ClariNet: A noise analysis tool for deep submicron design

Rafi Levy*, David Blaauw, Gabi Braca*, Aurobindo Dasgupta, Amir Grinshpon*, Chanhee Oh, Boaz Orshav, Supamas Sirichotiyakul, and Vladimir Zolotov

Motorola Inc. Austin, TX, *Motorola Semiconductor Israel Ltd. Tel Aviv, Israel

Abstract

Coupled noise analysis has become a critical issue for deep-submicron, high performance design. In this paper, we present, ClariNet, an industrial noise analysis tool, which was developed to efficiently analyze large, high performance processor designs. We present the overall approach and tool flow of ClariNet and discuss three critical large-processor design issues which have received limited discussion in the past. First, we present how the driver gates of a coupled interconnect network are represented with accurate linear models. Second, we show how to speed the analysis of large designs by using noise filters based on reduced interconnect representations and then pruning the nets coupled to a signal net. Third, we show how to incorporate logic and timing correlations into noise analysis to reduce its pessimism. We present the results from several industrial circuits, including a large high performance microprocessor design and a DSP design.

1 Introduction

For large, high performance designs, functional noise failures have become a significant design and verification issue. Due to the non-uniform scaling of interconnects cross-coupling capacitance between wires is becoming an increasingly dominant fraction of total wire capacitance, causing an increase in cross-coupled noise effects. At the same time, the quest for higher performance circuits has pushed designers to use more aggressive but less noise-immune circuit structures, such as dynamic logic and unbuffered latches. The combination of higher cross-coupling noise and more noise sensitive circuit structures has resulted in a significant noise problem, making effective noise analysis methods critical.

During noise analysis, nets are divided into two classes: *aggressor* and *victim* nets. A victim net is a net on which noise is injected by one or more neighboring nets through cross-coupled capacitances. The nets that inject noise onto a victim net are considered its aggressor nets. We broadly categorize noise into two types: *functional noise* and *delay noise*. Functional noise occurs when a victim net is intended to be at a stable value, and noise is injected on to the net, causing it to glitch. The glitch may propagate to a state element, such as a dynamic node or latch, altering the circuit state and causing a functional failure. The second type, delay noise, occurs when victim and aggressor nets simultaneously transition causing the transition delay of the victim net to be altered. In this paper, we focus only on functional noise analysis.

DAC 2000, Los Angeles, California

We present a new analysis tool called ClariNet. To address the large number of nets in a processor design, ClariNet uses linear models of the aggressor and victim driver gates and simulates the resulting linear circuit with PRIMA[4]. Noise from different aggressors of a victim net is combined using linear superposition and is propagated through the victim receiver gate. If the noise at the output of the receiver gate exceeds a predetermined noise threshold, a noise failure is reported to the user. For efficiency, the linear models of the driver gates and the noise immunity of the receiver gates are stored in pre-characterized tables.

In this paper, we focus on three critical issues that come to the forefront in a effective noise analysis tool: accurate and efficient linear modelling of aggressor and victim driver gates, efficient noise filters and reducing the pessimism of noise analysis through the use of logic and timing correlations in the circuit.

Aggressor and victim driver models: Accurate linear models for switching gates have been proposed [5] and can be used to model the aggressor driver gate. However, the victim driver in noise analysis is stable and therefore requires a separate linear model. We present a new algorithm for efficiently generating an accurate linear model for a victim driver. Our algorithm automatically detects the driver state which results in the lowest gate conductance and therefore the worst case noise analysis.

Depending on the structure of the victim driver gate, it might not precharge a victim net to the full rail voltage (Vdd or Gnd). This introduces a voltage drop in addition to the noise that is injected by aggressor nets. We present a new algorithm to efficiently determine the maximum and minimum pre-(dis)charge voltage of a victim driver gate and show how to incorporate this additional voltage drop in noise analysis.

Noise Filters: We present a number of new *interconnect reduction* techniques that allow very fast filtering of nets which have clearly less than the allowable noise voltage. We propose three successive circuit structures, each with increasing circuit detail and analysis run time. The filters are guaranteed to be conservative, meaning they will overestimate the actual noise. We also propose methods to prune the number of aggressor nets that couple to a victim net with a small impact on accuracy of the analysis. Using these techniques, ClariNet can process more than 1000 nets per minute, allowing it to analyze a complete high performance microprocessor in a few hours.

Logic and Timing Correlations: Since noise analysis uses an inherently conservative approach, many reported violations are unrealizable in practice. It is therefore essential that a noise analysis tool reduces the pessimism of the analysis as much as possible by accounting for the *logic* and *timing correlations* of the victim and aggressor nets. Logic and timing correlations constrain aggressor and victim nets such that only a subset of all aggressors can switch simultaneously and inject noise into a victim net. Traditionally, timing windows of aggressor nets have been used for timing correlations. In this paper we also consider static windows and observability windows for victim nets in order to further reduce the number of falsely reported failures. For logic correlations, we propose an efficient BDD-based algorithm to enumerate the number of aggressor subsets satisfying user specified logic correlations.

2 Overview and Previous Work

Dynamic noise analysis methods were first introduced [7]. In these methods, the worst case AC noise pulse that occurs on a net is calculated for every signal net. The receiver gate of the signal net is simulated with the AC noise pulse and is checked for noise attenuation. In [6] a method is proposed to propagate noise from one net to the next, as shown in Figure 1. Gate *g1* is simulated with noise at its input net *n1*, and the resulting noise at the output of *g1* is added to the cross-coupling noise induced on net *n2*. Whenever noise is propagated through a gate, the gate is verified to attenuate the noise. This analysis method requires all the circuit elements to be processed in topological order from the inputs to the outputs of the circuit, and multiple iterations may be needed to converge in the presence of feedback in the circuit.

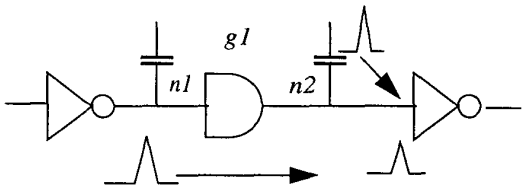


Figure 1. Noise due to propagation and coupling

A simplified approach that is commonly used in industrial settings is to simulate gate *g1* with the noise pulse on net *n1* and check that the output of *g1* does not exceed a given allowable global noise threshold, V_{Allow}. This noise threshold is set such that every gate satisfying the design rules attenuates noise if its output noise voltage is less than V_{Allow}. When analyzing net *n2*, the propagated noise from *g1* is conservatively assumed to be V_{Allow}. The effective noise on the net is then the sum of the cross-coupled noise and V_{Allow}. This approach allows independent analysis of nets *n1* and *n2*, and eliminates the need for a topological order of nets and iteration over feedback loops for convergence.

In ClariNet, we use this second approach for efficiency reasons. However, the presented algorithms for driver modeling, filtering, and accounting for timing and logic correlations apply to either approach. We analyze four types of noise: *low-undershoot, low-overshoot, high-undershoot* and *high-overshoot*. Low (high) noise refers to noise when the victim state is low (high). Undershoot (overshoot) noise refers to noise when the aggressor nets are falling (rising).

ClariNet iterates over all nets to analyze their noise. First, the victim net is reduced to a simplified network which is guaranteed to overestimate the noise. The calculated noise is compared against a designer specified acceptable noise value and if it is smaller, the victim net passes the noise analysis. Three filters with increasing complexity are used sequentially to quickly eliminate those nets that are guaranteed to pass noise analysis. If the net does not pass noise filters we linearize the aggressor and driver gates. The noise on the victim net is calculated using linear superposition where the noise induced by each aggressor is simulated while grounding the other aggressor voltage sources. We then use the logic and timing correlations to determine the subset of aggressors that induce the maximum possible noise. The combined noise from these aggressors is added to the propagated noise from the previous stage which is a predetermined noise threshold voltage. This aggregate noise pulse is propagated through the victim receiver gate by simulating it. If the noise peak at the output of the receiver gate is greater than the predetermined noise threshold, a noise failure is reported. For faster execution, the aggregate noise peak can be compared against a pre-characterized table of AC noise margins of the receiver gate.

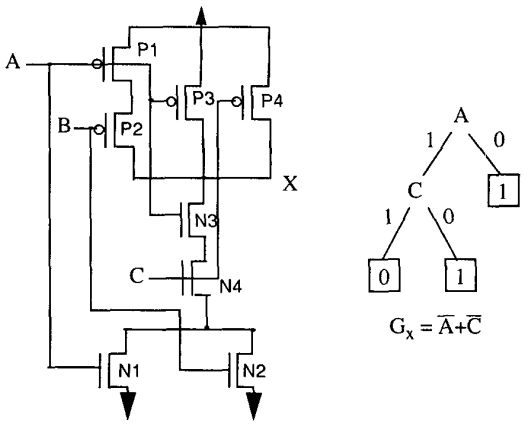
3 Victim Driver Models

In ClariNet, we model the driver gates by linear models which allows the use of fast linear simulation and the use of linear superposition. For the aggressor we use a standard Thevenin model[5]. For the victim net, the driving gate has a table state and all conducting transistors are operating in the linear region. The gate is therefore accurately modeled with a linear resistor, called the *holding resistance* with $R = 1/g_{ds}$, where g_{ds} is the channel conductance. For a complex gate, the equivalent conductance of a network of transistors is needed. Also, the holding resistance of such a gate depends on its input state. Therefore, the worst case input state, resulting in the highest holding resistance, must be determined. These two issues are solved in ClariNet as follows.

Efficient holding resistance calculation: Given an input state of a gate we represent each ON transistor on a path from the output node to the rail by its equivalent linear resistor and disconnect all other transistors. The equivalent linear resistance of a transistor is obtained using pre-calibrated tables where resistance is stored as a function of transistor width. The equivalent holding resistance is calculated by performing iterative series/parallel reductions and star-delta transformations. If the network cannot be solved using these simple reductions, the holding resistance is calculated by solving a set of linear equations.

During a noise spike at the gate output node, non-linearity of the transistor causes an increase in its channel conductance. Hence, to ensure a conservative analysis, the pre-calibrated resistance is generated using the channel conductance of the devices biased at $V_{ds}=V_n$, where V_n is the expected worst-case allowable noise.

Worst-case input state calculation: We propose the following



R(P1) = R(P2) = 400, R(P3) = 200, R(P4) = 240
$H_x = \overline{A}*\overline{B} + \overline{A} + \overline{C} = \overline{A} + \overline{C}$, $L_x = A*C*(A+B) = A*C$,
$G_x = H_x*\overline{L}_x = \overline{A} + \overline{C}$
Max holding resistance = max(R(P3), R(P4)) = 240

Figure 2. Equivalent holding resistance calculation.

algorithm for calculating the gate state which results in the worst case holding resistance, without enumerating all possible input states:

- Build the logic function for the gate output (x) to be high using BDDs. $\mathbf{G_x} = \mathbf{H_x} * \overline{\mathbf{L}}_\mathbf{x}$ where $\mathbf{H_x}$ and $\mathbf{L_x}$ are the pull-up and pull-down functions for output x.
- For each path in the BDD $\mathbf{G_x}$ that reaches a leaf node 1:

1. Assign known input values from the BDD path.
2. Find the transistor set **U**, where **U** contains all transistors which 1) have a gate node with an unknown state and 2) lie on a path from x to Vdd consisting of transistors in either ON or unknown state, i.e. connect to both Vdd and x through a potentially conducting path.
3. For each transistor in **U**, if a *feasible* assignment of the gate node of the transistor exists, assign the transistor gate node this state. A gate node assignment of a transistor **t** is said to be *feasible* if it turns OFF the transistor **t** and does not turn ON any other transistors in **U**. A feasible assignment guarantees that the gate conductance is not increased and is therefore conservative.
4. Repeat steps 2 and 3 until no more feasible assignment can be made.
5. If **U** is not empty, generate the set **S**, where **S** contains the enumeration of all possible state of the gate input node of transistors in **U**. **S** is therefore the set of worst-case input vectors associated with the current BDD path.

- Let **V** be the union of all **S** sets. For each input vector **v** in **V**, calculate the equivalent holding resistance $\mathbf{R_{hold}(v)}$ of the gate under input state **v** and return the maximum value.

In the BDD function $\mathbf{G_x}$ of the OAI gate in Figure 2 two BDD paths to a leaf node 1 exist: {A=0} and {A=1, C=0}. If we enumerate all unknown inputs, the set of input vectors that are considered is ABC={000, 001, 010, 011, 100, 110}. In the proposed algorithm, for the BDD path {A=0}, we identify that P1 and P3 are ON. In step 3, we find U={P2, P4}, both of which are turned OFF under the feasible assignment B=1, C=1, resulting in the vector ABC={011}. For the BDD path A=1, C=0, we identify that P1 and P3 are OFF while P4 is ON and **U** is empty. Hence the set of input vectors **V** which is considered for holding resistance calculation is ABC={011, 1X0}.

The complexity of the underlying problem, and hence the proposed algorithm is inherently exponential in the worst case. In practice, however, the algorithm is very efficient since it dramatically reduces the number of input vectors to be considered.

Victim Voltage Calculation: Special precharge (predischarge) driver structures are commonly used that do not driven the victim net to a full Vdd (or Gnd) voltage, but rather with a voltage drop. In this case, there already exists a DC noise equivalent to this voltage drop on the victim net, and the net is significantly more sensitive to additional noise. We propose a new algorithm to detect the maximum and minimum precharge and pre-discharge voltage of a victim driver. The maximum and minimum precharge (predischarge) voltage is used as the initial victim voltage for high (low)-overshoot and high (low)-undershoot noise analysis respectively. We examine the case when output x is high. The case where output x is low is analogous. We first construct the logic constraint function $G_x = H_x * \overline{L}_x$, which represents the input states that result in a high output state. In addition, we generated the logic constraint function $f_x = h_x * \overline{L}_x$, where h_x is the pull-up function of the driver gate with all NMOS transistors removed. The function f_x represents the input states that result in a high output state without any Vt drop. The maximum and minimum precharge voltage is now determined by comparing the satisfiability of G_x and f_x as shown in Table 1.

When G_x=0, there is no valid input assignment that precharges the gate output and the gate is disconnected. When G_x is satisfiable, and f_x=0 (the second row in Table 1), there does not exist a pull-up path that consists of only PMOS transistors, i.e. all conducting paths contain at least one NMOS transistor, and the gate always has a voltage drop. When f_x is satisfiable (the third and the fourth row in Table 1), pull-up paths that consist of only PMOS transistors exist and therefore the maximum voltage is Vdd. To

G_x	f_x	$G_x * \overline{f}_x$	Min Voltage	Max Voltage	Note
0	x	0	-1	-1	High output is not feasible
Satisfiable	0	-	V_{dd}-V_{t0n}	V_{dd}-V_{t0n}	Always has V_t drop
Satisfiable	Satisfiable	0	V_{dd}	V_{dd}	No V_t drop
Satisfiable	Satisfiable	Satisfiable	V_{dd}-V_{t0n}	V_{dd}	V_t drop possible

Table 1. Min. and max. precharge voltage calculation.

determine whether a voltage drop is also possible, we examine the function $G_x*\overline{f}_x$. When $G_x*\overline{f}_x$=0 (the third row in Table 1), all assignments that satisfy G_x also satisfy f_x and all pull-up paths consist of PMOS transistors only and a voltage drop is not possible. On the other hand, if $G_x*\overline{f}_x$ is satisfiable (the fourth row in Table 1), there exist some assignments that satisfy G_x and not f_x, and some pull-up paths contain at least one NMOS transistors and the minimum voltage is Vdd-V_{t0n}.

The proposed approach has the favorable property that it does not require enumeration of input states of the gate. Note that the analysis makes the simplifying assumption that the voltage drop across a path is constant, regardless of the number of NMOS transistors that are in series. If needed, paths with different number of NMOS transistors in series can be analyzed by enumerating the input vectors that satisfy G_x (for the second row in Table 1) or $G_x*\overline{f}_x$ (for the fourth row in Table 1) at the cost of additional run time.

If the victim driver has a possible voltage drop the equivalent holding resistances for the gate with and without a voltage drop can be significantly different. The holding resistance under input states without a voltage drop is obtained by executing the presented algorithm on the logic constraint function f_x instead of G_x. The holding resistance with a voltage drop is obtained by using the logic function $G_x*\overline{f}_x$, instead of in G_x.

Multiple Drivers: It is common that a victim or aggressor net is driven by multiple driver gates. In practice, not all drivers of a net will be active at the same time, and hence, we need to determine which drivers should be applied simultaneously. For the worst case noise condition, we can apply all aggressors drivers of an aggressor net and only the single victim driver with the largest holding resistance for a victim net. However, this leads to very pessimistic results and a large number of false violations. To determine the exact worst case set of aggressor and victim drivers requires extensive logic and state-space analysis with excessive run time. We therefore propose a heuristic that uses the fact that tristate gates typically do not drive a net simultaneously while non-tristate gates are always driving a net.

Victim Net Drivers: If both tristate and non-tristate drivers are connected to a victim net, the realistic worst case noise occurs when all non-tristate drivers are driving the victim net while all tristate drivers are assumed to be in tristate mode and are effectively disconnected. A single simulation is therefore needed with the holding resistance of all non-tristate drivers connected to the victim net.

If all victim drivers are tristate, we assume that at all times, at least one driver is in a driving state and is holding the victim net stable. Each victim driver is simulated in turn, each time disconnecting the other drivers, and the highest noise pulse at each sink node is recorded.

Aggressor Net Drivers: If some of the drivers are tristate, the realistic worst case noise will occur when only one of the tristate gates is driving the net, while all other tristate gates are in tristate mode. On the other hand non-tristate drivers are always driving an aggressor net when it switches. For an aggressor net, we therefore apply one tristate driver and all non-tristate drivers using the following algorithm:

1. Simulate each tristate driver with: 1) all other tristate drivers disconnected. 2) all non-tristate drivers connected to the aggressor net and their voltage sources shorted.
2. Among all tristate drivers, identify the worst one.
3. Determine the noise contribution of each non-tristate driver by simulating it with: 1) all other non-tristate drivers and the worst tristate driver connected with their voltage sources shorted. 2) all tristate drivers, except for the worst one, disconnected.
4. Use the superposition of all noise waveforms obtained from step 3 and from the worst tristate driver in step 2.

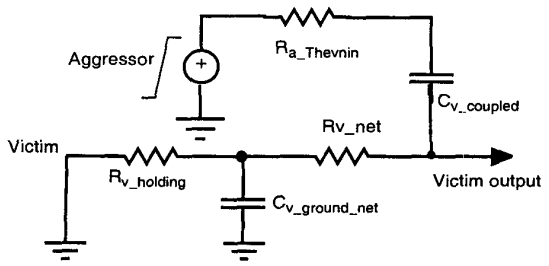
4 Noise Filters

Due to the large number of nets in circuit designs and the enormous number of interconnect and cross-talk elements the cost of loading and simulating all nets is prohibitively high. Therefore, it is essential to have efficient noise estimation techniques to quickly filter out nets that are guaranteed to pass noise analysis. The noise estimation method presented in [1] has the disadvantage that it requires the expensive loading of the full subcircuit consisting of the victim net, aggressor nets, and all their constituent interconnect and cross-talk elements. Also, it significantly overestimates the noise for fast aggressor edge rates, due to the assumption of an infinite ramp. An extension of this method, proposed in [3], suffers from the first disadvantage and also does not guarantee conservatism.

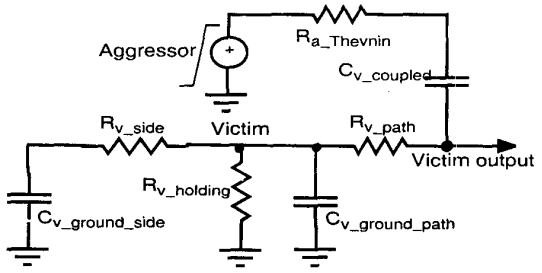
In ClariNet, we use a simple structural filtering strategy that offers both efficiency and accuracy. The coupling capacitance between the aggressor and victim nets ($C_{v\_coupled}$) is lumped between the aggressor source and victim sink to ensure a conservative estimate. Similarly, capacitances to ground from the aggressor and victim nets are lumped as far away as possible from the aggressor source or victim sink since they decrease the noise value.

The three filters used in ClariNet are each applied successively until either the victim net passes noise analysis for any filter or until it fails all the filters and is simulated in full detail. Each successive filter is less conservative but requires a higher run time.

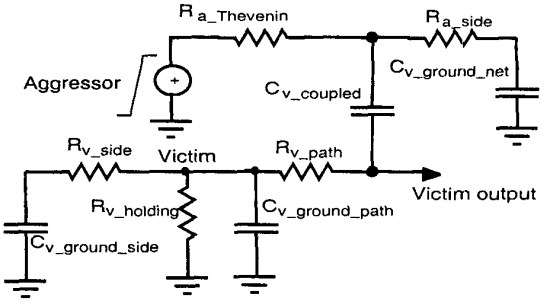
Filter 1: As drawn in Figure 3(a), the total resistance on the victim net is given by R_{v_net}. The grounded capacitance on the victim net is lumped together as $C_{v\_ground\_net}$ and connected away from the victim sink. The lumped resistance and capacitance values are usually stored with the extraction data and can be directly loaded without having to load every element of the net, thereby making this filter the fastest. A default aggressor and victim driver model that is conservative is used in this filter to avoid computing or loading the driver models. The Thevenin resistance, $R_{a\_Thevenin}$, of the aggressor driver model is the parallel equivalent of the Thevenin resistance of all possible aggressor drivers. The Thevenin voltage source used is the Thevenin voltage of the aggressor driver with the fastest rise time. This filter structure can be solved analytically under the conservative simplification that $R_{a\_Thevenin} = 0$. The closed form expression for peak noise has the form $k_1 \cdot \frac{(1-e^{aT})}{a \cdot T} + k_2 \cdot \frac{(1-e^{bT})}{b \cdot T}$, where k_1, k_2, a and b are constants determined by the values of the resistances and capacitances in the filter, and T is the transition time of the aggressor ramp. The



(a) **Filter 1**



(b) **Filter 2 with more detailed model of the victim net**



(c) **Filter 3 with more detailed model of the aggressor net**

Figure 3. Filters used in ClariNet.

derivation and exact expression are presented in [8].

Filter 2. As shown in Figure 3(b), the second filter includes more details of the victim net than used in the first filter while the aggressor net remains modeled with a lumped default model. We load the RC elements of the victim net and calculate the total resistance on the path from a victim source to a victim sink where the noise is measured, denoted by R_{v_path}, and lump the capacitance to ground on this path, denoted by $C_{v\_ground\_path}$. All other resis-

tances and capacitances of the victim net are lumped together and are denoted as R_{v_side} and $C_{v\_ground\_side}$. $R_{v\_ground\_side}$ $C_{v\_ground\_path}$ are connected away from the victim sink at the victim driver. R_{v_side} is the maximum of the resistances calculated for each path from the victim driver to any grounded capacitance not on a path to the victim net. It can be shown that calculating R_{v_side} in this manner yields a conservative estimation. Since it is necessary to load the elements of the victim net, the requiring run time in filter 2 is greater than filter 1. The noise overestimate by this filter is less than filter 1.

Filter 3. In filter 3 the details of the aggressors are included as shown in Figure 3(c). The grounded capacitance of all the aggressors are lumped together and are denoted by $C_{a\_ground\_net}$. To be conservative, $C_{a\_ground\_net}$ is connected away from the aggressor source. R_{a_side} is calculated in a manner similar to R_{v_side}. This filter has the maximum loading time because it has to read in every element from all the aggressor nets. However, it is also the most accurate of all filters with the least amount of noise overestimation.

Aggressor Pruning. The number of aggressors coupled with a single victim net can be extremely high, averaging several hundred for some circuits. The user can instruct ClariNet to use one or more of the following three criteria to reduce the number of aggressors.

1. The number of highly coupled aggressor nets is limited to a specified number with all other aggressors being discarded.
2. Using an approximate noise estimation similar to filter 1, ClariNet can discard aggressor nets that induce a combined noise that is less than a user defined value.
3. All aggressors that have a ratio of total coupling capacitance to the victim grounded capacitance that is smaller than a user specified threshold are discarded.

The coupling capacitance from the victim to all discarded aggressors are grounded with the underlying assumption that they are not switching.

5 Logic Correlations and Timing Windows

All nets may not induce noise simultaneously, because of logic and timing restrictions imposed by the circuit. Taking these restrictions into account results in trimming down the set of aggressor nets as described below.

Logic Correlations. In our noise analysis tool, we consider the following logic constraints: invert (logical negation), same (logical identity), imply (logical implication), one-hot (one and only one net can be at logic value 1) and one-cold (one and only one net can be at logic value 0).

The logic constraints between the nets are either provided by the designer or automatically extracted from the circuit. However, logic correlations under a zero delay assumption are not conservative for noise analysis. For example, the output of a two-input NAND gate, where the inputs are switching in opposing directions with some delay, may glitch, whereas zero-delay logic constraints would predict a stable value. Therefore, designers should only specify logic constraints for signals that are glitch free. For automatically extracted logic constraints, we restrict ourselves to only pairwise relationships across single gates to avoid this problem.

For efficient representation, logic correlations among aggressors and a victim are stored in the form of single BDD, by converting each constraint into a boolean equation and combine them. Each variable in this BDD represent the logic value of a net.

Timing Correlations. In addition to logic correlations, the victim and the aggressors may have restrictions in the temporal domain due to signal delays in the circuit.

An activity window for an aggressor net is defined as the interval from the earliest time to the latest time the net can switch. Typically, activity windows are obtained from static timing analysis by propagating the early and the late arrival times of the circuit inputs (or latch outputs in sequential designs) along all paths to the outputs (or latch inputs). Similarly, we obtain sensitivity windows by performing backward propagation of required times at circuit outputs or latch inputs. An observability window of a victim net is defined as the interval from the earliest required time and the latest required time. Note that the sensitivity window really needs to be calculated using gate delays under partial transitions instead of full rail-to-rail transitions because the noise pulses do not necessarily result in full transitions. This requires a timing analysis with special delay models, and is difficult since propagation delay under a partial transition is dependent on the noise height. Alternatively, we can use propagation delay under full transition but add some margin to the windows to account for variation in gate delay.

Figure 4 shows the algorithm used for combining the logic and timing constraints to reduce the number of aggressors. Note that c1

```
1.  Construct logic constraint BDD C by logical-ANDing individual constraints
2.  Reduce C by asserting victim logic state
3.  Intersect sensitivity window and activity windows to find a
    superset S of all aggressors that can simultaneously switch;
4.  Sort the set s in S in descending order of SUM of all aggressors in s
5.  max_valid_noise = 0;
6.  For each s in S
7.    max_noise = SUM (noise from all aggressors in s);
8.    if (max_valid_noise < max_noise)
9.       For each pair of logic constraint (c1, c2) in C such that
    c1!= c2
10.         Identify MUST_SWITCH, STABLE, and
    MAY_SWITCH set of aggressors;
11.         Remove aggressors in STABLE set from set s;
12.         next if any aggressor in MUST_SWITCH does not
    belong to set s;
13.         valid_noise = SUM (noise from all aggressors in s);
14.         if (max_valid_noise < valid_noise) max_valid_noise =
    valid_noise;
```

Figure 4. Logic and Timing correlation algorithm

and c2 in step 9 are two different traversals to 1-leaf nodes of the logic constraint BDD C and correspond to logic constraints before and after the aggressors switch, respectively. MUST_SWITCH is defined as the set of aggressors whose logic values in c1 and c2 are the same as the intended aggressor switching. STABLE is the set of aggressors whose logic values do not change in c1 and c2. MAY_SWITCH is the set of aggressors which do not appear in both c1 and c2.

6 Results

The noise analysis methodology described in this paper has been implemented and applied to a number of industrial circuits. Although our noise analysis uses linearized models for victim and aggressor driver gates and a fast interconnect evaluation engine, the accuracy of noise estimation is very good. Table 2 compares the maximum noise values on nets in circuit **dsp** to the values obtained from SPICE simulation with real drivers. Noise peak

voltages from ClariNet are shown to be within 4% of those from SPICE simulation.

Nets	Noise peak from ClariNet	Noise peak from SPICE	error (%)
net_1	692.3	677.1	2.3
net_2	691.8	668.9	3.4
net_3	687.7	673.6	2.1
net_4	683.7	658.7	3.8
net_5	695.9	681.0	2.2
net_6	682.6	667.1	2.3

Table 2. Accuracy of Noise Analysis.

In Table 3, details of the interconnects for some circuits are shown. Circuits **dsp, control** and **adder** are custom-designed functional blocks for a DSP core, a communication processor and a microprocessor, respectively. Circuit **chip** is a high speed PowerPC$_{TM}$ microprocessor.

The noise analysis results for the circuits are presented in Table 4. Note that our multi-stage filtering allows us to screen out as much as 96% of total nets without detailed analysis.

Circuit	# of nets	#RC elements per net			#aggressors per net		
		max.	avg.	min	max.	avg	min
dsp	46,035	47,927	105	6	5,612	19	1
control	30,795	114,418	2,787	628	1,039	391	277
adder	1,168	11,639	133	21	886	19	3
chip	406,404	58,835	305	16	1,596	41	2

Table 3. RC interconnect data

For circuit **adder** and **chip**, we show run times for two analyses; one with noise filtering and the other without. Run time with filtering is dramatically smaller. Note that both analyses use the presented methods for aggressor pruning, since without it, the run time is extremely large and analysis is infeasible. The significant aggressors are defined as the set of aggressor nets whose noise contribution is at least 90% of the total noise at a victim.

Circuit	# nets	# nets filtered	#nets fully analyzed	# nets failed	avg. # sign. aggressors	Run time
dsp	46,035	44,242	1,793	301	4.3	11 h
control	30,795	26,393	4,402	1,859	3.7	15 h
adder w/ filter	1,168	1,068	100	87	8.8	31 s
adder w/o filter	1,168	0	1,168	87	8.4	101 s
chip w/ filter.	406,404	335,182	71,222	2,356	6.9	5.5h
chip w/o filter.	406,404	0	406,404	2,356	6.7	20.3h
chip w/o correlation info	406,404	335,182	71,222	2,777	6.9	8.2h

Table 4. Noise Analysis Results

To demonstrate the effectiveness of using logic and timing correlation in reducing the number of false noise violations, we show two analysis results for **chip** example: one with the correlation information and the other without.

CPU time for noise analysis is reasonable and less than 15 hours for each circuit we presented here. Note that the run times for **dsp** and **control** are large compared to that of **chip**, although the designs are much smaller. This is because for these examples the linearized driver models are generated on-the-fly and each noise waveform at the victim sink is propagated through the receiver gate using simulation. Driver characterization and noise propagation account for more than 70% of the entire run time in each example. For the circuit **chip**, driver and receiver models are generated apriori and on-the-fly noise propagation is not needed.

7 Conclusion

In this paper, we presented a functional noise analysis tool, ClariNet, which incorporates a practical and comprehensive strategy for analyzing coupling noise effects in large, high-speed digital designs. We presented new algorithms for critical issues in noise analysis: the accurate and efficient modeling of driver gates, noise filtering for the efficient processing of very large designs, and logic and timing correlation for reducing false noise violations. We demonstrated the use of the tool on a number of industrial designs, including a DSP and PowerPC$_{TM}$ processor. Our results show that the accuracy of the linear models generated by our proposed methods is within 4% of full non-linear spice simulation. We also show how aggressor pruning and noise filtering allow ClariNet to analyze a 400,000 net design in under 6 hours, and how incorporating logic and timing correlation into noise analysis reduces the number of false violations.

References

[1] A. Devgan, "Efficient coupled noise estimation for on-chip interconnects," Proc. IEEE Intl. Conf. Computer-Aided Design, pp. 147-151, Nov. 1997.

[2] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela and J. Dunning, "Transistor-level sizing and timing verification of domino circuits in the PowerPC$_{TM}$ microprocessor", Proc. Intl. Conf. Computer Design, pp. 143-148, 1997.

[3] M. Kuhlmann, S. S. Sapatnekar and K. K. Parhi," Efficient crosstalk Estimation," Proc. IEEE Intl. Conf. Computer Design, pp. 266-272, Oct 1999.

[4] A. Odabasioglu, M. Celik and L. T. Pileggi, "PRIMA: Passive reduced-order interconnect macromodeling algorithm," Proc. Intl. Conf. Computer-Aided Design, pp. 58-65, 1997.

[5] J. Qian, S. Pullela and L. T. Pillage, "Modeling the effective capacitance for the RC interconnect of CMOS gates," IEEE Trans. Computer-Aided Design, pp. 1526-1555, Dec 1994.

[6] K. L. Sheppard, V. Narayanan, P. C. Elementary and G. Zheng, "Global Harmony: Coupled noise analysis for full-chip RC interconnect networks," Proc. Intl. Conf. Computer-Aided Design, pp. 139-146, 1997.

[7] J. M. Zurada, Y. S. Joo, S. V. Bell, "Dynamic noise margins of MOS logic gates", Proc. IEEE ISCAS, pp. 1153-1156, 1989.

[8] V. Zolotov et al," Closed form noise filter expressions", Tech Report, Motorola.