Non-Iterative Switching Window Computation for Delay-Noise

Bhavana Thudi, David Blaauw

University of Michigan, Ann Arbor, MI 48109

ABSTRACT

In this paper, we present an efficient method for computing switching windows in the presence of delay noise. In static timing analysis, delay noise has traditionally been modeled using a simple switch-factor based noise model and the computation of switching windows is performed using an iterative algorithm, resulting in an overall run time of $O(n^2)$, where n is the number of gates in the circuit. It has also been shown that the iterations converge to different solutions, depending on the initial assumptions, making it unclear which solution is correct. In this paper, we show that the iterative nature of the problem is due to the switching-factor based noise model and the order in which events are evaluated. We utilize a delay noise model based on superposition and propose a new algorithm with a run time that is linear with the circuit size. Since the algorithm is non-iterative and does not operate with initial assumptions, it also eliminates the multiple solution problem. We tested the algorithm on a number of designs and show that it achieves significant speedup over the iterative approach.

Categories and Subject Descriptors: B.7.2 **[Integrated Circuits]**: Design Aids - Verification.

General Terms: Algorithms, Theory, Verification.

Keywords: Cross-talk noise, Superposition, Switching Window.

1 INTRODUCTION

Noise from cross-coupling capacitance between neighboring nets has become a dominant factor in static timing analysis. *Delay noise*, which is the topic of this paper, occurs when noise is injected on a *victim net* when the victim transitions. It is critical that delay noise is accurately accounted for and a number of methods to compute the impact of noise on circuit delay have been proposed [1 - 4].

In order to reduce the pessimism of noise analysis, timing windows [2] are often computed to determine which aggressor nets can switch simultaneously with the victim net. However, the computation of timing windows in the presence of delay noise exhibits a well known "chicken and egg" problem [5]. The delay noise depends on the overlap of victim and aggressor timing windows. However, the timing windows depend on the delay of the circuit, which, in turn, is impacted by the delay noise. An iterative approach has been used to solve this problem. For instance, timing windows are initially computed without coupling noise and are then updated with delay noise and computed in multiple iterations until convergence.

Typically a so-called *switch factor* model is used where for both victim and aggressor nets, the coupling capacitance is replaced with a grounded capacitance. The value of the grounded capacitance is equal to the coupling capacitance, multiplied by a constant k, referred to as a switch factor [3][4].

As shown in Figure 1(a), we consider two coupled nets and their initial timing windows, computed without coupling noise. If, after this initial window computation, aggressor and victim timing windows overlap, as shown in Figure 1(b), the switch factor for computation of the early edge of window m is set to 0 and that for the computation of the late edge of l to 2. This has the effect of increasing the size of the windows, which can cause the windows of other nets to become overlapping. Multiple iterations are therefore needed to reach convergence. Also, an individual pair of nets may require multiple iterations to converge, since with each iteration, the region of overlap between the two nets increases as shown in Figure 1(c).

It was shown in [5] that an iterative computation of timing windows is guaranteed to converge with a maximum number of iterations of $O(n)$, where n is the number of circuit elements, leading to a worst-case run time complexity of $O(n^2)$. In [6], different window scheduling approaches are explored to reduce the number of iterations, however, the worst-case number of iterations remains $O(n)$. In practice, the number of iterations is typically less, but can still reach 5 - 10 for circuits with significant coupling. As coupling noise increases with technology scaling, the number of required iterations is expected to grow. It was later shown in [7] that the solution of the iterative computation depends on the assumption used to compute the initial windows. If the initial windows are computed in the absence of delay noise, versus with delay noise, the iterative approach will converge to different solutions. For instance, in Figure 1(d) two windows are shown that do not overlap in the absence of delay noise, whereas they overlap if the initial windows were computed assuming the presence of delay noise.

In this paper, we present a new approach for computing circuit performance in the presence of coupling noise, which differs from the discussed approach in two fundamental ways.

- We use a delay noise model based on linear superposition. Superposition has been used extensively in functional noise computation and has also been proposed for delay noise computation [6][8][10]. In a switch factor based model, delay noise is a function of the timing windows at the victim and aggressor nodes themselves, creating a cyclical dependency between the timing windows of the nodes. In the superposition based model, delay noise is a function of the timing windows at the driver *inputs* of the victim and aggressor nets, thereby removing this immediate cyclical dependency.

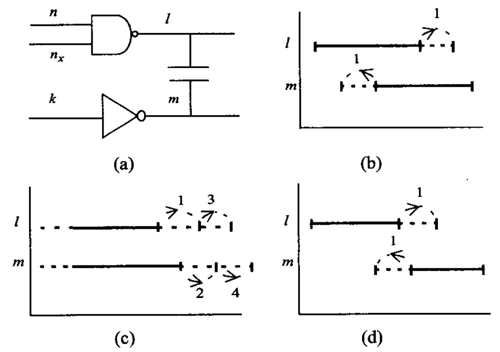

Figure 1. Coupled net and timing window computation. Arrows show change in window size over iterations as numbered.

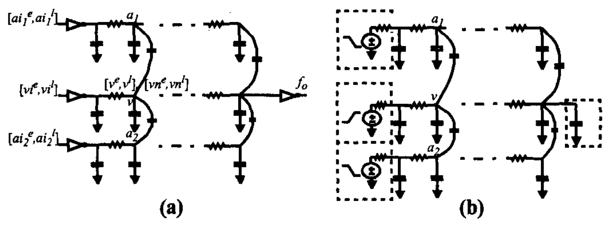

Figure 2. Noise cluster with switching windows and linear model for superposition

- Instead of traversing the timing graph in topological order, based on the structure of the circuit, we use a time-sort based algorithm where early and late window edges are scheduled as separate events and are processed in non-decreasing order.

In general, time progresses in monotone increasing fashion in the proposed algorithm and the worst-case run time is linear with the circuit size. We implemented the proposed algorithm and show that the algorithm achieves speedups of up to 5X over the iterative approach.

2 OVERVIEW

Given a victim net and aggressor nets, we construct linear Thevenin models for the victim and aggressor driver gates [10][11] as shown in Figure 2. Using superposition, each of the voltage sources is simulated in turn, while the other voltage sources are shorted. The voltage waveforms observed at the receiver gate input from all simulations are then added together to obtain the combined waveform. Figure 3(a) shows the *noiseless transition*, when only the victim driver voltage source of v is simulated, the noise pulses obtained by simulating each of the aggressor drivers a_1 and a_2, the *composite noise pulse* obtained by adding the two noise pulses, and the *noisy transition*, obtained by adding the composite pulse to the noiseless transition. The use of linear superposition has the advantage that the noise waveform induced by each aggressor can be shifted to search for the worst-case alignment with respect to the noiseless transition without requiring re-simulation of the network.

The noisy victim transition, and hence the impact of noise on delay, is a function of the alignment of the noise pulses. It is important to note that this alignment is constrained by the timing window at the *input* of the aggressor and victim drivers. Unlike the switch factor noise model, delay noise is not a function of the timing window at the aggressor and victim nodes themselves, thereby breaking the cyclical dependency between delay noise computation at a victim node on the timing window at that same node.

Based on this observation, we model the circuit using a so-called *causality* graph, where an edge from node n to node m indicates dependence of the delay at node m on the timing window at node n. Figure 4(a) shows a simple circuit and its causality graph. Vertex n_3 in the causality graph has edges incident from vertices n_1 and n_2, indicating that, if the timing windows at both n_1 and n_2 are defined, the timing window at node n_3 can be computed. Note that since the causality graph of the circuit in Figure 4(a) is acyclical, the timing windows for this circuit can be computed in a single topological traversal of the causality graph with linear run time, where as multiple iterations might be required using a switch factor delay model.

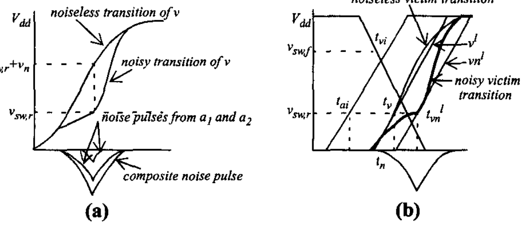

Figure 3. Victim waveform computation and alignment.

Although the specific example in Figure 4(a) is acyclical, it is, in general, possible that the causality graph is cyclical, such as the one shown in Figure 4(b). However, the nature of a cycle in the causality graph is such that it invloves the delay of *atleast one or more gates*(i.e., the delay of both gates *g2* and *g3* in Figure 4(b)). Therefore, if an event occurs at a particular node in a cycle and propagates along this cycle, the effect on itself will be delayed due to the delay of the gates in the cycle. Based on the properties of the proposed delay model, such self-induced noise cannot cause a signal transition to become earliar. This characteristic forms the basis for the proposed non-iterative solution for timing-window computation. By processing events in a time ordered fashion, the earliest unprocessed event can be completely determined, without the need to consider the impact of this event on itself. Since it is not always possible to determine which signal transition occurs first in a cycle, an incorrect ordering of event evaluation along a cycle may occur, leading to the need for roll-back of time. However, we will show that such roll-back of time does not lead to multiple iterations over the cycle, but simply the adjustment of where in the cycle event processing is started. This ensures that the algorithm remains linear in its run time complexity.

It is important to note, however, that the non-iterative nature of the algorithm depends on the properties of the delay model which is explained in the next section.

3 DELAY MODEL AND PROPERTIES

As shown in Figure 2(a), the input of the victim driver has early and late events vi^e and vi^l, where an early event corresponds to a signal transition at the start of a timing window and a late event corresponds to a signal transition at the end of its timing window. Similarly, the aggressor driver inputs have early and late events ai^e and ai^l. The victim node has early and late *noiseless* events v^e and v^l and early and late *noisy* victim events vn^e and vn^l. The event pairs for different nodes are shown in Figure 2(a). The noisy events vn^e and vn^l at victim v are considered as victim input events for the fanout node *fo* and as aggressor input events for nodes coupled to the fanout node *fo*.

Each event $e = (t_e, s_e)$ is defined in terms of its arrival time t_e when the signal transition crosses the so-called *switching voltage* v_{sw} and transition time s_e which is the time interval between the 20% and 80% V_{dd} crossing times. Given the victim input early event $vi^e = (t_{vi}^e, s_{vi}^e)$ and late event $vi^l = (t_{vi}^l, s_{vi}^l)$, the timing window at that node is given by the time interval $[t_{vi}^e, t_{vi}^l]$. In order the ensure that all gate delays are positive, our delay model uses separate switching voltages for rising and falling transitions. We set the switching voltage for a rising transition $v_{sw,r} = V_{t,n}$, as shown in Figure 3(a), and the switching voltage for a falling transition $v_{sw,f} = V_{dd} - V_{t,p}$, where $V_{t,n}$ and $V_{t,p}$ are the NMOS and PMOS threshold voltages.

The noisy early and late events vn^e and vn^l are a function of the alignment of the noise pulses relative to the noiseless victim transition. In [8], it was shown that, for a late event computation, the interconnect delay is maximized by aligning all aggressor noise pulses at the point where the noiseless victim transition reaches $v_{sw} + v_n$, where v_n is the height of the composite noise pulse (Figure 3(a)). Similarly, for early events, the peak of the composite noise pulse is aligned at the point where the noiseless victim transition reaches $v_{sw} - v_n$. If the alignment of the noise pulses is constrained by timing windows, an optimal alignment may not be possible, thereby reducing the impact of noise on the arrival time. Also, if the noise pulse height exceeds $V_{dd} - v_{sw}$ for a late event, the optimal alignment is no longer well defined since $v_{sw} + v_n > V_{dd}$. In this case, we consider

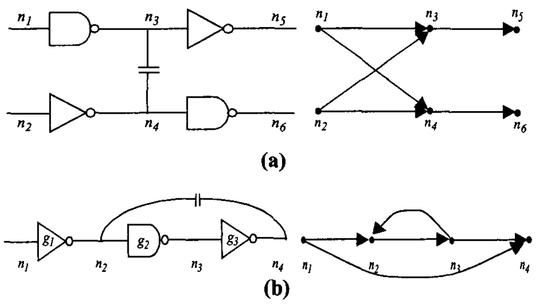

Figure 4. Simple circuits and their causality graph representation.

the optimal alignment to be when the peak of the composite noise pulse is positioned at the end of the noiseless victim transition, where it reaches V_{dd}. Similarly, for an early event, we consider the optimal alignment of a noise pulse that exceeds v_{sw} to be at the start of the noiseless victim transition.

We use the following model to compute a noisy victim late event $vn^l = (t_{vn}^l, s_{vn}^l)$, given a noiseless victim late event $v^l = (t_v^l, s_v^l)$ and one or more noise pulses. We define the noisy victim event $vn^l = (t_{vn}^l, s_{vn}^l)$ such that t_{vn}^l is equal to the time point where the noisy transition crosses v_{sw} and such that $s_{vn}^l = s_v^l$. As illustrated in Figure 3(b), we have effectively shifted the ramp approximation of the noiseless victim transition to the point where the noisy victim transition crosses the switching voltage, while keeping its transition time constant. A noisy early event is computed similarly. Although a number of other models for abstracting the arrival time and transition time from the noisy transition are possible, the described model is a common model that is conservative for typical noise pulse shapes and has useful properties for timing window computation.

We present the following important properties of the discussed delay noise model:

Property 1. A signal transition at a victim input with arrival time t_{vi} produces a noiseless signal transition at the victim node with arrival time t_v which is later than t_{vi}: $t_v > t_{vi}$.

Property 2. A signal transition at an aggressor input with arrival time t_{ai} produces a noise pulse on a victim node where the time of the start of the noise t_n is later than t_{ai}: $t_n > t_{ai}$.

Both Property 1 and Property 2 follow directly from setting separate switching voltages $v_{sw,r}$ and $v_{sw,f}$ such that the output transition of a gate starts after the input transition reaches the switching voltage.

Property 3. If the aggressor input arrival time is earlier than the noiseless victim event, $t_{ai} < t_v$, it follows that the noisy victim arrival time t_{vn} will fall after the aggressor input event, $t_{vn} > t_{ai}$.

Property 3 follows directly from Property 2 and from the fact that a noise pulse cannot hasten a victim transition to occur before the start of a noise pulse. Property 3 holds for early events but a similar property exists for late events; intuitively, it states that an aggressor input transition cannot cause a victim to switch earlier than itself. Next, we introduce a property which is useful for proving that the proposed method has a linear run time.

Property 4. Consider a noise pulse with optimal input alignment t_{ai}^{opt} resulting in a noisy late event time t_{vn}^{opt}. Given that the slope of the leading edge of the noise pulse is steeper than the slope of the victim transition, we consider a suboptimal noise pulse alignment t_{ai} and the subsequent noisy *late* event vn with event time t_{vn}. If $t_{ai} > t_{ai}^{opt}$, it follows that $t_{vn} = t_v$ and if $t_{ai} \le t_{ai}^{opt}$, it follows that $t_{vn} \le t_{vn}^{opt}$.

```
timing_window_computation()
schedule early and late events for primary inputs
for (t = 0; scheduled events left; t++) {
  while (events left at time t) {
    select event e at node n from queue at time t;
    for (all fanout nodes l of n) {
      if (e is an early event) victim_early_event(e);
       else if (e is a late event && for all fanin nodes of l a late event is defined,
               victim_late_event(e);
      for (all nets m coupled to l)
       if (e is an early event) aggressor_early_event(e);
  }
  select check point event cp at time t;
  if (cp is an early cp event) cp_early_event(cp);
  else if (cp is a late cp event) cp_late_event(cp);
  }
}
```

Figure 5. Time sort algorithm for computing timing windows.

Property 4 shows that if the noise pulse is aligned earlier than its optimal alignment t_{ai}^{opt}, the impact of the noise on the delay of the victim will be reduced and hence $t_{vn} \le t_{vn}^{opt}$. On the other hand, if the alignment of the noise is later than the optimal alignment t_{ai}^{opt}, the noise will not impact the delay of the victim and hence $t_{vn} = t_v$.

4 TIME SORT ALGORITHM

In the proposed algorithm, events are processed ordered by non-decreasing arrival times. We refer to the time t for which events are being processed as the *current time.* Due to space restrictions, we limit our discussion to nets with a single aggressor, although the algorithm can be easily extended to nets with multiple aggressors. Figure 1(a) shows the victim net l with aggressor net m. The victim net has a victim driver gate with one or more input nets n and the aggressor net has an aggressor driver with one or more inputs k. However, for clarity, we will restrict our explanation to aggressor drivers with a single input since the extension to aggressor drivers with multiple inputs is straightforward. The algorithm processes early and late events at victim driver input node n and aggressor driver input node k and computes a new noisy event at the victim node l. An event at node n is first considered as a victim input event, resulting in a new event at node l, and is then considered as an aggressor input event resulting in a possible change of the early and late events at node m.

The overall algorithm is shown in Figure 5 and consists of 5 event processing steps. We first illustrate the general approach of the algorithm and then discuss each procession step in detail in Section 4.1. All early and late events are processed as victim input events in functions *victim_early_event*() and *victim_late_event*(). For a victim input event vi at node n, the noiseless event v at the victim node l is first computed. If the early event ai at aggressor input k is defined (i.e. it has an early event time $t_{ai}^e < t$), we compute the optimal alignment time t_{ai}^{opt} at input k for this aggressor m, as illustrated in Figure 6(a) for early events. If the optimal alignment time $t_{ai}^{opt} \le t$, we superimpose the

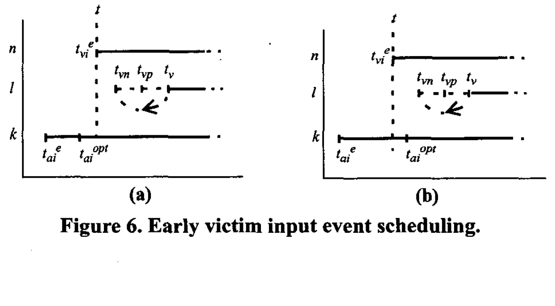

Figure 6. Early victim input event scheduling.

noise from aggressor *m* on the noiseless event *v* at node *l* and a new noisy event *vn* at *l* is computed and scheduled. During the computation, the noise alignment is constrained by the timing window at node *k*. On the other hand, if the optimal aggressor input alignment time $t_{ai}^{opt} > t$, as shown in Figure 6(b), there exists ambiguity as to whether the aggressor can transition at the optimal time, since the time window at *k* could end before t_{ai}^{opt}. In this case, we schedule a so-called *check point event cp* at a point in time when the ambiguity is resolved. The early and late check point events are scheduled much like regular events and are processed by functions *cp_early_event*() and *cp_late_event*(). For early events, only a single check point is needed whereas for late events, multiple check points may be required. However, we show in Section 4.1 that the maximum number of required check points is independent of the circuit size and can be considered a constant for run time complexity analysis. In practice, the number of check points was found to be small.

We consider the impact of noise from aggressor *m* on node *l* only if the aggressor input *k* has a defined early event at time *t* as shown in Figure 6. If the early event at aggressor input *k* is later than the victim input event *vi* at node *n* (i.e. $t_{ai}^{e} > t_{vi}$), we initially schedule event *vn* without considering the noise from aggressor *m* and then later update event *vn*, if necessary, when processing the early event *ai* at aggressor input *k* in function *aggressor_early_event*(). Early and late events at a node maybe scheduled multiple times, due to processing of new aggressor early events when $t_{ai}^{e} > t_{vi}$. However, we will show that the worst-case number of event updates is limited by the number of aggressor nets that couple to a victim net and is independent of the circuit size and is typically small. Also, the sorting and scheduling of new events can be performed in constant time by discretizing time.

Although an event can be *scheduled* multiple times, the objective of the algorithm is to *process* each event only once at a node. It can be shown that the following two conditions are sufficient to ensure for this:

- **Condition 1**: When scheduling a new event, the arrival time t_{vn} of this event *vn* falls after the current time *t*: $t_{vn} > t$.
- **Condition 2**: When scheduling a new event *vn*, a previously scheduled event of that type has not already been processed.

Condition 1 ensures that a newly scheduled event *vn* falls in the future and does not require a roll-back of the current time. Condition 2 ensures that a new event is not scheduled after a previous event of the same type has already been processed for a node. As shown in Section 4.1, Condition 1 and 2 are satisfied in the proposed algorithm in all but two cases. Under certain timing window alignments, it is possible that in function *victim_early_event*(), Condition 1 is not met, and in function *aggressor_early_event*(), Condition 2 is not met. In these cases, the current time *t* is rolled back and the event in question, as well as other processed events that it spurred, must be reprocessed. We will show, however, that for a particular circuit node, time roll-back can occur only once and the number of reprocessed events is fixed based on the topology of the circuit, and is not a function of the circuit size. Hence, the worst-case run time of the algorithm is linear with the circuit size. Also, we found that time roll-back and event reprocessing are extremely rare in actual circuits, while the number of iterations required in the existing approach is often quite significant. Finally, since the algorithm is non-iterative and does not operate using an initial overlapping or non-overlapping timing window assumption, it does not exhibit the multiple solution problem of the iterative approach.

4.1 Event Processing and Scheduling

In this Section, we present the event processing steps in more detail and also discuss their adherence to Condition 1 and 2.

```
victim_early_event($vi^e$)
1  if (early event time $t_{ai}^e$ is defined) {
2    compute optimal switching time $t_{ai}^{opt}$ at k resulting in noisy event time $t_{vn}$
3    if($t_{ai}^{opt} \le t$) {
4      if($t_{ai}^e \le t_{ai}^{opt} \le t_{ai}^l$) compute event vn at l, with optimal alignment
5      else compute event vn at l, with nearest alignment in[$t_{ai}^e$, $t_{ai}^l$]
6      if ($t_{vn} < t_{vp}$) remove vp and schedule vn
7    } else {
8      if ($t_{ai}^l$ is defined) {
9        compute noisy event vn at l, with aggressor input aligned at $t_{ai}^l$
10       if ($t_{vn} < t_{vp}$) remove vp and schedule vn
11     } else
12       if ($t_{vn} < t_{vp}$) remove vp and schedule checkpoint event cp at time $t_{vn}$
13 } else {
14   schedule event v
15 }
```

Figure 7. Victim early event processing.

Victim Early Event

In this scheduling step, we consider the response of the victim node *l* due to an early event at one of its driver inputs *n*, as shown in pseudo-code in Figure 7. Note that at time *t* node *l* may already have an early event scheduled, due to one of the other driver inputs of *l*. We first compute the noiseless victim event *v* at *l* due to event vi^e at *n*, as illustrated in Figure 6(a). We then consider the aggressor *m* and its driver input *k*. If the aggressor input early event time t_{ai}^{e} is defined (i.e. $t_{ai}^{e} < t$), we determine the noise pulse injected from node *m* on node *l* and find the switching time t_{ai}^{opt} at node *k* that will result in the optimal alignment of this noise pulse (line 2). If $t_{ai}^{opt} < t$, as shown in Figure 6(a), we compute the earliest noisy event *vn*, constrained by the switching window [t_{ai}^{e}, t_{ai}^{l}] at node *k* (lines 4-5) and compare its arrival time t_{vn} with the current early event time t_{vp} (line 6). If $t_{vn} < t_{vp}$ we remove the existing event *vp* and schedule the new event *vn*. If $t_{ai}^{opt} > t$, we check if the event window at aggressor input *k* has ended. If it has, we align the noise based on the latest point in the aggressor input window t^{l}_{ai} (lines 8-10). Otherwise, there is ambiguity whether the optimal alignment can occur (since it falls in the future) and we schedule a check point event for node *l* at time t_{vn} (line 12), which is processed in function *cp_early_event*(). The victim noisy event is then scheduled when this check point event is processed and the ambiguity is resolved. The processing steps in *cp_early_event*() are similar to those in *victim_early_event*() and the discussion is omitted for brevity. Note that if the early event on *k* is not defined, we ignore this aggressor and compute its effect on node *l* in function *aggressor_early_event*().

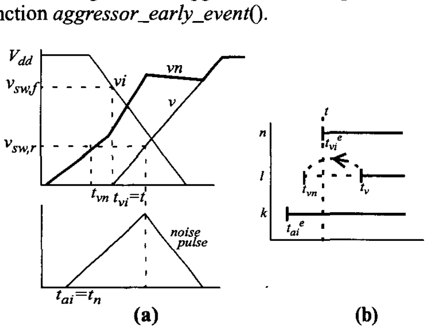

Figure 8. Event scheduling for noise heights exceeding v_{sw}.

```
victim_late_event($vi^l$)
1  if (early event time $t_{ai}^e$ is defined) {
2      compute switching time $t_{ai}^{opt}$ at k for optimal noise alignment
3      if ($t_{ai}^{opt} \le t$) {
4          compute vn at l, with k at nearest alignment in [$t_{ai}^e$, $t_{ai}^l$]
5          schedule vn
6      } else {
7          if ($t_{ai}^l$ is defined) {
8              compute noisy event vn at l, with aggressor input k aligned at $t_{ai}$
9              schedule vn
10         } else {
11             compute late event vn' at l, with aggressor input k aligned at t
12             schedule check point event cp at time $t'_{vn}$ for node l
13         }
14     }
15 } else {
16     schedule event v
17 }
```

Figure 9. Victim late event processing.

We only schedule event *vn* if a previously scheduled event *vp* has an event time $t_{vp} > t_{vn}$ and has not been processed, which satisfies Condition 2. Also, if the noise height is less than switching voltage v_{sw}, it is clear that the noisy victim event time t_{vn} must fall after the victim input early event time t_{vi} and hence, Condition 1 is met. However, if the noise pulse is larger than the switching voltage v_{sw}, it is possible that $t_{vn} < t$, if $t_{ai}^e < t_{vi}^e$ and Condition 1 is not met, as shown in Figure 8(a). This means that the victim early time t_{vn} occurs before the victim input early event time $t_{vi}^e = t$ as shown in Figure 8(b). In this case, events with arrival times in the window [t_{vn}, t_{vi}^e], that were already processed and that depend directly or indirectly on the transition at node *l* must be re-processed. We recursively examine all events at the fanout nodes of net *l* or coupled to the fanout nodes of net *l* and reschedule any that fall in the time window [t_{vn}, t_{vi}^e]. The rescheduling has two important properties. An early event *vn* at a victim node will be rescheduled only once. This is intuitively clear from the fact that Property 1 and 3 ensure that a change in *vn* cannot cause one of the aggressor *input* nodes of *l* or its victim input node to become earlier than itself. A detailed proof is given in [9]. Secondly, the number of rescheduled events is independent of circuit size. This is clear from the fact that the size of the rescheduling window [t_{vn}, t_{vi}^e] is bounded by the transition time of the leading edge of the noise pulse as illustrated in Figure 8(a) which is independent of circuit size. The number of events that need to be scheduled is therefore independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be very small in practice and required no more than 4 events to be rescheduled in all the benchmark circuits.

Victim Late Event

In this scheduling step, we consider the response of the victim *l* due to a late event at one of its driver inputs *n*. The computation is analogous to *victim_early_event*(), and is shown in pseudo-code in Figure 9. If the aggressor input early event ai^e has not occurred at time *t*, we compute the noiseless victim transition and schedule it (line 16). On the other hand, if $t_{ai}^e < t$, we compute the optimal alignment time for aggressor input *k* for a noisy late event at the victim node. If the optimal alignment $t_{ai}^{opt} \le t$, as is illustrated in Figure 10(a), we compute the noise on node *l* with the optimal alignment at the aggressor input *k* constrained by its switching window of [t_{ai}^e, t_{ai}^l] (lines 4,5). If $t_{ai}^{opt} > t$ and the switching window of *k* has not yet ended at time *t*, as shown in Figure 10(b), there is again ambiguity about whether the optimal alignment can occur and we schedule a check point for a future point of time to determine if the optimal alignment is possible. However, contrary to the early event time computation, the check point is scheduled at the noisy victim late event time t'_{vn}, computed with the aggressor input aligned at time *t* at node *k* (lines 11, 12). When this check point event is processed, we schedule the victim noisy late event if the ambiguity is resolved or we place another check point if there is ambiguity still. The processing steps in *cp_late_event*() are similar to those in *victim_late_event*() and the discussion is omitted for brevity. Since event time t'_{vn} is computed with the most optimal alignment that is certain to be possible, t'_{vn} is a lower bound on the final value of t_{vn}.

It is important to note that a late event at the victim input *n* is only processed if all other driver inputs of victim node *l* have defined late events, as shown in line 7 in Figure 5. This is necessary to ensure that Condition 2 is met. If one or more driver inputs n_x have not completed their event window at time *t*, the latest noiseless event on *l* will lie after the current time *t*, based on delay model Property 1, and it is not necessary to process the event on *n*. Therefore, a late event *vi* is processed only for the fanin node of *l* with the latest early event time. Also, based on delay model Property 1, the noiseless victim event time t_v due to a processed late event *vi* falls after *t*. Since the impact of noise on node *l* will only increase the late arrival time, the noisy late event time $t_{vn} > t$, satisfying Condition 1. To satisfy Condition 2, it is sufficient to note that event *vn* is the first late event scheduled for node *l*. Hence, no earlier scheduled event could have been processed.

Aggressor Early Event

In this step, we compute the impact of an aggressor early event on early and late victim events. We only process an aggressor early event if the victim event has already been processed. If the victim input early or late event has not been processed, the impact of the aggressor is accounted for in the functions *victim_early_event*() and *victim_late_event*(). When an aggressor early event is processed, the current time $t = t_{ai}^e$, as illustrated in Figure 11. We first compute the optimal alignment time t_{ai}^{opt} at the aggressor input node *n*. If $t_{ai}^{opt} < t$, we compute the new victim event *vn* with aggressor input aligned at time $t_{ai}^e = t$ and schedule it. If the optimal alignment time falls after the current time *t*, there is again ambiguity on whether the optimal alignment can occur and we schedule a check point in similar fashion to function *victim_early_event*() and *victim_late_event*().

From Property 3 it follows that, if the noiseless event *v* is later than the current time, $t_v > t = t_{ai}$ then, $t_{vn} > t$, which satisfies Condition 1 for early events as shown in Figure 11(a). Also, if $t_v < t = t_{ai}$, the noise will not impact the victim early transition time, and $t_v = t_{vn}$. Hence, no new event will be scheduled. This satisfies both Condition 1 and 2 for early victim events. For late victim events, it is again easy to see that if the noise impacts the victim late transition time, it follows that $t_{vn} > t_{ai} = t$, which satisfies Condition 1 as shown in Figure 11(b). However, Condition 2 is not met for the com-

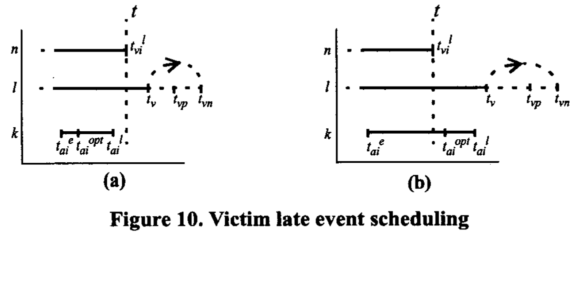

Figure 10. Victim late event scheduling

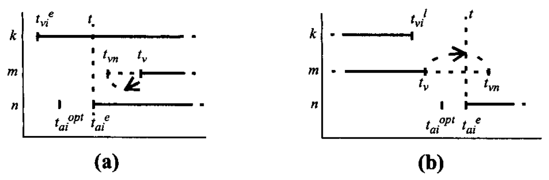

Figure 11. Early aggressor event processing for (a) early victim events and (b) late victim events.

putation of the victim late event, meaning that a previously scheduled late event *vp* at the victim *m* may have arrival time $t_{vp} < t$ and has already been processed. In this case, we need to reschedule the victim late event *vp* at node *m*, as well as any events dependent on it with event times in the window $[t_{vp}, t_{ai}^e]$. However, we show in [9] that this rescheduling again has the property that each event is rescheduled only once, and the number of rescheduled events is independent of circuit size, and hence, the run time complexity of the algorithm remains linear with circuit size.

5 RESULTS

The proposed time-sort algorithm was implemented and tested for the ISCAS85 benchmark circuits and a 64-bit ALU. All circuit were synthesized with a 0.18 μm Artisan library using Synopsys Design Compiler. The characteristics of these circuits are shown in Table 1. The average ratio of coupling capacitance to grounded capacitance for a net was 25%. Coupling capacitance was generated randomly between nodes in the circuit. Table 1 shows the percentage of nets with one or more couplings and the average number of coupling capacitances per net. For the larger circuits, an average number of couplings per net of 10 was used, which corresponds to the number of expected couplings for nets in high performance designs.

The traditional iterative approach was also implemented and Table 1 compares the results from the two methods. The reported run times are on a Pentium IV 1.8GHz PC running Linux. For the iterative method, the number of iterations needed for convergence ranged between 3 and 11. The improvement in run time ranged from 2.31 to 5.01 and increased with circuit size and the number of couplings in a circuit. Roll-back occurred only for circuit c7552, causing recomputation of two events for both cases of roll-back. This demonstrates that time roll-back is extremely rare and that in practice each net is processed only once. Table 1 also shows the percentage of events that required one or more check points (*% Check point events*) and the average number of check points for these events (*Avg # check points*). Note that a check point does not represent multiple processing of an event, but only multiple updates to an event time before the event is processed.

Table 1. Results for ISCAS85 combinational Circuits

Circuit	Iterative Approach		Proposed Time Sort Approach					speed up
	Run Time (sec)	# Iterations	Run Time (sec)	% Check Point Events	Avg # check points	# Roll Backs	# Events reprocessed	
c432	0.08	5	0.02	18.05	0.62	0	0	3.89
c499	0.33	5	0.12	19.54	0.57	0	0	2.71
c880	0.17	3	0.03	21.91	0.69	0	0	4.67
c1355	0.34	5	0.15	27.19	0.89	0	0	2.31
c1908	0.31	6	0.08	26.64	0.93	0	0	3.59
c2670	0.65	7	0.19	21.83	0.62	0	0	3.27
c3540	1.37	6	0.52	31.30	1.36	0	0	2.63
c5315	1.43	5	0.45	30.52	1.01	0	0	3.18
c6288	4.9	11	0.98	25.89	1.12	0	0	5.01
c7552	2.81	6	0.93	27.13	1.16	2	4	2.99
alu64	3.65	6	0.73	31.14	1.29	0	0	4.98

Finally, we compared the timing analysis results for the two methods. The iterative method converged to different solutions depending on the initial overlap assumption in all of the 11 circuits. The percentage difference in the window size ranged from 1% to 6%. As expected, we found that the proposed time sort algorithm obtained a result that falls between the two solutions of the iterative approach.

6 CONCLUSIONS

We have presented a new timing window computation method for static timing analysis in the presence of cross-coupling noise. The proposed method is based on delay noise computation using linear superposition and early and late event processing ordered by arrival times. We have shown that the proposed algorithm has linear run time with circuit size as opposed to the $O(n^2)$ worst-case complexity of the current iterative approach. Since the algorithm is non-iterative and does not use an initial coupling assumption for timing window computation it eliminates the multiple solution problem present in the iterative approach. We demonstrated the proposed method on benchmark circuits with extensive capacitive coupling and show that it obtains significant speedup over an iterative method.

Acknowledgements

This research was supported by SRC contract 2001-HJ-959 and NSFgrant CCR-0205227.

References

[1] L. H. Chen, M. Marek-Sadowska, "Aggressor Alignment for Worst-Case Crosstalk Noise," IEEE Trans. on Computer-Aided Design, vol. 20, pp. 612-621, May 2001.

[2] R. Arunachalam, K. Rajagopal, L. T. Pileggi, "TACO: timing analysis with coupling" Proc. DAC, pp. 266-269, June 2000.

[3] A. B. Kahng, S. Muddu, E. Sarto, "On Switching Factor Based Analysis of Coupled RC Interconnects," DAC, pp 266-269, June 2000.

[4] P. Chen, D. A. Kirkpatrick, K. Keutzer, "Miller Factor for Gate-Level Coupling delay calculation", Proc. ICCAD 2000.

[5] S. Sapatnekar, "A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing," IEEE Trans. on CAD, Vol 19, No. 5, pp. 550 - 559, 2000.

[6] P. Chen, D. A. Kirkpatrick, K. Keutzer, "Switching Window Computation for Static Timing Analysis in the Presence of Crosstalk Noise," ICCAD, pp. 331-337, 2000.

[7] H. Zhou, N. Shenoy, W. Nicholls, "Timing Analysis with crosstalk as fixed points on Complete Lattice", Proc. ACM/IEEE DAC 2001.

[8] F. Dartu, L. T. Pileggi, "Calculating Worst-Case Gate Delays Due to Dominant Capacitance Coupling," Proc. ACM/IEEE DAC, 1997

[9] B. Thudi, D. Blaauw, "Efficient Switching Window Computation for Cross-Talk Noise," Proc. ACM/IEEE TAU 2002

[10] P.D. Gross, R. Arunachalam, K. Rajagopal, L.T. Pileggi, "Determination of worst-case aggressor alignment for delay calculation," Proc. IEEE/ACM ICCAD, pp. 212-219, November 1998.

[11] S. Sirichotiyakul, D. Blaauw, C. Oh, R. Levy, V. Zolotov, J. Zuo, "Driver Modeling and Alignment for Worst-Case Delay Noise", Proc. ACM/IEEE DAC, pp. 720-725, 2001.