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].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2003, June 2-6, 2003, Anaheim, California, USA. Copyright 2003 ACM 1-58113-688-9/03/0006...$5.00.

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 window, 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.](image-url)
• 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 ignored. 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 acyclic, 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.

Although the specific example in Figure 4(a) is acyclic, 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 involves the delay of at least one or more gates(i.e., the delay of both gates \( g_2 \) and \( g_3 \) 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 earlier. 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 \( v^e \) and \( v^l \), 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 \( a^e \) and \( a^l \). The victim node has early and late noiseless events \( v^e \) and \( v^l \) and early and late noisy victim events \( v_{n}^e \) and \( v_{n}^l \). The event pairs for different nodes are shown in Figure 2(a). The noisy events \( v_{n}^e \) and \( v_{n}^l \) at victim \( v \) are considered as victim input events for the fanout node \( f_0 \) and as aggressor input events for nodes coupled to the fanout node \( f_0 \).

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 \( e^r = (t_{e,r}, s_{e,r}) \) and late event \( e^l = (t_{e,l}, s_{e,l}) \), the timing window at that node is given by the time interval \( [t_1, t_2] \). In order to 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 \( v_{sw,r} = V_{tr} \) as shown in Figure 3(a), and the switching voltage for a falling transition \( v_{sw,f} = V_{td} \) where \( V_{tr} \) and \( V_{td} \) are the NMOS and PMOS threshold voltages.

The noisy early and late events \( v_{n}^e \) and \( v_{n}^l \) 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 \( v_{sw} + v_{np} \), where \( v_{np} \) 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 \( v_{sw} - v_{np} \). 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 \( v_{sw} + v_{np} > V_{dd} \). In this case, we consider
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 \( v_{nl}^l = (t_{vn}^l, s_{vn}^l) \), given a noiseless victim late event \( v^l = (t^l, s^l) \) and one or more noise pulses. We define the noisy victim event \( v_{nl}^l = (t_{vn}^l, s_{vn}^l) \) such that \( t_{vn}^l \) is equal to the time point where the noisy transition crosses \( v_{sw} \) and such that \( s_{vn}^l = s^l \). 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 \( t_a \) is later than \( t_{ai} \); \( t_a > t_{ai} \).

Both Property 1 and Property 2 follow directly from setting separate switching voltages \( v_{swr} \) and \( v_{swf} \) 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, \( t_v > t_{ai} \).

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 \( v_{nl} \) with event time \( t_{vn} \). If \( t_{ai} > t_{ai}^{opt} \), the propert...
noise from aggressor \( m \) on the noiseless event \( v \) at node \( l \) and a new noisy event \( v_n \) 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 \( cp_{early}(t) \) and \( cp_{late}(t) \). 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 \( v_i \) at node \( n \) (i.e. \( t_{ai}^{opt} > t_{vn} \)), we initially schedule event \( v_n \) without considering the noise from aggressor \( m \) and then later update event \( v_n \), if necessary, when processing the early event \( ai \) at aggressor input \( k \) in function \( aggressor_early(t) \). Early and late events at a node may be scheduled multiple times, due to processing of new aggressor early events when \( t_{ai} > t_{vn} \). 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 \( v_n \) falls after the current time \( t \): \( t_{vn} > t \).
- **Condition 2**: When scheduling a new event \( v_n \), a previously scheduled event of that type has not already been processed.

Condition 1 ensures that a newly scheduled event \( v_n \) 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 \( victim_early(v) \), Condition 1 is not met, and in function \( aggressor_early(t) \), 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.

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 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 \( v^e \) 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 \( t_{ai}^{opt} \) is defined (i.e. \( t_{ai}^{opt} < t \)), 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 \( v_n \), constrained by the switching window \( t_{ai}^{opt}, t_{ai}^{opt} \) at node \( k \) (lines 4-5) and compare its arrival time \( t_{vn} \) with the current early event time \( t_{vn} \) (line 6). If \( t_{vn} < t_{vn} \), we remove the existing event \( v_n \) and schedule the new event \( v_n \). If \( t_{vn}^{opt} > t \), 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 \( t_{ai}^{opt} \) (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 at node \( l \) (line 12), which is processed in function \( cp_{early}(t) \). The victim noisy event is then scheduled when this check point event is processed and the ambiguity is resolved. The processing steps in \( cp_{early}(t) \) are similar to those in \( victim_early(v) \) 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 \( aggressor_early(v) \).

Figure 8. Event scheduling for noise heights exceeding \( v_{sw} \).
Secondly, the number of rescheduled events is independent of circuit node to become earlier than itself. A detailed proof is given in [9].

Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Furthermore, the rescheduling window was found to be independent of circuit size, ensuring the linear run time of the algorithm. Moreover, the noise pulse as illustrated in Figure 8(a) which is independent of timing node for aggressor input aligned at t.

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 at n has not occurred at time t, we compute the noiseless victim transition and schedule it (line 16). On the other hand, if t ≤ t, we compute the optimal alignment time for aggressor input n for a noisy late event at the victim node. If the optimal alignment t opt n ≤ t, as is illustrated in Figure 9, we compute the noise on node l with the optimal alignment at the aggressor input k constrained by its switching window of [tv, tvopt] (lines 4, 5). If t 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 tv 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 tv vn is computed with the most optimal alignment that is certain to be possible, tv vn is a lower bound on the final value of tv 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, 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 tv i 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 tvn > 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 = 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 victim v is later than the current time, t v > t ai, the noise will not impact the victim early transition time, and t vn = t ai. 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 comm.
independent of circuit size, and hence, the run time complexity of rescheduled only once, and the number of rescheduled events is that this rescheduling again has the property that each event is 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 6.09 seconds. The proposed method obtained significant speedup over an iterative method. We demonstrated the proposed method on benchmark circuits with extensive capacitive coupling and show that it obtains significant speedup over an iterative method.

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 improvement in run time ranged from 2.31 to 6.09 seconds.

Table 1. Results for ISCAS85 combinational Circuits

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Iterative Approach</th>
<th>Proposed Time Sort Approach</th>
<th>speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Run Time (sec)</td>
<td>% Check Point Events</td>
<td>Avg # check points</td>
</tr>
<tr>
<td>c432</td>
<td>0.08</td>
<td>0.12</td>
<td>0.02</td>
</tr>
<tr>
<td>c499</td>
<td>0.33</td>
<td>0.12</td>
<td>0.02</td>
</tr>
<tr>
<td>c880</td>
<td>0.17</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>c1355</td>
<td>0.34</td>
<td>0.15</td>
<td>0.02</td>
</tr>
<tr>
<td>c1908</td>
<td>0.31</td>
<td>0.08</td>
<td>0.02</td>
</tr>
<tr>
<td>c2670</td>
<td>0.65</td>
<td>0.19</td>
<td>0.02</td>
</tr>
<tr>
<td>c3540</td>
<td>1.37</td>
<td>0.52</td>
<td>0.02</td>
</tr>
<tr>
<td>c5315</td>
<td>1.43</td>
<td>0.45</td>
<td>0.02</td>
</tr>
<tr>
<td>c6288</td>
<td>1.91</td>
<td>0.98</td>
<td>0.02</td>
</tr>
<tr>
<td>c7552</td>
<td>2.81</td>
<td>0.93</td>
<td>0.02</td>
</tr>
<tr>
<td>alu64</td>
<td>3.65</td>
<td>0.73</td>
<td>0.02</td>
</tr>
</tbody>
</table>

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 NSF grant CCR-0205227.

References