Timing Arc Based Logic Analysis for False Noise Reduction

Murthy Palla[1,2], Jens Bargfrede[2], Stephan Eggersglüß[3], Walter Anheier[1], Rolf Drechsler[3]
[1] ITEM, University of Bremen, Bremen, Germany
[2] Infineon Technologies AG, Munich, Germany
[3] Institute of Computer Science, University of Bremen, Bremen, Germany
palla@ieee.org, Jens.Bargfrede@infineon.com, {segg, drechsle}@informatik.uni-bremen.de, anheier@item.uni-bremen.de

ABSTRACT

The problem of calculating accurate impact of crosstalk on a circuit considering its inherent logic and timing properties is very complex. Although it has been widely studied, it still lacks an efficient solution. As a result, state–of–the–art crosstalk calculators use simplistic and overly pessimistic models resulting in the overestimation of crosstalk effects. Such pessimism in crosstalk analysis often leads to the triggering of false violations and consequently an inefficient use of design resources.

The main contribution of this paper is a novel technique called *Timing Arc Based Logic Analysis* (TABLA) that serves as an efficient means to calculate realistic crosstalk bounds. TABLA uses timing arcs as basic elements to perform an efficient temporal logic analysis employing the min–max timing model using dedicated solvers for logic and timing. Additionally, a procedure to generate powerful conflict clauses is proposed to improve the run time of the overall analysis. The proposed technique has been tested in an industrial environment on benchmark circuits as well as on an industrial design, and results are provided.

Categories and Subject Descriptors

4.2 [**CAD for Circuits, Devices and Interconnect**]: Timing and Behavioral Modeling—*Timing analysis and methodologies*

General Terms

Static timing analysis, crosstalk, false noise analysis, SAT

1. INTRODUCTION

With the shrinking design geometries and increasing clock frequencies, the effects of crosstalk on delay and noise are increasing, thus making the accuracy of crosstalk analysis ever more critical. However, because of the complexity of the problem of considering the inherent logic and timing properties of a circuit in estimating its crosstalk, conventional crosstalk models often make a simplistic assumption that all the potential aggressors of any given victim net or path can simultaneously induce crosstalk. This unrealistic worst–case crosstalk model leads to an overly conservative and pessimistic estimation of the circuit crosstalk. The amount of overestimated 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 can render certain switching scenarios impossible are often simply ignored. Industrial strength crosstalk analysis tools try to avoid this for simple cases by considering the simple logic correlations arising from inverter and buffer cells, for instance in clock trees. Yet this is far from being sufficient as is apparent from [1, 4, 10, 13, 11, 12]. 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.

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 (also termed functional noise [4]) can be attenuated by subsequent cells, crosstalk delay never vanishes but sums up in the overall path delay and slack [4]. False calculated slew change due to crosstalk also distorts the analysis of subsequent cells in the path and, finally, the timing check calculation.

The false noise problem has been widely studied. In [4, 10], the problem is simplified by assuming a zero-delay circuit model. The approach in [1] proposes to independently account for logical and temporal false noise. Such a pure logic analysis under the zero delay assumption or independent accounting of logic and timing can be optimistic in the presence of glitches in the circuit. Hence, these approaches are conservative only for domino circuits [5] or specially designed glitch free circuits.

The work presented in [7] addresses the problem of estimating real crosstalk by considering a given set of *Logic Exclusivity* (LE) constraints using a gain based backtracking method. The effectiveness and accuracy of this method in reducing false noise depend on whether or not the set of LE constraints used cover the entire circuit and the circuit delay model used. At the same time, the deduction of all LE constraints for a given circuit is a complex problem by itself depending on the circuit delay model used and is not dealt in this paper.

To capture the effect of glitches, an analysis of timing, logic and their interdependence must be performed. Such an analysis is often called temporal logic analysis. The problem of temporal logic analysis to verify the validity of a crosstalk scenario can be described as a satisfiability problem. This problem is solved in [2, 11, 13] by converting it into a Boolean SAT instance, which leads to a potential exponential blow-up of variables and clauses, and then solving it using a state-of-the-art SAT solver, e.g. [3, 9]. These approaches often suffer from degraded performance due to the loss of high level semantics of the underlying timing problem and large SAT instances.

In this paper, we propose a novel technique called TABLA that efficiently handles the temporal logic analysis problem by using dedicated solvers for timing and logic, contrary to [13, 11]. To

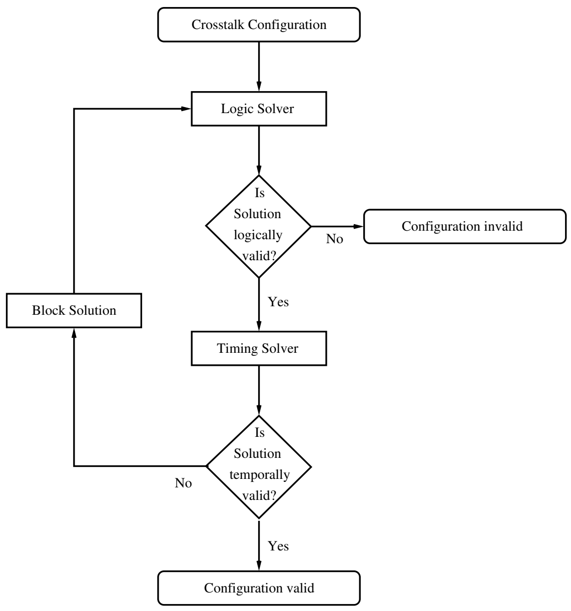


Figure 1: TABLA: top level flow chart

account for the interdependence of timing and logic, these solvers are tightly coupled by the use of common variables and data structures, which is possible because of the suitability of timing arcs to handle both logic and timing. As the higher level timing information is handled without any transformation into Boolean logic, this method maintains utmost accuracy without trading it off for run time. Although TABLA is applied on the false noise problem here, it is generic and can also be applied to other temporal logic analysis problems like, for instance, false path analysis.

2. KEY IDEA

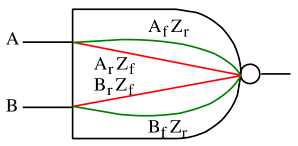
The major emphasis of this paper is on the fact that timing data can be handled more efficiently by using a dedicated timing solver rather than a logic solver working on the transformed problem. Hence, TABLA uses two separate but coupled solvers for logic and timing. We use MiniSat [3] as logic solver and our own timing solver built based on the principles explained in this paper.

TABLA uses a less restrictive logic solver that assumes the unbounded delay model (all gates and interconnects can assume an indefinite delay) [8] and a timing solver that assumes the min–max delay model designed to verify the temporal validity of the solutions provided by the logic solver. It performs a logic driven timing analysis that verifies the underlying logic and timing constraints. The logic solver acts as a master entity and proposes solutions that are valid from the logical perspective to the timing solver. The timing solver then checks if the logically valid solution is valid also temporally. If so, the crosstalk configuration is considered as valid. Otherwise, the timing solver indicates this to the logic solver, which would then verify if another logically valid solution exists. If so, it is handed over to the timing solver and the loop continues. Otherwise, the configuration is considered invalid. Thus, TABLA searches for all logically and temporally valid switching scenarios.

The higher level flow of TABLA is depicted in Figure 1. It should be noted that TABLA verifies the validity of a single crosstalk scenario. It is embedded in a branch and bound based optimization algorithm to perform false noise analysis as described in [4] and [10].

3. TIMING ARC BASED ENCODING

One of the fundamental differences between TABLA and other approaches in [4, 10, 13, 11] is that TABLA exploits the logical behavior of timing arcs of the cells of the circuit, whereas the others deduce logic correlations among the nets of the circuit. A timing



$A_r Z_f$	–	A rise & Z fall Condition: $B = 1$
$A_f Z_r$	–	A fall & Z rise Condition: $B = 1$
$B_r Z_f$	–	B rise & Z fall Condition: $A = 1$
$B_f Z_r$	–	B fall & Z rise Condition: $A = 1$

Figure 2: Timing arcs and side input conditions of a NAND gate

arc is a pin-to-pin timing path through one gate. Each timing arc has a start pin and an end pin. The start pin can be an input pin or an inout pin, and the end pin can be an output pin or an inout pin.

The set of all timing arcs of any given cell is an exclusive list of all possible paths through it. Depending on the complexity of the gate, there can be timing arcs with similar transition scenarios, but with different side input conditions. Consider the example of a NAND gate, which has four timing arcs as depicted in Figure 2.

Each timing arc represents a possible transition scenario at an input-output pin pair of its corresponding logic gate, and also the logical conditions under which such a transition scenario can actually happen. Further, it also represents the possible latency of the cell for this input-output transition scenario. This property of a timing arc makes it suitable for a closer integration of the logic and timing analysis of a circuit as explained in the following.

4. CIRCUIT TIMING GRAPH

A circuit timing graph is a directed acyclic graph with its vertices representing the nodes of the circuit and edges representing the timing arcs or nets connecting them. Its vertices and edges are assigned properties of their respective circuit elements that are required for TABLA. Using a graph structure to store the circuit information is helpful for the quick retrieval of data in a topological order. The information held by the vertices and edges of the graph and their classification into different types are described below.

Vertices

Each node of the circuit has two corresponding vertices in the circuit timing graph representing 'rise' and 'fall' transitions. These vertices are attributed the information corresponding to them such as the timing window, the maximum transition time and edge type. The two vertices belonging to a node of the circuit are called its paired vertices.

Based on their position in the graph, vertices are further classified into four types – input, net receiver, timing arc receiver and virtual. The names of these types indicate the kind of nodes they correspond to.

Edges

Edges of the circuit timing graph correspond to either a timing arc or a net branch, which are the basic delay elements of a circuit. Each timing arc of the circuit has a corresponding edge in the timing graph. Similarly, each net branch has two corresponding edges connecting the 'rise' driver vertex to its 'rise' receiver vertex and the 'fall' driver vertex to the 'fall' receiver vertex.

Edges of the circuit timing graph can be classified into three categories – timing arc, net branch and virtual. The names of these types indicate the circuit elements they correspond to.

5. LOGIC SOLVER

The logic solver considers correlations among the timing arcs of a circuit and proposes logically valid solutions to the timing solver (described in Section 6) for the verification of their temporal validity. If a logically valid solution of the logic solver is found to be temporally invalid, then a conflict clause is added to the instance that blocks this solution and possibly other similar solutions.

It is not sufficient to consider only the correlations among timing arcs to determine which static value a net can or must assume if none of the timing arcs connected to it are active. These values are required in the process of verification of the side-input conditions that must be satisfied for a timing arc to be active. Hence, each net is assigned a Boolean variable to store the static value on it. Deduction of the related constraints is explained in Section 5.2.

In order to make sure that glitches of the circuit are considered by TABLA, and that the underlying SAT instance is simple, the unbounded delay model is used by the logic solver. The unbounded delay model assumes an indefinite delay on the cells and interconnects of the circuit. As a result, the modeling of logic gets very simple. On the other hand, because of the overly conservative assumption of infinite timing windows, this model is less restrictive with respect to false noise reduction, unless it is combined with a method that considers more accurate timing windows, as done in this work.

5.1 Variable assignment

The activity of each of the timing arcs of the circuit under consideration is represented using a Boolean variable. An assignment of logic 1 to a Boolean variable implies that its corresponding timing arc is active and similarly, an assignment of logic 0 implies that the timing arc is not active.

In order to control the signal values on the input[1] nodes and the aggressors of a given crosstalk configuration, additional variables are defined. Imaginary timing arcs are assumed at the inputs, and the Boolean variables assigned to these timing arcs that control the input switching values are called virtual timing arc variables. Similarly, a Boolean variable per aggressor is assigned to control its activity.

To store the static values on nets that have all the timing arcs connected to them inactive, we define a Boolean variable for each net of the circuit. The value on this variable is invalid if any of the timing arcs connected to its corresponding net is active. The values of these variables are forced by the side-input conditions of active timing arcs and the circuit logic itself. This will make sure that two timing arcs whose side-input conditions contradict each other are not active at the same time.

The number of variables of the underlying SAT instance is in the order of the sum of the number of timing arcs and the number of nets of the circuit, and is hence very low compared to the number of variables required in the circuit unrolling process as in the case of [13].

5.2 Deduction of constraints

Constraints that verify the logical consistency of timing arcs are deduced. We consider three kinds of constraints: timing arc constraints, logic constraints and side-input constraints. In what follows, the procedure to deduce these three types of constraints is explained.

Timing arc constraints involve timing arc variables and they check the consistency of timing arcs. They can be further classified into two types: back-tracking constraints and single switching constraints.

- *Back-tracking constraints* ensure that a timing arc is active only if at least one of the timing arcs connected to its input is active. This is achieved by making backward implications from any timing arc to the corresponding timing arcs of the immediately preceding instances of its fan-in cone. Consider the example of timing arcs e and f of the circuit in Figure 3. If timing arc e has to be active, then either timing arc b or timing arc c should be active. Similarly, if timing arc f has to be active, then either timing arc a or timing arc d should be active. Hence, we capture the implications $e \Rightarrow b+c$ and $f \Rightarrow a+d$. They can be written in the form of constraints $(\overline{e}+b+c)$ and $(\overline{f}+a+d)$.

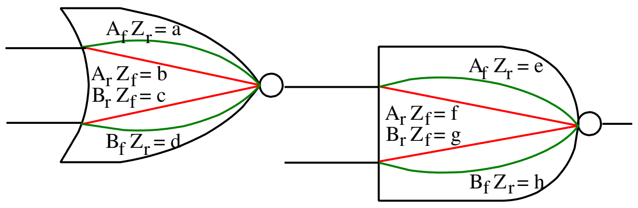


Figure 3: Example – deduction of constraints for TABLA

- *Single switching constraints* ensure that the signals on primary inputs and sequential cell outputs switch only once. For example, consider the inputs of the NOR gate in the circuit in Figure 3 to be primary inputs. The number of transitions possible on the nodes of this gate can then be restricted to one by adding the constraint clauses $(\overline{a}+\overline{b})$ and $(\overline{c}+\overline{d})$ to the CNF. It should be noted that this constraint does not exclude multiple input switching (MIS) within the circuit.

Logic constraints involve the net variables and they check the logical consistency of the static values assigned to the nets of the circuit. Logic constraints for all the gates of the circuit are deduced in a similar way to [10].

Side-input constraints verify if the side-input conditions of the corresponding timing arcs are satisfied. They involve both timing arc and net variables. These constraints verify the logical consistency of timing arcs when the side-inputs assume static values.

The constraint clauses for the entire circuit are deduced in a pre–processing step and are stored in the database. This information is then retrieved for the verification of individual crosstalk constellations in the actual procedure.

6. TIMING SOLVER

The timing solver evaluates the logically valid solutions provided by the logic solver for their temporal validity. As these solutions were proven to sensitize the crosstalk scenario under the unbounded delay assumption, it is now required to verify whether they hold also under the more realistic min–max delay assumption.

The timing solver uses the min–max delays of the timing arcs and nets obtained from our propriatory *Static Timing Analysis* (STA) tool. It performs an STA kind of propagation of timing windows from inputs towards the outputs in a topological order [6]. However, it differs from STA in that it checks the logical and temporal validity of timing arcs for their consideration in the calculation of timing windows, making them more precise. These timing windows are called 'logical timing windows' as they consider the inherent logic of the circuit and are conditional to an input switching vector, which in turn is obtained from the logic solver. If the timing windows of the victim and its aggressors overlap, then the configuration is considered as valid. Otherwise, the logic solver is asked to propose a new solution.

In what follows, the step-by-step process of the temporal validity check performed by the timing solver is explained.

6.1 Preprocessing steps

Identifying the fan-in cone: The coupled fan-in cone [13] of the crosstalk cluster is identified to obtain the fan-in timing graph.

Ordering of the vertices of the timing graph: The vertices of the fan-in timing graph are ordered topologically and then processed in this order.

Identification of the input switching vector: The input switching vector specific to the solution provided by the logic solver can be identified by checking the values on the Boolean variables corresponding to the timing arcs connected to the inputs of the fan-in timing graph.

[1]Throughout this paper, the term 'input' is used to refer to a primary input or a sequential cell output. As we consider combinational logic clouds, sequential cell outputs also serve as inputs to these clouds. The similar argument holds also for the term 'output'.

6.2 Coloring of vertices

TABLA uses a vertex coloring mechanism to avoid redundancies in calculation of timing windows. A vertex can be either red or green. A red vertex indicates that it is ready for processing, and a green vertex indicates that the information on it is up-to-date. Initially, all the vertices of the fan-in graph are colored red. An exception to the coloring mechanism holds for the input vertices. These vertices are always processed and are used to identify if the subsequent vertices connected to them need to be processed. The coloring process for each type of vertex is explained in 6.3.

6.3 Propagation of timing windows

Processing of a vertex involves the update of its attributes, i.e., the timing windows and edge literal information corresponding to it. The processing of any vertex depends on its type. After any vertex is processed it is colored green. The procedure to process different kinds of vertices is explained below.

Input vertices: An input vertex is processed irrespective of its color. As a first step, the type of transition that the corresponding input node assumes is obtained from the input switching vector. Depending on the type of transition, the timing window and edge literal information are updated. If the current values are different from the previous ones, then the out-edges of this vertex are colored red to indicate that they have to be updated.

Net receiver vertices: A net receiver vertex is processed only if it is red colored. The timing window on this vertex is obtained by delaying the timing window on its immediate predecessor by the min–max delay possible on the edge connecting them. Let the timing window on the immediate predecessor be $(t_{pre_{min}}, t_{pre_{max}})$ and the delay on the connecting edge be (d_{min}, d_{max}). Then, the timing window on this vertex is calculated as:

$$(t_{min}, t_{max}) = (t_{pre_{min}} + d_{min}, t_{pre_{max}} + d_{max}) \tag{1}$$

The edge literal of a net receiver vertex is set to the edge literal value of its immediately preceding vertex. After this vertex is processed, the vertices connected to its out-edges are colored red to indicate that they have to be updated.

Timing arc receiver vertices: A timing arc receiver is processed only if it is red colored. The timing arc attached to the timing arc receiver vertex is checked if it can be active as explained in Section 6.4. If it can be active, then the timing window of this vertex is also calculated using Equation (1). If the timing arc is temporally inactive but logically valid as per the SAT model from the logic solver, then a conflict clause is added to the SAT instance to block this and other similar scenarios. A conflict clause is a clause added to the logic solver to block a solution or a set of solutions of the SAT instance. A conflict that invalidates a timing arc to propagate timing windows is called a propagation conflict. The procedure to deduce corresponding conflict clauses is explained in Section 6.6.1.

The edge literal value of a timing arc receiver vertex is determined by the timing arc output edge. After this vertex is processed, the vertices connected to its out-edges are colored red to indicate that they need to be updated.

6.4 Checking if a timing arc can be active

Before any timing window is propagated through a timing arc, it is checked whether the timing arc can be active. This is done by verifying if the timing window to be propagated has a non-zero width and by checking the satisfiability of the side-input condition of the timing arc with the accurate timing information available. A timing arc is identified to be inactive if at least one of the following conditions is met:

1. The timing window to be propagated through the timing arc has zero-width, i.e., there is no event to be propagated.
2. The switching times of the side-inputs of the timing arc input are such that the side-input condition of the timing arc is violated. This can happen if a side-input can be proven to assume a controlling value during the entire period in which the timing arc input can switch.

Procedure 1 BACK_TRACK: Algorithm to back-track the fan-in cone to find the reason for conflict

Input: The set of conflicting vertices, V_c;
Output: A SAT clause, C to identify the reason for the conflict

```
1:  for each vertex $v$ of $V_c$ do
2:      if vertex $v$ is already processed then
3:          continue
4:      if vertex $v$ is an input vertex then
5:          Let $e_{tarc}$ be the timing arc variable connected to $v$
6:          if $e_{tarc}$ is active then
7:              Add literal $\overline{e_{tarc}}$ to $C$
8:          else
9:              Add literal $e_{tarc}$ to $C$
10:     else
11:         for each in-edge $e$ of vertex $v$ do
12:             if $e$ is a net-branch edge then
13:                 Vertex set $V = \{v\}$
14:                 $C = C \cup$ BACK_TRACK($V$)
15:             else
16:                 if $e$ is active then
17:                     $C = C \cup$ BACK_TRACK($V$)
18:                 else
19:                     Add literal $e$ to $C$
20:     Mark vertex $v$ as processed
21: return $C$
```

6.5 Checking the timing window overlap

After the propagation of the input switching vector towards the victim and aggressor nets, it is verified if the logical timing windows calculated overlap in such a way that the crosstalk scenario under consideration is sensitized. If the configuration is invalid, then a conflict clause is deduced as explained in Section 6.6. The result and the conflict clause(s), if any, are given as feed-back to the logic solver.

6.6 Deduction of conflict clauses

Deduction of efficient and small conflict clauses is the key to improve the speed of the logic solver, and in our case, the speed of the whole false noise analysis procedure itself. The easiest way to include a conflict clause is to exclude the input combination of the proposed solution. However, such a conflict clause is, in general, too long and therefore not powerful. A powerful conflict clause should exclude as many invalid switching scenarios as possible, i.e., it should be as short as possible.

The first step in deducing a conflict clause is to identify the location of the conflict, i.e., to find the set of conflicting vertices. This is followed by the process to identify the reason for the conflict by back-tracking from the conflicting vertices to the input vertices. This procedure is explained step-by-step in the pseudocode listed in Procedure 1.

The aim of Procedure 1 is to deduce a conflict clause with as few literals as possible. Lines 16 to 19 of the procedure are of particular significance here as they add the literal of an inactive timing arc to the conflict clause without further back-tracking, which excludes possibly a large number of input literals from the conflict clause.

There are two types of conflicts: propagation and top-level conflicts.

6.6.1 Propagation conflicts

A propagation conflict occurs if a timing window cannot be propagated through a timing arc that is active according to the logically valid SAT model being verified. There are two types of propagation conflicts: side-input conflicts and zero-width conflicts.

- A **side-input conflict** occurs if any timing arc of the circuit that is active as per the logically valid solution cannot actually be active because one of its side-inputs does not satisfy the side-input condition after the consideration of timing windows. Deduction of a side-input conflict clause involves identification of the conflicting vertices and then using the back-tracking algorithm listed in Procedure 1.

- A **zero-width conflict** should ensure that at least one of the immediately preceding timing arcs has to be active, so that the calculated timing window has a non-zero width. However, this is already covered by the timing arc constraints (see Section 5.2), and hence no action is required.

6.6.2 Top level conflicts

A top level conflict occurs when the timing windows of the inputs corresponding to the input switching vector, when propagated forward towards the crosstalk cluster (the victim and its aggressor nodes), do not overlap in such a way that the crosstalk scenario under test is sensitized. There can be several top-level conflicts for each temporally invalid configuration depending on the number of aggressor/victim pairs whose switching windows do not overlap. A top level conflict on an aggressor-victim pair indicates to the logic solver that the aggressor cannot induce crosstalk on the victim for the crosstalk type under consideration. Similarly, a top level conflict on an aggressor-aggressor pair blocks them from simultaneously inducing crosstalk on the victim.

7. IMPLEMENTATION AND RESULTS

TABLA has been implemented in C++ in a false noise analysis framework within our STA reference tool as a prototype 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 it is a common practice to perform false noise analysis as a post-processing step to reduce the number of violations reported by STA [4, 5, 13, 11], 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 and on an industrial design, both implemented in a 90nm technology. The experiments were conducted single-threaded on a Sun Sparc Solaris 10 workstation with 64GB RAM, 16 CPUs of 1.84GHz each and a CPU factor of 84. 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 often done by commercial STA tools. Such simplifications are possible and would have a positive impact on speed and solvability of the approach. As we did not use any approximate heuristics that trade–off accuracy and speed, as proposed in [4], a further improvement in speed can be achieved with their application.

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 scenarios. The circuits were tested using a coupling capacitance based weight factor similar to [10] to estimate the impact of an aggressor on its victim. The validity of using a linear model for estimating crosstalk is justified experimentally in [4].

Table 1 shows the results of TABLA on several ISCAS89 benchmarks. Column 2 gives maximum and average number of aggressors, Columns 3 and 4 give the average pessimism reduction in terms of the number of aggressors considering standard STA window overlap (SWO) and TABLA, respectively. For SWO based filtering, all aggressors with their switching windows (arrival time windows expanded on either sides by 50% of the transition time) not overlapping with the switching window of the victim are considered invalid.

Column 5 of Table 1 shows the improvement in aggressor count reduction achieved by TABLA with respect to SWO. Columns 6–8 give the number of nets processed versus those unsolved when a cut–off limit of 10 seconds per net was used. Finally, Columns 9–11 give the total CPU time to process all nets with the use of large conflict clauses vs. powerful conflict clauses (see Section 6.6), and the run time improvement achieved by using powerful conflict deduction mechanism. The CPU times shown are rounded of to two decimal places, while the exact values are used to calculate the improvements shown in Column 11.

The use of powerful conflict clauses, in general, provides improvement also in terms of solvability and pessimism reduction because of the cut–off time. This is a direct consequence of the fact that TABLA can solve more nets and reduce more pessimism in the given time, with the use of powerful conflict clauses. Because of space constraints, these results are not provided. The solvability and pessimism reduction values provided in Table 1 correspond to TABLA using the powerful conflict clause generation mechanism.

To demonstrate the benefit of TABLA in terms of crosstalk delay, we have also run it on the 15 most critical nets of an industrial design with about 65K standard cells. We used an L–BFGS–B[14] based optimization tool to accurately align the aggressors and calculate the crosstalk delay on these nets without any false noise filtering, with aggressor filtering based on SWO and with TABLA. For these experiments unbounded optimization was used, i.e., aggressor alignment was not restricted by timing windows. Our optimization tool employs analogue simulation to compute crosstalk delay values and derivatives w.r.t. aggressor shifts as inputs for the L–BFGS–B routine. This leads to highly accurate results at the cost of extreme run times. Thus, the number of nets to be compared was limited by our optimization tool and not by TABLA.

Table 2 shows the results of TABLA on 15 nets of the industrial design that gain a maximum benefit from false noise reduction. It highlights the significance of false noise analysis in reduction of crosstalk pessimism for nets that are worst affected by crosstalk. Columns 2–4 give the number of aggressors considered by the different approaches, Columns 5–7 show the corresponding crosstalk delays and Columns 8 and 9 the improvement that TABLA achieved over SWO in picoseconds and percentage, respectively. A significant additional amount of crosstalk is filtered by TABLA as compared to SWO.

The improvement of TABLA in reduction of crosstalk pessimism achieved over SWO shown in Column 5 of Table 1 gives the average improvements of each circuit. It can be understood from Column 9 of Table 2 that the actual improvement for certain nets can be as large as 100%. Such an improvement plays a crucial role in fixing violations on critical paths.

These results clearly indicate the significant benefit in terms of timing slack to be gained by the reduction of false noise. Many state-of-the-art temporal logic analysis methods [13, 11] resort to simplifications such as coarse time slices, interval merging or aggressor lumping to achieve solvability whereas TABLA solves most of the nets with a 10s cut–off limit without such simplifications indicating its efficiency.

8. CONCLUSIONS

In this paper, we propose a novel technique called TABLA to perform temporal logic analysis of a circuit to reduce false noise. The technique uses the accurate min–max timing model and handles glitches implicitly due to its timing arc based approach. Because of its conflict driven approach as well as the tight coupling of two domain specific solvers, the false noise problem is handled very efficiently. The method is integrated into the false noise analysis framework of our STA tool and tested on several ISCAS89 benchmarks as well as on an industrial design. The proposed technique is demonstrated to achieve great benefit in terms of reduction of crosstalk pessimism. The generation of better conflict clauses and the use of circuit information in the SAT solver are future research topics expected to yield further significant speed improvements.

9. REFERENCES

[1] M. Becer, V. Zolotov, R. Panda, A. Grinshpon, I. Algol, R. Levy, and C. Oh. Pessimism reduction in crosstalk noise aware STA. In *IEEE/ACM Int'l Conf. on Computer Aided Design*, pages 954–961, 2005.

Table 1: Results of TABLA on various ISCAS89 benchmarks

Circuit	# Pot. Aggrs.	Avg. Aggr. Count Reduction (%)			Solvable nets			Total CPU time (in mins)		
	(max/avg)	SWO	TABLA	Improvement	# processed	unsolved nets		Large conflicts	Powerful conflicts	% Improvement
						#	%			
s208.1	27/6	24.21	33.89	9.68	72	0	0.00	0.02	0.01	61.64
s298	37/6	31.18	42.98	11.80	92	0	0.00	0.02	0.02	32.09
s344	26/7	37.60	47.30	9.70	122	1	0.82	0.40	0.17	57.15
s349	33/7	39.10	48.60	9.50	101	0	0.00	0.32	0.16	48.70
s382	49/6	39.74	46.38	6.65	116	1	0.86	0.48	0.30	38.53
s386	42/8	33.80	43.08	9.28	111	0	0.00	0.05	0.03	37.37
s400	40/6	35.01	45.72	10.71	154	1	0.65	0.63	0.26	58.40
s420.1	43/6	31.43	40.26	8.83	124	0	0.00	1.75	0.28	84.05
s444	62/9	30.63	42.20	11.57	127	1	0.79	1.20	0.27	77.62
s499	51/7	39.92	52.96	13.04	161	20	12.42	13.70	4.78	65.09
s510	67/5	30.77	45.39	14.62	181	0	0.00	1.41	0.11	92.06
s526	36/7	28.17	40.02	11.84	160	0	0.00	1.57	0.23	85.59
s526n	60/9	30.28	42.57	12.29	167	0	0.00	2.01	0.24	88.21
s635	64/12	32.65	44.28	11.63	158	3	1.90	7.75	1.53	80.22
s641	42/8	40.45	43.84	3.39	163	9	5.52	8.64	2.36	72.65
s713	52/9	33.24	39.65	6.41	155	3	1.94	4.82	0.85	82.37
s820	63/9	29.50	47.00	17.50	239	1	0.42	4.19	0.65	84.59
s832	76/7	29.89	43.50	13.61	272	2	0.74	4.46	0.72	83.92
s838.1	75/7	38.64	45.73	7.09	246	11	4.47	7.27	2.03	72.06
s938	64/9	37.58	45.58	8.00	242	18	7.44	8.24	2.89	64.87
s991	60/9	31.74	35.19	3.45	294	3	1.02	5.86	1.00	82.94
s1196	85/11	32.40	42.94	10.54	377	23	6.10	18.72	5.68	69.67
s1238	84/10	29.94	41.33	11.39	352	35	9.94	18.70	7.20	61.49
s1494	123/14	31.17	44.63	13.46	431	24	5.57	7.65	5.11	33.14
s3271	72/7	30.76	38.81	8.05	771	19	2.46	11.53	5.08	55.90
s4863	126/12	39.36	42.45	3.09	983	47	4.78	48.41	10.72	77.85
s5378	91/11	43.28	48.43	5.15	976	47	4.82	65.31	10.02	84.66
s6669	88/8	37.04	41.82	4.77	1245	81	6.51	57.88	15.89	72.55
s9234.1	96/9	45.59	49.58	3.99	909	58	6.38	52.45	13.80	73.68
s13207.1	204/12	44.71	50.23	5.52	2307	181	7.85	121.58	40.48	66.70
s15850.1	201/14	51.34	56.01	4.67	2412	140	5.80	136.54	33.08	75.77
s38417	335/15	45.00	47.76	2.76	6744	681	10.10	503.00	153.98	69.39
s38584.1	226/11	41.51	45.90	4.39	5570	612	10.99	468.00	146.74	68.65
Averages		35.69	44.42	8.74	–	–	3.64	–	–	68.47

Table 2: Results of TABLA on selected critical nets of a 90nm industrial design.

Net #	Aggr. count			Crosstalk delay (ps)			Improvement	
	unfiltered	SWO	TABLA	unfiltered	SWO	TABLA	in ps	%
1	5	5	0	91	91	0	91	100
2	17	17	0	96	96	0	96	100
3	7	7	1	116	116	40	76	65
4	9	9	1	144	144	40	105	73
5	13	13	2	145	145	59	85	59
6	9	4	2	148	92	8	84	57
7	9	4	2	154	95	9	87	56
8	9	7	1	192	191	7	184	96
9	9	7	1	203	202	7	195	96
10	16	10	1	251	224	115	109	44
11	16	10	1	254	227	116	111	44
12	12	11	2	258	228	138	89	35
13	16	9	1	264	130	52	78	29
14	16	9	1	270	131	53	78	29
15	16	9	1	270	131	53	78	29

[2] P. Chen and K. Keutzer. Towards true crosstalk noise analysis. In *IEEE/ACM Int'l Conf. on Computer Aided Design*, pages 132–138, 1999.

[3] N. Eén and N. Sörensson. An extensible SAT solver. In *Int'l Conf. on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 502–518, 2004.

[4] A. Glebov, S. Gavrilov, R. Soloviev, V. Zolotov, M. R. Becer, C. Oh, and R. Panda. Delay noise pessimism reduction by logic correlations. In *IEEE/ACM Int'l Conf. on Computer Aided Design*, pages 160–167, 2004.

[5] A. Glebov, S. Gavrilov, V. Zolotov, C. Oh, R. Panda, and M. Becer. False-noise analysis for domino circuits. In *Design, Automation and Test in Europe*, pages 784–789, 2004.

[6] R. B. Hitchcock, Sr. Timing verification and the timing analysis program. In *Design Automation Conf.*, pages 594–604, 1982.

[7] R. Li, A. Shey, and M. Laudes. Incorporating logic exclusivity (LE) constraints in noise analysis using gain guided backtracking method. In *IEEE/ACM Int'l Conf. on Computer Aided Design*, pages 783–789, 2008.

[8] M. K. Michael and S. Tragoudas. Generation of hazard identification functions. In *Int'l Symp. on Quality Electronic Design*, pages 419–424, 2003.

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

[10] M. Palla, J. Bargfrede, K. Koch, W. Anheier, and R. Drechsler. Adaptive branch and bound using SAT to estimate false crosstalk. In *Int'l Symp. on Quality Electronic Design*, pages 508–513, 2008.

[11] Y. Ran, A. Kondratyev, K. Tseng, Y. Watanabe, and M. Marek-Sadowska. Eliminating false positives in crosstalk noise analysis. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 24(9):1406–1419, 2005.

[12] R. Tayade and J. A. Abraham. Critical path selection for delay test considering coupling noise. In *IEEE European Test Symp.*, pages 119–124, 2008.

[13] K. Tseng and M. Horowitz. False coupling exploration in timing analysis. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 24(11):1795–1805, 2005.

[14] C. Zhu, R. Byrd, P. Lu, and J. Nocedal. L–BFGS–B: A limited memory FORTRAN code for solving bound constrained optimization problems. In *Tech. Report, NAM-11, EECS Department, Northwestern University*, 1995.