An Event-Driven Approach to Crosstalk Noise Analysis

Pirouz Bazargan-Sabet, Patricia Renault
LIP6 – University of Paris 6, 4 Place Jussieu – 75005 Paris – France
pirouz.bazargan-sabet@lip6.fr, patricia.renault@lip6.fr

Abstract

Crosstalk noise evaluation consists in analysing the effect of the transition of a signal - called aggressor - on its neighbours - called victims. This evaluation is based on a model that estimates the peak noise produced on each victim. However, from a victim's point of view, not all its aggressors are able to make a transition at the same time. Thus, determining the active aggressors of a victim at a given time is a key point to get a realistic estimation of the noise. In this paper, we present a static analysis method for crosstalk noise evaluation. The purpose is to calculate the "maximal noise configuration" that is likely to occur of each signal. The proposed approach is based on an event-driven algorithm. First, the instability periods of signals are calculated. Then, using these periods, the maximal noise configuration is obtained for each victim.

1. Introduction

The development of recent submicron processes has offered the designers the opportunity of investigating the integration of several millions of transistors on a single circuit. Concepts such as system on a chip, hardware-software codesign, architectural synthesis and many other new items in system design have received a considerable impulse thanks to these technologies.

In counterpart, with submicron processes, digital designs are suffering from the apparition of new causes of misfunction, neglected until now. In the same time, some other sources of failure are more pronounced. As a consequence, a great emphasis is put on the verification step to certify the correctness of a design. To cover the risk of error in regards of these issues new post-layout verification tools are required.

These tools have to face a double fold challenge: First, develop and implement a specific model to check the respect of the specifications in a particular field; Second, manage the amount of information inside a data organization suited for the application of this model. Of course, these two aspects are not independent from each other. The amount of information depends on the model and the feasibility of the verification depends largely on the data organization and on the computational resources required by the model. Therefore, a suitable trade-off must be found between the precision of the model and the desired performance to make the verification of a multi-million-transistor circuit reachable within a reasonable delay.

The crosstalk noise is one of the emerging problems in submicron design that may cause timing and, in some extreme cases, functional failures in the circuit. In a crosstalk coupling, a noise is injected on a wire whenever a signal in its near environment makes a transition. This noise is due to a parasitic capacitor that appears between two wires routed one near the other. The signal that makes the transition is called the *aggressor*, and the signal affected by the noise the *victim*.

Let's consider two signals *A* (aggressor) and *V* (victim) driven by two inverters (Fig. 1). During the transition of *A*, if *V* is in a steady state, the noise has the form of a spike and is absorbed by the *V*'s driver after some delay. On the contrary, if *V* is making its transition in the same time as *A*, the crosstalk noise leads to a shorter or a longer transition delay.

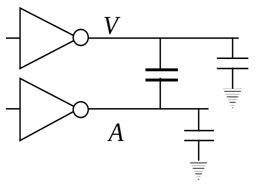

Fig 1: Two signals in crosstalk coupling

In a real circuit, the coupling scheme is much more complex. A given signal may be coupled with several thousand signals. However, at a given time, all the aggressors are not making a transition. The aggressors that are in a steady state and do not contribute to the noise produced on the victim are called *silent aggressors*. The others are *active aggressors*. In turn, each aggressor may have many other couplings. When a given signal is considered as a victim, all the other victims of its aggressors are referenced as *secondary victims*.

In submicron processes, many factors contribute to the apparition of the crosstalk noise: the shape of wires, the reduction of the distance between wires, the greater number of metal layers, etc. These factors tend either to reduce the capacitance to the ground or, to increase the coupling capacitance. Another parameter that determines the importance of the noise is the impedance of the aggressor and the victim's drivers.

In this paper, we detail the effective implementation of a crosstalk noise evaluation model inside a verification tool. The next section gives an overview of our noise estimation model already published in a previous paper. In the section 3, we focus on the problem of the static crosstalk analysis and the method used to determine the active aggressors of a victim signal. This static analysis method represents the main contribution of this paper. Section 4 presents the software architecture of the tool. Some experimental results are shown in section 5. The last section exposes the concluding remarks and the future works.

2. Crosstalk Noise Model

Many models have been proposed to estimate the peak value of the noise produced on a victim. Some of them take into account the resistance-capacitance of the interconnect, others focus on studying the noise produced by the simultaneous transition of several aggressors [1] [2] [3] [4].

In a previous paper [5] we have proposed a model that ignores the RC of the interconnect but gives a satisfying estimation of the peak produced by several aggressors using a simple approach. The proposed model takes into consideration some second order effects such as the existence of silent aggressors and secondary victims. The detailed description of this model is beyond the scope of this paper. However, for completeness we present an overview of this one:

Let's consider the case of Fig 2, where the victim signal V is in crosstalk coupling with 2 active aggressors A_1 and A_2 and a silent aggressor A_{silent}. Each active aggressor has a secondary victim, respectively V_1 and V_2.

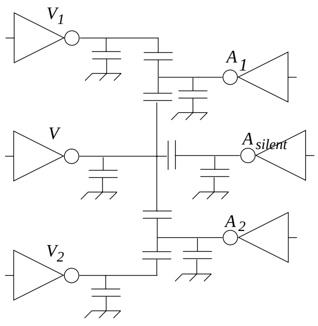

Fig 2: Victim (V) surrounded by its aggressors

Like many other models, in our approach, a first approximation consists in replacing signal's drivers by a simple resistor (Fig 3). However, the victim and the aggressors are not approximated in the same manner. The transistors of the victim's driver are in their linear region whereas, the aggressors make the most part of their transition in the saturated mode. Thus, two different equivalent resistors are calculated for each driver. A first resistance is used when the signal is supposed to be in a steady state (the victim, silent aggressors and secondary victims). A second resistor approximates the drivers of a signal when it is taken as an aggressor.

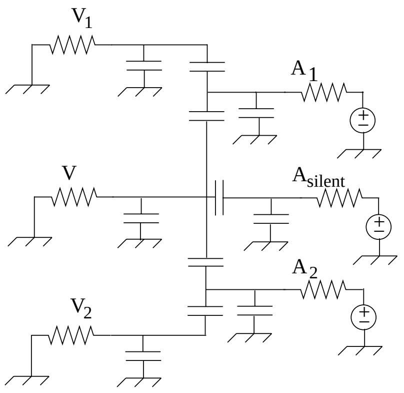

Fig 3: Drivers replaced by equivalent resistors

In the next step, for each victim, the silent aggressors are replaced by an equivalent capacitor. Then, for each of the remaining active aggressors, their secondary victims are replaced by an equivalent capacitor.

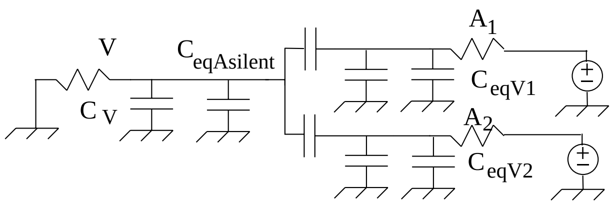

Fig 4: Approximation by equivalent capacitors

The presence of a secondary victim coupled to an aggressor has an impact on the transition of this later. Actually, a victim influences its aggressor by increasing its transition delay. Therefore, the capacitor equivalent to a secondary victim is calculated such as the transition of the aggressor in presence of the victim have the same shape as the transition of the aggressor loaded by the additional equivalent capacitor.

In a last approximation step, each active aggressor is replaced by an equivalent current source where the current decreases in an exponential form:

$$i = I_c e^{-t/\tau_c}$$

The two parameters I_c and τ_c are calculated such as the peak produced on the victim occur at the same time and have the same value as the peak induced by the corresponding aggressor.

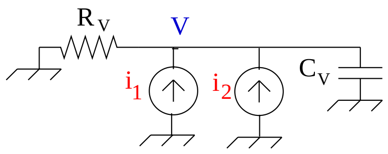

Fig 5: Final equivalent circuit

Then, the effect of the different current sources are added and the victim's waveform is expressed:

$$v(t) = R_v \sum_{k=1}^{n} I_{C_k} \frac{\tau_{C_k}}{\tau_v - \tau_{C_k}} (e^{-t/\tau_v} - e^{-t/\tau_{C_k}})$$

Finally, the peak noise is obtained when v(t) reaches its maximal value $\frac{d}{dt}v(t) = 0$.

3. Static Crosstalk Noise Analysis

As we have seen, for each victim, determining which of its aggressors may commute in the same time is a key point in the noise analysis. In this section, we detail a technique that aims to make the distinction between active and silent aggressors. Through this technique the noise configurations of each victim are calculated. For a signal, a *noise configuration* is defined as a subset of its aggressors that may commute in the same time. The falling transition of active aggressors produces a negative noise on the victim. Thus, such noise configuration is called a *negative* noise configuration. A noise configuration composed of aggressors that receive a rising transition represents a *positive* noise configuration.

An obvious way of determining the noise configuration is to consider that all the aggressors of a given victim are active aggressors. This elementary method is far the most efficient since it doesn't require any computation. However, in a multi-million-transistor circuit, a signal may have several thousands of aggressors. Thus, the assumption that all these aggressors could make their transition in the same time is far to be realistic. Therefore, considering the noise configuration as the entire set of the aggressors leads to an overestimated worst case.

As a consequence, a further analysis is required to determine the noise configurations that can actually be produced on a signal. This analysis relies on timing concerns. The calculation of instability periods is the first step to determine the signals' noise configurations.

3.1 Instability Periods

The instability periods are computed from:

- primary inputs' instability periods;
- gate propagation delays.

The primary inputs' instability periods are given by the designers. The gate propagation delays are obtained from a hierarchical timing analysis tool HiTAS. A brief overview of this tool is given in section 4.

Given the propagation delay through the gates, it is possible to compute the time, in the clock period of the circuit, at which a signal makes its transition. Yet, this calculation depends on the logic configuration of the inputs and on the internal state of the circuit. As it is not reasonable to examine all the logic combination of the inputs and the internal states to obtain the worst noise configuration, a static approach must be used. Like in static timing analysis, in static crosstalk noise analysis, the configurations of the inputs and the internal state of the circuit are not considered. The analysis consists in examining a typical clock cycle, assuming that, regardless of the logic configuration, all potential transitions are effective.

Let's consider a gate driving a signal *S* with the inputs I_1 , …, I_n. According to the Boolean function of the gate, up to four propagation delays may be computed for each input:

- FRD_i: Falling edge of I_i to the rising edge of *S*
- FFD_i: Falling edge of I_i to the falling edge of *S*
- RFD_i: Rising edge of I_i to the falling edge of *S*
- RRD_i: Rising edge of I_i to the rising edge of *S*

Besides, these propagation delays are not constant values and depend on the configuration of the other inputs. However, they can be restricted inside a lower and an upper bound. Then, given the transition time of an input, a transition interval, also called *instability period*, can be attributed to the output. In turn, this information can be used to calculate the transition interval of the signals that depend on *S*.

In detail, each signal may have several instability periods depending on the transition intervals of its inputs. Moreover, two kinds of instability must be considered: the instability due to a falling transition of the signal and the instability due to its rising transition. Figure 6 gives the example of a 2-input Nand gate. The rising transitions of the inputs induce two falling instability periods on the output. The falling transitions of the inputs are merged to a unique rising instability period on *S*.

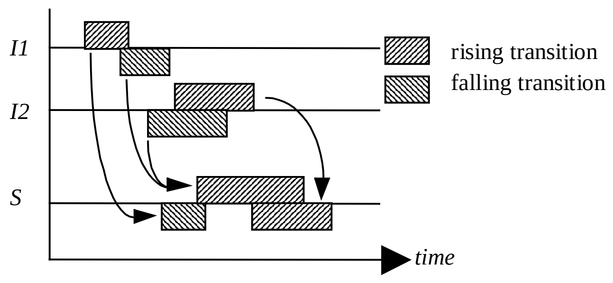

Fig 6: Instability periods

Several techniques with variable degree of complexity may be used to obtain the instability periods of signals. Our approach consists in using a symbolic event-driven simulation to compute the transition intervals. An event-driven simulator can be seen as composed of two independent parts. A first part, the *simulation engine*, includes the propagation mechanism and insures the coherence of the simulation. In an event-driven simulator, this part is a loop containing two main functions *Update()* and *Execute()* also called the δ-loop. During the Update phase, the simulation time is advanced and all the transactions for the current time are extracted from the scheduler. Then, the current values of signals are updated. If an event is detected on a signal during this update, all the processes that depend on that signal are resumed. During the Execute phase resumed processes are executed and transactions are send to the scheduler.

The set of values a signal can take, the notion of transaction and event are defined in the second part. This second part, the *evaluation engine*, also describes the evaluation function. This function determines how the value of a process's output is calculated from its inputs.

In our approach, this second part is set to compute the instability periods. An instability period is characterized by two time bounds: the beginning of the instability and its end.

The value attributed to a signal represents its state: stable or unstable. This value is coded on two natural integers. A first integer, RS (rising stability), signals the stability in regard of rising edges. The second, FS (falling stability), concerns the stability in regard of falling edges. The value 0 means that the signal is stable. Positive non null integers denote the unstability.

A transaction consists in incrementing or decrementing one of these two integers. An event is detected when the signal's current value becomes null (when the signal switches from unstable to stable) or when this value changes from 0 to a non null value (when the signal switches from stable to unstable). In other terms, there are 4 types of event:

- beginning of a rising instability period;
- end of a rising instability period;
- beginning of a falling instability period;
- end of a falling instability period.

The last point consists in setting the evaluation function. Whenever an input of a gate receives an event, the execution of the evaluation function creates one or two transactions on the gate's output. The produced transactions depend on the Boolean function of the gate. If the event is the beginning of a rising instability period, the transaction consists in incrementing the value of RS and/or FS. If a rising edge of the input is able to cause a rising edge of the output, RS is incremented. In the same way, FS is incremented if the Boolean function is such that a falling transition can be produced on the output from a rising edge of the input. The transaction delay is the minimum value of RRD or RFD for the corresponding input. For a "end of rising instability period" event the transaction decrements the value of RS and/or FS and the transaction's delay is the maximum value of RRD or RFD.

Figure 7 shows the example of an Xor gate followed by an inverter. The beginning of rising instability of *I1* induces two transactions on *S* since a rising edge on an xor's input may generate a rising or a falling edge on the output. In the same manner, the beginning of the falling instability of *I1* creates two new transactions on *S*. However, *S* is already in an instability period and these two transactions do not generate any event.

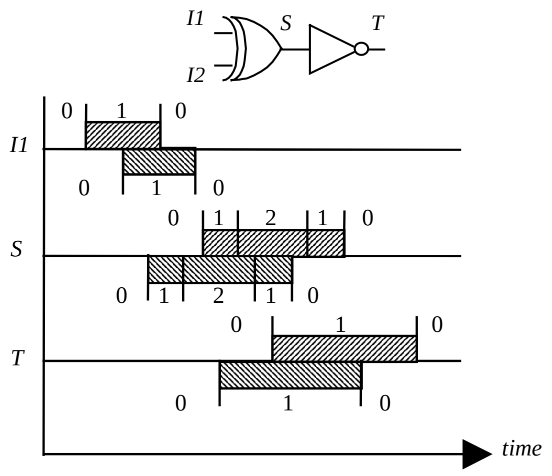

Fig 7: Simulation of instability periods

The simulation starts with initializing the simulator’s scheduler with the instability periods of the circuit's inputs. These data are given by the designers. For a sequential circuit, the instability periods of the clock are defined from the clock frequency and take into account the clock skew. Then, the simulation begins with the Update phase. The clock input receives a first event :

"beginning of rising instability period". In the next Execute step, all the registers of the circuit are resumed. These evaluations produce a set of transactions on the registers' output. Then, the simulation continues through the next δ-cycle propagating the instability of registers' output through the combinatory logic.

3.2 Noise Configurations

The second step is to determine the signals' noise configurations with instability periods.

Let's consider a victim V having m aggressors $A_1,\dots,A_m$. Let $R_1, \dots, R_m$ be the rising and $F_1,\dots,F_m$ the falling instability periods of $A_1,\dots A_m$.

The subset $\{A_{i1},\dots,A_{ip}\}$ represents a positive noise configuration if and only if $\bigcap_{k=1}^{p} R_{ik} \neq \varnothing$. The same subset constitutes a negative noise configurations if and only if $\bigcap_{k=1}^{p} F_{ik} \neq \varnothing$.

Figure 8 shows the example of a signal coupled to 4 aggressors A_1, …, A_4.

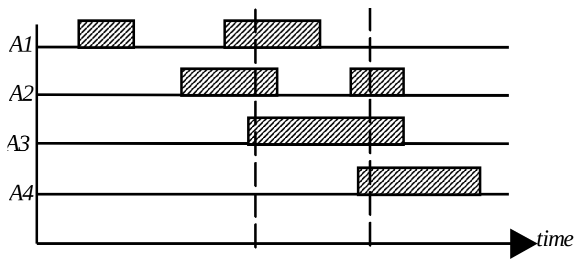

Fig 8: Noise configurations

In this example, 11 noise configurations can be detected: $\{A_1\}$, $\{A_2\}$, $\{A_3\}$, $\{A_4\}$, $\{A_1,A_2\}$, $\{A_1,A_3\}$, $\{A_2,A_3\}$, $\{A_2,A_4\}$, $\{A_3,A_4\}$, $\{A_1,A_2,A_3\}$ and $\{A_2,A_3,A_4\}$.

Nevertheless, it is unnecessary to compute the noise produced by each of these configurations. Actually, it is obvious that the peak generated by the first configurations will be less than the 2 last since each of the 9 first sets is included either in $\{A_1,A_2,A_3\}$ or in $\{A_2,A_3,A_4\}$. These two configurations are called *Maximal Noise Configuration (MNC).*

A maximal noise configuration is a noise configuration that is not included in any other configuration. Although the instability periods are useful for a static noise analysis, the main objective in our analysis is to determine the set of maximal noise configurations for each signal.

In our proposed approach, the set of maximal noise configurations is calculated inside the same event-driven simulation process that determines the instability periods.

From the simulator's point of view, a circuit is seen as a directed graph where the vertices are the gates and the edges the signals that connect the output of a gate to the input of another one. This connection is called *functional forward dependency* or *link*.

To set the simulator for the crosstalk noise analysis, this graph is extended with crosstalk links. A crosstalk coupling between two signals X and Y creates two *crosstalk forward dependencies* : one from X to Y and an another from Y to X.

Figure 9.a shows the example of a simple circuit containing four gates and a crosstalk coupling.

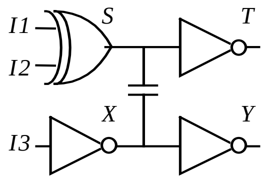

Fig 9a: Simple circuit

The resulted graph is presented in Fig 9.b.

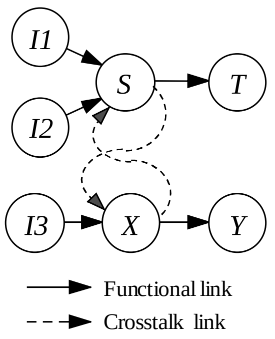

Fig 9b: Simulation graph

As mentioned in 3.1, functional links are used to calculate the instability periods. Noise configurations are calculated using crosstalk links.

The evaluation engine of the simulator is modified as follows.

First, four new data are added to the value of each signal : current positive noise configuration (CPNC), current negative noise configuration (CNNC), maximal positive noise configurations (MPNC) and maximal negative noise configurations (MNNC).

When a signal receives an event, the simulation engine resumes all the processes that depend on that signal. The processes resumed due to a functional link are evaluated using the functional evaluation function as defined in 3.1. Evaluations caused by a crosstalk link are performed by the crosstalk evaluation function.

This last evaluation function operates on noise configurations. A "beginning of instability" event on a signal *A* makes *A* being considered as an active aggressor and appends it to the current noise configuration of the signal. An "end of instability" event on *A* removes *A* from the current noise configuration. In addition, before altering the current noise configuration, this one is compared to the different maximal noise configurations identified until this point. If it is not included in any maximal noise configuration then, the current configuration is maximal and is added to the maximal noise configurations. Besides, appending the new configuration may remove from the maximal noise configurations list a previous configuration that has become non maximal.

4. Software Architecture

A Static Crosstalk Noise Evaluation tool based on the proposed technique has been implemented. A simplified flow chart of this tool is given in Fig 10. The inputs are an extracted transistor-capacitor netlist and the inputs' instability periods. The output is a sorted list of signals with respect of decreasing peak noise that they can receive.

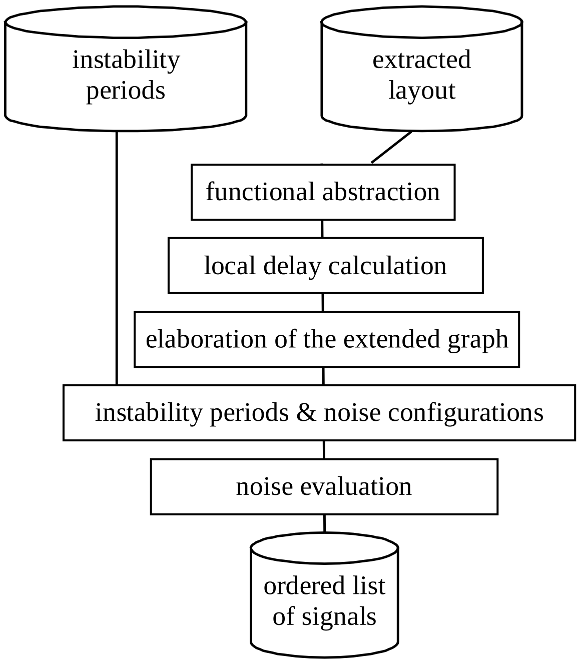

Fig 10: Simplified flow chart

As mentioned in section 2, the crosstalk noise model generates on an directed gate netlist. In a first step, a functional abstractor (YAGLE)[6] [7] is used to convert the transistor netlist into a gate netlist. YAGLE recognizes the different transistors that compose a gate and gather them into a structure called cone. Then a set and reset boolean function is attributed to each cone. Unlike other functional abstractor that use pattern matching technique to identify gates, YAGLE is based on a formal approach. For each signal that controls the gate of a transistor, YAGLE build the paths that connect this signal to V_{DD} or to V_{SS}. Each path also called branch contains a set of transistors in series. and its conduction condition gives a part of the set or reset boolean function of the cone. Memorization elements are identified through a more complex process using a formal boolean analysis. Using this technique YAGLE is able to abstract circuits designed through a full custom methodology or built from an unknown cell library.

In the next step, a static timing analysis (HiTAS) is called to calculate the delays of each cone regardless of the crosstalk noise. First, for each cone, HiTAS [8] [9] identifies the branch that leads to the worst delay. Then, the current delivered by the branch is evaluated using the MCC model. MCC (Short Channel MOS) is a transistor model developed at University of Paris 6 - LIP6 that gives a simple expression of the drain current as a function of v_{DS} and v_{GS}. The propagation delay is calculated taking into account the slope of the input and some second order effects such as the Miller capacitance. This tool can deal with circuits of extremely high integration density.

Then, the extended graph of the circuit is elaborated. This graph includes the functional links as well as the crosstalk links. The instability periods and the maximal noise configurations are calculated using the proposed event-driven simulation technique.

In the last step, for each maximal noise configuration of a given victim, the peak noise is estimated using the noise evaluation model. Then, the maximal peak noise is determined and saved into the output file.

5. Results

The static analysis technique exposed here above can be explored from 2 points of view: the correctness and the efficiency.

Checking the correctness of the proposed method consists in verifying that for all the signals of a circuit, the noise configurations that has been flagged as unfeasible through the analysis may effectively not be produced. Obviously, on a real circuit, where a signal can present several thousands of aggressors, such a prove remains out of reach. However, on a simple circuit including a series of transistors and a limited number of aggressors, the experience shows that the noise configurations rejected by the analysis are ineffective.

The efficiency of the method can be explored through several experiences.

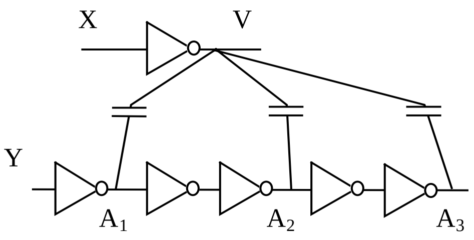

Fig 11: Smple circuit

This verification tool has been used to perform noise evaluation on several circuits ranging from a few thousands to up to 1.2 million transistors. Small circuits where an electrical simulation is reachable, have been used to assess the accuracy of the noise model. The relevance of the proposed static analysis approach has been experimented on larger circuits.

In Fig. 12, the number of active aggressors contained in the maximal noise configuration is compared to the total number of aggressors. The comparison concerns the 200 signals that show the highest peak in a 1.2 million-transistor circuit designed with a 0.25 µ process. As it could be expected, the static analysis reveals that even if a victim can presents several thousands of aggressors, only a few number of them can be active in the same time.

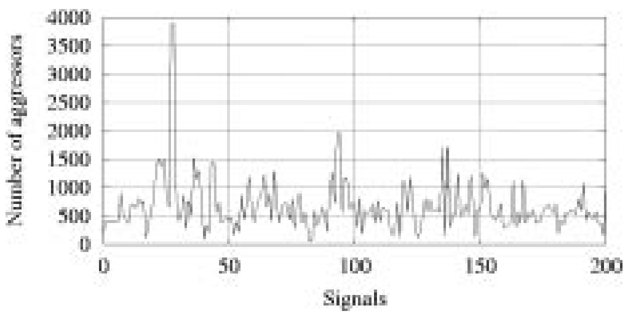

Fig 12a: Total number of aggressors

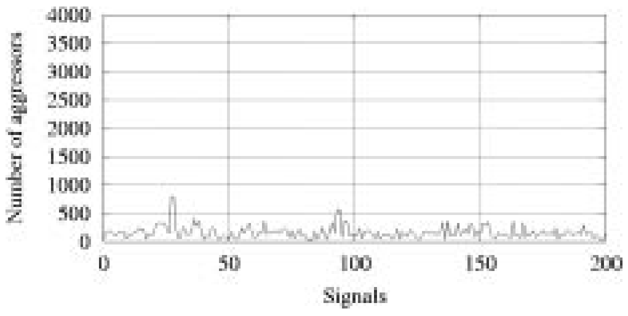

Fig 12b: Number of aggressors in Maximal Noise Configuration

Fig. 13 compares the peak noise produced by the maximal noise configuration and the estimated noise assuming that all the aggressors are active. Although the number of active aggressors has been largely reduced through the proposed approach, the same reduction is not observed for the peak noise. This means that for several victims, the major part of the noise is due to a few number of aggressors coupled to the victim through a strong crosstalk capacitance.

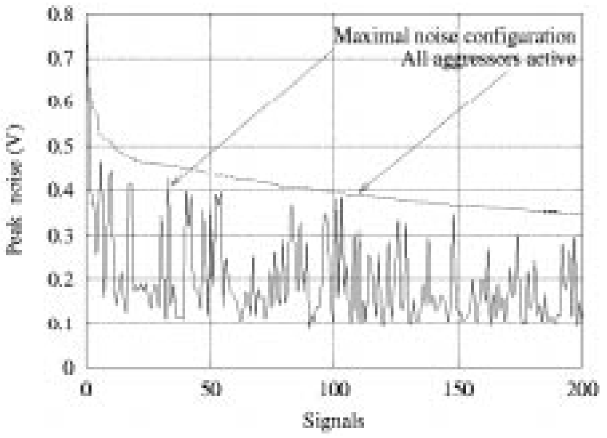

Fig 13: Peak noise

The performance of the tool in terms of computation time is measured in Fig. 14.

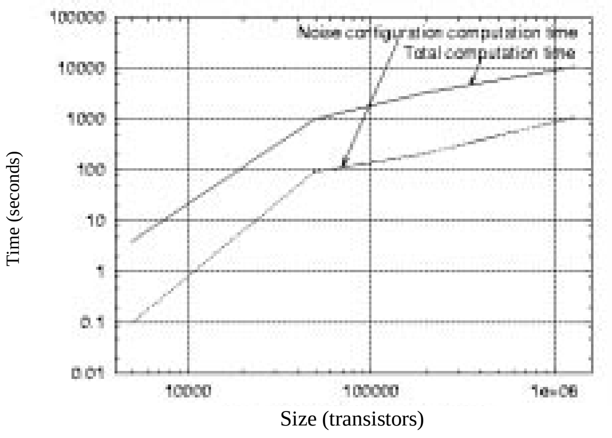

Fig 14: Computation time

The experience shows that for real size designs the total computation time remains linear with the complexity of the circuit. This time can be divided into three major parts. Over 50% of the time is spent in reading the input file, making the functional abstraction and preparing the data structures. Computing the maximal noise configurations through the proposed static analysis represents the second part. The last part is devoted to the estimation of the peak voltage using the noise evaluation model. As shown in the figure, the static analysis requires only about 10% of the time and does not impact in a significant way the global performance of the tool. Yet, it brings a significant improvement in the relevance of the estimated noise.

6. Conclusion

Signal integrity is becoming a major issue in the verification process of high performance designs. Crosstalk noise is one of the factors that may cause timing and functional failure in the circuit. Crosstalk noise evaluation is based on a model that tends to estimate the peak noise voltage produced on a victim when the signals in its environment make a transition. However, considering that all the signals in crosstalk coupling with a given victim may switch in the same time is far to be realistic.

In this paper we have presented a static analysis technique that determines the "maximal noise configuration" that is likely to occur for a given victim. This analysis is based on an adaptation of the classic event-driven algorithm. The experience shows that this static analysis can be performed within less than 10% of the total computation time required by the crosstalk verification tool. Moreover, for life size circuits, the analysis reveals that, at a given time, the major part of the aggressors are not active and only a few numbers of a victim's aggressors participates to the production of the noise. The proposed crosstalk verification tool can be improved following two directions. First, determining the maximal noise configuration may be refined through a deeper analysis that takes into account the Boolean function of gates and the correlation between signals. However, this kind of analysis requires an important amount of computation resource and time and may compromise the global performance of the verification. The second improvement concerns the noise model. For deep submicron processes a more accurate model has to be develop taking into consideration the RLC of the interconnect.

7. References

[1] A.B. Kahng, S. Muddu, D. Vidhani, "Noise and delay uncertainly studies for coupled RC interconnects", ASIC/SOC, IEEE, 1999, pp. 3.

[2] A.B. Kahng, S. Muddu, N. Pol, D. Vidhani, "Noise Model for Multiple Segmented Coupled RC Interconnects", ISQED, 2001, pp. 145-150.

[3] P. Chen, K. Keutzer, "Towards True Crosstalk Noise Analysis", ICCAD, 1999.

[4] A. Devgan, "Efficient Coupled Noise Estimation for On-Chip Interconnects", ICCAD, 1997

[5] P. Renault, P. Bazargan-Sabet, D. Le Du, "A MoS Transistor Model for peak voltage calculation of crosstalk noise", ICECS, 2002

[6] A. Lester, P. Bazargan-Sabet, A. Greiner, "Second generation Functional Abstractor for CMOS VLSI Circuits", ICM, 1998

[7] A. Lester, "Abstraction Fonctionnelle des Circuits Numériques VLSI avec une méthode formelle basée sur une extraction de réseau de portes", Ph.D. dissertation, University of Paris 6, 1999.

[8] A. Hajjar, R. Marbot, A. Greiner, P. Kiani, "TAS: An Accurate Timing Analyser for CMOS VLSI", EDAC, IEEE, 1991, pp. 261-265.

[9] A. Hajjar, "Modelisation des temps de propagation et analyse temporelle statique", Ph.D. dissertation, University of Paris 6, 1992