Top-k Aggressors Sets in Delay Noise Analysis

Ravikishore Gandikota, Kaviraj Chopra, David Blaauw, Dennis Sylvester, Murat Becer*
University of Michigan, Ann Arbor, MI. email: {gravkis, kaviraj, blaauw, dennis}@eecs.umich.edu
*CLK Design Automation, Littleton, MA. email: murat@clkda.com

ABSTRACT

We present, in this paper, novel algorithms to compute the set of "*top-k*" aggressors in a design. We show that the computation of the set of top-*k* aggressors is non-trivial, since we must consider all permutations of aggressors that are coupled to a critical path. Also, different sets of aggressors contribute different amounts of noise to each critical path and a brute-force enumeration to obtain the set of top-*k* aggressors has impractical runtime. Our proposed approach uses two key techniques to reduce the runtime complexity: Firstly, we model the delay noise propagated from a victim net to its fanout net by a so-called *pseudo aggressor,* which simplifies our problem formulation significantly. Secondly, we define a *dominance* property for aggressor sets, which imposes a partial ordering on the aggressor sets and allows us to efficiently prune the enumeration space. We then demonstrate the effectiveness of our proposed algorithm on benchmark circuits.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Algorithms, Design

Keywords

Crosstalk, delay noise, static timing analysis

1 INTRODUCTION

Coupling noise has become an important concern in nano-meter designs. *Delay noise* which models the impact of noise on circuit delay is of particular concern for high-performance designs. To address this issue, static noise analysis was first introduced in [1],[2] and has been the focus of significant research effort. Since delay noise requires both the aggressor and the victim nets to switch at the almost the same time, *timing windows* were defined to indicate the time range in a clock period within which an aggressor/victim net can transition. It was observed early on that the computation of delay noise and timing windows poses a *chicken-and-egg* problem. Delay noise cannot be computed before timing windows are defined and, vice versa, accurate timing windows cannot be computed without information about the delay noise.

An iterative method was suggested in [3],[5] for computing the delay noise in a design. The iterations start by either assuming that all aggressor-victim timing windows have an overlap or none of them have any overlap. The circuit delay is then computed by iteratively updating the delay noise and timing windows. It was shown in [3] that this iterative method is guaranteed to converge. It was observed in [4] that the problem formulated above has solutions which are fixpoints on a complete lattice. Also, a number of methods to identify and eliminate *false aggressors,* which cannot impact the delay of a victim due to logical and timing correlations in the circuit, have been proposed in [10],[11].

DAC 2007, July 4–8, 2007, San Diego, California, USA.
Copyright 2007 ACM 978-1-59593-627-1/07/0006...$5.00.

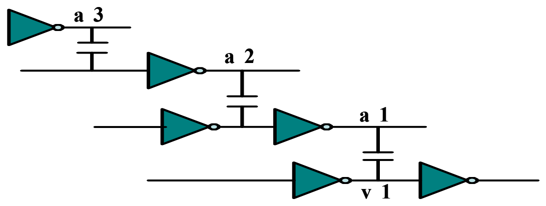


Figure 1. Indirect aggressors *a*2, *a*3 affect the timing window of primary aggressor *a*1 in the delay noise analysis of victim *v*1.

Despite the pruning of false aggressors, the total number of aggressors that contribute to delay noise in a circuit can be extremely high. For a victim net, the so-called *primary* aggressors couple noise directly on to the victim transition. Furthermore, *indirect* aggressors are coupled to primary aggressors and can impact the timing window of a primary aggressor. This increase in the timing window of a primary aggressor may cause an overlap with the victim timing window, resulting in an increased delay noise. In Figure 1, for instance, noise from aggressor *a*2 can increase the timing window of *a*1 such that it now overlaps with that of victim *v*1. Therefore, *a*2 is an indirect or *secondary* aggressor of victim *v*1. Similarly, *a*3 is a *tertiary* aggressor of victim *v*1. Note that in this example, the noise analysis algorithm would require 3 iterations for convergence. The fact that industrial noise analysis tools report the need for 3-4 iterations for convergence, shows that noise from indirect aggressors contribute to delay noise in industrial designs.

Since we must consider primary aggressors coupled to each node along a circuit path and also those coupled to the fanin cone of the primary aggressors, it is clear that the number of aggressors that can potentially contribute to the circuit delay is huge. However in practice, designers often limit the number of aggressors which can switch simultaneously due to one of the following reasons:

- Delay noise that involves hundreds of precisely timed noise events is considered unlikely and consequently ignored.
- A noise event involving hundreds of aggressors is less probable than that involving a few aggressors. With limited resources for fixing delay noise, the latter event must be given a higher priority.

A very common approach to limit the total number of aggressors considered in delay noise is by restricting the set of primary aggressors for each victim to a few (say 10) by choosing those aggressors which exhibit the maximum amount of coupling with the victim. However, this approach to reduce the number of aggressors may lead to unpredictable results. Firstly, the total number of aggressors contributing to delay noise will vary from one path to another. Also, there is no consistent manner of restricting the total number of indirect aggressors that contribute delay noise due to noise iterations.

In this paper, we present a new algorithm for computing the set of "top-*k*" critical aggressors. This information will provide a feed-

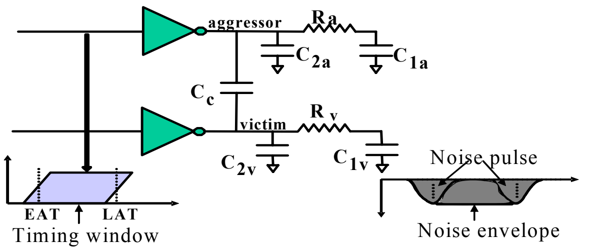


Figure 2. Aggressor timing window and resulting noise envelope

back to the designer regarding the set of aggressors that contribute most strongly to the circuit delay. The concept of the set of top-*k* aggressors is analogous to the top-*k* critical paths commonly reported in traditional static timing analysis, but comes in two flavors, (1) the *top-k aggressors elimination* set, and (2) the *top-k aggressors addition* set. We introduce both of them below:

Top-*k* aggressors elimination set: Given traditional noise analysis, a top-*k* aggressors elimination set is a set of *k* aggressor-victim couplings which, when not considered in noise analysis, would reduce the delay noise of the design by a maximum amount. This information is vital to a designer in situations where only a limited number of aggressor-victim couplings can be fixed. For instance, if a designer can eliminate only 10 coupling situations (e.g., through shielding or spacing), then the top-10 aggressor elimination set exactly points to the set of 10 aggressor-victim couplings which must be fixed to obtain the maximum reduction in delay noise. Hence, the top-*k* aggressors elimination set ensures that the maximum delay noise improvement is achieved for the performed effort. It is true that fixing a particular aggressor-victim coupling may perturb the overall physical implementation and may result in other new couplings. Nevertheless, the availability of the top-*k* aggressors elimination set is key in each cycle of delay noise mitigation.

Top-*k* aggressors addition set: Given a timing analysis without delay noise, the top-*k* aggressors addition set is the set of *k* aggressor-victim couplings whose delay noise, when added to the noiseless timing analysis, will result in the maximum circuit delay. The top-*k* aggressors addition set is useful if the designer wants to restrict the noise analysis to no more than *k* aggressor-victim couplings switching together. Alternatively, it can also be used to identify sets of aggressors which must be given a higher priority while fixing aggressors for delay noise mitigation.

In this paper, we show that the computation of the top-*k* aggressors addition and elimination sets are dual problems. The analysis is complicated by the fact that both primary and indirect aggressors must be considered for inclusion in the top-*k* aggressor sets. For correctness, in addition to the critical path, the analysis must also include near-critical paths. Furthermore, we must model the propagation of delay noise efficiently. The proposed algorithm uses two novel concepts to provide a tractable solution: Firstly, we model the propagated delay noise with a *pseudo aggressor* and secondly, we prune the enumeration space by using a *dominance* relationship which imposes a partial order on the aggressor sets. The proposed algorithm is able to achieve practical runtimes for large values of *k* on all tested benchmark circuits. In comparison, brute-force enumeration could not generate sets with *k* greater than 3, even for the smallest benchmark circuit. The remainder of this paper is organized as follows. We describe the problem in detail in Section 2 and then describe the proposed algorithm for top-*k* enumeration in Section 3. We present the results in Section 4 and finally conclude the paper in Section 5.

2 PROBLEM DESCRIPTION

The goal of this work is to identify, for a given *k*, the set of *k* aggressors (top-*k* aggressor set) which must be fixed for optimally minimizing the noise violations in a design. Such a technique can (potentially) be employed in the inner loop of design optimization and therefore runtime efficiency is key. Conventionally, linear Thevenin driver models were used to perform static noise analysis efficiently. Recently, non-linear current source based driver model have been proposed [9] to achieve the accuracy demanded by sign-off timing analysis. However, a single victim-aggressor alignment in such a framework requires a non-linear solver and it is clearly difficult to achieve an efficient runtime. Therefore, we make engineering decision to use the linear noise framework in our analysis. Note that linear noise analysis is still used in the industry [12] in applications (such as ours) where accuracy can be traded for runtime efficiency. For the sake of completeness, we will now briefly review the framework of linear noise analysis.

In static timing analysis (STA), timing windows are computed by propagating the fastest and the slowest switching signals for each net from the inputs to the outputs. *Early arrival time* (EAT) of a net is defined as the earliest time at which a switching transition reaches 50% of supply voltage (referred to as t_{50}). Similarly, *latest arrival time* (LAT) is the latest possible t_{50} and forms the other extreme of a timing window. A *noise envelope*, which bounds the noise coupled from an aggressor to the victim net, is obtained by sweeping the aggressor input transition within its timing window and observing the peak noise coupled to the victim net. In this work, we compute the 'trapezoidal' noise envelope by combining the noise pulse coupled from an aggressor switching at its EAT and its LAT and subsequently connecting their noise peaks (as shown in Figure 2).

The worst-case delay noise due to an aggressor is obtained by superimposing the corresponding noise envelope with the latest victim transition waveform (i.e. t_{50} = LAT) and observing the increase in t_{50}. If multiple aggressors are coupled to the victim, we construct noise envelopes for each aggressor individually and add them together to create a combined noise envelope (as shown in Figure 3). The worst-case delay noise due to all aggressors is similarly obtained by superimposing the combined noise envelope. The worst-case aggressor alignment problem has been extensively studied in literature ([5],[6],[7]). However, the problem of computing the set of top-*k* aggressors has not been addressed, to our knowledge. A brute-force manner of generating the top-*k* aggressors elimination set is by simply running the noise analysis multiple times and eliminating *k* aggressors in each run. In this case, a total of ${}^{r}C_{k}$ noise analysis runs are required, where *r* refers to the total number of aggressors in the circuit, which is prohibitively expensive.

The computation of top-*k* aggressors set is non-trivial and complicated due to several factors and two of which are discussed below. First, note that with increasing cardinality, the top-*k* aggressors sets could be non-monotonic. For instance, if the top-3 aggressors set in an arbitrary design contains aggressors {*a*1, *a*5, *a*9}, then the aggressor set with the next higher cardinality (i.e. top-4) may not contain any these aggressors. Although counter-intuitive, this

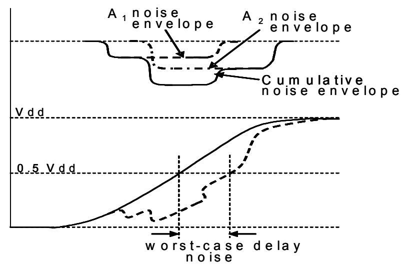


Figure 3. Worst-case delay noise due to two aggressors

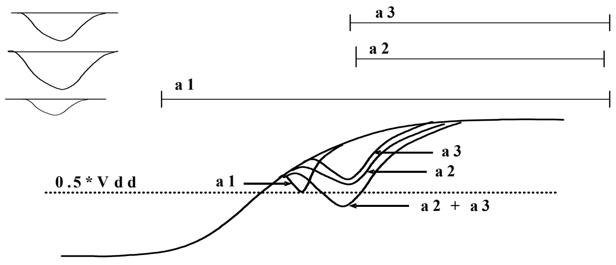


Figure 4. Non-monotonicity in aggressor lists for aggressors a1, a2, and a3 (Timing windows and noise pulses as shown.)

property arise due to the fact that the alignment of aggressors affect the delay noise (as shown in Figure 4). Although aggressors *a*2 and *a*3 have larger noise pulses than a1, their timing windows restrict their alignment to the left and they do not produce any delay noise if they switch individually (t_{50} doesn't change). Hence, the top-1 aggressor set is {*a*1}, although its noise pulse is smaller than both *a*2 and *a*3. However, when we consider the top-2 aggressors set, the delay noise due to the combined switching of {*a*2,*a*3} is greater than that of {*a*1,*a*2} and {*a*1,*a*3}. Therefore, the top-2 aggressors set is {*a*2,*a*3}. This example shows that adding an aggressor to the top-*k* aggressors set may not necessarily produce the top-*k+1* set.

Secondly, the worst-case aggressor-victim alignment at a victim net is affected by the delay noise propagated from the fanin cone of the victim driver. Hence, the top-*k* aggressors set at any victim *v* must consider aggressors coupled to the transitive fanin cone of victim *v*. Similarly, the impact of aggressors coupled to the transitive fanin cone of the primary aggressors must also be considered as they can change the timing window of the primary aggressors. Therefore, the primary aggressors must be considered both by themselves as well as acting in concert with tertiary aggressors that increase their timing window. A primary aggressor acting by itself is referred to as a first *order* aggressor. Primary aggressors are assigned an order $p = t+1$, where *t* is the number of aggressors coupled to the transitive fanin cone of the primary aggressor.

3 PROPOSED APPROACH

We will focus on the algorithm to compute the top-*k* aggressors **addition** set, since it is conceptually simpler to understand. In Section 3.4, we show how the proposed algorithm can be modified to instead compute the top-*k* aggressors elimination set. We use implicit enumeration to iteratively compute the desired top-*k* aggressors set in a bottom-up manner. During the i^{th} iteration ($i < k$), a super-set of all aggressor sets of cardinality *i* - defined as $list_i$ is constructed. Elements of $list_i$ can potentially be a subset of the desired top-*k* aggressors set and the $list_i$ is generated for all values of *i* from 1 through *k*. Finally, the set of aggressors in $list_k$ which has the maximum delay noise is reported as the top-*k* aggressors set.

The enumeration of $list_i$ in the i^{th} iteration is based on two key concepts: First, we use "pseudo aggressors" to model the shift observed in a victim transition due to noise coupled to the transitive fanin cone of the victim driver. This allows us to model all aggressors coupled to the fanin cone of a victim by using 'pseudo' noise envelopes that are similar to the noise envelopes of primary aggressors. Hence, if $list_i$ is computed at the input of a victim driver, then the $list_i$ can be propagated to the victim net by using these pseudo aggressors. This leads to an efficient problem partition i.e. find the $list_k$ for each victim and propagate it a topological order.

Second, we use the concept of dominance to aggressively prune the solution search space. The dominance property imposes a partial ordering on the aggressors of a victim. If the noise envelope *N*1 of aggressor *a*1 entirely encompasses the noise envelope *N*2 of another aggressor *a*2, then the delay noise due to *N*1 is never less than that due to *N*2. Therefore, while computing the top-1 aggressor set, aggressor *a*1 will be chosen as compared to *a*2. In other words, aggressor *a*1 dominates aggressor *a*2. This dominance property can easily be extended to pseudo aggressors and higher order aggressors. Note that dominance based pruning dramatically improves runtime for large values of *k*. We will now describe the concepts of pseudo aggressors and dominance in more detail and then present the algorithm to compute top-*k* aggressors set.

3.1 Pseudo input aggressors

Delay noise propagated from the input of a victim driver affects the alignment of the downstream victim nets with their respective primary aggressors. Therefore, the top-*k* aggressors set for a victim net may comprise of both the primary aggressors and the aggressors coupled to its fanin cone. A brute-force approach to compute the top-*k* aggressors set for a victim could be: (1) Select *p* ($p < k$) aggressors coupled to the fanin cone of the victim and compute the delay noise on the victim. (2) Select $k - p$ primary aggressors, recompute the worst-case alignment and the delay noise. (3) Enumerate all possible permutations of *p* and select that set of aggressors which results in maximum delay noise on the victim transition. Clearly, the brute-force method is prohibitively expensive as there can be numerous sets of fanin aggressors and cannot employed for circuits of practical size. Therefore, we introduce the concept of pseudo input aggressors which allows us to traverse the circuit in a topological order while propagating top-*k* aggressor sets.

The noiseless and noisy input and output transitions occurring due to delay noise at some of its fanin node(s) are shown, in Figure 5, for a typical victim net. We know that a noisy transition on a victim input may cause a noisy output transition which can alter the worst-case alignment of the victim with its primary aggressors. For the propagation of the top-*k* sets in a topological order, we wish to break the dependence of the concerned victim output transition on the noisy input transition at its fanin. This is achieved by representing the worst-case set of aggressors coupled to the fanin of the victim by an abstract *pseudo-noise envelope*. A pseudo input noise envelope is defined as the waveform obtained by subtracting the noiseless victim transition from the delayed noisy victim transition.

Using the principle of linear superposition, the delayed victim transition can be constructed by appropriately superimposing this pseudo noise envelope with the original noiseless victim transition. Note that this pseudo noise envelope has a shape that is somewhat similar to the noise envelope obtained from primary aggressors. It may so happen that, due to the circuit layout, the nets present in the topological fanout of a victim net are its primary aggressors. Note that the delay noise induced by these aggressors can accurately be modeled by their pseudo input aggressors. Also, even if the delay noise propagated from victim input exceeds the slew of the victim transition, superposition would report delay noise accurately.

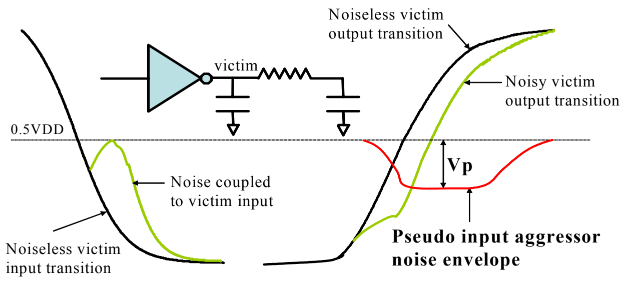


Figure 5. Pseudo aggressor noise envelope

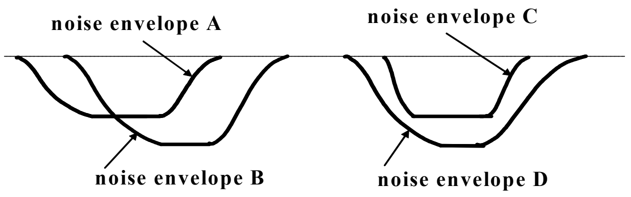


Figure 6. Dominance of two aggressor noise envelopes.

3.2 Aggressor Dominance

During the bottom-up enumeration procedure, in the ith iteration ($i > 1$), we have several candidate aggressor sets, that form the $list_i$, which are potential subsets of the desired set of top-k aggressors. In a naive implementation the $list_i$ can easily blow up. Therefore, it is important to keep the cardinality of $list_i$ under check and this is achieved by using the notion of dominance.

- **Dominance**: For any given victim, if the noise envelope of aggressor A encapsulates that of any aggressor B, then aggressor A is said to dominate aggressor B.

As shown in Figure 6, the noise envelope *D* dominates the envelope *C*, whereas both *A* nor *B* are mutually non-dominated. The usefulness of dominance follows from the following theorem.

Theorem 1. *Consider two possible aggressor sets P and Q with the same cardinality such that P* ***dominates*** *Q. A higher cardinality aggressor set obtained by adding any additional aggressor 'a' to Q would never couple a greater delay noise on the victim than that obtained by adding 'a' to P.*

Proof: Given P dominates Q, the combined noise envelope of P must encapsulate that of Q. Now, the combined noise envelope P ∪ *'a'*, obtained by adding the noise envelopes of P and 'a', must at each point in time either encapsulate or be equal to that of the combined noise envelope Q ∪ *'a'*. As the magnitude of noise of Q ∪ *'a'* is never greater that of P ∪ *'a'*, the noisy victim transition obtained by superimposing Q ∪ *'a'* with the noiseless victim transition will always have a higher (lower) voltage for a rising (falling) victim transition than P ∪ *'a'*. Therefore, the noisy victim t_{50} of Q ∪ *'a'* must always be earlier than that of P ∪ *'a'*. Consequently, the delay noise of P ∪ *'a'* is always larger than that of Q ∪ *'a'*. □

Consequently, we do not need to propagate dominated aggressor such as aggressor *C* (in Figure 6), since we can always replace it with aggressor *D* which produces a higher delay noise. This observation naturally leads to the creation of irredundant lists, defined as:

- **Irredundant Lists:** If $list_i$ be the list of all possible aggressors sets each having cardinality i, then the *irredundant list* $I\text{-}list_i$ is a subset of $list_i$, such that all aggressor sets $x \in I\text{-}list_i$, are not dominated by any aggressor set $y \in list_i$.

In other words, $I\text{-}list_i$ consists of all sets of non-dominated aggressors of cardinality *i.* The fact that the desired top-*k* aggressors set is a subset of $I\text{-}list_k$, reduces our search space significantly.

Finally, we show how the dominance property can be applied to the aggressor noise envelopes. We first identify a time interval (referred to as the dominance interval) within which a noise envelope has to encapsulate another noise envelope for it to dominate the other aggressor. The lower bound of the dominance interval is the t_{50} of noiseless victim, since a noise envelope that ends before the t_{50} will not induce any delay noise. For the other boundary of the dominance interval, we compute an upper bound on the delay noise by performing standard noise analysis by assuming all aggressors to

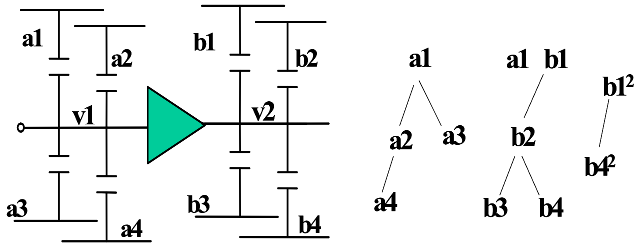


Figure 7. Partial Order between noise envelopes

have infinite timing windows. In practice, we find that a large number of noise envelopes dominate each other within the dominance interval, thereby significantly reducing the search space.

3.3 Algorithm to compute the top-k addition aggressors

In this subsection, we explain the algorithm for computing the top-*k* aggressors set. The pseudo-code of the proposed algorithm is given in Figure 9. As discussed earlier, for a desired value of *k*, the goal of the algorithm is to iteratively construct an $I\text{-}list_k$. The top-*k* aggressor set is that aggressors set which belongs to $I\text{-}list_k$ and causes the maximum delay noise. Now, in the i^{th} iteration of the algorithm, we compute the corresponding $I\text{-}list_i$ by operating on the $I\text{-}list_{i-1}$ computed in the previous iteration. Using the concept of pseudo-aggressors and Theorem 1 we can implicitly propagate irredundant lists through the circuit in topological order. The irredundant list at the sink node of the circuit gives us the desired $I\text{-}list_i$. For ease of understanding, we explain the steps of algorithm in more detail using an example (as shown in Figure 7).

In the first iteration we would like to find all non-dominated aggressors of cardinality one. Figure 7 shows two victim nets *v*1 and *v*2, with *v*1 being the input to the driver of *v*2. Victim *v1* is coupled to four primary aggressors *a*1-*a*4 and similarly *v*2 is coupled to aggressors *b*1-*b*4. Running traditional noise analysis, we obtain the timing windows and the noise envelopes of all the primary aggressor on each victim net. The partial ordering on the noise envelopes of the aggressors based on the dominance is also shown in Figure 7, where aggressor *a*1 dominates all the other primary aggressors (i.e. *a*2*, a*3*, a*4) and *b*1 dominates all primary aggressors (i.e. *b*2*, b*3*, b*4). Since *v*1 is a primary input it has no pseudo input aggressors. The irredundant list of cardinality one (as shown in Figure 8) contains only one set that contains aggressor {(*a*1)}.

For victim *v*2*,* we propagate only aggressor *a*1 as a pseudo input aggressor, since it is the only aggressor present in the irredundant list for *v*1. If the driver had multiple inputs, we would compute the top-*k* aggressors sets for each input independently and finally select the one which results in the maximum victim **output** arrival time. In the example, pseudo input aggressor *a*1 is not dominated by aggressors *b*1-*b*4 and hence, $I\text{-}list_1$ for victim *v*2 includes two aggressor sets {(*a*1), (*b*1)}.

Irredundant set	v1	v2	Possible reduced v2 set
K = 1	(a1)	(a1) (b1)	(a1) (b1)
K = 2	(a1, a2)(a1, a3)	(a1, b1) (b1,b2) (a1, a2)(a1, a3) (b1^2)	(a1, b1) (b1,b2) (a1, a3) (b1^2)
K = 3	(a1,a2,a3) (a1,a2,a4)	(b1,b2, b3) (b1,b2,b4) (a1, a3, b1) (b1^2, a1) (a1,a2,a3) (a1,a2,a4)	(b1,b2, b3) (a1, a3, b1) (b1^2, a1) (a1,a2,a3)

Figure 8. Creating higher order Irredundant sets

Now, *I-list*$_2$ can be computed by the explicit enumeration of all possible pairs of aggressors. However, a more effective approach is to reuse the information from the previous irredundant list (i.e *I-list*$_1$). In the example, both sets ($a1$, $a2$) and ($a1$, $a3$) are added to the *I-list*$_2$ for the victim $v1$. Using Theorem 1, we can ignore dominated sets such as (a2, a4).

I-list$_2$ for victim $v2$ is computed by first adding aggressor $b2$ to each aggressor set in *I-list*$_1$. Next, we add the pseudo input aggressors of cardinality 2. The set of non-dominated aggressors propagated from $v1$ is $\{(a1,a2), (a1,a3)\}$. Thirdly, we account for higher order aggressors by using a partial ordering of aggressors that have an order 2 (i.e. whose timing window has increased due to one additional aggressor). The innate cardinality of such an aggressor is 2 and is denoted as $b1^2$. Note that the height of noise envelope of an order 2 aggressor is the same as its order 1 counterpart. However, the width of the envelope increases due to its larger timing window. Using the property of dominance, we find that $b1^2$ dominates every other order 2 aggressor and is added to *I-list*$_2$.

Note that at this point, *I-list*$_2$ for $v2$ contains 5 entries (see Figure 8). However, some of these entries may dominate each other and can therefore be reduced further as shows in the right most column of Figure 8. Similarly, we generate the *I-list*$_3$ by operating on *I-list*$_2$. This procedure is repeated for k iterations. Finally, we choose the *I-list*$_k$ of the sink node and superpose the noise envelopes from all the aggressor sets present in *I-list*$_k$ with the latest sink transition waveform. The top-k aggressors set is the one which belongs to *I-list*$_k$ of the sink such that it results in the worst-case delay noise.

3.4 Top-*k* aggressors elimination set

In this section, we briefly discuss how the analysis for finding the top-k aggressors addition set can be easily modified to find the top-k aggressors elimination set. For the latter, we assume that all aggressors are present in the design and we wish to find the set of k aggressors such that fixing them will reduce the delay noise by a maximum amount. The key difference in both algorithms is that for aggressors addition set, we start with noiseless timing windows and for the aggressors elimination set, we start with noisy timing windows. Therefore, the noise envelopes of the primary aggressors are expanded since their timing windows contain delay noise. The dominance property remains the same and irredundant lists of aggressor sets are computed in a similar manner.

```
for (all i < k)
    for each victim net (in topological order)
  1. Create list_i containing aggressors sets of cardinality i by adding an additional aggressor to each aggressor set ⊂ I-list_{i-1};
  2. Add pseudo input aggressors of cardinality i to list_i; If a gate has multiple inputs, select that aggressor which results in the latest output arrival time;
  3. Add higher order aggressors of cardinality i to list_i;
  4. Use Theorem 1 and partial ordering on list_i to compute I-list_i;
  5. Superimpose the noise envelopes of all aggressor sets ⊂ I-list_i with the noiseless victim waveform and select the aggressor set that gives the worst-case delay noise; Propagate this set as the pseudo-input aggressors for downstream gates.
return the top-k aggressor set from I-list_k of the sink node
```

Figure 9. Pseudo-code for the proposed algorithm

Table 1. Validation of proposed approach with brute-force enumeration

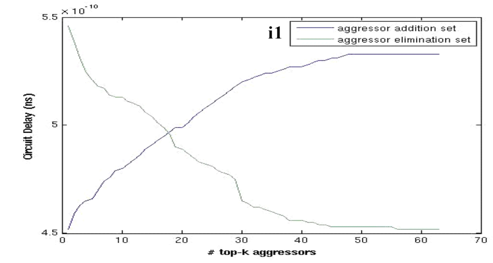
	brute-force		Top-*k* elimination set	
k	ckt delay	Runtime	ckt delay (ns.)	runtime
1	.743	0.12	.743	.01
2	.722	9.65	.722	.01
3	.709	621.4	.709	.02
4	-	-	.703	.06

However, while superposing the noise envelope we are trying to reduce the delay noise in the design. To do this we first define a *total noise envelope* as the noise envelope due to all aggressors coupled to the victim net with their largest timing windows, such that it results in the maximum noise delay in the design. The superposition of aggressor sets in the irredundant list requires: (1) Subtract the noise envelope from the total noise envelope, (2) Superpose the resulting envelope with the noiseless victim transition, and (3) Select that aggressor set which results in the smallest delay noise. The overall algorithm functions in the exactly the same manner and the top-k aggressor set is selected from the *I-list*$_k$ of the sink node.

4. RESULTS

In this section, we show experimental results of the proposed top-k aggressors addition and elimination algorithm by using a prototype noise analysis tool implemented in C++. A 0.13mm standard cell library was used for synthesis and technology mapping. The synthesized designs were placed and routed by using a commercial APR tool and the distributed RC was extracted by using a commercial parasitic extraction tool.

A brute-force method, as explained earlier, was also implemented to verify the proposed algorithm. We ran both the proposed algorithm and the brute-force algorithm on all circuits and observed that the enormous complexity of the brute-force method resulted in its failure to generate the top-k aggressors sets of size greater than 3 in 1800 sec. However, the proposed algorithm was able to generate sets of size up to 50 with tractable runtimes. As shown in Table 1, for values of $k \leq 3$, the top-k aggressors set computed by proposed algorithm was consistent with brute-force method. It can be observed that about 2 orders of magnitude runtime speedup was achieved over the brute-force approach.



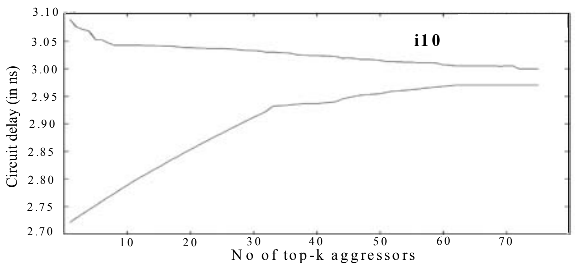


Figure 10. Aggressor addition vs. aggressor elimination set for circuits i1 and i10

In Figure 10 we present the convergence trend of the delay noise for circuits *i1* & *i10*, while computing both the top-*k* aggressors addition and elimination set for different values of *k* ranging form 1 to 75. The delay and runtime results for both algorithms are shown in Table 2 (a) and (b). Although the worst-case complexity of the proposed algorithm is exponential, in practice it can be seen that due to the efficient pruning of search space, the runtime of the algorithm grows at much smaller rate. In fact, the analysis for top-50 aggressor sets, for all benchmark circuits, completes in less than 100 sec.

5. CONCLUSION

In this work, we introduced the concept of top-*k* delay aggressors for fixing noise violations. The proposed problem is non-trivial and a naive brute-force implementation leads to impractical runtimes. Addressing this issue, we proposed the concept of pseudo aggressor that allows us to propagate possible top-*k* candidate aggressors in an organized manner. Furthermore, we proposed the dominance of noise envelopes which imposes a partial ordering on aggressors and enables us to efficiently prune the enumeration space. Based on these concepts we implemented an implicit enumeration algorithm for identifying the set of top-*k* aggressors. Experiment results show that the proposed algorithm achieves a speedup of a couple of orders of magnitude without compromising accuracy. Future work includes extension to non-linear driver models and finding a ‘good’ value of k for reasonably fixing noise violations in a design.

ACKNOWLEDGEMENT

This work has been supported by funding from NSF.

REFERENCES

[1] K. L. Shepard and V. Narayanan, “Noise in deep submicron digital design,” IEEE/ACM International Conference on Computer-Aided Design, pp. 524-531, 1996.

[2] K. L. Shepard, V. Narayanan, P. C. Elmendorf, G. Zheng, “Global harmony: coupled noise analysis for full chip RC interconnect networks,” IEEE/ACM International Conference on Computer-Aided Design, pp. 139-146, 1997.

[3] S. S. Sapatnekar, “A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, pp. 550-559, 2000.

[4] H. Zhou, “Timing analysis with crosstalk is a fixpoint on a complete lattice,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22, pp. 1261-1269, 2003.

[5] P.D. Gross, R. Arunachalam, K. Rajagopal, L.T. Pileggi, “Determination of worst-case aggressor alignment for delay calculation,” IEEE/ACM International Conference on Computer-Aided Design, pp. 212-219, 1998.

[6] T. Xiao, M. M. Sadowska, “Worst delay estimation in crosstalk aware static timing analysis,” International Conference on Computer Design, pp. 115-120, 2000.

[7] V. Rajappan, S.S. Sapatnekar,“An efficient algorithm for calculating the worst-case delay due to crosstalk,” International Conference on Computer Design, pp.76-81, 2003.

[8] D. Blaauw, S. Sirichotiyakul, C. Oh, “Driver modeling and alignment for worst-case delay noise”, IEEE Transactions on VLSI Systems, Vol 11, pp. 157-166, 2003.

[9] J.F. Croix and D.F. Wong, “Blade and razor: cell and interconnect delay analysis using current-based models,” IEEE/ACM Design Automation Conference, pp. 386-389, 2003.

[10] K. P Belkhale, A. J. Suess, “Timing analysis with known false subgraphs”, IEEE/ACM International Conference on Computer-Aided Design, pp. 736-739, 1995.

[11] D. Chai, et. al., “Temporofunctional Crosstalk Noise Analysis,” Design Automation Conference, pp. 860-863, 2003.

[12] R. Levy, et. al., “ClariNet: a noise analysis tool for deep submicron design,” Design Automation Conference, pp. 233-238, 2000.

[13] I. Keller, K. Tseng, N. Verghese, “A robust cell-level crosstalk delay change analysis,” IEEE/ACM International Conference on Computer-Aided Design, pp. 147-154, 2004.

Table 2. (a). Delay and runtime results shown for benchmarks circuits for top-*k* addition set of aggressors

ckt	# gates	# nets	# coupling caps	Circuit Delay (in ns.)							Runtime (in s)							
				0	5	10	20	30	40	50	0	5	10	15	20	30	40	50
i1	59	46	232	.452	.466	.480	.499	.520	.527	.534	.01	.01	.01	.02	.06	.32	.65	.89
i2	222	221	706	.582	.604.	.636	.667	.696	.711	.726	.01	.05	.15	.21	.48	.81	1.44	1.68
i3	132	126	551	.413	.428	.444	.459	.489	.504	.521	.01	.02	.08	.09	.17	.46	.73	1.12
i4	236	230	1181	.661	.674	.689	.716	.743	.764	.779	.04	.13	.17	.58	.92	1.82	3.64	6.78
i5	204	138	1835	.958	.984	1.01	1.03	1.06	1.08	1.11	.02	.26	.82	1.18	2.52	6.86	13.2	15.4
i6	735	668	7298	.861	.898	.924	.960	.971	1.01	1.03	.09	.72	2.36	2.94	3.57	4.12	26.1	38.4
i7	937	870	9605	.823	.843	.862	.898	.932	.964	.993	.15	.61	4.12	9.09	13.9	19.5	41.8	68.1
i8	1609	1528	10235	1.47	1.50	1.52	1.54	1.57	1.58	1.59	.21	.67	2.37	5.23	9.42	16.3	37.1	66.9
i9	1018	955	14140	1.52	1.56	1.59	1.65	1.71	1.75	1.78	.18	.68	3.17	6.42	12.1	24.9	43.9	75.6
i10	3379	3155	18318	2.71	2.75	2.78	2.85	2.91	2.94	2.96	.46	.78	4.28	6.41	13.8	27.5	55.6	81.5

Table 2. (b). Delay and runtime shown for benchmark circuits for the top-*k* elimination set of aggressors

ckt	# gates	# nets	# coupling caps	Circuit Delay (in ns.)								Runtime (in s)							
				all agg.	5	10	20	30	40	50	no agg.	1	5	10	15	20	30	40	50
i1	59	46	232	.546	.521	.513	.489	.465	.456	.453	.452	.01	.01	.02	.03	.15	.49	.79	1.12
i2	222	221	706	.743	.695	.671	.649	.633	.625	.621	.582	.03	.07	.20	.38	.71	1.38	2.47	3.13
i3	132	126	551	.529	.471	.453	.438	.431	.427	.421	.413	.03	.07	.10	.13	.29	.71	1.24	1.93
i4	236	230	1181	.801	.763	.746	.716	.712	.702	.697	.661	.04	.15	.21	.63	1.15	2.47	4.19	7.63
i5	204	138	1835	1.21	1.11	1.09	1.04	1.01	1.00	1.00	.958	.04	.46	.97	2.63	3.42	7.31	13.2	18.7
i6	735	668	7298	1.05	.976	.960	.955	.949	.936	.921	.861	.16	.62	2.27	3.84	4.29	12.5	28.5	42.5
i7	937	870	9605	1.12	1.09	1.06	1.05	1.04	1.04	1.02	.823	.20	.69	4.27	10.2	14.8	25.4	47.2	71.5
i8	1609	1528	10235	1.64	1.61	1.59	1.58	1.57	1.55	1.54	1.47	.24	.72	3.68	6.72	12.5	21.7	42.7	79.3
i9	1018	955	14140	1.84	1.81	1.80	1.78	1.77	1.76	1.75	1.52	.31	.78	3.65	7.36	13.4	26.4	41.8	75.9
i10	3379	3155	18318	3.09	3.05	3.04	3.03	3.03	3.02	3.02	2.71	.41	1.23	5.43	8.76	16.4	34.6	62.8	91.4