Delay Noise Pessimism Reduction by Logic Correlations

A. Glebov\(^1\), S. Gavrilo\(v\)\(^1\), R. Soloviev\(^1\), V. Zolotov\(^2\), M. R. Becer\(^3\), C. Oh\(^3\), R. Panda\(^3\)

\(^1\)Microstyle, Moscow, Russia,
\(^2\)IBM T.J. Watson Research Center, Yorktown Heights, NY,
\(^3\)Freescale Semiconductor, Austin, Texas,

Abstract

High-performance digital circuits are facing increasingly severe signal integrity problems due to crosstalk noise and therefore the state-of-the-art static timing analysis (STA) methods consider crosstalk-induced delay variation. Current noise-aware STA methods compute noise-induced delay uncertainty for each net independently and annotate appropriate delay changes of nets onto data paths and associated clock paths to determine timing violations. Since delay changes in individual nets contribute cumulatively to delay changes of paths, even small amounts of pessimism in noise computation of nets can add up to produce large timing violations for paths, which may be unrealistic. Unlike glitch noise analysis where noise often attenuates during propagation, quality of delay noise analysis is severely affected by any pessimism in noise estimation and can unnecessarily cost valuable silicon and design resources for fixing unreal violations. In this paper, we propose a method to reduce pessimism in noise-aware STA by considering signal correlations of all nets associated with an entire timing path simultaneously, in a path-based approach. We first present an exact algorithm based on the branch-and-bound technique and then extend it with several heuristic techniques so that very large industrial designs can be analyzed efficiently. These techniques, which are implemented in an industrial crosstalk noise analysis tool, show as much as 75% reduction in the computed path delay variations.

1 Introduction

Due to process scaling, cross-coupling capacitance has become a dominant portion of total interconnect capacitance. As a result, delay variation due to coupling noise injected by switching neighboring nets, called delay noise [1], has become a significant component of the total interconnect delay. The net on which noise is injected is referred to as a victim net, and the nets which inject noise are called aggressor nets. The collection of a victim net and all its aggressor nets is called a noise cluster. The switching of an aggressor net may either speed-up or slow-down the victim net transition depending on the switching direction. For reliable chip design, it is necessary to predict the worst delay variation due to cross coupling noise and take it into account during timing verification.

Delay noise depends greatly on how many aggressor nets switch and how their transitions are aligned among themselves and with respect to the victim’s transition. An infinite number of switching scenarios is possible, depending on input signal arrivals, process variation, environment parameters, and more importantly on the logical operation of the circuit. Due to this, predicting the exact worst case noise is extremely difficult. Therefore, noise analysis tools compute a conservative estimate of possible delay variation. Typically, it is assumed that all aggressor nets switch in the same direction at the worst alignment time [2], [2]. However, many switching scenarios are prohibited in reality due to timing and logical relations between the victim and aggressor signals. More often than one would assume, the worst case switching scenario does not occur due to such signal correlations, thus rendering the delay variation estimated by naive noise analysis approaches very pessimistic. This pessimism, also called false noise, results in false functional and timing violations. A false functional violation can occur due to an overestimation in the magnitude of crosstalk noise and, likewise, false or excessive timing violation can occur due to an overestimated noise situation resulting in an overestimation in victim net’s delay variation. False noise sounds false alarm for the chip designer or integration engineer and wastes precious design and silicon resources, which are spent for fixing non-existing problems.

Timing and logic correlations have been very successfully used to reduce pessimism in functional noise analysis, where a noise glitch is responsible for flipping the logic state of a quiet victim net. Timing correlations are computed by propagating the switching windows obtained through static timing analysis [1], [4], [3]. This technique has been shown to be relatively simple and effective. However, this approach does not identify situations where all aggressor nets cannot switch at the same time in the same direction due to logical constraints in the circuit. In order to eliminate all false noise failures, both timing and logic correlations of the signals must be taken into account. In [4], this problem was represented as a search for a worst-case 2-vector test using a Boolean Constraint Optimization formulation. A test pattern generation approach was proposed in [3]. Due to very high computational complexity, these methods are not suitable for false noise analysis of large size designs. In [5], an approach based on simple logic implications (SLI) [6] was developed. An SLI expresses a logic relationship between a pair of signals. Initially SLIs are generated for logic gates and then more implications are derived by forward and backward propagation of available SLIs. [7] proposed to use the resolution method, originally used for mechanical theorem proving[8]. The resolution method of [7] was shown to have advantage over [5] as it can model logic correlations between multiple signals and can extract logic correlations from transistor level circuit description without an explicit logic function extraction.

The above mentioned logic correlation techniques have targeted mainly functional noise analysis, though pessimism is a more serious problem for delay noise than for functional noise. In case of functional noise, a noise glitch generated at some stage is often attenuated and filtered out during propagation through stages of logic gates due to their low pass filtering property. Thus the effect of pessimism is greatly weakened in glitch noise analysis. On the other hand, pessimism shows up strongly in delay noise due to the additive contribution of stage delays to path delays. The idea of extending logic correlation techniques to delay noise analysis may seem simple since the process of deriving logic constraints is the same. However, as we will show later, the difficulty lies in the application of such logical information in delay noise analysis. In delay noise analysis, logical constraints need to be considered among a much larger set of signals than is necessary for glitch analysis. Also, a maximal set of aggressors needs to be selected such that the effect of noise is maximized globally and conserva-
tively over several signal stages of a timing path. Since an aggressor can interact with multiple victim nets on a timing path and each interaction can contribute different amount of delay change relative to other aggressors, the problem of finding such an aggressor set becomes difficult. Based on a key observation about the relationship between noise pulse magnitudes and induced delays, we will construct a linear model for path delays based on noise amplitudes so that the problem can be optimized very efficiently. With this linear model, the problem is formulated as a maximum weight independent set for hypergraphs.

A maximum realizable aggressor set (MRAS) is a set of aggressors of a net which can switch simultaneously and inject a combined noise on the victim without violating the circuit’s inherent logic constraints. It was shown that computing MRAS is an NP complete problem[5]. Despite the exponential complexity, enumerative traversal of the ROBDD of constraints was still usable for glitch noise analysis, as each noise cluster was analyzed separately and a typical noise cluster consisted only about 10 aggressor nets[7]. However, in order to achieve maximum possible pessimism reduction in delay noise analysis, it is necessary to consider all victim nets of an analyzed path and all their aggressor nets together. As a result, we need to compute MRAS from about 100 or more nets depending on the length of the timing path. Thus more sophisticated techniques than direct enumeration are necessary. In this paper, we provide a branch-and-bound, as well as, several heuristic techniques to address this need.

Without loss of generality, we discuss only delay increase. Application of the method for delay decrease is similar. Also, for simplicity of discussion, we do not consider complications arising due to the same net acting both as victim and aggressor. The proposed techniques were implemented in an in-house industrial noise analysis tool[2] and benchmarked using several high performance microprocessor circuit modules and ISCAS benchmark circuits. The current implementation of logic constraints generation uses a zero-delay assumption which is conservative only for glitch-free circuits, obtained, for instance, through special transistor sizing methods. However, the proposed technique of computing maximum realizable aggressor set is independent of how logic constraints are generated and can be applied to logic constraints generated by any other technique, including user-defined ones. Our implementation can process circuits described both at transistor and gate level, making it convenient to handle custom and compiled blocks. The benchmarks show that the proposed technique is very efficient and helps to cut down as much as 100% of timing violations reported by traditional delay noise analysis.

The remainder of this paper is organized as follows. Section 2 discusses our delay noise model, shows the range of its applicability and formulates the problem of delay noise pessimism reduction. Section 3 briefly explains the concept of logic constraints and their generation by various techniques. Section 4 presents and compares several new algorithms of computing maximum realizable aggressor sets for large clusters. Section 5 shows experimental results and reports the efficiency of different false noise analysis techniques. Section 6 contains our concluding remarks.

2 Delay noise pessimism reduction

2.1 Delay noise model

Figure 1 shows a simple noise cluster and signal waveforms with noise-induced delay degradation. Aggressor nets inject a combined noise pulse into the victim net. This pulse combines with non-noisy victim transition creating the noisy signal which is delayed with reference to the non-noisy victim transition. If the injected pulse is large and narrow in comparison to non-noisy victim transition, the noisy victim waveform may cross the delay measurement point (which is 50% of supply voltage, usually) multiple times, causing problems for delay definition and modeling. On the other hand, the noisy signal at the output of victim receiver gate is usually monotonic due to the low pass filtering properties of logic gates. Therefore, we measure signal delay variation at the output of the victim receiver gate.

Generally, a victim net has many aggressors injecting noise pulses. In noise analysis, it is common to use linear models of victim and aggressor drivers in order to compute the combined noise pulse using superposition. To obtain a conservative estimate, all noise pulses are aligned at their peaks as shown in Figure 2.

Figure 1. Crosstalk noise injection

Figure 2. Combined noise pulse waveform

In a general case, the signal delay variation $\Delta D$ due to noise is a non-linear function of noise pulse parameters: height $h_n$, width $w_n$, etc. However, for small noise values, this function can be approximated by a linear expression:

$$\Delta D = \frac{\partial D}{\partial h_n} \cdot \Delta h_n + \frac{\partial D}{\partial w_n} \cdot \Delta w_n + \ldots$$  

(EQ 1)

At the noise-aware STA stage, one usually deals with small noise pulses because any large noise pulse must have been analyzed and fixed irrespective of its impact on timing in order to prevent any functional problem. Thus the above linear approximation is sufficiently accurate for delay noise analysis. Linear victim and aggressor driver models together with the above linear approxima-
tion of delay variation allow us to use superposition principle to compute the combined effect of noise injection by several aggressors. This is crucial for efficient computation of a realistic global set of aggressors that will maximize a victim path delay. The total delay variation $\Delta D$ can be represented as a sum of delay variations $\Delta D_i$ contributed by each aggressor net:

$$\Delta D = \sum_i \Delta D_i$$

(EQ 2)

For efficiency, we first accurately compute total delay variation due to all aggressors of a victim net, and also the magnitude of individual noise pulses induced by each aggressor. We then estimate each aggressor’s individual contribution to the net’s delay variation based on their noise height ($h_i$) proportions as below:

1. Compute noise pulse height $h_i$ of each aggressor $a_i$.
2. Compute total delay variation $\Delta D$ due to the combined noise pulse.
3. Estimate delay variation due to each aggressor using:

$$\Delta D_i = h_i \cdot \frac{\Delta D}{\sum_j h_j}$$

(EQ 3)

2.2 Delay noise analysis

Correct operation of a circuit depends on delays of signal propagation from one memory element to another through combinational logic. Too long (short) signal propagation delay results in setup (hold) timing violations. Signal propagation delay is different for different signal paths and can be computed by summing up delays of stages making up the path. Usually a circuit is analyzed as a whole with a static timing analysis tool which computes the circuit delay and a set of paths with timing violations. Noise is injected on each net on a signal propagation path by its aggressor nets as shown in Figure 3. The injected noise pulses affect the stage delays and thus the entire path’s delay. Noise-aware static timing analysis consists of the following steps:

1. Compute net delays in the presence of crosstalk noise.
2. Annotate circuit nets with computed delays.
3. Perform timing analysis.

Total path delay variation is the sum of delay variations of each net (Figure 3) on the path. Assuming linear approximation of delay variation due to crosstalk noise and using formula (EQ2), path delay variation $\Delta D_p$ can be expressed as a sum of delay variation for each net, $i$, on the path due to each of its aggressor nets, $A_i$:

$$\Delta D_p = \sum_i \sum_j \Delta D_{i,j}$$

(EQ 4)

where $\Delta D_{i,j}$ is delay variation of net $i$ on path $P$ due to noise injected by aggressor $j$. Figure 4 shows the error due to this linear approximation by comparing the estimated $\Delta D_{i,j}$ calculated by EQ 3 against the actual $\Delta D_{i,j}$ when only that aggressor switches. Note that this error decreases as the number of switching aggressors approaches the actual number of aggressors used to calculate $\Delta D$. Even in worst case error situation, as when estimating the delay noise contribution of a single aggressor based on delay noise due to all aggressors, the average error in the estimated delay noise is not more than 15%. Based on our experience, total delay noise on a net due to all aggressors is up to 40% of non-noisy net delay. Thus our approximation will have only less than 6% error, which is acceptable for use as a fast estimate in our pessimism reduction algorithms.

2.3 Pessimism reduction

Reduction of pessimism in noise analysis relies on the fact that not all signal combinations and transitions are always possible in a circuit. During signal propagation through path $P$, each net $v_i$ of this path, considered a victim, transitions from its initial state $x_i^1$ to its final state $x_i^F$. We define a signal propagation path $P$ as a set of nets and their transitions. Thus, two paths consisting of same nets but having different transitions (i.e. one rising and one falling) will be considered as separate paths. The maximum realizable aggressor set (MRAS) is a set of aggressor nets which can switch simultaneously without violating any logic correlations and inject noise into the victims on $P$ so that maximum path delay variation results. We can reduce the problem of finding the maximum weight independent set (MWIS) in a hypergraph to the problem of finding MRAS in linear time by mapping the hypergraph vertices to aggressor nets, mapping the hypergraph edges to logic constraints and assigning weights equal to the total noise delay contribution of each aggressor to the path. Thus, the optimization problem is at least as complex as finding MWIS in an hypergraph.

Consider a signal propagation path $P$ consisting of victim nets $v_1$, $v_2$, ..., and each $v_i$ with a set of aggressors $A_i=[a_{i,1}, a_{i,2}, ...]$. During signal propagation, $v_i$ transitions from $x_i^1$ to $x_i^F$. Aggressors
### 3.2 Sources of signal correlations

There are two sources of logic correlations in a digital circuit: correlations between input signals and correlations due to circuit structure itself. The first type is related to coding of input signals, for example, one hot (one cold) coding where only one signal from a given group is high (low) at any time. Such constraints need to be specified by a circuit designer. The second type follows from the logic functionality of the circuit. For instance, a NAND gate implementing \( x = \overline{a \cdot b} \) has three logic constraints: \( \overline{a} \cdot \overline{b} \), \( \overline{a} \cdot b \), \( a \cdot \overline{b} \). It means that the output cannot be \( 0 \) if any of the input signals is \( 0 \) and it cannot be \( 1 \) if all inputs are \( 1 \).

If a circuit is represented at transistor level, it is often difficult to convert it to a logic level representation. In such cases, it is more convenient to build logic correlations for the entire circuit from certain basic signal constraint governing the operation of MOS transistors[7]. They are as follows:

- nMOS transistor (\( s \)-source, \( g \)-gate, \( d \)-drain) has two constraints:
  - \( g \cdot s \cdot d \), \( s \cdot \overline{g} \cdot \overline{d} \)
  - \( s \cdot \overline{g} \cdot \overline{d} \), \( \overline{s} \cdot g \cdot \overline{d} \)

These constraints are consequences of the MOS transistors’ operation in their on state. For example, constraint \( g \cdot s \cdot d \) for an nMOS means that if its gate is high, it is impossible to keep its source at high while keeping the drain low.

### 3.3 Derivation of logic constraints

Using an initial set of logic constraints (for transistors or gates), we apply logic calculus to derive more signal correlations in the circuit. Derivation of new logic relations from the existing ones is one well studied problem in mechanical theorem proving [8]. One of
the efficient techniques for this is the resolution method. It is based on recurring application of the resolution rule:

$$a \cdot B = 0, \bar{a} \cdot C = 0 \rightarrow B \cdot C = 0$$  \hspace{1cm} (EQ 6)

where $B$ and $C$ are arbitrary logic expressions.

For false noise analysis, the resolution rule is applied to the terms of (EQ6) which specify logic constraints. Therefore $B$ and $C$ are Boolean products of circuit signals or their negation. Unlike the classical resolution method that derives new true sentences from known true sentences, our formulation is applied to a set of false sentences (logic constraints or prohibited signal combinations) and derives new false sentences. The resolution rule fully covers all the laws of logic calculus. For brevity, the equality sign and boolean constant $\theta$ are omitted from the resolution rule:

$$a \cdot B, a \cdot C \rightarrow B \cdot C$$  \hspace{1cm} (EQ 7)

As said before, the resolution rule can be applied both at transistor and logic levels. At transistor level, the resolution rule is applied to signals of each DC-connected component (DCCC) separately. The derivation starts from a set of constraints for individual transistors and continues recurrently adding new logic constraints constructed from pairs of existing ones[7]. Figure 7 demonstrates the derivation of logic constraints for a 2-input NAND gate.

![Figure 7. DCCC logic constraints calculation by resolution](image)

Derivation of logic constraints at logic level starts from the constraints obtained for DCCCs. Since logic circuits of real digital blocks are usually very large, uncontrolled application of the resolution rule in all possible ways results in wastage of time and effort for generation of the same constraints multiple times. We use a heuristic approach described in [7] to generate new logic constraints by combining existing ones. Constraints are propagated backward and forward through logic gates where they are combined with signal correlations of those gates by the resolution rule. Figure 6 illustrates generation of initial and derived constraints.

4 Computation of worst possible delay noise

4.1 Maximum realizable aggressor set

Computation of the worst possible noise requires finding the maximum realizable aggressor set. Consider a signal propagation path $P$ consisting of victim nets $\{v_1, v_2, \ldots\}$ and a set of aggressor nets $A = \bigcup A_i$ where $A_i$ is a set of aggressors injecting noise on $v_i$. Assume that each victim net $v_i$ transitions from $x_i^I$ to $x_i^F$ and each aggressor $a_j$ transitions from $y_i^I$ to $y_i^F$. Let the vectors of victim net’s initial and final states be $X^I$ and $X^F$ and that of aggressors be $Y^I = \{y_1^I, y_2^I, \ldots, y_j^I, \ldots\}$ and $Y^F = \{y_1^F, y_2^F, \ldots, y_j^F, \ldots\}$. Given a set of logic constraints $C$ for the circuit, we can obtain two sets of logic constraints $C^I$ and $C^F$ by substituting initial and final state values $X^I$ and $X^F$ for the victim nets in $C$. The problem of computing maximum possible delay variation (delay noise) $\Delta D_{\text{max}}$ and the corresponding maximum aggressor set $A_R$ can now be formulated as an optimization problem:

$$\text{Compute } \Delta D_{\text{max}} = \max \sum_{a_j \in A} \Delta D_{i,j}$$  \hspace{1cm} (EQ 8)

such that vectors $Y^I$ and $Y^F$ of initial and final states of aggressors $A_R \subseteq A$ satisfy logic constraints $C^I$ and $C^F$ respectively.

Constraints $C^I$ and $C^F$ may have terms with one or more variables. Single variable terms can prohibit aggressors from injecting noise, independent of the state of other aggressors. For example, constraint $\bar{a} \in C^I$ prohibits aggressor $a$ from having initial state 0 and consequently from having rising transition $b \rightarrow 1$. 2-variable terms are SLIs which prohibit pairs of aggressors from simultaneously injecting noise into a victim net. If all constraints are only SLIs, we can then build a constraint graph $GC=(A, EC)$ whose set of vertices $A$ is the set of aggressors. Each vertex $a_i$ has a weight equal to the delay noise contribution by aggressor $a_i$. Edge $e = (a_i, a_j) \in E_C$ corresponds to SLI $a_i \land a_j$ which prohibits aggressors $a_i$ and $a_j$ from simultaneous noise injection.

Finding the MWIS for the graph $GC=(A, EC)$ (Figure 5(a)) is NP complete. Therefore the optimization problem with multi-variable constraints which requires finding MWIS for a hypergraph (Figure 5(b)) is no easier.

4.2 Exact solution approach

When the aggressors set is small (about 10 aggressors), all aggressor combinations can be enumerated and checked against the constraints. For this approach, it is important to be able to quickly check the satisfiability of logic constraints. In [7], this problem for functional noise was solved by constructing a characteristic ROBBD describing the logic constraints. This approach, however, is not practical for simultaneous consideration of a large number of victim-aggressor clusters as required in a path-based delay noise analysis. The number of victim and aggressor nets related to a timing path can easily exceed 100. If only SLI constraints are generated, then the algorithm in [9] can be used, but such SLIs very much under-constrain the problem.

For an exact solution of the problem with a general set of constraints (i.e. multi-variable term constraints), we propose a depth-first traversal of a decision tree shown in Figure 8 which is improved by branch-and-bound. The validity of any branching is verified with constraints. At any stage of traversal, unnecessary traversal of a subtree is eliminated using a currently optimal selec-
branch of the decision tree: combinations including aggressor $a_{i+1}$ (step 1) and combinations excluding aggressor $a_{i+1}$ (step 2). The function may be called recursively in each of these 2 steps. Unnecessary branching is avoided by:

- checking in step 1.1 that the current selection is not expandable with $a_{i+1}$ without violating some logic constraints

**Property 2:** If a set of aggressors $A_r$ can inject delay noise $w(A_r)$ and satisfy logic constraints $C$, then the worst realizable aggressor set $A_R$ will inject noise $w(A_R)$ equal to or higher than $w(A_r)$, i.e. $w(A_r) \leq w(A_R)$.

**Property 3:** Let an aggressor set $A$ be divided into two disjoint subsets $A_1$ and $A_2$ s.t. $A = A_1 \cup A_2$, and $A_r \subseteq A_1$ and, $A_r \subseteq A$ are realizable sets. If inequality

$$w(A_r) > w(A_{r+1}) + w(A_2) \tag{EQ 9}$$

is valid, then the MRAS, $A_R$ is not a subset of $A_{r+1} \cup A_2$ ($A_R \nsubseteq A_{r+1} \cup A_2$).

The proposed branch and bound algorithm is described below.

**Algorithm: Branch_and_Bound($A$, $i$, $A_r$, $A_s$)**

**Input:** Set of complete aggressors $A$=$\{a_1, a_2,...\}$, index of last processed aggressor $i$, current selection $A_s$=$\{a_1, a_2,...\}$ which is a partial set of realizable aggressors, and current MRAS $A_r$=$\{a_1, a_2,...\}$ which is the set contributing maximum noise among the partial selections considered thus far.

**Output:** Current MRAS $A_r$=$\{a_{i+1}, a_{i+2},...\}$

1. Set $UNPROCESSED$=$\{a_{i+1}, a_{i+2},..., a_{\mu}\}$. If $UNPROCESSED$ is NOT empty, then:
2.1 If adding $a_{i+1}$ to $A_r$ does not violate the constraints and $w(A_r) + w(UNPROCESSED) > w(A_r)$ then:

- 1.1.1 Set $A_r' = A_r \cup \{a_{i+1}\}$
- 1.1.2 If $w(A_r') > w(A_r)$, update current MRAS: $A_r = A_r'$
- 1.2.3 Try to improve the current MRAS $A_r$:

$$A_r = Branch_and_Bound(A, i+1, A_r', A_r)$$

2. Set $UNPROCESSED$=$\{a_{i+2}, a_{i+3},..., a_{\mu}\}$. If $UNPROCESSED$ is NOT empty, then:
2.2 If $w(A_r) + w(UNPROCESSED) > w(A_r)$, try to improve the current MRAS $A_r$:

$$A_r = Branch_and_Bound(A, i+2, A_r, A_r)$$

3. Return $A_r$

The function $Branch_and_Bound(A, i, A_r, A_s)$ assumes that aggressors $\{a_1, a_2,...,a_i\}$ have been processed by previous calls and processes the reminder $UNPROCESSED$=$\{a_{i+1}, a_{i+2},..., a_{\mu}\}$. It tries to expand a given partial selection $A_s$=$\{a_1, a_2,..., a_i\}$ using more aggressors from the UNPROCESSED set. The function considers two branches of the decision tree: combinations including aggressor $a_{i+1}$ (step 1) and combinations excluding aggressor $a_{i+1}$ (step 2). The function may be called recursively in each of these 2 steps. Unnecessary branching is avoided by:

- checking in steps 1.1 and 2.1 that the current selection $A_s$, even if extended with all unprocessed aggressors, cannot inject noise greater than the current MRAS $A_r$.

The branch and bound procedure is invoked at the root (index 0) of the decision tree: $A_r=Branch_and_Bound(A, 0, {}, {})$. The efficiency of the algorithm can be improved by partitioning the aggressor set $A$ into disjoint subsets $A_k$, such that any logic constraint includes aggressors only from one of the subsets. Then each subset $A_k$ can be processed independently, thus reducing the problem size. Another way to improve efficiency is to make the initial call as $A_r=Branch_and_Bound(A, 0, {}, A_j)$ where $A_j$ is an approximation of MRAS computed by some other fast heuristic approach.

### 4.3 Heuristic approaches

The branch and bound algorithm always finds an MRAS exactly, but in the worst case it has exponential run time. Therefore, it is useful to have fast heuristic algorithms that are able to find a conservative approximate MRAS $A_j$ which produces higher noise than an exact MRAS $A_r$.

One of the approaches is to modify the branch and bound algorithm so that it computes a constantly improving approximation of the MRAS. It uses the computation of an upper bound of the maximum realizable noise $w(A_j)+w(UNPROCESSED)=w(A_j)$ in steps 1.1 and 2.1. At run time, the algorithm updates the best current upper bound and the corresponding aggressor set. If stopped, the algorithm outputs the best approximation of the maximum realizable aggressor set and the delay noise injected by it. One useful criterion for stopping the search is achieving a delay noise estimate such that the total path delay is lower than the required path delay. Another useful criterion is maximum error of the achieved approximation:

$$e = w(A_r) + w(UNPROCESSED) - w(A_j) \tag{EQ 10}$$

Another class of heuristic algorithms can be built using Property 1 of MRAS which allows safely excluding any logic constraints from consideration. We exclude some of the logic constraints so that we can partition the whole aggressor set into subsets $A_k$ such that any logic constraint involves aggressors only from one of these subsets, $A_k$. Then, we find maximum realizable aggressor set for each subset $A_k$ separately. The simplest partitioning considers each noise cluster (victim net $v_i$ and its aggressors $a_{ij}$) separately. The size of noise clusters is sufficiently small to use either branch and bound, or exhaustive enumeration of aggressor combinations. A better approximation can be achieved by more sophisticated partitioning algorithms for constraint hypergraph $H(A, E)$. For example, partitioning can be done by algorithms used for circuit partitioning in VLSI design[10]. The solution computed by partitioning can be improved by considering some of the constraints between aggressor partitions. The following is a simple representative of this kind of algorithms. It works on a given set of aggressor partitions and builds an approximation of MRAS by merging pairs of aggressor partitions.

**Heuristic Algorithm:**

**Input:** A set of aggressor partitions $A$=$\{A_1, A_2,...\}$ (an even number of partitions is assumed for simplicity), and a set of inter-
partition SLIs $C$, i.e. SLI constrained aggressors from different partitions.

Each constraint $c_{i,j} \in C$ prohibits simultaneous noise injection by aggressor pair $a_{i,p} \in A_i$ and $a_{j,q} \in A_j$ belonging to different partitions.

Each constraint $c_{i,j}$ between aggressors $a_{i,p}$ and $a_{j,q}$ is assigned a weight $w(c_{i,j}) = \min(w(a_{i,p}), w(a_{j,q}))$

Output: Set of aggressors $A_s \subseteq A$, i.e. a conservative approximation of MRAS.

1. Set $A_s = \emptyset$
2. Repeat until $A$ is empty
   1.1. Find constraint $c_{i,j}$ with maximum weight prohibiting aggressors $a_{i,p} \in A_i \subseteq A$, $a_{j,q} \in A_j \subseteq A$ from simultaneous noise injection
   1.2. Compute maximum realizable aggressor sets $A_{i,R}$, $A_{j,R}$, $A_{i',R}$, $A_{j',R}$ for aggressor sets $A_i$, $A_{i'} = A_i \setminus a_{i,p}$, $A_j$, $A_{j'} = A_j \setminus a_{j,q}$, respectively
   1.3. If $w(A_i \cup A_j) > w(A_{i'} \cup A_{j'})$
      set $A_s = A_i \cup A_{i,R} \cup A_{j,R}$
      else
      set $A_s = A_j \cup A_{i,R} \cup A_{j,R}$
   1.4. Exclude aggressor partitions $A_i$ and $A_j$ from $A$
3. Compute approximation of maximum realizable noise as $w(A_j)$

The algorithm iterates through pairs of aggressor partitions and the most significant inter-partition SLI for each partition pair. For each partition pair, the algorithm checks all possible ways to satisfy the inter-partition SLI and computes maximum realizable aggressor set for the union of these partitions. The algorithm can be extended to consider many SLIs between aggressor partitions, merging more than two partitions together and handling more complex constraints. Any of these extensions explores all possible ways to satisfy inter-partition constraints and selects the one injecting the highest noise. In our experiments, we implemented the version considering up to two SLIs simultaneously and merging up to two aggressor partitions.

5 Implementation and experimental results

The proposed pessimism reduction technique was implemented in an industrial noise analysis tool [2]. The system was architected using a separate logic pessimism reduction engine. First, the noise analysis tool performs delay noise analysis without logic information. For each net, it computes delay variation due to cross coupled noise and supplies this information to a static timing analysis tool. Additionally for each net, it computes delay variation due to each aggressor. This information is computed during noise analysis and does not require any additional computation time. The timing analysis tool finds paths violating timing constraints. The pessimism reduction engine computes logic correlations for the whole circuit and then processes each signal propagation path separately. For each signal propagation path, it first excludes any aggressor which cannot inject noise into victim nets because of logic correlations with the victim nets. Then it solves the MRAS problem using the remaining aggressors. After computing the MRAS, the noise analysis tool performs SPICE simulation of each noise cluster and computes accurate net delays and path delays in the presence of cross coupled noise. This is to avoid any inaccuracy due to the linear noise model which is used solely for optimization.

We implemented the following algorithms, a performance comparison of which is provided in Table 1.

1. A noise cluster partition algorithm which solves the MRAS problem for each noise cluster separately, ignoring logic constraints for aggressors across clusters (column 3).
2. An improved noise cluster partition algorithm (described in Section 4.3) which improves the above algorithm by taking into account logic constraints between aggressors of different noise clusters (column 4).
3. A full-path MWIS algorithm which computes the maximum weight independent set of a constraint graph for the set of aggressors of the whole path using SLIs (binary logic constraints) only (column 5).
4. An exact Branch and Bound algorithm (column 6)

We applied the above to several ISCAS benchmarks and industrial circuits (see Table 1, columns 1 and 2 for circuit name and number of transistors.) Columns 3 - 6 show the percentage of total path delay noise remaining after applying the above 4 algorithms. Results are averaged over 20 paths randomly selected from the set of longest paths. As it can be seen, the exact method of branch and bound (column 6) reduces the average total path delay variation due to crosstalk by up to 75% and by 44% on average. This method uses all derived logic constraints including the ones that contain more than two variables and exhaustively goes through the decision tree using Branch and Bound to get the optimal result. Using logical correlations only within the nets of separate clusters and ignoring any inter-cluster correlations (column 3), gives the least pessimism reduction as expected. This also shows that using a path based approach that benefits from the correlations between nets of different noise clusters in the signal propagation path significantly increases the opportunity to reduce pessimism. The heuristic algorithm and the SLI-only constraint graph approach gives improved pessimism reduction, respectively. Although the results of the faster algorithms are sub-optimal compared to branch and bound, they offer a good trade-off between pessimism reduction and the run time. In our experience, the performance of branch and bound is unacceptable for more than 100 aggressors. The con-

<table>
<thead>
<tr>
<th>circuit</th>
<th>#transistors</th>
<th>% delay noise remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>separate clusters</td>
</tr>
<tr>
<td>c499</td>
<td>1572</td>
<td>88.61</td>
</tr>
<tr>
<td>c1355</td>
<td>2260</td>
<td>87.91</td>
</tr>
<tr>
<td>c1908</td>
<td>3322</td>
<td>91.24</td>
</tr>
<tr>
<td>c2670</td>
<td>4990</td>
<td>88.65</td>
</tr>
<tr>
<td>cnt_1</td>
<td>350</td>
<td>68.83</td>
</tr>
<tr>
<td>cnt_0</td>
<td>340</td>
<td>70.28</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>63.08</td>
</tr>
<tr>
<td></td>
<td>352</td>
<td>69.23</td>
</tr>
<tr>
<td>c_1</td>
<td>1008</td>
<td>94.06</td>
</tr>
</tbody>
</table>

Table 1: Results of experiments on delay noise analysis
straint graph based method is applicable for problems with up to 300 aggressors. The heuristic algorithm is useful virtually for any larger problem size.

In Table 2, we present some additional details on our experiments. In the table, column 1 shows the circuit names and column 2 gives the number of SLIs between victim and aggressor nets and number of SLIs among aggressor nets respectively, averaged over 20 critical paths. Column 3 shows the total number of hyperedges (multi-variable constraints) and SLIs. Column 4 presents the percentage reduction in the number of paths violating timing for the branch and bound method. Column 5 shows the critical path delay before and after the application of branch and bound approach. Column 6 shows the CPU times for the heuristic and branch-and-bound techniques. As can be seen in Table 2, logic correlation based pessimism reduction in noise-aware STA reduces the number of timing violating paths by up to 100% and by 61% on average in the test cases.

We also analysed a very large block (consisting 620,000 transistors) of a low power chip. We generated 35,562 logic constraints which included 26,423 SLIs and processed about 1100 signal propagation paths. The average noise pessimism reduction for this test case was about 12% with constraint graph approach and 17% with the branch and bound algorithm. For some paths, the noise estimation was reduced up to 31%. The branch and bound algorithm was applied only to paths with less than 200 aggressors. The total run time for constraint graph algorithm was about 56 hours on a Sun UltraSparc 10 workstation. This test case had significantly longer signal propagation paths and fewer signal correlations compared to other high performance microprocessor blocks (in Table 1) and thus the gains realized were smaller compared to the results in Table 1.

### 6 Conclusions

In this paper, we presented several novel techniques for path-based delay noise pessimism reduction using a linear model of delay noise and logic correlations between circuit signals. The accuracy of linear delay noise model for the purpose of optimization was validated and justified with experimental data. We developed and implemented several exact and heuristic algorithms for solving the maximum realizable aggressor set problem for large sets of victim and aggressor nets related to signal propagation paths. Their efficiency and accuracy were demonstrated on ISCAS benchmarks and some industrial circuits. The proposed techniques are incorporated into an industrial noise analysis tool which has been used for functional and delay noise analysis and repair of many high performance designs. The proposed methodology is shown to reduce pessimism in delay noise estimation and to avoid very expensive noise reduction measures for many nets.

### 7 References


<table>
<thead>
<tr>
<th>circuit</th>
<th>#SLIs: vict-agg/agg-agg</th>
<th>#multi-constr./SLIs</th>
<th>violat. reduc %</th>
<th>delay (ns) before/after reduction</th>
<th>CPU (s) sep.clust, br&amp;bnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>c499</td>
<td>16/56</td>
<td>5995/4595</td>
<td>33.3</td>
<td>14.38/13.98</td>
<td>25/59</td>
</tr>
<tr>
<td>c1355</td>
<td>25/43</td>
<td>3929/3441</td>
<td>33.3</td>
<td>19.79/19.13</td>
<td>21/46</td>
</tr>
<tr>
<td>c1908</td>
<td>10/11</td>
<td>5043/3320</td>
<td>100</td>
<td>28.01/27.55</td>
<td>45/87</td>
</tr>
<tr>
<td>c2670</td>
<td>8/13</td>
<td>12064/8111</td>
<td>50.0</td>
<td>29.58/28.45</td>
<td>84/180</td>
</tr>
<tr>
<td>cnt_1</td>
<td>407/568</td>
<td>7113/926</td>
<td>71.4</td>
<td>16.63/15.68</td>
<td>95/567</td>
</tr>
<tr>
<td>cnt_0</td>
<td>540/740</td>
<td>9579/750</td>
<td>100</td>
<td>15.33/14.38</td>
<td>120/613</td>
</tr>
<tr>
<td>cnt_1</td>
<td>394/515</td>
<td>6149/886</td>
<td>80.0</td>
<td>17.48/16.29</td>
<td>45/256</td>
</tr>
<tr>
<td>cnt_0</td>
<td>576/729</td>
<td>7557/899</td>
<td>37.5</td>
<td>18.21/16.76</td>
<td>23/143</td>
</tr>
<tr>
<td>cla</td>
<td>4/13</td>
<td>1028/435</td>
<td>66.7</td>
<td>12.15/11.98</td>
<td>30/168</td>
</tr>
</tbody>
</table>

Table 2. Delay noise analysis statistics