Generation of Shorter Sequences for High Resolution Error Diagnosis Using Sequential SAT

Sung-Jui (Song-Ra) Pan and Kwang-Ting Cheng
Dept. of Electrical & Computer Engr., U. of California, Santa Barbara, Santa Barbara, CA 93106
{*srpan, timcheng*}@*ece.ucsb.edu*

John Moondanos and Ziyad Hanna
Intel Corporation
{*john.moondanos, ziyad.hanna*}@*intel.com*

Abstract

Commonly used pattern sources in simulation-based verification include random, guided random, or design verification patterns. Although these patterns may help bring the design to those hard-to-reach states for activating the errors and for propagating them to observation points, they tend to be very long, which complicates the subsequent diagnosis process. As a key step in reducing the overall diagnosis complexity, we propose a method of generating a shorter error-sequence based on a given long error-sequence. We formulate the problem as a satisfiability problem and employ a SAT solver as the underlying engine for this task. By heuristically selecting an intermediate state S_i which is reachable by the given long sequence, the task of finding the transfer sequence from the initial state to the target state can be divided into two easier tasks - finding a transfer sequence from the initial state to S_i and one from S_i to the target state. Our preliminary experimental results on public benchmark circuits show that the proposed method can achieve significant reduction in the length of the error sequences.

1 Introduction

Constrained or guided random patterns are still heavily used to detect errors in a modern design, although recent improvements in formal methods such as bounded model checking (BMC) have made it possible to verify certain properties and generate counter-examples for large industrial designs. The counter-examples are often too complex to be manually inspected by the designers. In [11], Ravi and Somenzi propose to guide the bounded model checker to minimize the sizes of counter-examples. However, in current practice, most of the counter-examples are primarily found by simulation-based verification instead of formal methods. Thus, both manual debugging and automatic design-error diagnosis for a sequential circuit often relies on a set of sequences, called *error sequences*, which reveal erroneous responses at the primary outputs. In general, longer error sequences imply higher complexity for diagnosis. An unnecessarily long error sequence (e.g. visiting a state multiple times, not taking the shortest path from the initial state to the required state for activating and propagating the error, etc.) generally degrades the diagnosis resolution. In contrast, short error sequences minimize the complexity and maximize the resolution of the diagnosis process.

D'Souza and Hsiao in [6] extended the region-based technique proposed in [3] for diagnosing sequential circuits. It attempts to identify the first cycle in which an error appears at either a flip-flop or a primary output to avoid re-simulating the entire sequence. After such a cycle is identified, a shorter sequence can be found for region-based diagnosis. Region-based diagnosis proposed in [3, 6] relaxes the single-error assumption and does not use any specific error models originally proposed in [1]. However, the knowledge of the expected value at each register in each cycle, which is required for these methods, is not typically available in the debugging phase. Usually the expected values are available only for the primary outputs. Therefore, we cannot accurately determine the cycle at which the state of the erroneous circuit first deviates from that of the good circuit.

In [5], the authors propose to eliminate redundant states in an error trace. A new trace can then be found by identifying the shortest path among the remaining unique states in the trace. One main problem with the method is that the new trace may not activate and propagate any errors to the primary outputs. This is because the process of identifying the redundant states in the original error trace and finding the shortest path for the remaining unique states is not based on a "golden model" of the design. Instead, it is based on the design model which contains bugs. Therefore, the new trace may not be a valid error trace. These simulation based techniques have been greatly enhanced in [4].

In this paper, we address the problem of generating a shorter error sequence from a given longer one. Error sequences are often identified by examining the simulation results of the erroneous designs on either functional or random patterns. We focus on formulating this test-generation problem as a satisfiability problem and employ a modern SAT solver [8], for this optimization. By selecting an intermediate

0-7803-9451-8/06/$20.00 ©2006 IEEE.

state S_i, which is reachable by the given long sequence, the task of finding the transfer sequence from the initial state and the state which reveals the error at the outputs can be divided into two sub-tasks - finding a transfer sequence from the initial state to S_i and one from S_i to the target state. This strategy can be recursively applied until every subproblem becomes solvable by the SAT solver. However, if the transfer sequences are derived this way and then cascaded together to form the final transfer sequence, we may get a "false" sequence that cannot activate the errors or propagate them to the outputs. This could happen if the circuit model used by the SAT solver is erroneous (because it is derived from the circuit under diagnosis). Therefore, each error sequence generated must be simulated to verify the matching. If the generated sequence does not result in the erroneous responses at the outputs, the procedure needs to be called again to generate a new transfer sequence. We propose an algorithm to select states with higher probability of exciting and/or propagating errors as the intermediate states for finding the transfer sequence. The experimental results on ISCAS89 and ITC99 benchmark circuits show that the proposed method can achieve a reduction rate as high as 99.94% in test length.

The rest of the paper is organized as follows. Section 2 gives a detailed statement of the problem. Sections 3 and 4 present the proposed method for generating a shorter test sequence. Section 5 shows the experimental results, and concluding remarks are given in Section 6.

2 Problem Formulation

There often exist multiple transfer sequences between any pair of states; some are significantly shorter than others. In addition, some states visited by an error sequence are not relevant for activating and for propagating errors to the outputs. Figure 1 shows the state diagram of a sequential circuit under diagnosis. Note that our method needs neither the construction nor the use of the state diagram. Our method takes the boolean netlist of the sequential circuit as the input and works primarily at the structural level. The example is mainly used for illustration of the concept. Suppose the transition function from state F to state B is erroneous and the states traversed by the error sequence is A–C–C–D–E–F–B. Our objective is to find a shorter sequence that can transfer the circuit from A to B and also propagate errors to the primary outputs. Because transitions A–C and C–D are not relevant for activating and propagating the error, states C and D need not be visited. A shorter sequence such as the one traversing states A–E–F–B would be a better sequence for diagnosis.

We can classify the states traversed by the error sequence into *irrelevant* states (such as states C and D) and *relevant* states (such as states E and F). The objective is to find a new error sequence that will visit all relevant states and avoid irrelevant states. However, because the knowledge of correct values at registers in each clock cycle is not available, we cannot determine precisely whether a state visited by the error sequence is relevant or not. Therefore, we need to simulate

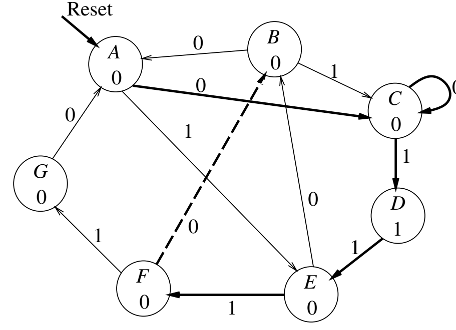

Figure 1: An example of the error sequence, A–C–C–D–E–F–B, where the state transition, F–B, is erroneous.

the generated sequence to verify whether it indeed activates and propagates the error to the primary outputs.

3 Overview of the Method

For a given error sequence, the initial state S_I and the final state S_F of a circuit C could be derived by simulation. Since our objective is to find a shorter error sequence from S_I to S_F rather than to search for all possible shorter sequences, a modern SAT solver becomes a more attractive method than the BDD-based techniques. We can apply a sequential SAT solver to find the shortest transfer sequence $T^{'}(S_I, S_F)$ between S_I and S_F. However, a sequential SAT solver [8, 9] usually cannot find the shortest solution. In order to find the shortest sequence, one strategy is to follow the principle of bounded model checking. We attempt to find a solution within i time frames where i is initially set to 1. If no solution exists, i is then increased by 1 for another run of the SAT solving. This iterative process continues until a solution is found. Without the exact information of relevant states, the sequence derived this way may be a "false" sequence that cannot activate and propagate the error to the outputs. By selecting intermediate states derived from the simulation results of an erroneous circuit, the SAT solving will be guided to generate a sequence that will visit relevant states. Also, the task of finding a shorter sequence from S_I to S_F can be divided into two sub-tasks that are more amenable to the SAT solving.

The proposed algorithm is shown in Figure 2. In Line 4, we select intermediate states, $S_{targets}$ derived from the simulation result of an erroneous circuit C on a given long error sequence V, as candidates to divide V into two sub-sequences, $T(S_I, S_i)$ and $T(S_i, S_F)$ where $S_i \in S_{targets}$. The detail of the state selection will be explained in Section 4. Then, we use a SAT solver to find the shortest transfer sequence, $T^{'}(S_I, S_i)$, to replace $T(S_I, S_i)$ (Line 5). Note that in finding the shortest sequence from S_I to every $S_i \in S_{targets}$, the learned conflict clauses generated in solving one target can be accumulated and reused in finding the sequences for other targets. However, $T^{'}(S_I, S_i)$ alone may not be able to activate and propagate errors to primary outputs, since errors in the given sequence may only be activated and propagated to primary outputs by $T(S_i, S_F)$. Thus, in Line 9–10, for each $S_i \in S_{targets}$, a new test sequence $V^{'}$ is generated by cascad-

ing $T^{'}(S_I, S_i)$ and $T(S_i, S_F)$ followed by re-simulation. If no error appears at the observation points, the new sequence $V^{'}$ is discarded and the shortest transfer sequence from S_I to the next $S_i \in S_{targets}$ is used to form the next test sequence for simulation and verification of its validity. On the other hand, if $V^{'}$ successfully produces errors at primary outputs, a valid new sequence has been found. We then continue to shorten the second part of the sequence, $T(S_i, S_F)$. This can be done by setting S_i as the initial state S_I (Line 13) before calling the process again (Line 3–12). This process continues (by iteratively setting new S_I's) until no reduction can be made.

```
0 # C: Circuit, V: Error Sequence
1 # S_I: Initial State, S_F: Final State
2 push S_F into S_targets.
3 while S_targets is not empty
4   Select intermediates states into S_targets.
5   Apply the sequential SAT solver [8]
6   Sort S_targets in a reverse chronological order, S'_targets
7   for each target S_i ∈ S'_targets
8       if solution of S_i exists
9           V' = Cascade_Sequences(T'(S_I, S_i), T(S_i, S_F)).
10          ErrorObserved = Simulate(V').
11          if ErrorObserved == TRUE
12              break
13  set S_I = S_i.
```

Figure 2: Algorithm for error sequence generation/minimization.

In order to achieve a higher reduction rate in test length, we process the state $S_i \in S_{targets}$ in a reverse chronological order. That is, the first S_i selected for transfer sequence generation is the farthest state S_i in $S_{targets}$ from S_I. If the resulting sequence cannot reveal the errors at observation points, then the next farthest target is selected as the target for transfer sequence generation.

4 Intermediate State Selection

Selection of intermediate states in Figure 2 affects the reduction rate of the test length. Consider the example in Figure 1. If states B and D are selected as the intermediate states to optimize the transfer sequence from state A to state B, then A–E–B and A–C–D–E–B, will be found as the shortest state sequences respectively. However, neither sequence can activate error (because the assumption was that the error only corrupted the transition from F to B). Therefore, both sequences will be rejected and no test length reduction will be achieved. On the other hand, if we select state F as the intermediate state, the valid sequence A–E–F–B will be found which could activate and propagate the error to the outputs. In the following, we propose a heuristic for identifying states with higher probabilities of successful activation and/or propagation of the errors to either primary outputs or registers in the next clock cycle. These states are preferred intermediate states for transfer sequence generation.

Starting from the erroneous outputs E, we trace backwards to find the fan-in registers of E. By analyzing the states visited by the error sequence V, we develop a metric to identify which fan-in registers, denoted as F_1, are irrelevant for activating or propagating the error to E, and which registers, denoted as F_2, are essential for error activation and propagation. For f_i to be in F_2, one of the following two cases must apply : (1) an erroneous value already exists in f_i, or (2) the value in f_i is error-free but is essential for error activation or propagation. Starting from the registers in F_2, we continue to trace backwards for one more cycle to identify their fan-in registers, which are essential for error activation or propagation. The iterative process continues until every state reached by the error sequence has been analyzed. Throughout this process, we can calculate the cycles in which a register is classified as an essential one (i.e. belonging to F_2). Suppose that, in M cycles, a register f_i is classified as F_2. Among these M cycles, assume f_i is at 0(1) for $m_1(m_2)$ cycles. If $m_1 > m_2$, we would guess that f_i being 0 is more likely to excite and/or propagate the errors than f_i being 1. Therefore, we would prefer to select an intermediate state with f_i being 0 rather than with f_i being 1. Likewise, if $m_2 > m_1$, we would prefer to select an intermediate state with f_i being 1.

As shown in Figure 3, for a given error sequence, the outputs with erroneous responses are set as the initial *error candidates*, E (Line 1). In this algorithm, E contains either outputs or registers that might have erroneous values. In Line 3, we trace backwards to find the fan-in registers F of signals in E. In order to determine whether $f_i \in F$ belongs to either category F_1 or category F_2 in each cycle, we complement the value in f_i (and keep value in all other inputs and registers intact - Line 5) and check whether the change can be propagated to E (Line 6). If any value of $e_i \in E$ is changed, we classify f_i as a category F_2 register. Then, we push f_i into E_{next} as a source for back-tracing in the next iteration (Line 12). In Line 8–11, we increment α_i^1 by 1 if the original value of f_i is 1; otherwise, α_i^0 is incremented by 1. α_i^k is the number of cycles that $f_i \in F$ being at value k has been classified as a category F_2 register. For a register whose α_i^0 is larger than α_i^1, heuristically, f_i being 0 has a higher probability to excite and/or propagate the error than that being 1. The whole process (Line 3–11) repeats until all states visited by the error sequence have been examined. After the calculation of α_i^k for each register is completed, the *relevance* β_j of a state S_j for error activation/propagation can be approximated by summing up the α_i^k of register $f_i \in S_j$ based on the value of f_i in S_j (Lines 15 – 20). Then, we sort the states based on their β_j from the highest to the lowest (Line 21). In Line 22, the states $S^{'}$ are selected from the sorted states in a chronological order. The selected states are returned for finding shorter transfer sequences (Line 23).

5 Experimental Results

We implemented our algorithm in the C++ language and tested it on some public benchmark circuits including IS-

```
# every α_i^k and β_i is initialized to 0.
# S: the simulation result of a given error sequence
1 Identify erroneous outputs as error candidates E.
2 repeat
3   Trace back from E to get their fan-in registers F.
4   for each register f_i ∈ F.
5       Invert the value of f_i and keep the others.
6       Simulate the circuit.
7       if any value of e_i ∈ E is changed.
8           if value(f_i) == 1
9               Increase α_i^1 by 1.
10          else if value(f_i) == 0
11              Increase α_i^0 by 1.
12          Push f_i into E_next.
13  set E = E_next.
14 until Empty(E) == TRUE.
15 for each state S_j ∈ S .
16  for each register f_i ∈ S_j
17      if value(f_i) == 1
18          β_j += α_i^1.
19      else
20          β_j += α_i^0.
21 Sort S based on β_i (from the highest to the lowest).
22 Select states S' from the sorted states in chronological order.
23 return S'.
```

Figure 3: Selection of intermediate states with higher probabilities of activation and/or propagation of the errors.

Ckt. Name	Num. of Registers
s9234	211
s38417	1636
b14	245
b20	490
b21	490
b22	703
or1k	1860

Table 1: Number of registers for each circuit

CAS89, ITC99, and the OpenRisc 1000 (a microprocessor) from the OpenCores organization [10]. The number of registers of each tested circuit is shown in Table 1 where or1k denotes the OpenRisc 1000. In this paper, we do not speculate about what kind of errors will be easier to detect by simulation-based or formal verification methods. Our objective is to show the effectiveness of our method to reduce the error sequence found by simulation. We generated several erroneous circuits from each benchmark circuit listed in Table 1 by randomly replacing several gates with different types of gates. Then, we used the Candence TestBuilder [13] as our random pattern generator to generate input sequences and used Mentor Graphics ModelSim to simulate the erroneous circuits. After a long error sequence had been identified, we applied the proposed method to generate a new shorter error sequence.

As discussed in Section 4, selection of intermediate states is critical to the test generation quality. As shown in Table 2, we compare our method with the method $Select(k)$ for which the state of every k cycles is selected as an intermediate state. For example, method $Select(20)$ means the reached states in every 20 cycles, starting from the initial state, will be included as the intermediate states. Len denotes the length of the error sequence found by simulation. The numbers before and after "/" in Columns 3, 4, and 5 denote the length of derived test sequence and the reduction rate in percentages with respect to the original test length. For the method $Select(k)$, if each time we fail to generate a transfer sequence for a target intermediate state S_i, the next target intermediate state would be the state reached k cycle earlier, S_{i-k}, under the original error sequence. Even if the transfer sequence for the new target is successfully found, the final error sequence will include $T(S_{i-k}, S_F)$ which is k patterns longer than $T(S_i, S_F)$. Therefore, the length of the final sequence would be increased by k. As shown in Column 4, since the method could not find a short error sequence from the initial state to the final state at which the erroneous responses are present at primary outputs for every single circuit in our experiment, the lengths of sequences for the method $Select(100)$ are always at least 100. In general, the reduction rate of $Select(k)$ can be improved by reducing k. Experimentally, changing k from 100 to 20 increases the reduction rate from 71.32% to 76.68%. In comparison to $Select(k)$, our method achieves average higher average reduction rate (78.49%) than both $Select(100)$ (71.32%) and $Select(20)$ (76.68%). For most cases, the improvement rate is much better than the average. Note that, for some circuits, such as or1k_[1,2], the results of $Select(20)$ are worse than those of $Select(100)$. The reason is that there are more targets for SAT solving for Method $Selct(20)$ than Method $Select(100)$. The sequential SAT solver simply cannot find enough solutions for selected targets to reduce test length within the time limit when time limit for each iteration is set to 1800 seconds.

The reduction rates for all tested circuits are shown in Table 3, where Len, Len', Rate, Ite., and Time denote the length of the original error sequence, the length of generated shorter sequence, the reduction rate in test length, the number of iterations for transfer sequence generation, and the average runtime for SAT solving in each iteration, respectively. As shown in Column 4 of Table 3, the proposed method can achieve more than 80% reduction rate in 20 out of the 36 cases, and the highest reduction rate is 99.94% on b14_9. The average reduction rate is 59.4%. There are 6 out of the 36 cases whose reduction rates are below 10%. One explanation for the low reduction rate is that there may be counters in the circuits, and specific values in the counters are required for activating and propagating the errors to the outputs. For counters, it is unlikely that any shorter sequences exist to reach a specific state.

6 Concluding Remarks

Although long test sequences may help bring the design to those hard-to-reach states for activating the errors and for propagating them to observation points, these sequences com-

Ckt.	Len	Ours	$Select(100)$	$Select(20)$
b14_1	369	21/94.31%	115/68.83%	13/96.48%
b14_2	693	77/88.89%	115/83.41%	75/89.18%
b14_3	874	24/97.25%	114/86.96%	32/96.34%
b14_4	2347	29/98.76%	111/95.27%	23/99.02%
b14_5	22796	110/99.52%	112/99.51%	46/99.80%
b14_6	30960	18/99.94%	114/99.63%	46/99.85%
b14_7	4634	24/99.48%	114/97.54%	32/99.31%
b20_1	645	505/21.71%	609/5.58%	127/80.31%
b20_2	1013	163/83.91%	215/78.78%	105/89.63%
s38417_1	1041	103/90.11%	193/81.46%	109/89.53%
s38417_2	2759	457/83.44%	244/91.56%	188/93.19%
s38417_3	1629	53/96.75%	132/91.90%	55/96.62%
s38417_4	1541	83/94.61%	167/89.16%	99/93.58%
s38417_5	2223	99/95.55%	206/90.73%	132/94.06%
s38417_6	1082	86/92.05%	165/84.75%	89/91.77%
or1k_1	2982	246/91.75%	252/91.55%	2865/3.92%
or1k_2	1230	21/98.29%	125/89.84%	360/70.73%
or1k_3	1183	1130/4.48%	1183/0.00%	1183/0.00%
or1k_4	4262	3070/27.97%	4262/0.00%	2661/37.56%
or1k_5	4931	4385/11.07%	4931/0.00%	4400/10.77%
Average	-	-/78.49%	-/71.32%	-/76.68%

Table 2: Comparison of three intermediate state selection methods.

plicate the subsequent diagnosis process. To reduce the complexity of diagnosis, we have proposed a method to generate a shorter error sequence, based on a known error sequence identified by simulation. We have further proposed to simplify the task of generating such a sequence, which brings the circuit from the initial state to the state that reveals the error at the primary outputs. This is accomplished by identifying an intermediate state S_i so that the final sequence is formed by two sub-sequences - one from the initial state to S_i and the other from S_i to the final state. The task of generating each subsequence is substantially simpler than the longer original task. We have developed a heuristic to select intermediate states which are more likely to lead to successful generation of error sequences. A sequential SAT solver was used as the underlying engine for finding these sub-sequences. The experimental results on ISCAS89 and ITC99 benchmark circuits show that the proposed method can achieve a very significant reduction rate in test length.

Ckt.	Len	Len'	Rate(%)	Ite.	Time
s9234_1	3548	24	99.32	2	287
s9234_2	7114	44	99.38	2	108
s9234_3	3955	18	99.54	2	687
s9234_4	9386	15	99.84	2	5
s38417_1	1041	103	90.10	4	943
s38417_2	2759	457	83.44	3	2530
s38417_3	1629	53	96.75	3	302
s38417_4	1541	83	94.61	3	712
s38417_5	2223	99	95.55	4	942
s38417_6	1082	86	92.05	3	1433
b14_1	369	21	94.31	2	28
b14_2	693	77	88.89	2	60
b14_3	874	24	97.25	3	59
b14_4	2347	29	98.76	3	77
b14_5	22796	110	99.52	2	63
b14_6	30960	18	99.94	4	56
b14_7	4634	24	99.48	4	80
b20_2	1013	163	83.91	4	7520
or1k_1	2982	246	91.75	7	5095
or1k_2	1230	21	98.29	5	1045
b20_1	645	505	22.71	5	2808
b20_3	2130	1902	10.70	5	3518
b20_4	2429	2273	6.42	3	2710
b20_5	939	719	23.43	6	3601
b21_1	433	267	38.34	3	2664
b21_2	2160	2096	3.96	2	3173
b21_3	870	704	19.08	3	2663
b21_4	1231	941	23.56	3	2893
b21_5	955	767	19.69	2	2929
b21_6	4252	3998	5.97	2	2936
b22_1	2138	2080	2.71	3	2792
b22_2	6552	6488	0.98	4	3079
b22_3	1527	1295	15.19	4	3243
or1k_3	1183	1130	4.48	2	3171
or1k_4	4262	3070	27.97	4	5263
or1k_5	4931	4385	11.08	2	4058
Average	-	-	59.4	-	-

Table 3: Experimental results on ISCAS89, ITC99, and or1k benchmark circuits.

References

[1] Magdy S. Abadir, Jack Ferguson, and Thomas E. Kirkland, "Logic Design Verification via Test Generation," in *IEEE Trans. on Computer-Aided Deisgn*, Jan. 1988, Vol. 7, No. 1, pp. 138–148.

[2] Moayad Fahim Ali, Andreas Veneris, Sean Safarpour, Magdy Abadir, Rolf Drechsler, and Alexander Smith, "Debugging Sequential Circuits Using Boolean Satisfiability," in *Proc. International Conference on Computer Aided Design*, 2004, pp. 204–209.

[3] Vamsi Boppana, Rajarshi Mukherjee, Jawahar Jain, Masahiro Fujita, and Pradeep Bollineni, "Multiple Error Diagnosis Based On Xlists," in *Proc. Design Automation Conference*, 1999, pp. 660–665.

[4] K. Chang, V. Bertacco, and I. Markov, "Simulation based Bug Trace Minimization with BMC-based Refinement," in *Proc. International Conference on Computer Aided Design*, 2005.

[5] Yirng-An Chen and Fang-Sung Chen, "Algorithms for Compacting Error Traces," in *Proc. Asia and South Pacific Design Automation Conference*, 2003, pp. 21–24.

[6] Anand L. D'Souza and Michael S. Hsiao, "Error Diagnosis of Sequential Circuits Using Region-Based Model," in *Proc. Intenal Conference on VLSI Design*, 2001, pp. 103–108.

[7] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in *Proc. Design, Automation and Test Conference in Europe*, 2002, pp. 142–149.

[8] Feng Lu, M. K. Iyer, G. Parthasarathy, Li.-C. Wang, K.-T. Cheng, and K.C. Chen, "An Efficient Sequential SAT Solver With Improved Search Strategies," in *Proc. Design, Automation, and Test Conference in Europe*, 2005, pp. 1102–1107.

[9] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proc. Design Automation Conference*, 2001, pp. 530–535.

[10] OpenRisc 1000 series, *http://www.opencores.org*

[11] K. Ravi and F. Somenzi, "Minimal assignments for bounded model check," in *Tools and Algorithms for the Construction of Analysis of Systems*, 2004, pp. 31–45.

[12] Alexander Smith, Andreas Veneris, and Anastasios Viglas, "Design Diagnosis Using Boolean Satisfiability," in *Asia and South Pacific Design Automation Conference*, 2004, pp. 218–223.

[13] Cadence TestBuilder, *http://www.testbuilder.net*