Adaptive Branch and Bound using SAT to Estimate False Crosstalk

Murthy Palla, Jens Bargfrede, Klaus Koch
Infineon Technologies AG
Munich, Germany

Walter Anheier, Rolf Drechsler
University of Bremen
Bremen, Germany

Abstract

Accurate crosstalk analysis has become a key issue in Static Timing Analysis of modern deep–submicron digital circuits. The inherent logic and timing properties of the circuit are often neglected in the crosstalk estimation process resulting in an overly pessimistic analysis. The problem of considering the logic correlations of the circuit to eliminate false crosstalk has been widely studied, but still lacks a very efficient solution also due to its NP–hard nature. In this paper, we propose a SAT solver based approach to efficiently solve the false crosstalk problem. We also propose a novel and very powerful bounding technique called Adaptive Bounding as well as an aggressor ordering technique called Simple Aggressor Ordering for the branch and bound method running on top of the SAT solver. These techniques are proven to drastically increase the speed of false crosstalk analysis to an extent that nets with hundreds of aggressors can be handled. The results of this approach on the ISCAS89 benchmark circuits are provided.

1. Introduction

Conventional crosstalk models often consider all the potential aggressors of any given victim net or path as valid. This unrealistic worst–case crosstalk model leads to an overly conservative and pessimistic estimation of the circuit crosstalk. The amount of overestimation of crosstalk (noise, delay or slew change) is called *false noise*.

The source of overall pessimism in crosstalk analysis stems from the independent accounting of each aggressor and victim net coupling. Timing and logic correlations which could render certain switching scenarios impossible are simply ignored. Industrial strength crosstalk analysis tools try to avoid the most obvious blunders by considering the simple logic correlations arising from single inverter and buffer cells. Yet this is far from being sufficient as is apparent from [1, 2, 3, 4, 5, 6, 7]. What needs to be done is to find the aggressor set that has a maximum crosstalk impact on the victim consisting only of those aggressors that can logically and temporally induce crosstalk simultaneously.

To demonstrate the concept of false noise, we consider the circuit in Figure 1 consisting of a victim V and three aggressors a_1, a_2 and a_3. In case of V switching from logic zero to one, the aggressors will increase the victim's delay, if they switch from logic one to zero in the vicinity of V's switching time. Conventional crosstalk analysis algorithms consider that the worst–case would occur with all the three aggressors switching in the same direction at the same time without considering whether this switching scenario is possible at all. However, a brief check of the circuit rules out this scenario as not all aggressors can simultaneously assume logic 1 or logic 0, which implies that they cannot simultaneously switch in the same direction. So, a constraint should be laid pruning such unrealistic scenarios for crosstalk analysis.

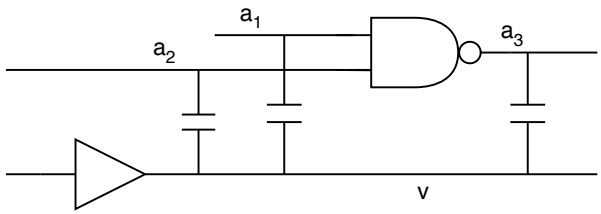

Figure 1. False noise example with impossible aggressor combination $\mathbf{a_1}$, $\mathbf{a_2}$ and $\mathbf{a_3}$ of victim $\mathbf{V}$.

False noise not only distorts the crosstalk analysis of the affected net, but also that of subsequent circuit elements. The impact of this error propagation depends on the crosstalk effect considered. While crosstalk noise could be attenuated by subsequent cells, crosstalk delay never vanishes but sums up in the overall path delay and slack [1]. False calculated slew change due to crosstalk even distorts the analysis of subsequent cells in the path and, finally, the timing check calculation.

Recently, new algorithms have been proposed to take into account more complicated logic and timing correlations for finding false noise and selecting the Maximum Realizable Aggressor Set (MRAS) [1, 2, 3]. In [3], Simple Logic Implications (SLIs) were used to store the relations between signal pairs. As SLIs cannot relate more than two signals at a time, false noise analysis using logic constraints was proposed in [1, 2]. The logic constraints of the circuit other than those of transistors or gates are deduced

0-7695-3117-2/08 $25.00 © 2008 IEEE
DOI 10.1109/ISQED.2008.14

IEEE computer society

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.

by multiple application of the resolution principle. As the functions specifying all possible logic constraints are often too large and cumbersome for practical purposes, this approach requires to neglect some logic constraints to make the problem tractable. As we see in the subsequent sections, we circumvent this problem by using a SAT solver, which uses *substitution* instead of *resolution* to tackle this problem. This ensures that no logic correlations of the circuit are left unconsidered. Moreover, dramatic performance improvements with respect to our implementation of the logic correlations based approach proposed in [1] are achieved.

In [5], a heuristic approach to order the aggressors fed to the branch and bound algorithm proposed in [1] was presented. This was shown to considerably improve the speed of the analysis. However, calculation of tendencies in the method proposed in [5] requires the deduction of logic constraints correlating aggressors using the resolution principle. To reduce this overhead, we propose an aggressor ordering technique based on the aggressor strengths. We call this technique Simple Aggressor Ordering (SAO). This is discussed in detail in section 3.2.

In [6, 7], an attempt to integrate logic and timing for crosstalk analysis has been made. [6] is based on slicing the timing windows and unrolling the circuit. [7] uses back propagation to check if a particular crosstalk scenario is valid. They use SAT solvers to solve the underlying constraint satisfiability problem, but do not use efficient optimization algorithms on the top to minimize the number of crosstalk scenarios that need to be verified. As a result, both these approaches suffer from degraded performance for large problem complexities.

In this paper, we propose a solution based on a SAT solver. We also propose a novel and very powerful bounding technique Adaptive Bounding (AB) for the branch and bound method which runs on the top of the SAT solver as an optimizer.

As mentioned earlier, false crosstalk could arise either due to negligence of logic correlations and/or timing correlations. In this paper, we consider only the false crosstalk arising due to the negligence of logic constraints of the circuit, assuming the zero–delay timing model, i.e., zero delay on gates and interconnects. Logic constraints generated under this assumption are conservative only for glitch free circuits, obtained, for instance, through special transistor sizing methods [1]. However, the core of the false noise analysis approach we propose is independent of the method used to generate constraints and, hence, is applicable for an analysis that integrates timing and logic constraints.

We have used a linear crosstalk model for our analysis. The strength or impact of an aggressor is quantified by the coupling capacitance between aggressor and victim. In a more realistic scenario, the strength of each aggressor could be estimated more accurately, for instance, by considering aggressor drive strength. However, the methods proposed are independent of the accuracy in modeling the crosstalk impact of each aggressor and could benefit from any improved accuracy in this domain.

The rest of this paper is organized as follows: section 2 describes various related background topics. Section 3 describes in detail the optimization of false crosstalk using the branch and bound method with the proposed techniques. Section 4 explains our implementation and the results of our false noise analysis on the ISCAS89 benchmark circuits. Finally, section 5 presents our concluding remarks.

2. Background

2.1. Logic constraints

Logic constraints are logically impossible combinations of signals. They are used to determine if a particular switching scenario of a circuit is logically valid. Mathematically, a logic constraint is a boolean clause with one or more literals, either inverted or uninverted. Contrary to [1, 2], we write logic constraints as disjunctions of literals to be able to write the logic function of the circuit in Conjunctive Normal Form (CNF). This means that a satisfiable assignment to the literals of any logic constraint in our model would make it logic 1, as opposed to the definition of a logic constraint in [1, 2].

Logic constraints of various combinational gates used in the circuit are generated a priori from their function definitions. At the first step of the false noise analysis, these generic constraints are mapped onto their corresponding instances in the design. For a more detailed discussion on the generation of basic logic constraints, the reader is referred to the papers [1, 2, 5].

2.2. Coupled fan–in cone

For the analysis of a victim net, it is not necessary to consider the logic constraints of the entire circuit. It is sufficient to consider that part of the circuit that can influence the victim and the aggressor signals. This part of the circuit is obtained by finding the fan–in cones of the victim and its aggressors and merging them [6]. This selection is called the coupled fan–in cone of the victim.

Coupled fan–in cones are often very large for victim nets near the primary outputs or the inputs of sequential cells. This could be avoided by fixing the maximum depth of the fan–in cones. We have observed that fixing the maximum cone depth to 15 instances has a significant impact on speed with only a little loss of accuracy. It should be noted that this solution is acceptable because it is still conservative.

For all the analyses we ran for the results of this paper, we did not restrict the maximum fan–in cone depth. Hence, all the figures we provide are highly accurate. Speed improvements are to be expected if the depth is reduced.

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.

2.3. Validation of crosstalk scenarios

For a crosstalk configuration to be valid, all the aggressors of that particular configuration should be able to switch simultaneously in the direction required to induce the crosstalk effect under consideration. As we use a zero–delay model, it is sufficient to check if the required pre–transition and post–transition values for the victim and its aggressors are satisfiable. In other words, we find the required pre–transition and post–transition values on the aggressor and the victim nets and verify that the assignments do not lead to any violations. If no violations occur, the configuration is said to be valid, and invalid, otherwise. Verifying the satisfiability of logic constraints is discussed in the next section.

2.4. Constraint satisfiability and SAT solvers

Satisfiability of the logic constraints has to be ensured to validate any crosstalk scenario as logically feasible. As this is a very critical step in terms of speed, we chose to take advantage of all the latest developments in the SAT solver domain by using a publicly available SAT solver. Modern SAT solvers like zChaff and miniSat have demonstrated their capabilities by winning many SAT competitions [8]. In our implementation we use miniSat, a conflict–driven SAT solver, for the analysis. However, the methodology and techniques provided here are independent of the solver and any conflict–driven SAT solver could easily be plugged–in.

Once the coupled fan–in cone of a given victim net is found, the logic constraints of its instances are fed to the SAT solver, which can then check the logical validity of a given aggressor configuration as explained in section 2.3. Each time the SAT solver is called, it updates its clause database with new clauses that are learnt during its search process. This improves its performance for subsequent queries on the same problem instance.

3 Finding maximal aggressor set

The ultimate goal of false noise analysis is to find a maximal set of valid aggressors that can simultaneously induce crosstalk on the given victim. In other words, the best of all the 2^n possible combinations of aggressors, where n is the number of potential aggressors, has to be found in such a way that the obtained solution is logically valid. This can be achieved by using the branch and bound algorithm as proposed in [1].

3.1 Branch and bound algorithm review

Branch and bound is an optimization method, mostly used for non–convex NP–hard problems. It explores the solution search space by pruning off non–promising or futile regions as it proceeds based on a bounding criterion and satisfiability. In the case of false noise analysis, the branch and bound explores a binary search tree of aggressors where each node of the tree indicates a specific selection of the aggressors already traversed. The validity of any branching, or, in other words, the corresponding crosstalk configuration, is determined using the SAT solver. The currently best solution is kept track of during the entire process.

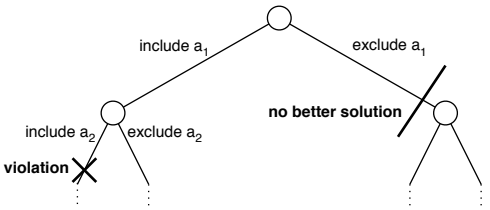

Figure 2. A sample branch and bound decision tree. The idea of search space pruning based on violations and a bounding criterion are depicted.

Figure 2 depicts the binary tree explored by the branch and bound algorithm. The tree is traversed recursively in a depth–first fashion. During the traversal, at each node, all further possible combinations (with the decisions taken at the higher levels fixed) which include and exclude the current node are explored by recursive calls to its sub–nodes. However, this search is implicit in some cases where it could be proven that further exploration is not necessary. In other words, before any further recursive calls to the branch and bound method, it is verified using the bounding criterion that a further exploration of the sub–tree is useful. The sub–tree of any node is explored further only if it can potentially yield a better solution than the current best.

Pruning can also be done based on the validity of configurations. If at any stage, a decision either to include or to exclude an aggressor proves to be invalid, the sub–tree corresponding to this decision branch is pruned.

Our branch and bound algorithm is based on the following properties of which the first two are inspired from [1]. Let $W()$ be the weight function which gives the overall crosstalk induced by a set of aggressors.

Property 1: Let A be the set of all potential aggressors. If $A_r \subset A$ is any set of realizable aggressors and A_R is the maximum realizable aggressor set (MRAS), then $W(A_r) \leq W(A_R)$.

Property 2: Let A be the set of all potential aggressors, and A_1 and A_2 be two disjoint subsets of A, such that $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$. If $A_{r_1} \subset A_1$, and $A_r \subset A$ are realizable aggressor sets such that $W(A_r) > W(A_{r_1} \cup A_2)$, then the maximum realizable aggressor set (MRAS) A_R is not a subset of $A_{r_1} \cup A_2$, i.e., $A_R \not\subset A_{r_1} \cup A_2$.

Property 3: Let A be the set of all potential aggressors and $A_r \subset A$ be a set of realizable aggressors. Assume that $A_c \subset A$ is a set of conflicting aggressors of which aggressor a has the least strength. If $A_s \subset A$ is any selection of ag-

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.

gressors such that $A_c \subset A_s$ and $W(A_r) > W(A_s - \{a\})$, then the maximum realizable aggressor set (MRAS) A_R is not a subset of A_s, i.e., $A_R \not\subset A_s$.

Procedure 1 ADAPTIVE_BNB: Pseudocode for the Adaptive Branch and Bound Algorithm

Input: An aggressor vector ordered using SAO, $A_v = \{a_1, a_2, \cdots, a_n\}$; Index of the aggressor to be processed, i; Current aggressor selection, A_s; Current MRAS, A_r; Weight function $W()$ to find the overall influence of a given aggressor set

Output: Current MRAS, A_r

```
1: Set unprocessed aggressor set A_u = {a_i, a_{i+1}, ..., a_n}
2: if A_u = ∅ return A_r
3: Set A_s' = A_s ∪ {a_i}; Set A_u' = A_u − {a_i}
4: if aggressor configuration A_s' is valid then
5:       W_UB = ADAPTIVE_UPPER_BOUND(A_s', A_u')
6:       if W(A_s') > W(A_r) then A_r = A_s'
7:       if W_UB > W(A_r) then
8:             A_r = ADAPTIVE_BNB(A_v, i+1, A_s', A_r)
9: else
10:      Get conflicting aggressor set, A_c from SAT solver
11:      Add A_c to conflict database
12: W_UB = ADAPTIVE_UPPER_BOUND(A_s, A_u')
13: if W_UB > W(A_r) then
14:      A_r = ADAPTIVE_BNB(A_v, i+1, A_s, A_r)
15: return A_r
```

3.2 Simple Aggressor Ordering (SAO)

The order in which the aggressors are traversed plays a vital role in the amount of solution search space that can be pruned. A tendency based ordering of aggressors is proposed in [5]. This approach is based on the switching tendencies of aggressors which can be calculated from the logic constraints correlating them. As this would be a considerable overhead with our approach, we propose and use a different ordering technique called Simple Aggressor Ordering, which orders the aggressors in the descending order of their strengths.

Our experiments have shown that this kind of ordering is essential not just for speed improvement, but in many cases also for feasibility. If the decisions on stronger aggressors are done in the beginning, many decisions on less significant aggressors can be avoided if their addition does not yield any improvement. The latter can be verified by computation of the bounding criterion.

3.3 Learning in Branch and Bound

In this paper, we introduce the idea of learning in Branch and Bound, which could aid the Adaptive Bounding strategy discussed in section 3.4. As mentioned in section 2.4, a conflict–driven SAT solver is used in our approach. Conflict–driven SAT solvers are based on learning from conflicts occurring during their search for a satisfiable solution. The idea of Conflict–Based Learning is very effective because the search process can be driven in such a way that the conflicts that once occurred do not occur again.

During the constraint satisfiability check, if an aggressor configuration is found to be invalid, the set of conflicting aggressors can be deduced from the conflict information obtained from the SAT solver. This information is stored in a database in the form of a list of conflicting aggressor sets for any given aggressor. As we see in the next section on Adaptive Bounding, this information can be exploited to calculate very tight upper bounds in the branch and bound process.

3.4 Adaptive Bounding

The power of branch and bound lies in its ability to prune futile search spaces. This can be improved by using a bounding strategy that calculates very tight upper bounds. Adaptive Bounding is such a technique that helps branch and bound to efficiently use knowledge that is already acquired during the process by adaptively taking care that a conflict that already occurred is considered in the estimation of upper bounds. This helps the Branch and Bound to efficiently foresee the necessity to explore an unexplored sub–tree.

Procedure 2 ADAPTIVE_UPPER_BOUND: Pseudocode to find upper bound using the Adaptive Bounding technique

Input: Aggressor set A_s (current aggressor selection); Aggressor set A_u (unprocessed aggressors); Weight function $W()$ to find the overall influence of a given aggressor set

Output: Adaptive Upper Bound, W_{UB}

```
1: Let A_UB = A_s
2: for each aggressor a of A_u do
3:       Let S_Ac be the set of all known conflicts of a
4:       Set conflictFlag = false
5:       for each conflict aggressor set A_c of S_Ac do
6:             if A_c − {a} ⊂ A_UB then
7:                   Set conflictFlag = true;
8:                   Break
9:       if conflictFlag = false then add a to A_UB
10: return W(A_UB)
```

As explained in section 3.1, during its process, the branch and bound algorithm needs to calculate an upper bound on the crosstalk due to a set of aggressors, say A_s, obtained from the current selection and a set of unprocessed aggressors, say A_u. A non–learning branch and bound ends up in finding the overall crosstalk due to $A_s \cup A_u$. However, this is not realistic if any of the conflicting aggressor sets from the conflict database are present in $A_s \cup A_u$.

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.

Adaptive Bounding is based on property 3 described in section 3.1. With the application of this technique, a more realistic upper bound could be calculated by considering only a maximal set of aggressors from A_u along with A_s. As the learning process in the branch and bound is supported by the conflict–driven mechanism of the SAT solvers, application of this technique does not add any considerable overhead to the optimization process. The Adaptive Bounding process is described in pseudocode in procedure 2.

4 Implementation and Results

The proposed approach has been implemented in C++ in a false noise analysis engine and tested in an industrial environment. It is designed in such a way that all the nets of a given circuit that have cross–coupling are processed for false noise to filter their unrealistic aggressors. Although false noise analysis is aimed to be performed as a post-processing step to reduce the number of violations reported by static timing analysis (STA) by many [1, 2, 3, 6, 7], we target to perform this during STA to make timing analysis more realistic, accurate and reliable.

We have tested the proposed approach on several ISCAS89 benchmarks. The experiments were conducted on a SUN sparc Sun–Fire–V890 workstation. All the nets that see cross–coupling with other nets are considered. No aggressors are filtered based on coupling capacitance ratios or absolute values as is done by some commercial STA tools. Such simplifications are possible and would have a positive speed impact on our approach. Also, because we did not use any approximate heuristics that trade–off accuracy and speed, as proposed in [1], one can expect a further improvement in speed with their application.

We have provided a comparison of the results of our SAT based false noise analysis with and without the use of the proposed Simple Aggressor Ordering and Adaptive Bounding techniques. The analysis is done for the crosstalk fall delay scenario, where the victim is assumed to make a falling transition while the aggressors are rising. Similar results can be produced for other crosstalk types. Columns 2 to 5 of Table 1 provide the number of cross–coupled nets of each circuit processed versus the number of unsolved nets for each case. A net is considered unsolved if the number of recursive branch and bound calls during the optimization process exceeds an upper limit of 10,000. It can be observed that the application of both SAO and AB (referred as SAO+AB in Table 1) made all the nets solvable for all the tested circuits.

Columns 6 to 9 of Table 1 give the average reduction of pessimism of each net in terms of the number of aggressors. It can be observed that the amount of pessimism reduced per net is larger when SAO and AB are applied. This is a direct consequence of their ability to solve all the nets thus yielding optimal results. Reduction in pessimism could also be quantified in a different way, for instance in terms of the reduction of the number of false timing violations, as has been done in [1, 2, 3, 6, 7]. In the future we will work on these issues.

Columns 10 to 15 of Table 1 signify the speed improvement achieved using the proposed methods. The speed has been measured using two estimates: the average number of recursive calls to the branch and bound method and the CPU time. As already mentioned, we declare a net as unsolvable by a particular approach if it required more than 10,000 branch and bound (BnB) calls. For each infeasible case the BnB call count is set to be 10,000 and the CPU time to reach this stage is measured. The efficiency of each of the proposed techniques SAO and AB in successfully filtering futile search spaces during the optimization process can clearly be observed. As another measure to estimate the overall time required for the analysis, we also compare the total CPU time required for the analysis of each circuit. As the lower bounds of BnB call counts and CPU times are used for infeasible cases, the average improvement figures provided in Table 1 are the lower bounds on the performance and accuracy gains achieved.

A comparison of the speed of our approach to the classical logic constraints approach [1] is not provided, because our implementation of this approach could not finish in most of the cases as we do not apply any approximate heuristics.

The results clearly show the efficiency of the proposed techniques in effectively increasing solvability and speed, and reducing pessimism. These techniques could also be applied on the top of the logic constraints approach proposed in [1] with considerable improvement in speed. However, we do not recommend this as the basic logic constraints approach is many factors slower than our SAT based approach even without the application of SAO and AB.

5 Conclusions

A SAT based approach to analyze false crosstalk has been proposed along with two powerful techniques, Simple Aggressor Ordering and Adaptive Bounding, that help to improve the speed of the optimization process. The proposed techniques have been implemented and tested on ISCAS89 benchmark circuits demonstrating their capability to drastically improve speed, accuracy and solvability without any trade–offs. The results provided show the necessity of such methods to solve the false noise problem in the first place.

References

[1] A. Glebov, S. Gavrilov, R. Soloviev, V. Zolotov, M. Becer, C. Oh, and R. Panda., Delay noise pessimism reduction by logic correlations, *International Conference on Computer Aided Design*, pages 160–167, 2004.

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.

Circuit	No of Nets				# Pot. Aggrs. (max/ avg)	Avg. Reduction in Aggr. Count (%)			Avg. no. of BnB Calls per net			Total CPU time (sec)		
	proc-essed	unsolved												
		SAT	SAO	SAO + AB		SAT	SAO	SAO + AB	SAT	SAO	SAO + AB	SAT	SAO	SAO + AB
s208.1	72	0	0	0	27/6	22.01	22.01	22.01	192	14	7	1.07	0.12	0.07
s298	92	2	0	0	37/6	19.41	20.49	20.49	346	19	8	3.36	0.24	0.14
s349	101	1	0	0	26/7	15.24	15.59	15.59	346	9	8	3.52	0.18	0.12
s386	111	9	0	0	33/7	18.89	22.70	22.70	996	39	10	13.49	0.55	0.29
s382	116	1	1	0	49/6	23.95	23.94	24.34	171	95	7	2.98	1.98	0.2
s344	122	3	0	0	42/8	20.31	21.15	21.23	519	27	9	7.69	0.5	0.21
s420.1	124	2	0	0	40/6	21.76	22.33	22.33	198	30	8	3.47	0.67	0.17
s444	129	3	0	0	43/6	23.55	24.53	24.53	337	17	7	6.2	0.36	0.25
s713	155	8	1	0	62/9	12.08	13.47	13.72	376	31	8	8.45	0.91	0.32
s400	155	2	0	0	51/7	25.94	26.53	26.53	721	90	11	18.42	2.99	0.55
s635	158	2	1	0	67/5	21.29	21.45	21.71	182	78	7	4.92	2.11	0.38
s526	160	5	0	0	36/7	20.80	21.90	21.90	560	26	9	13.22	0.74	0.3
s499	161	3	1	0	60/9	36.43	36.85	37.27	393	106	12	9.43	2.8	0.48
s641	163	15	1	0	64/12	10.36	12.68	12.88	1463	162	14	44.15	5.54	0.79
s526n	167	6	0	0	42/8	22.79	24.40	24.40	591	44	10	14.64	1.27	0.41
s510	182	15	0	0	52/9	29.43	34.05	34.05	1080	75	13	27.98	2.14	0.61
s820	239	24	3	0	63/9	23.19	27.49	28.22	1112	321	14	48.8	15.23	1.07
s938	243	4	1	0	76/7	19.64	20.08	20.21	296	87	9	14.87	5.47	0.98
s838.1	246	5	2	0	75/7	18.11	18.49	18.81	345	96	9	16.86	5.92	0.87
s832	272	25	6	0	64/9	22.83	25.93	27.11	1040	313	14	59.33	17.44	1.33
s991	294	4	0	0	60/9	13.21	13.60	13.60	253	29	10	18.86	2.67	1.27
s1238	354	41	7	0	85/11	22.82	27.54	28.67	1345	370	18	118	37.05	3.43
s1196	378	36	7	0	84/10	25.59	29.67	30.79	1177	362	18	112	36.33	3.62
s1494	431	59	27	0	123/14	23.44	27.35	31.01	1578	824	32	195	96.98	7.75
s3271	775	17	1	0	72/7	19.64	20.34	20.38	367	36	8	134	16.99	5.42
s9234.1	909	73	30	0	126/12	16.11	17.55	18.79	937	461	20	588	290	18.63
s5378	977	84	2	0	91/11	14.19	16.52	16.60	1142	128	14	699	92.71	14.01
s4863	984	58	5	0	88/8	10.95	12.33	12.48	700	91	11	506	90.23	18.42
s6669	1248	72	1	0	96/9	8.15	9.27	9.29	680	61	11	952	112	22.08
s13207.1	2310	187	57	0	204/12	16.02	17.63	18.54	946	356	20	3147	1284	103
s15850.1	2415	191	94	0	201/14	12.38	13.28	14.41	922	471	26	3653	1860	148
s38584.1	5580	599	213	0	335/15	11.39	12.76	13.64	1240	474	25	24048	9809	661
s38417	6763	516	135	0	226/11	11.01	12.28	12.78	884	276	15	24826	8472	596
Improvement (%) – SAT vs. SAO+AB (min/max/avg)						0 / 32.33 / 11.37			95.72 / 99.04 /97.82			92.28 / 98.21 / 96.24		

Table 1. Solvability, aggr. count reduction, avg. no. of BnB calls and CPU times for various ISCAS89 benchmarks

[2] A. Glebov, S. Gavrilov, V. Zolotov, R. Panda, C. Oh, and D. Blaauw, False–noise analysis using resolution method, *International Symposium on Quality Electronic Design*, 2002.

[3] A. Glebov, S. Gavrilov, D. Blaauw, S. Sirichotiyakul, C. Oh, and V. Zolotov, False noise analysis using logic implications, *International Conference on Computer Aided Design*, pages 515–521, 2001.

[4] R. Arunachalam, R. D. Blanton, and L. T. Pileggi, False coupling interactions in static timing analysis, *Design Automation Conference*, 2001.

[5] M. Palla, K. Koch, J. Bargfrede, M. Glesner, W. Anheier, Reduction of crosstalk pessimism using Tendency Graph Approach, *International Conference on Computer Design*, 2006.

[6] K. Tseng, M. Horowitz, False coupling exploration in timing analysis, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 11, pages 1795–1805, 2005.

[7] Y. Ran, A. Kondratyev, K. Tseng, Y. Watanabe, M. Marek-Sadowska, Eliminating False Positives in Crosstalk Noise Analysis, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 9, pages 1406–1419, 2005.

[8] The International SAT Competitions web site, http://www.satcompetition.org

[9] N. Eén, N. Sörensson, An Extensible SAT–solver, *International Conference on Theory and Applications of Satisfiability Testing*, 2003.

Authorized licensed use limited to: IEEE Xplore. Downloaded on April 7, 2009 at 00:36 from IEEE Xplore. Restrictions apply.