Spatial and Temporal Design Debug using Partial MaxSAT

Yibin Chen, Sean Safarpour, Andreas Veneris
Department of Elec. and Comp. Eng.
University of Toronto, Toronto, Canada
{yibin,sean,veneris}@eecg.toronto.edu

Joao Marques-Silva
Electronics and Computer Science
University of Southampton, Southampton, UK
jpms@ecs.soton.ac.uk

ABSTRACT

Design debug remains one of the major bottlenecks in the VLSI design cycle today. Existing automated solutions strive to aid engineers in reducing the debug effort by identifying possible error sources in the design. Unfortunately, these techniques do not provide any information regarding the time at which the bug is active during an error trace or counter-example. This work introduces an automated debug technique that provides the user with both spatial and temporal information about the source of error. The proposed method is based on a Partial MaxSAT formulation which models errors at the CNF clause level instead of the traditional gate or module level. Thus, error sites are identified based on erroneous implications that correspond to locations both in the design and in the error trace. Experiments demonstrate that we can provide this additional information at no extra cost in run time and are able to prune about 61% of all simulation time frames from the debugging process. When compared to a trivial formulation we observe a performance improvement of up to two orders of magnitude and 5× on average when using the proposed formulation.

Categories and Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer-aided design (CAD)

General Terms: Algorithms, Verification

Keywords: Design Debugging, Maximum Satisfiability

1. INTRODUCTION

As design tools and methodologies for today's Systems-on-Chip (SoCs) and VLSI designs become increasingly sophisticated, designing a bug free circuit remains the exception rather than the norm. Functional verification tasks pose a major bottleneck in the design process, consuming up to 70% of the design effort [1]. A multitude of methodologies, formal and semi-formal techniques exist for verifying design functionality [2, 3]. These methods and techniques verify whether a design implements its given specification. However, once the design fails verification, the root cause of the failure must be identified and rectified manually. As the complexity of digital designs steadily increases, the cost of design debug becomes substantial and unpredictable due to the overwhelmingly manual nature of the debugging process [4] and the increased intricacy of the design cycle. The tremendous time-to-market pressures of today's applications make automated design debug tools essential.

Today, design debug comprises of heavy manual engineering tasks such as examining stimulus traces, analyzing the design components, and back-tracing design signals. The result of this arduous debug effort is a set of components and circumstances responsible for the functional failure. Traditionally, automated debug solutions for hardware designs have been proposed based on simulation [5], path tracing [6], and Binary Decision Diagrams (BDDs) [7]. More recently, advances based on formal engines such as SAT [8], QBF [9], and MaxSAT solvers [10] have been successful at helping the engineers. Without exception, all existing automated debug techniques identify components (gates or modules) in the design for manual analysis. Thus they provide spatial debug information with respect to the error. Surprisingly, none provide temporal information, that is, *when* during the verification phase the error is *active* (*i.e.* it is excited and its effects are propagated to an observation point). Temporal information is very important for designers when determining how to remove the error and correct the design [11].

This work presents a novel alternative formulation to the automated debugging problem for sequential circuits where the solution is not limited to spatial error sources. More specifically, the proposed technique identifies errors both *spatially* and *temporally* thus localizing where and when during the verification trace the errors are active.

The basis of our technique is a departure from conventional debugging frameworks where errors are modeled as either gates or modules [6,8,11]. Instead, errors are modeled as implications or clauses in the Conjunctive Normal Form (CNF) representation. By identifying a set of CNF clauses as the root cause of an error, we can determine where in the design and when in the simulation trace the erroneous gates are excited to cause the wrong behavior.

Specifically, our approach builds an unsatisfiable Boolean formula from the design, the verification trace and the expected behavior. We use a Partial MaxSAT solver to extract the maximal subset of clauses that is satisfiable and complement this set to derive the minimal set of clauses whose removal make the CNF formula satisfiable. This minimal set represents a set of potential error sources which if corrected allow the circuit to pass verification.

Our major contributions are summarized as follows:

- A novel method for efficiently determining spatial and temporal error sources.
- A formulation of the design debug problem using a Partial MaxSAT encoding with minimal overhead.
- An error cardinality model based on error excitations and propagations.

To demonstrate the effectiveness of the proposed technique, we develop an automated debug framework using the Partial MaxSAT solver in [12]. We show that our technique accurately identifies time frames where errors are excited and their effect propagated to the outputs. We also show that our formulation is superior to a non-optimized clause

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GLSVLSI'09, May 10–12, 2009, Boston, Massachusetts, USA.
Copyright 2009 ACM 978-1-60558-522-2/09/05 ...$5.00.

level MaxSAT approach resulting in speedups of up to two orders of magnitude with an average of 5×. Versus an existing MaxSAT-based debugger, our technique is more effective as it provides additional temporal information while demonstrating a competitive 1.29× speedup.

The remainder of this paper is structured as follows. Section 2 discusses some of the previous contributions to the field. In Section 3 we provide the necessary background regarding automated design debugging and Partial MaxSAT. Our proposed debugging approach is given in Section 4. Section 5 extends on ideas from Section 4 by introducing a new cardinality model for sequential circuits. Finally, our experiments and conclusions are given in Sections 6 and 7.

2. PREVIOUS WORK

Existing formal SAT-based techniques for design debugging can be grouped into two broad categories. Approaches that are based on *satisfiability* (i.e. finding a satisfying assignment to the CNF problem) and those that are based on *unsatisfiability* (i.e. identifying which parts of the CNF problem cannot be satisfied). For this paper we will mainly focus on approaches based on unsatisfiability as the debugging problem is naturally unsatisfiable and recent advances in Maximum Satisfiability solvers are showing promising results for industrial applications [13].

There are two major contributions to the field of design debugging using unsatisfiability. In [10] the first MaxSAT formulation for design debug is introduced. The use of clauses for identifying exact error locations in combinational circuits is presented but deemed impractical. Instead, for sequential circuits, it is shown that MaxSAT can be used as a powerful tool to group clauses together for a quick over-approximation of the solutions. Error locations are modeled at the gate level as clauses are grouped across time frames. By increasing the granularity of errors, Safarpour et al [10] combine a groupings based MaxSAT formulation with an exact SAT based debugger to achieve performance gains.

Furthermore the solver [14] used by [10] significantly differs from the solution technique presented here. In [14] all satisfiable subsets are enumerated, independent of their size, using disjunctions of relaxation variables. This paper presents an algorithm which enumerates MaxSAT solutions using no relaxation variables.

In [15], unsatisfiable cores are used to speed up the debugging process for multiple fault diagnosis problems. This approach extracts a set of unsatisfiable cores from the CNF problem and prunes potential error locations not contained in any of the cores. A SAT based exact debugger is then used to find the actual error locations from the reduced problem.

Both contributions focus on using unsatisfiability during pre-processing to improve performance. Error sources are modeled as physical locations in the design (gates or modules) and the final solution is returned by a secondary SAT based debugger. Our approach differs from previous approaches in that we do not attempt to balance the use of unsatisfiability and satisfiability for performance gains. Instead we use a Partial MaxSAT solver on sequential circuits to find the exact error location in the design without the need for an additional solver. Our formulation models errors at a finer level of granularity offering a better resolution than other approaches in addition to being the first formulation able to locate suspects in time. Even though the search space of our problem is significantly increased, our experiments show that the impact on run time is insignificant due to advances made in modern MaxSAT solvers.

3. BACKGROUND

3.1 Automated Design Debugging

Design debugging occurs at the early stages of the design cycle when the implementation of a design does not meet the specification. At this stage, the RTL design has failed verification (simulation or formal). Given an erroneous circuit, a sequence of input values (stimulus trace) and expected output values, design debugging seeks to find a set of error sources in the design, that if corrected can rectify the problem. This is similar to fault diagnosis which focuses on locating defects in silicon [6]. While the techniques presented here are also applicable to fault diagnosis, we will mainly focus our discussion on design debugging.

Traditionally, error sources and their corresponding correction models are represented at either the gate or module level [8, 11]. For SAT [8] and QBF [9] based debug, the problem is converted to CNF using techniques that may [16] or may not [17] take into account circuit information. The stimulus and expected behavior are then used to constrain the resulting CNF problem. Additional constraints are added to limit the maximum number of errors, that can be simultaneously activated. Finding a satisfying assignment to the resulting formula effectively finds a set of possible error locations E_l. A debugger is limited to finding the set of all sites E_q functionally equivalent to E_l, such that $|E_q| \leq N_e$. N_e is a user defined cardinality providing the upper bound on the number of errors that can be active in the circuit simultaneously [8]. Sets of error locations are said to be *functionally equivalent* if they cannot be functionally distinguished from each other under a given stimulus trace [6].

3.2 Maximum Satisfiability

This section reviews MaxSAT and its extensions, and briefly overviews recent algorithms for MaxSAT, capable of handling large complex problem instances.

Given an *unsatisfiable* CNF formula Φ, the MaxSAT problems consists of identifying an assignment to the problem variables such that the number of satisfied clauses in Φ is maximized [18]. The MaxSAT problem is a well-known NP-Hard optimization problem.

In the *Partial MaxSAT* problem the CNF formula is organized into a set of *hard* clauses, which must be satisfied, and a set of *soft* clauses, which may or may not be satisfied, *i.e.* $\Phi = \Phi_H \cdot \Phi_S$. For Partial MaxSAT problems the objective is to find an assignment that satisfies all the hard clauses and that maximizes the number of satisfied soft clauses.

In the remainder of this paper, *hard* clauses will be represented in square brackets and *soft* clauses in round brackets. For example, consider the following formula:

$$\Phi = [x_1 + \overline{x_2}][x_3] \cdot (\overline{x_1})(x_2)(\overline{x_3} + x_1) \quad (1)$$

The first two clauses are hard clauses, and so must be satisfied, whereas the remaining three clauses are soft clauses and may or may not be satisfied.

In the recent past [18], the most effective MaxSAT algorithms have been based on branch-and-bound search (B&B), supported by effective lower bounding and dedicated inference techniques. Nevertheless, most of the experimental evaluation associated with B&B MaxSAT solvers assumed random and handmade problem instances, which unfortunately often bear little relationship with hard industrial instances. As a result, recent work has addressed alternative approaches, aiming the use of MaxSAT algorithms in industrial settings, and focusing on instances derived from realistic applications. The most effective algorithms are based on solving MaxSAT with unsatisfiable sub-formula identification and relaxation [12,19,20]. The relaxation of the clauses in each unsatisfiable sub-formula is achieved by associating a relaxation variable with each such clause. Cardinality constraints are used to constrain the number of relaxed clauses.

4. DEBUGGING SEQUENTIAL CIRCUITS WITH PARTIAL MAXSAT

For the design debugging problem we are primarily interested in sequential circuits specified using logic gates and state elements. We use time frame expansion to model the behavior of state elements over a finite number of clock cycles k. This technique effectively transforms the sequential circuit into a combinational circuit, otherwise known as an

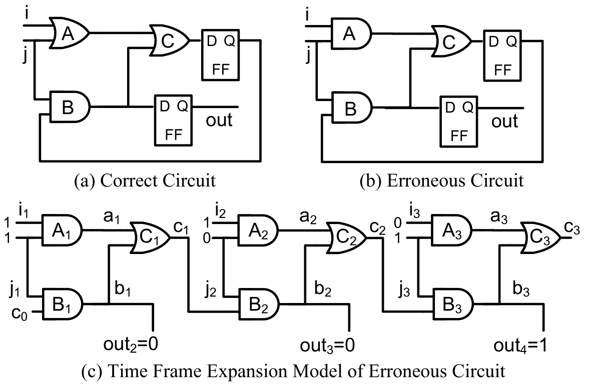


Figure 1: Correct and erroneous circuit for Example 1

Iterative Logic Array (ILA) [6], by replicating the combinational portion of the circuit k times. Adjacent time frames are connected by their respective next and current state variables. Let $ILA_k(C)$ be the ILA obtained by expanding a buggy sequential circuit C over k time frames. Let $CNF(ILA_k(C))$ denote the Boolean formula obtained by translating each gate in $ILA_k(C)$ into their respective CNF representation as in [17]. We can formulate the debugging problem as a Partial MaxSAT problem as follows:

$$\Phi = \prod_{i=1}^{k}[I_i][O_i] \cdot [IS] \cdot CNF(ILA_k(C)) \qquad (2)$$

Where $I_1, I_2, \ldots, I_k$ is the sequence of input constraints provided by the stimulus trace, $O_1, O_2, \ldots, O_k$ is the sequence of constraints based on the expected output values, and IS is the initial state.

As the erroneous circuit cannot produce the expected output response, Φ is inherently unsatisfiable. The complement of the solution set obtained from a Partial MaxSAT solver is the minimal set of clauses whose removal satisfies Φ (i.e. the clauses corresponding to a possible error). Due to the many-to-one mapping between clauses and gates in the CNF, this solution set also corresponds to a minimal collection of logic gates in C that may be responsible for the discrepancy between the observed and expected behavior. The input, output, and initial state constraints are specified as hard clauses, as indicated by the square brackets, since their removal trivially satisfies the CNF formula. We can also specify trusted portions of the circuit (such as adders and multipliers) as hard clauses to reduce the solution space. For the remainder of this paper the phrase 'MaxSAT solution' will refer to the set complement of the Partial MaxSAT solution.

The most basic Partial MaxSAT formulation is one that relates each clause found as a solution back to its gate or module level representation. The following example demonstrates this process for a simple circuit.

Example 1 *Fig. 1(a) and (b) give an example of a correct and erroneous circuit. Gate A in the correct circuit was wrongly implemented by an AND gate. Fig. 1(c) shows the time frame expansion model of the erroneous circuit under a set of input and correct output constraints and an initial state of 0. The time frame number is given by the subscript.*

Note that that since the output is registered, the output values provided by b_1, b_2, and b_3 are constrained by the expected output values in the next time frame. The CNF representation of the ILA from Fig. 1(c) is given below along with the gates represented by each set of clauses.

$$[i_1][j_1][\overline{c_0}][\overline{b_1}]$$
$$A_1\text{: } (i_1+\overline{a_1})(j_1+\overline{a_1})(\overline{i_1}+\overline{j_1}+a_1)$$
$$B_1\text{: } (j_1+\overline{b_1})(c_0+\overline{b_1})(\overline{j_1}+\overline{c_0}+b_1)$$
$$C_1\text{: } (\overline{a_1}+c_1)(\overline{b_1}+c_1)(a_1+b_1+\overline{c_1})$$
$$[i_2][\overline{j_2}][\overline{b_2}]$$
$$A_2\text{: } (i_2+\overline{a_2})(j_2+\overline{a_2})(\overline{i_2}+\overline{j_2}+a_2)$$
$$B_2\text{: } (j_2+\overline{b_2})(c_1+\overline{b_2})(\overline{j_2}+\overline{c_1}+b_2)$$
$$C_2\text{: } (\overline{a_2}+c_2)(\overline{b_2}+c_2)(a_2+b_2+\overline{c_2})$$
$$[\overline{i_3}][j_3][b_3]$$
$$A_3\text{: } (i_3+\overline{a_3})(j_3+\overline{a_3})(\overline{i_3}+\overline{j_3}+a_3)$$
$$B_3\text{: } (j_3+\overline{b_3})(c_2+\overline{b_3})(\overline{j_3}+\overline{c_2}+b_3)$$
$$C_3\text{: } (\overline{a_3}+c_3)(\overline{b_3}+c_3)(a_3+b_3+\overline{c_3})$$

One of the possible solutions returned by the MaxSAT solver is $(j_2+\overline{a_2})$ from A_2, correctly implying that the behavior of gate A at time frame 2 is the cause of the error. Since the erroneous behavior due to the incorrect gate A also propagates through gates C and B, the clauses $(a_2+b_2+\overline{c_2})$ and $(c_2+\overline{b_3})$ are also possible MaxSAT solutions. In fact, these 3 solutions are the only solutions obtainable for this trace.

4.1 Obtaining Multiple Solution Sets

Typically, a Partial MaxSAT solver will only return a single solution per call. In Example 1, any one of the three solution clauses could be returned by the solver as all of them are of minimal cardinality. For the debug problem, we are generally interested in obtaining all sets of clause level solutions of cardinality $\leq N_c$, where N_c is a user defined constant. This requires a mechanism to effectively block previously found solutions during iterations. For solutions of cardinality one, this can be trivially done by denoting them as hard clauses instead of soft clauses in the CNF.

For cardinality m solutions however ($m > 1$), a set of solution clauses $S_c = \{Cl_1, Cl_2, \ldots, Cl_m\}$ needs to be blocked. This means that for future solutions at least one of the clauses in S_c must evaluate to true in the satisfying assignment of the satisfiable subset of clauses obtained by MaxSAT. Thus S_c can be blocked as a solution by adding one additional hard clause $Cl_c = [S_c] = [Cl_1 + Cl_2 + \cdots + Cl_m]$ to the CNF.

Using $\overline{maxsat(\Phi)}$ to denote a solver returning a minimal set of unsatisfiable clauses, the algorithm to obtain all sets of clause level error sources S_c such that $|S_c| \leq N_c$ is given in Algorithm 1.

Algorithm 1 The mxs_solve algorithm

```
mxs_solve(Φ,N_c)
1: error_sources ⇐ ∅
2: S_c ⇐ overline{maxsat}(Φ)
3: while |S_c| ≤ N_c and S_c ≠ ∅ do
4:     Φ ⇐ Φ · [S_c]
5:     error_sources = error_sources ∪ {S_c}
6:     S_c ⇐ overline{maxsat}(Φ)
7: end while
8: return error_sources
```

4.2 Extracting Temporal Information

The MaxSAT solutions also provide valuable information regarding the temporal location of the errors. In Example 1, all the solution clauses originate from gates A_2, C_2, B_3 in either time frame 2 or 3. Thus, further analysis or correction by the engineer can focus on clock cycles 2 or 3 within a stimulus trace or counter-example. Incidentally, these solution clauses describe a propagation path from the error source in time frame 2 to the observed error at the output in time frame 4. Even though gate A is also erroneous in time frame 1, the value of a_1 actually matches its expected value. Thus the error is undetectable in time frame 1. In contrast, existing automated debug solutions only provide spatial information (gates A, B and C) regarding error suspects.

Our experiments show that our method can reduce the number of time frames requiring analysis for debug by an average of 61%. Furthermore, we can measure the frequency that a certain time frame is implicated from the set of solutions returned. Experimentally, time frames which are implicated by solutions more frequently are more likely to contain the actual error excitation. In the example given, two out of three solutions implicate the actual erroneous time frame. For longer simulation traces this analysis allows the designer to shorten the debugging process by prioritizing their efforts to a small selection of simulation time frames.

4.3 Using Literal Information

Another benefit of modeling error sources at the clause level is that a solution clause returned by MaxSAT does not merely present a specific gate as erroneous. It also provides additional information regarding the nature of the problem. For instance, consider the solution $(a_2 + b_2 + \overline{c_2})$ from Example 1. Since the removal of this clause makes the CNF problem satisfiable, this clause must evaluate to false in the satisfying assignment. The following observations can be deduced regarding the nature of the error.

1. The error could have been caused by the incorrect behavior of gate C.
2. The literal $\overline{c_2}$ implies that setting $c = 0$ (gate C) only in time frame 2 by some circuit modification would rectify the problem.
3. The literals a_2 and b_2 imply that changing either the output value of gate A or gate B in time frame 2 from 0 to 1 would rectify the problem.

Of these points, only observation 1 is returned by traditional automated debug techniques. Observations 2 and 3 are unique to clause level debugging as they reason about implications in a particular time frame.

Further analysis can be made for the input literals in observation 3. Due to the output constraint imposed on on gate B at time frame 2 ($out_3 = 0$), setting $b_2 = 1$ is not a viable option. In general, this means that these additional *implied* solutions should be checked against other clauses to ensure the correcting assignment does not cause another clause to evaluate to 0 due to multiple fanouts. Thus, gate A in time frame 2 is the only other potential error source implied by this solution.

Since each clause level solution effectively returns a small cluster of gates as possibly erroneous we can effectively add the extra gates found to the set of error sources returned by Algorithm 1. Continuing with our above example, this means that both the gates A_2 and C_2 are added as potential error candidates to the *error_sources* set after finding the clause $(a_2 + b_2 + \overline{d_2})$.

5. MODELING ERROR CARDINALITY AS EXCITATIONS AND PROPAGATIONS

In existing gate level debugging approaches, the maximum cardinality of solution sets N_g is given by the user [8]. That is, N_g represents an estimate on the maximum number of simultaneous gate level error sources active in the circuit. In this section we establish a new way to express error cardinality which describes error sources at a finer level of granularity. To do so, we must first find the relationship between between N_g and N_c.

Consider the case where Φ is derived from a single clock cycle simulation of a circuit C. We can treat C as a combinational circuit. Let E be the set of all functionally equivalent gate level error locations of cardinality $\leq N_g$. We define m_{cl} to be the maximum number of clauses generated for any gate in the circuit. For example, for a circuit with only 2-input AND/NAND gates and NOT gates, $m_{cl} = 3$. Due to the minimality of solutions returned by $\overline{maxsat}(\Phi)$ every element in E is found by *mxs_solve* for $N_c = N_g \cdot m_{cl}$.

For circuit problems optimized through Boolean Constraint Propagation (BCP), where all unit literal clauses are removed, m_{cl} can be further be replaced by $(m_{cl} - 1)$ in the above upper bounds.

Theorem 1: In BCP optimized circuit problems with no unit literal clauses the maximum number of clauses that can be unsatisfiable per gate is $m_{cl} - 1$.

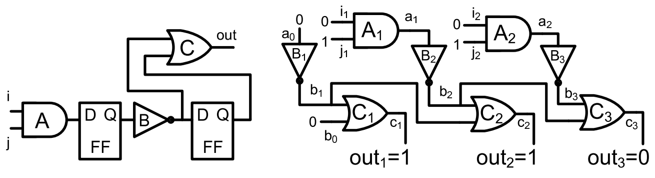


Figure 2: Erroneous circuit and its respective time frame expansion model for Example 2

Proof: Every clause in the minimal CNF representation of a single output logic gate G with output variable y must either contain the literal y or $\overline{y}$. Both the literals y and $\overline{y}$ must appear in $CNF(G)$ at least once as otherwise $CNF(G)$ could be reduced to a unit literal clause by BCP. Therefore, no matter what value y is assigned at least one clause in $CNF(G)$ is satisfied. Thus MaxSAT can return a maximum of $(m_{cl} - 1)$ clauses per gate. ■

For sequential circuits, since a gate is replicated k times in $ILA_k(C)$, in the worst case $N_c = (m_{cl} - 1) \cdot N_g \cdot k$ to find all gate level solutions of cardinality $\leq N_g$. Notice that k actually denotes the maximum number of times that a gate level error is active, *i.e.* the gate level error is excited and its effect is propagated to the output [6]. Since in general a gate level error is not active for every clock cycle, k can be effectively replaced by k_{ep}, where $k_{ep} \leq k$ denotes the expected number times the error site is active.

Since errors are modeled at the clause level, errors from the same or different gate level sources are not distinguished. Thus, once a clause cardinality N_c is specified, *mxs_solve* finds all clause level errors irrespective of their corresponding gates. As a result, N_c can be more appropriately specified as $N_c = (m_{cl} - 1) \cdot N_{ep}$, where N_{ep} is the maximum expected number of error excitations and propagations for a given stimulus trace. This is in contrast to previous definitions of gate level error cardinality which demand the estimated number of error locations that are active in the circuit at once. Similar to N_g, the user can provide an estimate for this number based on trace length and the complexity of the problem [8].

Example 2 shows a simple circuit where using a N_c independent of the gate level locations provides valuable debug information.

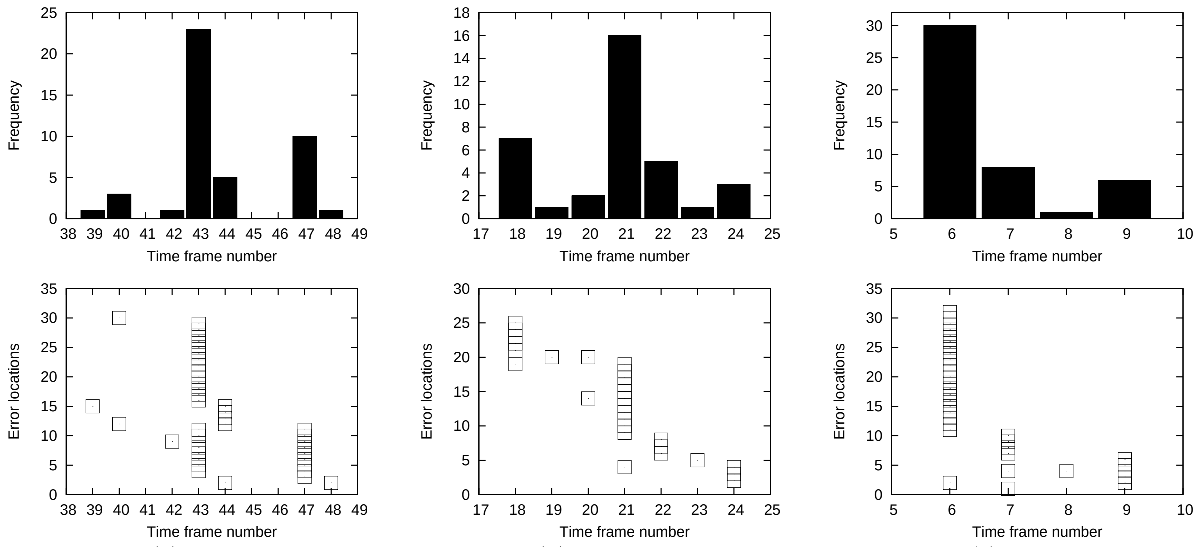
Example 2 *Consider the erroneous circuit in Fig 2(a). Some gates irrelevant to the problem are omitted in the time frame expansion model of Fig 2(b). The correct circuit is derived by replacing gate A with an OR gate so the actual number of error excitations and propagations is 2. Suppose $m_{cl} = 3$ and we guess $N_{ep} = 2$. Consequently, mxs_solve with $N_c = (m_{cl} - 1) \cdot N_{ep} = 4$ returns a total of 9 solutions (all of cardinality 2). Three of these solutions are given below with their gate level representations shown for clarity.*

$$S_1 = \{A_2 : (i_2 + \overline{a_2}), B_2 : (a_1 + b_2)\}$$
$$S_2 = \{A_1 : (i_1 + \overline{a_1}), A_2 : (i_2 + \overline{a_2})\}$$
$$S_3 = \{C_3 : (\overline{b_3} + c_3), C_3 : (\overline{b_2} + c_3)\}$$

These three solutions return three different types error sources. In the case of S_1, the two possible error locations are gates A and B in time frame 2. For S_2 the spatial location is gate A, while the temporal locations are time frames 1 and 2. Finally, S_3 implicates gate C in time frame 3. Note that additional solutions can be inferred when taking into consideration input literals.

6. EXPERIMENTS

In this section we demonstrate the effectiveness of our temporal and spatial debug techniques. The techniques described in this paper are implemented using C++, Perl and the Partial MaxSAT solver from [12]. All experiments are run on a 2.20GHz Intel Core2 Duo machine with 2GB of memory and a timeout of 6 hours. In total five educational



(a) cct4 (b) divider-2 (c) fpu-1

Figure 3: Temporal debug information based on frequency of time frames found for circuits cct4, divider-2, and fpu-1

circuits (`cct1-5`) and three circuits obtained from OpenCores.org [21] (`divider`, `fpu`, `rsdecoder`) are presented. These circuits are modified at the RTL level where a single Verilog error is inserted. The buggy circuits are first simulated to verify that the errors are detected. They are then synthesized and converted to CNF using the method in [17]. The input and output constraints are captured from simulation.

As described in Section 4 providing temporal debug information such as the most likely time frame when an error is active is crucial. The temporal information provided by our technique is illustrated in Fig. 3. The histograms in Fig. 3 show the frequency a time frame is implicated by a literal in solution clauses. The likelihood of an error being active at a particular time frame is indicated by the height of the bars. The scatter plots underneath the histograms illustrate which error locations are found for which time frame. The y-axis lists all unique error locations found by the algorithm and the x-axis shows the time frames during which these locations can be excited to cause the error.

Fig. 3(a) shows the results for `cct4` which is a large state machine with a wrong state transition. In total 30 unique error locations are found. The graphs show two visible spikes at time frames 43 and 47. The actual error location was excited in time frame 43.

For `divider-2` (Fig. 3(b)) a constant assignment is inserted into the datapath and excited by the testbench at time frame 21. We see a wider distribution of possible erroneous time frames as the erroneous behavior is propagated through multiple pipeline stages

Similarly, for `fpu-1`, an RTL operation is replaced with another in the datapath. The peak in Fig. 3(c) correctly implies that the bug is excited in time frame 6. As the error propagates through the datapath the number of possible error sources for the observed bug decreases.

These graphs can allow the designer to focus on these locations during these time frames to correct the problem. For all three circuits our technique correctly indicates the time frame during which the bug is excited and propagated to the output. The frequency of the erroneous time frame, *i.e.* when the error is active, is more than double that of the next highest data point for the three circuits shown.

These histograms and scatter plots can therefore be used to reduce the amount of time spent on analyzing simulation waveforms by reducing the search space to a few clock cycles. For example in `cct4`, if the engineer is aware of when state transitions occur, the error source in the state machine may be deduced even without any spatial information.

Table 1 summarizes the performance of our technique on all the sample circuits. For the larger circuits multiple interesting error instances are considered (four for `divider`, three for `fpu`, two for `rsdecoder`).

Columns 1 to 3 give the instance of the buggy circuit, the number of gates, and the number of state elements, respectively. Columns 4 and 5 indicate the length of the stimulus trace in clock cycles and the number of literals in the MaxSAT formulation. The number of equivalent gate level error locations is given in column 6.

Columns 7 to 10 provide run time information for the basic Partial MaxSAT formulation as described in Section 4. For each clause, only the gate that it was derived from is considered a solution. Column *max card* provides the maximum cardinality N_c required to find all error locations at the clause level. The run time to find all equivalent error locations using this method is summarized in columns *iter* (number of iterations), *time/iter* (average solver time per iteration), and *total* (total run time).

Columns 11 to 14 present the cardinality requirements and run time results for the case where clause literals are interpreted as solutions as presented in Section 4.3. The improvement in run time of this method over the basic formulation is given by the *improv* column. On average we observe a speedup of around 5× for finding all equivalent error locations when comparing against the basic formulation with best results reaching up to two orders of magnitude (`cct4` and `fpu-1`). Finally, the percentage of time frames pruned due to temporal information is given in the last column. Our formulation is able to prune about 61% of all possible error excitation time frames on average. Note that this number does not take into consideration visible spikes in frequency as given by the histograms in Fig.3 which could further reduce the time to find the actual erroneous time frame.

Consider `divider-2` with 5670 gates and 424 state elements which is simulated for 27 clock cycles resulting in a MaxSAT formulation with 176,481 literals. The inserted error results in 15 equivalent error locations at the gate level. The clause based method finds all 15 error locations after 64 iterations with an N_c of 2. In contrast, our literal based method finds all error locations in 13 iterations requiring an N_c of 1. Since the error trace ran for a total of 27 time frames and our formulation found 7 possible time frames during which the error could be excited the percentage of possible error excitation time frames pruned is 74%.

We also implemented the debug problem using groupings similar to the method presented in [10]. In this variation,

Table 1: Runtime comparison between clause level MaxSAT formulations with and without literals analysis

Circuit and error information						Debug using only clause information				Debug using literals information					
circuit	# gates	# DFF	# CLK	# lits	# equiv error locs	max card	# iter	time/iter (sec)	total (sec)	max card	# iter	time/iter (sec)	total (sec)	improv	%tf pruned
cct1	225	16	15	4,020	7	2	13	0.57	7.45	1	11	0.56	6.19	1.20	60%
cct2	282	24	15	5,117	3	2	7	0.92	6.47	2	4	0.85	3.40	1.90	73%
cct3	234	11	32	8,567	12	2	14	1.52	21.33	2	10	1.51	15.08	1.41	85%
cct4	450	8	50	23,586	19	3	450	6.61	2,974.57	3	21	6.22	130.63	22.77	86%
cct5	226	8	17	4,305	11	2	12	0.71	8.48	2	8	0.70	5.61	1.51	71%
divider-1	5,670	424	39	255,134	126	-	TO	-	-	2	115	52.78	6,069.39	>3.56	11%
divider-2	5,670	424	27	176,481	15	2	64	32.48	2,078.99	1	13	30.52	396.77	5.24	74%
divider-3	5,670	424	12	78,347	19	2	77	13.70	1,055.26	1	21	15.40	640.94	3.26	17%
divider-4	5,670	424	21	137,319	8	1	11	27.68	304.43	1	6	27.68	166.05	1.83	86%
fpu-1	19,868	715	12	255,594	25	2	325	58.22	18,920.49	1	19	48.46	920.76	20.55	67%
fpu-2	19,868	715	12	255,594	6	2	12	49.87	598.49	1	5	48.456	242.28	2.47	67%
fpu-3	19,868	715	12	255,594	29	2	54	51.67	2,789.97	1	24	48.51	1,164.23	2.40	58%
rsdecoder-1	10,753	521	17	200,702	11	1	11	55.14	606.59	1	9	53.50	481.52	1.26	76%
rsdecoder-2	10,753	521	24	283,374	60	2	89	78.89	7,020.90	1	73	58.03	4,236.15	1.66	17%
													Average:	**5.07**	**61%**

Table 2: Runtime comparison against gate level debug with groupings

	Gate level groupings		Our technique		
circuit	# iter	time (sec)	# iter	time (sec)	improv
cct4	19	114.40	21	130.63	0.87
cct5	11	8.42	8	5.61	1.50
divider-1	126	7164.61	115	6069.39	1.18
fpu-1	25	1331.62	19	920.76	1.45
rsdecoder-1	11	1228.36	9	481.52	2.55
Total:		9847.41		7607.91	1.29

solutions are found based on grouping clauses at the gate level over all time frames by using additional CNF variables. For a fairer performance comparison we used the Partial MaxSAT solver of [12] instead of the solver [14] originally used by [10]. Experimentally our Partial MaxSAT solver outperforms the solver of [14] by a large margin for our sample circuits. The results are not presented here due to space constraints. A representative sample of the results comparing our run times against gate level groupings is presented in Table 2. Since errors are modeled at the clause level our formulation generally has a larger search space and finds additional solutions not provided by a formulation based on gate level cardinality. However, since a single clause may imply multiple gate level error locations, in all cases except for `cct4`, fewer iterations are required by our proposed technique to find all equivalent error locations.

With the solve time per iteration almost identical between the two approaches, our technique results in an overall performance improvement of 1.29× for the sample of circuits given. Furthermore, since the work in [10] also groups clauses for gates across all time frames, no temporal debug information can be deduced from the solutions. As a result, our formulation not only provides a speedup, but also provides valuable temporal debug information to the user.

7. CONCLUSION

This work introduces a technique for debugging sequential circuits using a Partial MaxSAT formulation which provides both temporal and spatial information about error locations. Temporal information is critical to the user during the debugging process. The proposed framework is a departure from traditional debug techniques as error sources are modeled at the clause and literal levels. Experiments demonstrate that the temporal locations can accurately locate when in an error trace the errors are active. Performance gains of orders of magnitude are observed in some cases, with an improvement of 5× on average.

8. ACKNOWLEDGMENTS

The authors would like to thank Mr. Duncan Smith for offering his technical expertise and help with implementation aspects of this framework.

9. REFERENCES

[1] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip Verification: Methodology and Techniques.* Kluwer Academic Publisher, 2000.

[2] E. Clarke, O. Grumberg, and D. Peled, *Model Checking.* MIT Press, 1999.

[3] V. Paruthi and A. Kuehlmann, "Equivalence checking combining a structural SAT-solver, BDDs, and simulation," in *Int'l Conf. on Comp. Design*, 2000, pp. 459–464.

[4] International Techonology Roadmap for Semiconductors, "ITRS 2006 Update," 2008, http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm.

[5] 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.

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

[7] 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.

[8] 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.

[9] H. Mangassarian, A. Veneris, S. Safarpour, M. Benedetti, and D. Smith, "A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test," in *Int'l Conf. on CAD*, 2007, pp. 240–245.

[10] S. Safarpour, M. H. Liffiton, H. Mangassarian, A. Veneris, and K. A. Sakallah, "Improved design debugging using maximum satisfiability," in *Int'l Conf. on Formal Methods in CAD*, 2007, pp. 13–19.

[11] K. Chang, V. Bertacco, and I. Markov, "Simulation-based bug trace minimization with BMC-based refinement," in *Int'l Conf. on CAD*, 2005, pp. 1045–1051.

[12] J. Marques-Silva and V. M. Manquinho, "Towards more effective unsatisfiability-based maximum satisfiability algorithms," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, May 2008, pp. 225–230.

[13] Max-SAT 2008, "http://www.maxsat.udl.cat," 2008.

[14] M. H. Liffiton and K. A. Sakallah, "On finding all minimally unsatisfiable subformulas," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 173–186.

[15] A. Sülflow, G. Fey, R. Bloem, and R. Drechsler, "Using unsatisfiable cores to debug multiple design errors." in *ACM Great Lakes Symposium on VLSI*, V. Narayanan, Z. Yan, E. Macii, and S. Bhanja, Eds. ACM, 2008, pp. 77–82.

[16] M. Velev, "Exploiting signal unobservability for efficient translation to CNF in formal verification of microprocessors." in *Design, Automation and Test in Europe*, 2004, pp. 266–271.

[17] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. on CAD*, vol. 11, pp. 4–15, 1992.

[18] F. Manyà, "Maxsat, hard and soft constraints," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and Walsh, Eds. IOS Press, 2008, in Press.

[19] Z. Fu and S. Malik, "On solving the partial MAX-SAT problem," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2006, pp. 252–265.

[20] J. Marques-Silva and J. Planes, "Algorithms for maximum satisfiability using unsatisfiable cores," in *Design, Automation and Test in Europe.* ACM Press, March 2008.

[21] OpenCores.org, "http://www.opencores.org," 2008.