Reduction of Crosstalk Pessimism Using Tendency Graph Approach

Murthy Palla*, Klaus Koch*, Jens Bargfrede*, Manfred Glesner†, Walter Anheier‡
*Future Design Systems Group, Infineon Technologies AG, Munich
{murthy.palla, klaus.koch, jens.bargfrede}@infineon.com
†Institute of Microelectronic Systems, Darmstadt University of Technology, Darmstadt, glesner@mes.tu-darmstadt.de
‡ ITEM, University of Bremen, Bremen, anheier@item.uni-bremen.de

***Abstract*— Accurate estimation of worst-case crosstalk effects is critical for a realistic estimation of the worst-case behavior of deep sub-micron circuits. Crosstalk analysis models usually assume that the worst-case crosstalk occurs with all the aggressors of a victim (net or path) simultaneously inducing crosstalk even though this may not be possible at all. This overestimated crosstalk is called false noise. Logic correlations have been explored to reduce false noise in [3], which also used branch and bound method to solve the problem. In this paper, we propose a novel approach, named Tendency Graph Approach (TGA), which preprocesses the logic constraints of the circuit to drastically speed up the fundamental branch and bound algorithm. The new approach has been implemented in C++ and tested on an industrial circuit in a current 90 nm technology, demonstrating that TGA considerably accelerates the solution to the false noise problem, and makes in many cases branch and bound feasible in the first place.**

I. Introduction

Ever since crosstalk has been considered in synthesizing deep sub-micron (DSM) circuits, designers kept wondering if the crosstalk predicted by the algorithms is that likely or even possible at all. As it turns out, indeed, quite a lot of crosstalk is not possible [3] [4] [5] [6] [7]. Such impossible crosstalk and its excess noise, or delay and 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 or 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 filtering out the most primitive logic correlations of single inverter and buffer cells. Yet, this is far from being sufficient. For this, consider the example 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.

False noise not only distorts the crosstalk analysis of the net where it originates, 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 [3]. False calculated slew change due to crosstalk even distorts the analysis of subsequent cells in the path and finally the timing-check calculation.

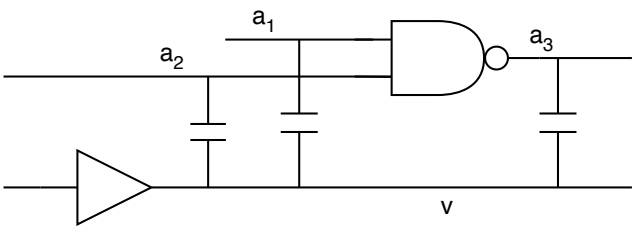

Fig. 1. False noise example with impossible aggressor combination $\mathbf{a_1}$, $\mathbf{a_2}$ and $\mathbf{a_3}$ of victim $\mathbf{v}$.

Recently, new algorithms have been proposed to take into account more complicated logic and timing correlations for finding false noise and selecting the max. (worst) realizable aggressor set (MRAS) [3] [4] [5]. What keeps them from entering mainstream is the NP-complete nature of the problem itself, thus a great need exists for heuristics which could accelerate the solution with tolerable pessimism. We propose such an algorithm which we name Tendency Graph Approach (TGA). It utilizes logic correlations of the involved cells in order to find a quantitative measure for the probability of switching constellations. This measure is then applied to sort and bipartition the graph for the branch and bound algorithm used to get the exact solution of false noise.

Contrary to the approximate heuristics proposed in [3], our approach does not require to ignore 'some' logic correlations among different subsets of aggressors and thus trade feasibility with pessimism and accuracy. Instead, all correlations can be taken into account and the exact solution is found. However, TGA could be combined with those heuristics, leading to even further accelerated computation.

1-4244-9707-X/06/$20.00 ©2006 IEEE

The proposed technique has been tested on a 90 nm industrial design, considering false noise in crosstalk delay, speed-up, high noise and low noise. It is shown that of the investigated cases, about fifty percent couldn't have been solved exactly without TGA in reasonable time. TGA makes such cases not only feasible, but leads to an overall acceleration of about 63.7 % on average and of up to 98.4 %. Further, of the potential aggressors, about 31 % on average were pruned out.

As mentioned earlier, false noise could arise due to the negligence of timing correlations and/or logic constraints of the circuit while estimating worst-case crosstalk. In this paper, we consider only false noise arising due to the negligence of logic constraints assuming zero delay. However, the proposed technique is independent on how logic constraints are generated and could be used with any other technique. Even though only digital gates are discussed, the proposed approach can be used on CMOS transistors as well.

The remaining part of the paper is organized as follows. Section II briefly discusses state of the art in false noise analysis. Section III describes the Tendency Graph Approach and its derivation. Section IV presents results for an industrial 90 nm design, followed by conclusions in V.

II. State of the art

A. Logic Constraints

Logic constraints are impossible combinations of signals. They indicate whether a particular combination of signals of a circuit is logically possible or not. Mathematically, a logic constraint is a boolean term with one or more literals, either inverted or uninverted, in a conjunctive form. The literals indicate the signal values on the corresponding nets of the circuit. If the signal values are such that the term (logic constraint) gives a logic '1', then the signal combination is logically impossible.

At the first step, logic constraints of the entire circuit are generated. We followed the methods described in [4] and [3] for this purpose. We consider an example here to explain the generation of logic constraints. However, the reader is referred to the original papers for detailed descriptions.

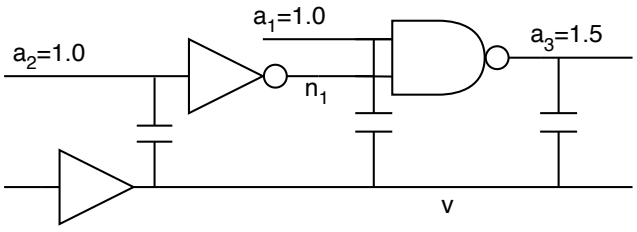

Fig. 2. Sample circuit for deduction of logic constraints. Nets $\mathbf{a_1}$, $\mathbf{a_2}$ and $\mathbf{a_3}$ are the potential aggressors of victim $\mathbf{v}$ with the numbers near them representing their strengths.

Figure 2 depicts a simple circuit with a 'NAND' gate and an inverter, and its logic constraints. The nets a_1, a_2 and a_3 are aggressors to the net v. The numbers near the aggressors are the strengths with which they impair the victim (they are used in the example of TGA in section III-G). In the logic constraint generation process, first the logic constraints for the primitive gates are generated. The logic constraints for the 'NAND' gate are $\overline{a_1}\cdot\overline{a_3}$, $\overline{n_1}\cdot\overline{a_3}$ and $a_1\cdot n_1\cdot a_3$. Similarly, the logic constraints for the inverter are $a_2\cdot n_1$ and $\overline{a_2}\cdot\overline{n_1}$. As n_1 is not an aggressor of v, the constraints consisting of the literal n_1 have to be dropped. However, we can avoid the loss of logical information resulting from the dropping of these constraints by using *resolution principle*, according to which, if A and B are two arbitrary logical expressions and a is a literal, then

$$a\cdot B=0,\ \overline{a}\cdot C=0\ \rightarrow\ B\cdot C=0$$

Applying resolution principle to the logic constraints containing n_1: $\overline{n_1}\cdot\overline{a_3}$, $a_1\cdot n_1\cdot a_3$, $a_2\cdot n_1$ and $\overline{a_2}\cdot\overline{n_1}$ results in the new logic constraints $a_2\cdot\overline{a_3}$ and $a_1\cdot\overline{a_2}\cdot a_3$. Thus, the final logic constraints for the aggressors a_1, a_2 and a_3 are $\overline{a_1}\cdot\overline{a_3}$, $a_1\cdot\overline{a_2}\cdot a_3$ and $a_2\cdot\overline{a_3}$.

B. MWIS Representation

Finding the actual worst-case set of aggressors and its induced crosstalk can be represented as a maximum weight independent set (MWIS) problem [3][4][5]. The logic constraints of the problem are translated into a hypergraph with logic constraints as hyper-edges and the aggressors as its vertices. Each vertex is weighted by the impact of its corresponding aggressor, turning the false noise problem into the problem of finding a maximal set of vertices of the hypergraph such that all the vertices of any of the hyper-edges are not selected simultaneously.

C. Exact Solution Approach

An exact solution approach to detect false noise should essentially use the branch and bound technique as proposed in [3]. The bounding strategy is based on the currently optimal solution and the upper bound on the noise from the unexplored subtree. If the upper bound obtained by considering all the elements of the unexplored subtree is less than the currently optimal solution, the subtree is dropped and the search is continued. Avoiding unnecessary branching using the bound strategy helps a lot in increasing the speed of the algorithm. However, the worst-case complexity of the algorithm is still $O(2^n)$, where n is the number of potential aggressors. Even though the worst-case complexity may seldom be reached, the problem could often get infeasible or just too costly for problems with a large number of potential aggressors. Hence, increasing the speed of the algorithm using heuristics without trading off its accuracy is mandatory. This can be achieved by using the proposed Tendency Graph Approach.

III. Tendency Graph Approach

Tendency Graph Approach is based on the branching heuristic 'fail first' which states: "To succeed, try first where you are most likely to fail" [1][2]. Fail first suggests the ordering of the elements fed to the branch and bound algorithm in a way that unnecessary branches are dropped out at an earlier stage, thus saving unnecessary explorations.

Tendency Graph Approach increases the speed of branch and bound algorithm by trying to minimize the effort required to determine whether an insoluble subtree is indeed insoluble. However, since branch and bound has an exponential worst-case complexity, for problems with a very large number of elements (typically greater than 50), the algorithm may not be feasible even after applying the heuristic. In such cases, additional, yet approximate solutions which aim at partitioning the set of aggressors into smaller sub-sets are discussed in [3].

A. What is Aggressor Tendency?

'Aggressor Tendency' or simply 'Tendency' of two aggressors is a quantitative measure of the probability with which they switch in the same or opposite direction (relative to each other) obtained from the logic constraints of the circuit. A negative tendency between two aggressors indicates that they tend to switch in opposite directions and similarly, a positive tendency indicates that they tend to switch in the same direction.

To make the idea of 'tendency' clearer, let us consider the simple example of a NAND gate with signals a and b at inputs and z at output. One of the three logic constraints of the NAND gate is $\overline{a} \cdot \overline{z}$ avoiding the combination of the signals $a = 0$ and $z = 0$. In other words, the logic constraint $\overline{a} \cdot \overline{z}$ invalidates one of the four possible combinations of the signals a and z. Out of the remaining three valid combinations of a and z, two combinations are with a and z having opposite values indicating that the probability that a and z are opposite in their values is greater than the probability that they are equal in their values. Said differently, a and z have the 'tendency' to switch in opposite directions.

B. Derivation of Aggressor Tendencies

A quantitative measure of 'tendency' is necessary to algorithmically use aggressor tendencies for the fail-first ordering of aggressors. For this purpose, let us consider a logic constraint with n literals $l_1 \cdot l_2 \cdots l_n$. Please note that the literals could either be inverted or uninverted, i.e., a literal l_i could, in general, be inverted or uninverted. Let $N(predicate)$ be a function determining the number of possible combinations of the values of the n literals of $\{l_1, l_2 \cdots l_n\}$ satisfying the *predicate*. Without the consideration of the logic constraint, for any two literals l_i, l_j from the literal set $\{l_1, l_2 \cdots l_n\}$, we have:

$$N(1) = 2^n, \tag{1}$$

$$N(l_i = l_j) = 2^{n-1} \tag{2}$$

and similarly,

$$N(l_i \neq l_j) = 2^{n-1} \tag{3}$$

With the logic constraint invalidating exactly one combination of the 2^n possible combinations of values of the containing literals, we have:

$$N(1) = 2^n - 1 \tag{4}$$

$$N(l_i = l_j) = \begin{cases} 2^{n-1} - 1 & \text{if both } l_i \text{ and } l_j \text{ are either inverted or uninverted} \\ 2^{n-1} & \text{otherwise} \end{cases} \tag{5}$$

and similarly,

$$N(l_i \neq l_j) = \begin{cases} 2^{n-1} - 1 & \text{if exactly one of } l_i \text{ and } l_j \text{ is inverted} \\ 2^{n-1} & \text{otherwise} \end{cases} \tag{6}$$

If $P(predicate)$ is the probability with which the *predicate* is true, we have (after the consideration of the logic constraint):

$$P(l_i = l_j) = \begin{cases} \dfrac{2^{n-1} - 1}{2^n - 1} & \text{if both } l_i \text{ and } l_j \text{ are either inverted or uninverted} \\ \dfrac{2^{n-1}}{2^n - 1} & \text{otherwise} \end{cases} \tag{7}$$

$$P(l_i \neq l_j) = \begin{cases} \dfrac{2^{n-1} - 1}{2^n - 1} & \text{if exactly one of } l_i \text{ and } l_j \text{ is inverted} \\ \dfrac{2^{n-1}}{2^n - 1} & \text{otherwise} \end{cases} \tag{8}$$

Let s_i store the information of whether the literal l_i is inverted or not, such that

$$s_i = \begin{cases} 1 & \text{if } l_i \text{ is inverted} \\ 0 & \text{otherwise} \end{cases}$$

where $i \in \{1,\ 2,\ \cdots,\ n\}$. We can then define the tendency, T_{l_i,l_j} of two literals l_i and l_j as

$$\begin{aligned} T_{l_i,l_j} &= P(l_i = l_j) - P(l_i \neq l_j) \\ &= (-1)^{s_i + s_j + 1} \cdot \frac{1}{2^n - 1} \end{aligned} \tag{9}$$

To summarize, tendency of two literals is the difference between the probabilities with which they are equal or different in their values. This information is useful in identifying aggressor pairs which tend to switch in the same direction or opposite directions.

C. What is a Tendency Graph?

A tendency graph is a graph that gives the switching tendencies of aggressors with respect to other aggressors. The vertices of a tendency graph represent the aggressors, and the weights of its edges represent the tendencies of the corresponding aggressors they connect. As explained in sections III-A and III-B, tendency is a probabilistic measure. If the weight of an edge is negative, it indicates that the two aggressors it connects tend to switch in opposite directions. Similarly, if the weight of an edge is positive, it indicates that the two aggressors it connects tend to switch in the same direction. In general, the edge weights of a tendency graph can either be negative or positive. A subgraph of a tendency graph with all its positive weighted edges removed is called its *negative tendency graph*. Similarly, a subgraph of a tendency graph with all its negative weighted edges removed is called its *positive tendency graph*.

D. Building a Tendency Graph

In order to build a tendency graph, the tendencies of various aggressor pairs are obtained from the logic constraints. If an aggressor pair is present in different logic constraints, the tendencies obtained from all these logic constraints are summed up to find its overall tendency. The tendency graph is build using this information as described in procedure 1.

Procedure 1 BUILD_TENDENCY_GRAPH

Input: Logic constraint set, *lCS*
Output: Tendency graph, tG

```
Create an empty graph tG
for each logic constraint lC of lCS do
    for each pair of literals l_i and l_j of lC do
        Calculate tendency T_{l_i,l_j} using formula (9)
        if l_i and l_j are not vertices of tG then
            add them to tG
        if edge (l_i, l_j) does not exist in tG then
            create edge (l_i, l_j) with weight T_{l_i,l_j}
        else
            add T_{l_i,l_j} to the weight of the existing edge
return tG
```

E. Bipartition of Tendency Graph

The aim of TGA is to partition the potential aggressors into two sets such that the aggressors of any set switch with a high probability in the opposite direction to the aggressors of the other set. This is achieved by bipartitioning the tendency graph. The tendency graph consists of information on which aggressors tend to switch in opposite directions and which tend to switch in the same. Further, the graph contains quantitative information on strength of the tendency. In the best case, bipartition can be done in such a way that the two aggressors of all the aggressor pairs connected by a negative tendency edge are in opposite partitions and the two aggressors of all aggressor pairs connected by a positive tendency edge are in the same partition. In other words, the best case is when any two aggressors connected by a negative weighted edge fall in opposite sets and any two aggressors connected by a positive weighted edge fall in the same set after bipartition. However, this is not achievable if the negative tendency graph is not bipartite and certain tendencies of the positive tendency graph contradict the negative tendency graph. In such cases, it is inevitable to lose some information by dropping some edges of the negative tendency graph to make it bipartite.

A bipartite graph, also called a bigraph, is a set of graph vertices that can be decomposed into two disjoint sets such that no two vertices in the same set are adjacent [9]. A graph is bipartite if and only if all its cycles are of even length [8]. In order to make the negative tendency graph bipartite, cycles with odd lengths have to be broken. This could be achieved as described in procedure 2.

Once the negative tendency graph is made bipartite, it is a simple job to bipartition it. In case of more than one connected component in the negative tendency graph, the question arises as to which sets the partitions of the individual components have to go. This can be decided easily by using the positive tendency graph. We find the most suitable set into which the aggressors must go based on the positive tendency information. The process of bipartitioning a tendency graph is described in procedure 3.

Procedure 2 MAKE_BIPARTITE

Input: Positive tendency graph, $posTG$
Output: Bipartite negative tendency graph, $posTG$

```
Find all the odd-length cycles of posTG using depth-first-search
Break the cycles by removing edges of least possible weight from posTG
return posTG
```

Procedure 3 BIPARTITION_TENDENCY_GRAPH

Input: Tendency graph, tG; Weight function, $W()$
Output: Vertex set pair, (A_s, A_w)

```
Create two graphs posTG and negTG with same vertices as tG and no edges
for each edge e of tG do
    if W(e) > 0 then
        add e to posTG
    else if W(e) < 0 then
        add e to negTG
Create two empty vertex sets A_s and A_w
Make negTG bipartite using procedure 2
for each connected component cC of negTG do
    Bipartition cC into partitions p_1 and p_2
    if A_s is not empty then
        if p_1 has more positive tendency to A_s than p_2 then
            add p_1 to A_s
            add p_2 to A_w
        else
            add p_1 to A_w
            add p_2 to A_s
    else
        add p_1 to A_s
        add p_2 to A_w
Insert all unconnected vertices in A_s
return pair(A_s, A_w)
```

F. Fail-first Ordering of Aggressors

After bipartition, the aggressors are ready for ordering based on the fail first principle. The overall strengths of the aggressors of each of the two partitions is then calculated. The aggressors are then ordered in such a way that the aggressors from the partition with greatest overall strength are before the aggressors of the other partition. The aggressors of each individual partition are again internally ordered in the descending order of their weights. The ordered aggressor

vector is then fed to the branch and bound algorithm to find the exact solution with much greater speed.

G. TGA on the Sample Circuit

To give an impression of the advantage of TGA over the fundamental approach, we solve the false noise problem of the sample circuit depicted in figure 2 and compare the results. The logic constraints for this example are $\overline{a_1} \cdot \overline{a_3}$, $a_1 \cdot \overline{a_2} \cdot a_3$ and $a_2 \cdot \overline{a_3}$, the generation of which was explained in section II-A. Figure 3(a) depicts the tendency graph obtained by applying procedure 1 on these logic constraints.

Now, the bipartition of this graph can be achieved by applying procedure 3. Figure 3(b) depicts the partitions of the tendency graph obtained after the bipartitioning process.

Fail-first ordering of these aggressors now results in the aggressor order "a_3, a_2, a_1". Feeding the aggressors in this order to the branch and bound algorithm results in 3 calls to the recursive branch and bound method. Instead, if the aggressors are fed in the order "a_1, a_2, a_3", the branch and bound method is called 7 times. Thus, a drastic speed-up of 57 % is evident in this simple example. For problems with larger number of aggressors, fail-first ordering with TGA may be considered as essential, if not just useful.

IV. Results

The proposed Tendency Graph Approach (TGA) has been implemented in C++ and tested on a 90 nm technology industrial design with about 70,000 standard cells. TGA is tested for various crosstalk scenarios (high-noise, low-noise, delay and speed-up) and the number of calls to the recursive branch and bound method with and without TGA are compared. Our TGA prominently dominated the rudimentary exact solution approach proposed in [3] for all the scenarios. Due to space constraints, we provide the results only for the crosstalk low-noise scenario. However, pessimism reduction for various scenarios in terms of number of aggressors is depicted in figure 4. All the nets of the circuit that are parasitically (capacitively) coupled to the corresponding victim net are considered to be potential aggressors. Victims with potential aggressors in the number ranging from 15 to 45 were chosen randomly from the circuit. Table I gives the reduction in crosstalk pessimism for low-noise scenario in terms of number of aggressors. A pessimism reduction of 31.03 % on average and up to 50 % in terms of the number of aggressors is observed in this case.

Victim (net name)	Number of aggressors		
	potential	real	reduction [%]
*22423	16	12	25.0
*6539	19	16	15.8
*9686	20	17	15.0
*5030	21	17	19.1
*23576	21	11	47.6
*21769	21	17	19.1
*25396	22	17	22.7
*39681	22	19	13.6
*29713	23	13	43.5
*7904	24	13	45.8
*7903	25	16	36.0
*3026	26	23	11.5
*33973	27	21	22.2
*53716	27	22	18.5
*5931	28	15	46.4
*51853	30	19	36.7
*7900	30	24	20.0
*33042	31	19	38.7
*2574	32	16	50.0
*7893	32	17	46.9
*39390	32	20	37.5
*28763	34	20	41.2
*5870	35	23	34.3
*46710	36	25	30.6
*51714	38	31	18.4
*2038	40	24	40.0
*7906	40	24	40.0
*32991	41	28	31.7
*32991	41	31	24.4
*7892	44	27	38.6
Average			31.03

TABLE I

Pessimism reduction for crosstalk low-noise in terms of number of aggressors. Potential aggressors are the nets that are capacitively coupled to the corresponding victim nets. Real aggressors are the aggressors from the maximum realizable aggressor set (MRAS) obtained after false noise reduction.

Table II shows the number of calls to the recursive branch and bound method with and without the application of TGA for the crosstalk low-noise scenario. A reduction of 63.7 % on average and up to 98.4 % in the number of calls to branch and bound is observed in this case.

One case (victim *39681) in table II shows more calls to the branch & bound with TGA. This can be explained as a case where the original order of aggressors is better than the fail-first ordering done by TGA. This has a very small probability to happen and is the only case where we have seen that.

A linear model for finding crosstalk was used, which means each aggressor was weighted by the amount of crosstalk it induces on the victim and the amount of crosstalk that a set of aggressors can simultaneously induce is the sum of their weights. For the purpose of testing, all aggressors were assumed to induce an amount of crosstalk equal to the capacitance with which they were coupled to their respective victims. For more realistic and accurate analysis, a more sophisticated weight function may be used, which could be easily incorporated into the implemented program and existing model.

The results indicate that the number of aggressors that can simultaneously induce crosstalk to make the real worst-case is up to 50 % smaller than the number of potential aggressors. The average reduction was about 31 %. This indicates a great reduction in pessimism in crosstalk analysis, thus leading to a better utilization of available resources. If timing correlations are also used in addition to logic constraints, the false noise could be reduced even further.

Victim (net name)	Number of calls to branch & bound		
	w/o TGA	with TGA	speed inc. [%]
*22423	4	2	50.0
*6539	29	6	79.3
*9686	20	15	25.0
*5030	92	44	52.2
*23576	757	436	42.4
*21769	26	13	50.0
*25396	124	44	64.5
*39681	181	278	−53.6
*29713	1184	339	71.4
*7904	> 10000	540	> 94.6
*7903	> 10000	1697	> 83.0
*3026	211	64	69.7
*33973	2232	779	65.1
*53716	> 10000	3194	> 68.1
*5931	2378	430	81.9
*51853	> 10000	4370	> 56.3
*7900	> 10000	7638	> 23.6
*33042	6970	742	89.4
*2574	455	105	76.9
*7893	1300	51	96.1
*39390	268	103	61.6
*28763	9129	822	91.0
*5870	> 10000	341	> 96.6
*46710	> 10000	160	> 98.4
*51714	> 10000	335	> 96.7
*2038	5019	83	98.3
*7906	> 10000	2302	> 77.0
*32991	> 10000	6804	> 32.0
*32991	> 10000	4336	> 56.6
*7892	> 10000	8438	> 15.6
Average			> 63.7

TABLE II

SPEED GAIN IN NUMBER OF CALLS TO BRANCH AND BOUND WITH USE OF TGA FOR CROSSTALK LOW-NOISE. CASES WITH MORE THAN 10000 CALLS WERE CONSIDERED INFEASIBLE, THUS THE CALCULATION WAS STOPPED WHEN THIS LIMIT WAS REACHED. ALL SUCH CASES COULD BE SOLVED WITH LESS THAN 10000 CALLS WHEN TGA WAS APPLIED.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown the need for accelerating the basic branch and bound algorithm used for computing false noise in [3] and proposed a novel approach called 'Tendency Graph Approach'. Contrary to the approximate heuristics proposed in [3], this approach speeds up the branch and bound algorithm for finding false noise without trading off accuracy and increases the applicability of exact solution approach for problems with aggressor sets of size up to 50. The new approach has been implemented in C++ and tested on an industrial 90 nm design. The analysis of randomly chosen nets from this design was demonstratively accelerated by the proposed approach. The maximum reached run time improvement was 98 %. In this paper, only false noise due to the negligence of logic constraints was considered. Still, a significant pruning of potential aggressors could be accomplished (on average of about 31 %). Future work could include also the dependency of signal transitions in regard to timing.

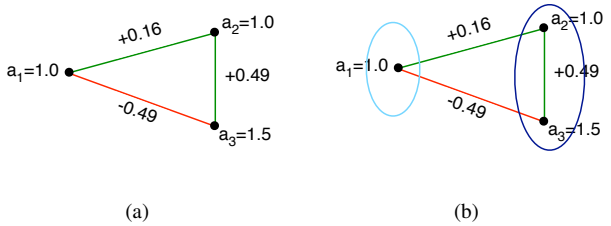

Fig. 3. Tendency graph for example in fig. 2: (a) before bipartition, (b) after bipartition. In (b), the vertices connecting the negative tendency edge are in opposite partitions. Aggressor $\mathbf{a_2}$ falls in the same partition as $\mathbf{a_3}$ because its positive tendency is greater towards $\mathbf{a_3}$ than $\mathbf{a_1}$.

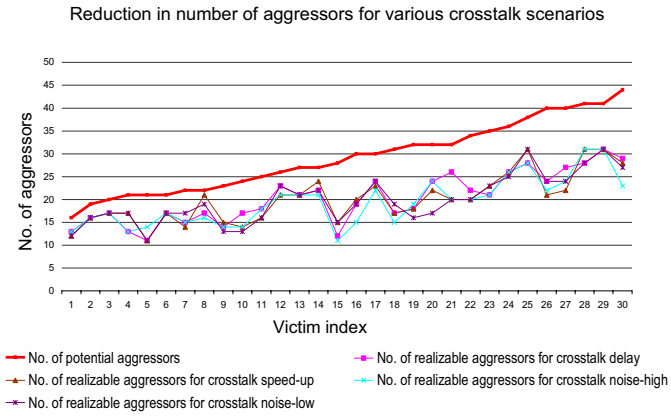

Fig. 4. Pessimism reduction for various crosstalk scenarios: high-noise, low-noise, delay and speed-up.

ACKNOWLEDGMENT

This work has been supported by the German Ministry of Education and Research (BMBF) within the project "LEONIDAS+" (Project ID 01M3074). The content is the sole responsibility of the authors.

REFERENCES

[1] Haralick R.M. and Elliott G.L., Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, 1980.

[2] Beck J.C., Prosser P., and Wallace R.J., Trying Again to Fail First, *Recent Advances in Constraints, Lecture Notes in Artificial Intelligence*, vol. 3419, Springer, 2005.

[3] 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 (ICCAD)*, 2004.

[4] 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 (ISQED)*, 2002.

[5] 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 (ICCAD)*, pages 515–521, 2001.

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

[7] D. Friedberg and K. Singhal, Removing pessimism from crosstalk analysis, February 2003. http://www.commsdesign.com.

[8] S. Skiena, Coloring Bipartite Graphs, section §5.5.2 of *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica*, page 213, Addison-Wesley, 1990.

[9] E. W. Weisstein, Bipartite Graphs, from MathWorld, A Wolfram Resource. http://mathworld.wolfram.com/BipartiteGraph.html.