in the following CNF.

$$\begin{gathered}(\overline{a}) \cdot (b) \cdot (d) \cdot (e)\\ (a + \overline{c} + \overline{y_A}) \cdot (b + \overline{c} + \overline{y_A}) \cdot (\overline{a} + \overline{b} + c + \overline{y_A})\\ (c + \overline{e} + \overline{y_B}) \cdot (d + \overline{e} + \overline{y_B}) \cdot (\overline{c} + \overline{d} + e + \overline{y_B}).\end{gathered}$$

IV. Extension to Sequential Circuits and Multiple Vectors

Debugging sequential circuits is similar to that of combinational circuits except that their behavior must be modeled for a finite number of clock cycles. These clock cycles are necessary to excite and observe the errors. A popular approach for modeling sequential circuits is to use the time-frame expansion technique or the Iterative Logic Array (ILA) representation. These techniques replicate a circuit's transition relation, called a time-frame, and connect the current-state and the next-state of adjacent time-frames together. In effect, the sequential circuit is transformed into an "unfolded" combinational circuit that can be debugged like any other combinational circuit.

Since the complexity of debugging increases exponentially with the number of error sources, debuggers must be careful not to consider the "replicated" gates across time-frames as unique error sources. For example, a single gate-level error in an ILA with 3 time-frames may appear to have 3 distinct error locations, however, replacing the functionality of a single gate in the original sequential circuit will fix the problem in all time-frames.

The proposed max-sat debugging technique can be extended to handle sequential designs efficiently. First, the sequential circuit is converted to an ILA and then translated into CNF. Similar to the previous formulation the CNF is then constrained with input stimulus and output response, I and O resulting in

$$\Phi = I \cdot O \cdot \mathrm{CNF}(\mathrm{ILA}(C)).$$

The second step is to account for the replication due to the ILA by grouping all clauses derived from the same gate but from any time-frame. As a result, clauses from a particular gate will be "enabled" and "disabled" at once irrespective of the time-frames they represent.

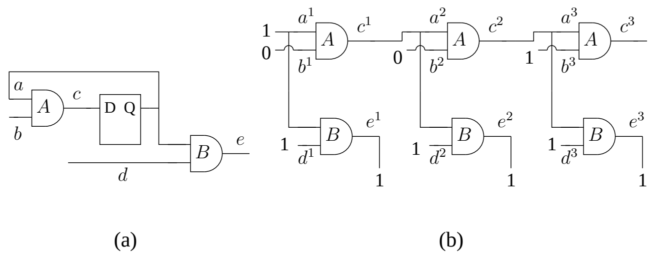


Fig. 2. Erroneous sequential circuit and its ILA representation

For example consider the erroneous sequential circuit shown in Fig. 2(a) and its ILA in Fig. 2(b). Here, the gate A has been erroneously implemented as an `AND` gate instead of an `OR` gate. As a result, the output of A in the first and second time-frames should be 1 instead of 0. Note that the input stimulus and correct response are also shown in Fig. 2(b). The corresponding CNF for the constrained ILA is shown below.

$$\begin{gathered}(a^1) \cdot (\overline{b^1}) \cdot (d^1) \cdot (e^1)\\ (a^1 + \overline{c^1}) \cdot (b^1 + \overline{c^1}) \cdot (\overline{a^1} + \overline{b^1} + c^1)\\ (a^1 + \overline{e^1}) \cdot (d^1 + \overline{e^1}) \cdot (\overline{a^1} + \overline{d^1} + e^1)\\ (\overline{c^1} + a^2) \cdot (c^1 + \overline{a^2})\\ (\overline{b^2}) \cdot (d^2) \cdot (e^2)\\ (a^2 + \overline{c^2}) \cdot (b^2 + \overline{c^2}) \cdot (\overline{a^2} + \overline{b^2} + c^2)\\ (a^2 + \overline{e^2}) \cdot (d^2 + \overline{e^2}) \cdot (\overline{a^2} + \overline{d^2} + e^2)\\ (\overline{c^2} + a^3) \cdot (c^2 + \overline{a^3})\\ (\overline{b^3}) \cdot (d^3) \cdot (e^3)\\ (a^3 + \overline{c^3}) \cdot (b^3 + \overline{c^3}) \cdot (\overline{a^3} + \overline{b^3} + c^3)\\ (a^3 + \overline{e^3}) \cdot (d^3 + \overline{e^3}) \cdot (\overline{a^3} + \overline{d^3} + e^3)\end{gathered}$$

In the above example, the clauses corresponding to gate A in both time frames 1 and 2 are responsible for the discrepancy between the actual and correct response. Specifically, these are $(b^1 + \overline{c^1})$ and $(b^2 + \overline{c^2})$. However, by grouping all clauses derived from gate A together and those from gate B together, irrespective of the time-frames, the single group solution is returned. Below is the modified CNF based on grouping clauses from gate A (B) together with the clause-selector variable y_A (y_B).

$$\begin{gathered}(a^1)\cdot(\overline{b^1})\cdot(d^1)\cdot(e^1)\\(a^1+\overline{c^1}+\overline{y_A})\cdot(b^1+\overline{c^1}+\overline{y_A})\cdot(\overline{a^1}+\overline{b^1}+c^1+\overline{y_A})\\(a^1+\overline{e^1}+\overline{y_B})\cdot(d^1+\overline{e^1}+\overline{y_B})\cdot(\overline{a^1}+\overline{d^1}+e^1+\overline{y_B})\\(\overline{c^1}+a^2)\cdot(c^1+\overline{a^2})\\(\overline{b^2})\cdot(d^2)\cdot(e^2)\\(a^2+\overline{c^2}+\overline{y_A})\cdot(b^2+\overline{c^2}+\overline{y_A})\cdot(\overline{a^2}+\overline{b^2}+c^2+\overline{y_A})\\(a^2+\overline{e^2}+\overline{y_B})\cdot(d^2+\overline{e^2}+\overline{y_B})\cdot(\overline{a^2}+\overline{d^2}+e^2+\overline{y_B})\\(\overline{c^2}+a^3)\cdot(c^2+\overline{a^3})\\(\overline{b^3})\cdot(d^3)\cdot(e^3)\\(a^3+\overline{c^3}+\overline{y_A})\cdot(b^3+\overline{c^3}+\overline{y_A})\cdot(\overline{a^3}+\overline{b^3}+c^3+\overline{y_A})\\(a^3+\overline{e^3}+\overline{y_B})\cdot(d^3+\overline{e^3}+\overline{y_B})\cdot(\overline{a^3}+\overline{d^3}+e^3+\overline{y_B})\end{gathered}$$

For debugging problems with multiple vectors, $\vec{I} = \{I_1, I_2, ...\}$, $\vec{O} = \{O_1, O_2, ...\}$, the union of the CNF problems for each vector results in a single constraint system. In other words the CNF corresponding to the circuit, C is again replicated for each vector. Similar to the approach for sequential circuit, all clauses derived from the same gate, regardless of which replica of C they occur in, must be grouped together and treated as a single error source. It should be noted that the groupings for multiple vectors and sequential circuits is in addition to the gate groupings discussed in Sec. III.

V. Debugging with Approximate Max-sat

In practice, debugging via an exact max-sat formulation may not be practical, as the number of groups and clauses under consideration can be quite high thus resulting in a "hard" max-sat problem. The proposed max-sat strategy can be easily modified to perform an over-approximation instead of finding exact solutions. The benefit of the over-approximation is that the speed and resolution trade-off can be adjusted for the problem: reducing the resolution or granularity of the solutions found yields decreased run-time.

The over-approximation is achieved by grouping clauses together as described in Sec. II-A and finding the MCSes in terms of the groups. Note that the groupings discussed here are in addition to those presented in Sec. III and IV. Different grouping strategies can be easily formulated ranging from random groupings to those based on a circuit's topology or structure. Similarly, groups can differ in cardinality from a single clause to thousands of clauses. For instance, a set of clauses can be grouped together if they are in the same fanout-free cone which is similar to the dominator debugging technique introduced in [10]. Another example is grouping based on a high-level modules derived from RTL similar to the technique of [3]. Intuitively, generating groups based on the circuit's structure or modularity may be advantageous as fewer solutions/suspects may be returned compared to arbitrary grouping schemes.

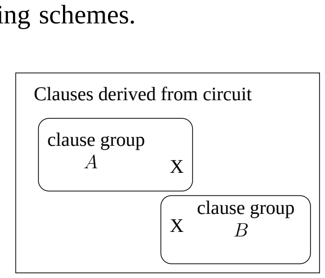


Fig. 3. Error masking in clause groupings

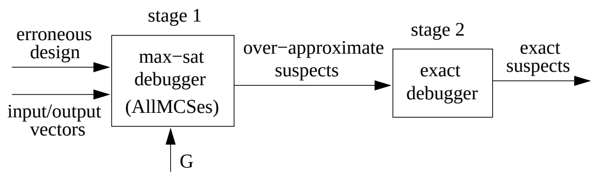


Fig. 4. Max-sat debugging framework

Grouping clauses may increase the effect of error masking, in which some error sources may not be detected as they are masked by others [3]. This also occurs in traditional diagnosis techniques when error-free models are used. For instance, consider the gates shown in Fig. 1 and a pair of errors on gates A and B. In this scenario, the single model-free error, A, masks the pair solution of A and B.

Similar scenarios can occur when grouping clauses together, especially if the groups are made arbitrarily. For instance, consider the CNF illustrated in Fig. 3 where some clauses are grouped in A and other are grouped in B. Further consider a pair of error clauses illustrated by the "X". Here, the single solution identifying group A masks the pair solution A and B. It should be emphasized that error masking is not unique to the proposed technique as it occurs in gate-level and hierarchical debugging as well [3].

A. *Efficient Max-sat Framework*

This section presents a performance optimized debugging framework using the discussed max-sat technique. The complexity of conventional debugging techniques such as SAT-based tools depend to a large extent on the number of suspects that must be considered. In the past, divide and conquer schemes based on the problem hierarchy have proven beneficial [3]. Here, the approximate max-sat approach can be used as a filter to remove the majority of the suspects by quickly finding over-approximate solutions. Subsequently, any exact debugging approach can be used and will benefit greatly by not having to consider all the original suspects during its analysis.

Any type of grouping can be used; however, in the remainder, clauses are grouped in sets of size G according to their corresponding circuit-level topology. Every group contains G clauses (except for one group that contains the remainder of the clauses in the CNF) from gates in close proximity to one another. For sequential circuits and multiple vectors, the group size is $G\times$ [the total number of replications] as described in Sec. IV. Fig. 4 illustrates the flow of the proposed framework where the suspects are first filtered by the max-sat engine and then processed by the exact debugger. The optimal value of G, found experimentally, determines how the debugging effort is divided between the two stages.

VI. Experiments

The proposed framework is implemented in C++ using the max-sat algorithm (AllMCSes) in [8] and the SAT-based debugging engine in [5] as a second stage debugger. Six combinational and ten sequential circuits from ISCAS85, ISCAS89 and ITC99 benchmarks as well as OpenCores.org [9] are used

Circuit and debugging info					Debug	Max-sat20+debug						
						max-sat				debug	total time	X
name	# gates	# vecs	# repl.	# error locs	time (sec)	# grps	# suspects	% susp red	time (sec)	time (sec)	(sec)	improv.
mot-comb1	2, 162	1	1	4	4.79	3	49	97.73%	0.03	0.05	**0.08**	59.88
mot-comb2	5, 487	1	1	13	54.50	13	178	96.76%	0.13	0.24	**0.37**	147.30
mot-comb3	11, 268	1	1	16	357.67	14	189	98.32%	0.27	0.47	**0.74**	483.34
c6288	3, 466	1	1	75	67.96	48	536	84.54%	0.45	1.23	**1.68**	40.45
c7552	2, 644	1	1	248	25.66	74	789	70.16%	0.11	3.11	**3.22**	7.97
c5315	1, 884	1	1	11	4.83	7	99	94.75%	0.04	0.07	**0.11**	43.91
rsdecoder	12, 041	1	2	11	572.68	7	126	98.95%	0.67	0.65	**1.32**	433.85
spi	2, 012	1	21	19	80.54	12	194	90.36%	1.15	2.99	**4.14**	19.45
erp	2, 449	1	3	13	36.09	11	179	92.69%	0.20	0.25	**0.45**	80.20
ac97	15, 599	1	6	4	[TO]	3	58	99.63%	2.22	1.45	**3.67**	> 980.93
reactimer	265	1	512	7	**51.81**	6	89	66.42%	47.58	6.15	53.73	0.96
divider	5, 248	1	15	4	1, 160.39	3	52	99.01%	14.58	1.32	**15.90**	72.98
b14	5, 695	1	22	45	1, 377.86	36	627	88.99%	11.17	50.75	**61.92**	22.25
b15	8, 938	1	13	32	[TO]	40	645	92.78%	96.99	65.82	**162.81**	> 22.11
s15850	10, 481	1	2	19	747.36	12	183	98.25%	0.53	0.71	**1.24**	602.71
s38584	21, 006	1	14	58	[TO]	34	566	97.31%	28.02	36.00	**64.02**	> 56.23
rsdecoder	12, 041	4	8	11	[TO]	7	126	98.95%	2.88	2.01	**4.89**	> 736.20
spi	2, 012	4	81	4	264.07	6	107	94.68%	4.95	4.39	**9.34**	28.27
erp	2, 449	4	12	4	73.71	5	101	95.88%	0.82	0.52	**1.34**	55.01
ac97	15, 599	4	23	4	[TO]	3	58	99.63%	9.95	5.05	**15.00**	> 240.00
reactimer	265	4	1, 745	6	**172.30**	6	89	66.42%	2, 845.80	21.48	2, 867.28	0.06
divider	5, 248	4	71	4	[TO]	3	52	99.01%	54.74	5.44	**60.18**	> 59.82
b14	10, 114	4	1, 216	–	[MO]	–	–	–	[MO]	–	–	–
b15	8, 938	4	62	–	[TO]	–	–	–	[TO]	–	–	–
s15850	10, 481	4	8	19	[TO]	12	183	98.25%	2.21	3.64	**5.85**	> 615.38
s38584	21, 006	4	178	35	[MO]	20	365	98.26%	626.45	376.62	**1, 003.07**	> 3.59

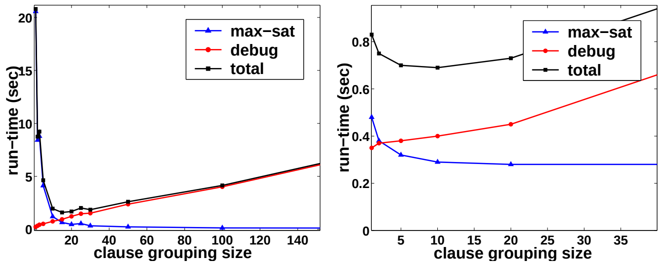
Fig. 5. Max-sat+debug versus standalone debugger

to construct several design debugging problems. The erroneous circuits are obtained by manually changing the functionality of a single gate at random. The failing test vectors are generated by running pseudo-random simulations until an erroneous response is observed. Experiments are conducted using both single and four failing test vectors. The performance of the proposed framework utilizing the max-sat pre-processing is compared against the efficiency of the SAT-based debugging engine in [5] without pre-processing. In all experiments, the size of the clause group error cardinality N_{cg} is set to one to find the single error sources. In addition to the groups created for the over-approximation, clauses are also grouped together based on the circuit replicas as discussed in Sec. IV. Experiments are conducted on a Pentium IV 2.8 GHz Linux platform with a 1GB memory limit and 3600 seconds time-out.

In order to determine the effectiveness of the overall debugging framework of Sec. V-A as a function of the group size G, experiments are conducted on several representative circuits. Fig. 6 (a) and (b) shows two such experiments, using circuit c6288 and mot-comb3, where three curves representing the run-times of the over-approximate max-sat stage, the exact debugging stage, and the combined run-times are presented for several group sizes. The run-time of max-sat increases abruptly as the group size becomes very small, and it reaches a maximum when the exact method is used (single-clause groups). However, as the group size increases, the run-time of the second stage debugger increases as it must consider many more suspects due to the over-approximation. The combined curve shows the total run-time of the overall framework is minimized with group sizes of roughly 10 to 20 clauses.

In the remaining, "max-sat20+debug" refers to the proposed framework with a grouping size of $G = 20$. For sequential designs and multiple vectors the actual number of clauses per group is 20 times the number of circuit replicas. Figure 5 compares max-sat20+debug to the standalone debugger of [5]. Rows $1-6$ report experiments with combinational circuits given a single failing test vector, and $7-16$ ($17-26$) report experiments with sequential circuits given one (four) failing test vector(s). The first four columns contain the circuit's name, its size in gates, the number of test vectors used, and the total number of circuit replicas needed. The fifth column (*# error locs*) gives the total number of potential error locations that could explain the faulty behavior of the circuit (the complete set). These are the locations expected to be returned by both approaches when available. The sixth column gives the run-time of the standalone debugger. An entry of [TO] denotes a time-out, and [MO] denotes a memory-out.

The remaining columns present the results of our proposed



(a) c6288 (b) mot-comb3

Fig. 6. Run-time versus clause grouping size

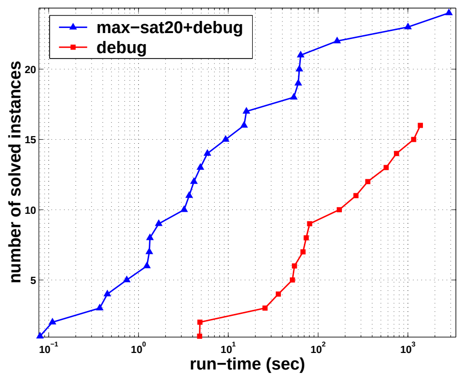


Fig. 7. max-sat20+debug versus debug

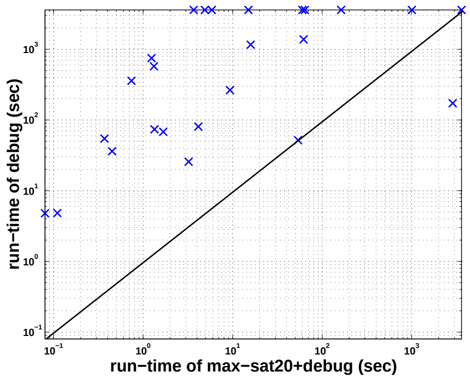


Fig. 8. max-sat20+debug versus debug

framework. The first four (*# grps*, *# suspects*, *% susp red*, and *time (sec)*) report the number of groups (of $20\times$ *# repl.* clauses) returned by the AllMCSes algorithm in any MCS; the number of *suspect* variables identified by those groups, each corresponding to a potential gate-level error source; the percent reduction in the number of suspect gates; and the run-time of this first stage. The true benefit of the proposed technique is evident when considering the number of suspects that are filtered by the first stage with relatively small run-time. For instance consider the circuit ac97 with a single vector. The approximation technique rules out 99.64% of the suspects in just 2.22 seconds. On average, the number of suspects is reduced by over 92%.

The run-time in seconds of the second stage debugger using the suspects of the first stage is shown in column *debug time (sec)*. Finally, the *total time (sec)* column shows the combined run-time of the proposed framework. This number is compared with the run-time of the standalone debugger in column six to get the improvements shown in the final column (*X improv.*).

These results demonstrate the overwhelming advantage of the proposed method over the standalone debugging engine as the run-times are reduced by an average of 200 times. For combinational circuits, the number of solved instances is increased from 16 to 24 out of 26, a 50% improvement, and for sequential circuits with one (four) test vector(s), the number of solved instances is increased from 7 (3) to 10 (8), a 43% (167%) improvement.

Fig. 7 plots the number of solved instances as a function of run-time on a logarithmic scale for max-sat20+debug and standalone debug. It can be seen that max-sat20+debug outperforms the standalone approach by roughly two orders of magnitude across all problems. Fig. 8 plots the total run-time of max-sat20+debug for each instance against the corresponding run-time of standalone debugger on a logarithmic scale. Clearly, most points lie above the 45^o line which indicate the better performance of the proposed framework. Points on the upper border indicate the instances solved by max-sat20+debug but unsolved by the standalone approach. The single point where the proposed framework fares essentially worse is caused by the large run-time of the first stage. Such cases can be addressed by increasing the group size G, thus reducing the difficulty for the AllMCSes algorithm.

VII. Conclusion

This work presents an efficient two stage debugging framework which uses a novel max-sat problem formulation. First, it is shown that the debugging problem can be solved exactly with a max-sat formulation. The approach is extended for sequential circuits and for problems with multiple vectors. An over-approximation technique is developed to take advantage of the strengths of the max-sat techniques. This technique considers groups of clauses together and can thus make decisions based on the groups instead of the individual clauses. The over-approximation technique is used as a pre-processing step that filters the majority of suspects and reduces the problem complexity drastically for any debugger used in the second stage. Experiments demonstrate overwhelming run-time improvements of two orders of magnitude on average.

References

[1] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[2] M. Abramovici, P. R. Menon, and D. T. Miller, “Critical path tracing - an alternative to fault simulation,” in *DAC ’83: Proceedings of the 20th conference on Design automation*, 1983, pp. 214–220.

[3] M. F. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler, “Post-verification debugging of hierarchical designs,” in *Int’l Conf. on CAD*, 2005, pp. 871–876.

[4] P. Bjesse and A. Boralv, “DAG-aware circuit compression for formal verification,” in *Int’l Conf. on CAD*, 2004, pp. 42–49.

[5] M. Fahim Ali, A. Veneris, S. Safarpour, R. Drechsler, A. Smith, and M.S.Abadir, “Debugging sequential circuits using Boolean satisfiability,” in *Int’l Conf. on CAD*, 2004, pp. 204–209.

[6] S.-Y. Huang, “A fading algorithm for sequential fault diagnosis,” in *DFT ’04: Proceedings of the Defect and Fault Tolerance in VLSI Systems, 19th IEEE International Symposium on (DFT’04)*, 2004, pp. 139–147.

[7] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, “Robust Boolean reasoning for equivalence checking and functional property verification,” *IEEE Trans. on CAD*, vol. 21, no. 12, pp. 1377–1394, 2002.

[8] M. Liffiton and K. A. Sakallah, “On Finding All Minimally Unsatisfiable Subformulas,” in *Int’l Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 32–43.

[9] OpenCores.org, “http://www.opencores.org,” 2006.

[10] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, “Fault diagnosis and logic debugging using Boolean satisfiability,” *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[11] A. Veneris and I. N. Hajj, “Design error diagnosis and correction via test vector simulation,” *IEEE Trans. on CAD*, vol. 18, no. 12, pp. 1803–1816, 1999.