Static Noise Analysis with Noise Windows

Ken Tseng
Cadence Design Systems
kentseng@cadence.com

Vinod Kariat
Cadence Design Systems
vin@cadence.com

ABSTRACT

As processing technology scales down to the nanometer regime, capacitive crosstalk is having an increasingly adverse effect on circuit functionality, leading to increasing number of chip failures. In this paper, we propose mapping the static crosstalk functional noise problem into the well understood static timing problem. The key differences between static noise and static timing analyses, namely the injection of noise, accurate noise window propagation and register sensitive window computation are the contributions of this work. We demonstrate the effectiveness of this approach in two industrial designs by achieving 5X reduction in functional noise failures over noise propagation without considering timing of the composite noise pulse envelope, and 30X reduction in functional noise failures over net based noise failure metrics.

Categories and Subject Descriptors

B.7.2 [Hardware]: Integrated Circuits - Design Aids, Reliability and Testing

General Terms

Algorithms, Reliability, Verification

Keywords

Noise, Signal Integrity, Crosstalk

1. INTRODUCTION

Early research on noise analysis in digital circuits focused on functional failures caused by charge sharing, crosstalk and leakage noise in dynamic circuits [1]. Recently noise induced chip failures are manifested in static cell based designs as well, where most of the research up to now has been focused on the noise effects on delay [2][3]. However, functional noise failures are real even in static CMOS logic. If the noise glitch at the input of a storage element is large enough and occurs at the right time, a functional violation will occur resulting in the wrong logic state being stored. Once noise violations are detected, they must be fixed by design changes that involve either buffer insertion, driver sizing or rewiring [4][5]. Existing techniques in noise analysis tend to overestimate noise violations, resulting in unnecessary design changes. Making the matter worse is the fact that crosstalk problems are heavily dependent on detail routing, hence the accurate analysis and design changes can only take place late in the design flow. Hence, reducing false noise violations is critical to design closure.

DAC 2003, June 2-6, 2003, Anaheim, California, USA.

We now review existing techniques in functional noise analysis. The most basic approach to functional noise analysis is to simply calculate the glitch noise magnitude at every net and report the nets with glitch magnitude exceeding a certain voltage threshold as potential violators [4]. This approach often reports hundreds or even thousands of false violations so it is difficult to use. Timing windows are often employed to filter aggressors that cannot switch simultaneously [6], which helps reduce false violations.

Shepard and Narayanan [1] suggested using noise stability as a conservative metric to measure the response of the receiving gates to a given noise glitch using circuit simulation. A violation is reported only when the receiving gate is amplifying the input noise, regardless of the magnitude of the input noise itself. This approach, though conservative, often reduces false violations by an order of magnitude or more over the previous approaches, as glitch noise is fundamentally AC in nature and CMOS gates are low pass filters. A characterization scheme has been proposed [7] to improve runtime while trading off some accuracy.

To further reduce false violations, two industrial noise analysis tools [8][11] allow noise to propagate through combinatorial cells and non-linearly combine with downstream crosstalk noise. In [8], coupling noise is performed at the cell level, whereas noise propagation with any additional coupling on cell output is performed at the transistor level. Noise stability is checked only at the inputs of registers. This further reduces false violations by another order of magnitude. In many industrial designs at 0.18um to 0.13um the number of failures reported is often much less than a hundred.

However, the timing of the noise glitch is not considered, hence any propagated noise is allowed to combine with any coupling noise. Further, the timing of the resulting noise glitch at the register input is not considered, resulting in many false violations. Sensitivity window calculation is mentioned in [10] by backward propagation from register inputs based on gate delays, yet the authors also pointed out that some amount of delay compensation is needed to account for the difference in propagation delay between a switching signal and a noise glitch. Since the delay difference can be cumulative through many levels of logic, the amount of compensation can be substantial for victim nets that are many levels upstream from the storage element.

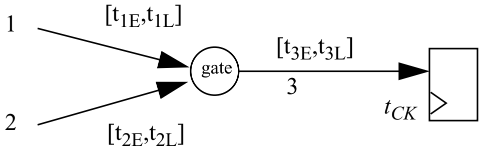
In this paper we propose forward propagation of functional noise windows. Noise window is defined as the worst case duration of time that significant noise glitches can occur on a given net. The noise window width is bounded by the union of the aggressors' timing windows, with adjustments for the coupled interconnect delay between aggressor and victim. Accuracy is achieved by utilizing circuit simulation techniques for noise propagation, while runtime is kept in check by judicious use of noise filtering to reduce the need for unnecessary noise propagation analysis. Once the glitch noise is propagated to the input of a register, an additional sensitive window based on the clock trigger time, with adjustment for setup and hold times, can be applied to further filter glitch noise that occur outside of the sensitive period of the register.

2. STATIC NOISE ANALYSIS

Functional noise exists in circuits that may or may not be detrimental to the correct functionality of the circuit. However, if a sizable functional noise occurs at the input of a register when the register is active, the wrong logic value could be stored in the register. Therefore, static noise analysis should keep track of the maximum noise value at every time point within a clock period, for every net in the design.

2.1 Comparison to Static Timing

Conceptually, noise glitches propagate through logic gates just as switching signals propagate through logic gates. The similarities and key differences are explained in this section. The terminology for a simplified static timing model is illustrated in Figure 1. Notice that this is intended to capture just the essence of static timing, with many details left out. Static timing analysis calculates the early and late arrival times in every net. For net 3 in the diagram, this is represented by the $[t_{3E}, t_{3L}]$ pair. The early and late arrival times at register inputs are checked against the clock arrival time t_{CK} and the setup and hold times to determine timing violation. An analogous diagram for static noise analysis is shown in Figure 2. Contrary to timing analysis, the early and late arrival times of glitch noise are not determined by the net's fanin, but by the net's aggressors' early and late arrival times. In other words,

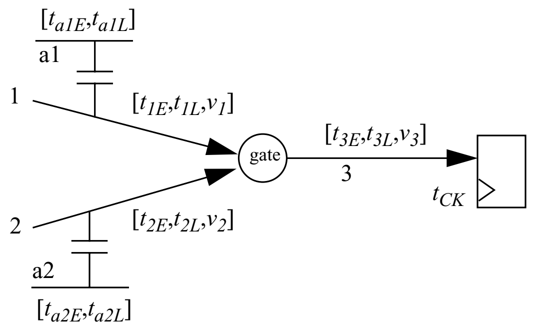


$$[t_{3E}, t_{3L}] = [\min(t_{1E}+t_D, t_{2E}+t_D), \max(t_{1L}+t_D, t_{2L}+t_D)]$$
$$t_{3L} < t_{CK} - t_{SETUP}$$
$$t_{3E} > t_{CK} - t_{CYCLE} + t_{HOLD}$$

Fig. 1. Simplified static timing model

the injection of noise and its associated noise window is determined not by circuit topology but by a net's capacitive neighbors.

In Figure 2, nets a1 and a2 are aggressors to nets 1 and 2 respectively. $[t_{a1E}, t_{a1L}]$ defines the early and late arrival times, or timing windows, of aggressor net a1. When a1 switches, a noise glitch is created on net 1 at time t_{Da1} delay from the switching point of a1 to the gate input of net 1. The peak magnitude of the noise on net 1 is represented by v_1. Hence, the noise window on net 1 is represented by the triplet $[t_{1E}, t_{1L}, v_1]$. A similar calculation is performed on net 2. Next we calculate the noise window on net 3. The detail mechanics of noise propagation is discussed in Section 2.3, but in essence we need to calculate the output noise waveform and the propagation delay from input noise peak to output noise peak. The propagation delay is represented by t_{D13} and t_{D23}. The output noise magnitude from input noises at net 1 and net 2 are represented by v_{13} and v_{23} respectively. The noise window at net 3 is the union of the noise window on net 3 contributed by net 1, and the noise window on net 3 contributed by net 2. Finally, when a noise window reaches a register input, the intersection of the noise window and the register's sensitive window forms the effective noise window $[t_{3E}', t_{3L}', v_3']$. The worst case noise waveform within the effective noise window is



$$[t_{1E}, t_{1L}, v_1] = [t_{a1E}+t_{Da1}, t_{a1L}+t_{Da1}, v_{a1}]$$
$$[t_{2E}, t_{2L}, v_2] = [t_{a2E}+t_{Da2}, t_{a2L}+t_{Da2}, v_{a2}]$$
$$[t_{3E}, t_{3L}, v_3] = [t_{1E}+t_{D13}, t_{1L}+t_{D13}, v_{13}] \cup [t_{2E}+t_{D23}, t_{2L}+t_{D23}, v_{23}]$$
$$[t_{3E}', t_{3L}', v_3'] = [t_{3E}, t_{3L}, v_3] \cap [t_{CK}-t_{SETUP}, t_{CK}+t_{HOLD}]$$

Fig. 2. Noise window model

used as propagated noise input to the register to determine noise immunity.

2.2 Noise Window Computation and Propagation

The method of calculating the worst case glitch noise, the associated noise window, and the propagation of the noise window, is illustrated using an example shown in Figure 3. We define VL noise to be the glitch noise above ground, and VH noise to be the glitch noise below the power supply level. Net A has a VH noise caused by capacitive coupling, which can be propagated onto net B as VL noise, and combine with additional coupling on net B from a rising transition on aggressor net C. The combined VL noise on B gets propagated onto D as VH noise, which again gets combined with more coupling noise from aggressor net E. Net D is the data input to a flip flop, which has a sensitive window as defined by the timing window of the clock padded with setup time on the early edge and hold time on the late edge. The resulting worst case combined noise on D is evaluated for noise stability on the flip flop storage node with respect to D to check for noise violation.

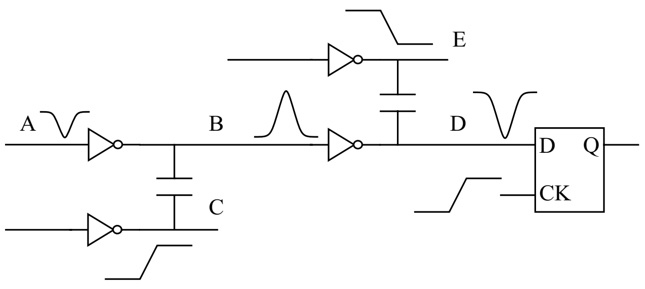


Fig. 3. Noise propagation and combining example

The corresponding timing diagram is shown in Figure 4, with arbitrary timing windows on aggressors C and E and clock CK to illustrate noise combining and noise window propagation.

The glitch noise on A is caused by coupling from aggresors that is not shown here. The noise window on A is propagated to B by

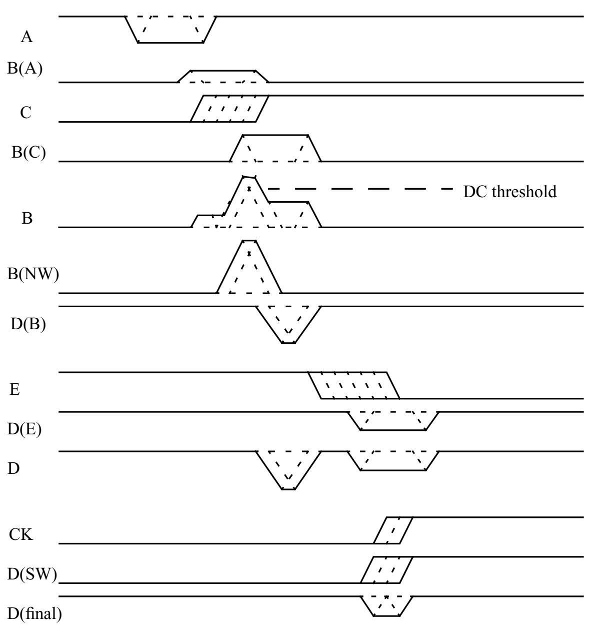


Fig. 4. Noise window propagation timing diagram

shifting the window by the delay of the noise glitch from A to B. B(A) indicates the noise constituent on B due to A.

Net C is an aggressor to net B and its timing window is shown above. Noise window on B due to coupling from C is shown below as B(C). By superposition of B(A) and B(C) we derive the peak noise envelope on B as shown in B below. A combined simulation of propagated noise from A and coupling noise from C on net B is performed to get accurate combined noise effects. This combined noise can be significantly larger than the superposition result due to the non-linear characteristics of the driving gate [11]. The noise envelope on B must be adjusted to account for this difference. From this noise envelope we derive the noise window for B. If the noise peak is less than a certain voltage threshold, normally defined to be the minimum threshold voltage of the receiving gates, it is safe to assume the noise will not be propagated to any of the receiving gates. A rare exception is when the noise is barely below threshold on the input of a single stage cell and the cell output has significant coupling occurring at the same time, then the output coupling noise can be underestimated. This issue of driver subthreshold weakening is currently ignored. Ignoring this exception, the noise window is derived to be the noise envelope region that exceeds the DC threshold, as shown in B(NW).

Similarly, noise on E is caused by a combination of crosstalk from aggressor E resulting in D(E), and the propagation of the noise window from B resulting in D(B). Again by superposition and combined simulation, we derive the combined noise envelope on net D. Since net D is the data input we further constrain the noise envelope by the sensitive window on D, which is shown below as D(SW) which is [t_{CK}-t_{SETUP} ,t_{CK}+t_{HOLD}]. The final noise envelope on D is shown below as D(final). The combined noise on D is then analyzed for noise stability by performing a circuit simulation of the flip flop with input noise on D. Note that had the noise on D arrive a bit earlier it would have completely missed the sensitive window, or had the noise on D arrive a bit later a much larger noise would affect the flip flop.

In timing window based noise analysis [6], the goal has been to derive the set of aggressors that produces the worst noise on a given net. However, we are interested in the entire noise versus time profile, where the bounds determine the noise window. At every time instant a worst case noise must be produced from all the possible noise sources, which may include coupling aggressors from neighboring nets and propagated noise sources from the driving gates. Under certain conditions, such as the presence of mutually exclusive timing windows or logic constraints, several mutually exclusive groups of noise sources must be analyzed to determine the worst case noise group. The method we use to determine the worst noise group is based on the linear summation of each individual noise source's noise contribution to the victim. The noise group that gives the maximum noise peak by linear summation will be selected. After the worst group is determined, then we do a single combined simulation with multiple noise sources acting together simultaneously. An accurate noise waveform will be computed considering the non-linearity of the driver under multiple noise sources.

If unbuffered latches (or flip-flops) are used in the design, an additional overshoot/undershoot coupling noise analysis must be performed on direct data inputs of the latches. Noise induced glitches below Gnd or above Vdd may cause the latch to lose its internal stored logic value even when the latch is inactive. Therefore, two separate analysis must be performed on the inputs of unbuffered latches. The first analysis is based on noise window and sensitive window, as explained above. The second, additional analysis focuses only on coupling noise on the unbuffered input. The propagated noise component is ignored as noise propagation cannot result in noise above Vdd or below Gnd based on device physics. Two separate noise immunity analyses are performed on the unbuffered latch and the worst case noise immunity is reported.

2.3 Noise Waveform Propagation

There have been attempts in the past to characterize noise propagation. There are two fundamental difficulties involved. First, the number of characterization parameters is large. To accurately model a noise waveform, at least two, may be three, parameters are needed. These include the peak and the width of the noise glitch, or the peak, leading and trailing slopes of the noise glitch if higher accuracy is desired. Together with output load, and the number of input to output arcs in a gate, the characterization tables can be very large. Second, the combined effect of multiple input propagated noise, and the combined effect of noise propagation with additional crosstalk on the output net cannot be easily modeled. Therefore, we decided to use transistor level simulations for noise window computations. However, we should note that the noise window method itself is independent of the noise propagation model. A noise propagation model based on characterization of input/output arcs and linear superposition can be employed in the noise window framework, but with the loss of accuracy as explained above.

Even though the peak noise in a design can be large, the percentage of nets with noise exceeding the threshold voltage is small. From experimental data with many large industrial designs, the number of nets that require transistor level noise propagation analysis, based on the minimum threshold voltage requirement, is typically less than 5% of the total nets in a design. Therefore, the runtime penalty for using transistor level simulations is minimal. Table 1 shows the percentage of total runtime consumed by transistor level circuit simulations in a static noise analysis tool. As shown in the table, the additional runtime due to transistor level simulations are minimal..

Special sensitization is required for noise propagation to guarantee the worst case combination of noise glitches are captured [9]. The sensitization algorithms employed in this work are proprietary, but similar concepts have been published in [12] and [13]. Basically, a circuit is partitioned into channel connected components, and

Table 1: Transistor level simulation runtime as a percentage of total noise analysis runtime

Process	# Nets	%Transistor simulation runtime
0.18 um	114,647	1.4%
0.15 um	174,009	0.4%
0.15 um	1,399,476	0.1%
0.13 um	457,091	0.9%
0.13 um	532,178	6.2%

sensitization equations, formulated using BDDs [14], for noise propagation are derived by logic extraction through a transistor level network. Once the propagated noise equations are formulated, the satisfiable vectors are enumerated for circuit simulation. In order to reduce the number of circuit simulations, vectors that are deemed electrically equivalent, with similar resistance and capacitance characteristics in the transistor level network, are simulated only once. The propagated noise sensitization concept is illustrated with a 2 input NAND gate as shown in Figure 5.

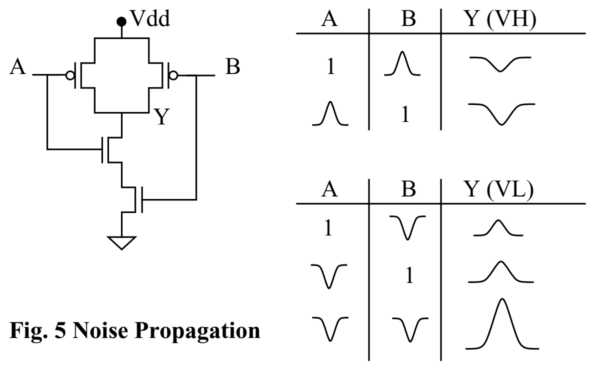


Fig. 5 Noise Propagation

The table on top of Figure 5 shows the two different sensitizations for input VL noise. At least one of the two NMOS transistors in the pulldown stack must be 'ON' for the input VL noise on the other input to cause a VH noise on the output. Assuming the input noise on A and B are equal, and the 4 transistors are well balanced, the output VH noise on Y due to A would be slightly larger than noise due to B because of the smaller output capacitance in the first case. This kind of subtle effects are very difficult to capture without exhaustive vector enumeration together with transistor level simulation. In the bottom table, three vectors are possible to sensitize output VL noise due to input VH noise. The worst case is both inputs A and B having input VH noise, if they both happen around the same time. In the previous section we described a technique using noise superposition to arrive at the combined noise window. Here we perform a combined simulation to get the accurate combined effects of both input noise. The resulting combined noise may be significantly higher than the superposition result due to the turning-off of the pulldown N-stack. The noise window magnitude and width are adjusted based on the combined simulation result.

In our noise propagation formulation we have chosen to ignore a second order noise effect to simplify the problem. An aggressor causes a noise glitch on a victim, and the noise glitch on the victim in turn causes a noise glitch on the victim's victim. This second order noise effect is shown in Figure 6. Noise windows can be easily extended to include glitch noise caused by both switching aggressors and *glitching* aggressors. An iterative noise analysis may consider these glitching aggressors. Since the noise impact from glitching aggressors are usually small, as shown in the Spice simulation results in Figure 6, the iterations should converge quickly. .

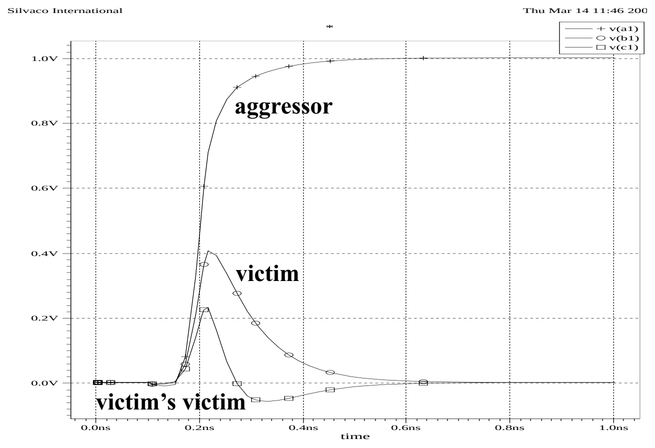


Fig. 6. Victim behaving as an aggressor

3. RESULTS

The methods described in this paper has been implemented in an industrial static noise analysis tool. We now present experimental results on two industrial designs. Design A consists of 15K cell instances, 3.6M coupling capacitances, 410K grounded capacitances and 405K resistances. It was implemented in 0.3um technology. The design is driven by a 2.0V power supply and 400MHz and 250MHz clocks. The results are reported in Table 2. This design has been fabricated and the silicon was functional, as correctly predicted by the noise analysis result that indicates the absence of noise violation.

Design B consists of 14K cell instances, 36K coupling capacitances, 55K grounded capacitances and 58K resistances. It was implemented in 0.12um technology. The design is driven by a 1.32V power supply and 300MHz and 150MHz clocks. The results are reported in Table 3.

First we perform static noise analysis using a voltage threshold as the noise immunity metric. We define the threshold to be 30% of supply voltage. Next we analyze the same designs using the noise sensitivity metric [1], which is conservative yet significantly reduce the number of reported failures. Then we run noise analysis again with noise propagation enabled, but without noise windows and register sensitive windows. Finally we run noise analysis with noise window propagation and register sensitive window enabled. In Tables 2 and 3 we report the number of noise violations as defined by the four different noise metrics. The results are listed under Vp (peak), dVo/dVi (sensitivity), prp (noise propagation) and nw (noise window propagation).

Unfortunately in the original designs the number of noise failures are relatively small, so whichever noise metric was used the

Table 2: Noise metric comparison for Design A

Design A	Vp	dVo/ dVi	prp	nw
Original	7	1	0	0
10x Coupling	1569	261	27	6

Table 3: Noise metric comparison for Design B

Design B	Vp	dVo/ dVi	prp	nw
Original	10	1	0	0
5x Coupling	2465	398	66	12

difference in the number of noise violations is not significant. This is possibly due to the designs not being routing congested. For routing congested designs at higher clock speeds we expect the number of noise failures to increase. To show the effectiveness of the noise window methodology, we artificially increased all coupling capacitances in the designs while keeping all other design parameters unchanged. As shown in Tables 2 and 3, the reduction in false noise violations using noise windows is more than 200 times over the peak noise metric. The improvement over the net based sensitivity metric [1][8][9] is also significant at over 30 times. The improvement over noise propagation without considering noise windows is 5 times.

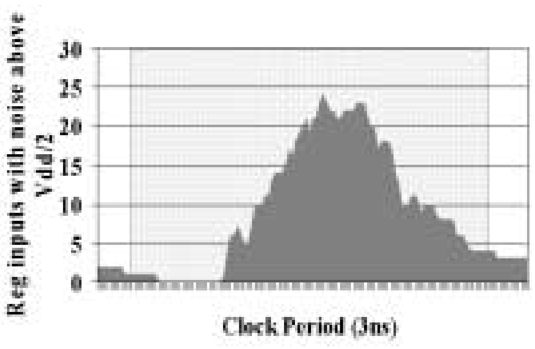


Fig. 7. Register input noise distribution over clock period

Figure 7 graphically suggests where this improvement is coming from for Design B. The figure plots the number of register input nets with significant noise (noise peak exceeding Vdd/2) against their arrival times at the register input within the clock period. As shown in the grey box, most noise in this design arrives at register inputs during the middle of the clock period when the clock is inactive, which gets pruned away by the register sensitive window, and will not result in functional failure.

The runtime impact of noise window analysis is insignificant and therefore is not reported here.

4. CONCLUSION

We have presented a method of noise propagation considering noise windows and the sensitive window of the receiving registers. Our results on industrial designs proved this method to be highly effective in reducing the number of reported false noise violations. We will continue to test this methodology on additional designs that may have more noise problems, and hopefully demonstrate the scalability of the methodology.

In terms of future direction, we plan to run the same analyses again considering small, or insignificant, aggressors. For a given victim net, the number of small aggressors can exceed a thousand depending on the coupling capacitance extraction threshold. If each small aggressor is analyzed separately the impact on runtime will be unacceptable. However, the timing behavior of small aggressors can be nicely captured in the noise window by superposition. A timed virtual aggressor can be created to model the combined effects of the small aggressors at any given time.

5. REFERENCES

[1] K. L. Shepard and V. Narayanan, “Noise in Deep Submicron Digital Design”, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 524-531, November 1996.

[2] P.Chen, K.Keutzer, “Towards True Crosstalk Noise Analysis”, International Conference on Computer-Aided Design, pp.132-137, 1999.

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

[4] Scheffer, L. “What is the appropriate model for crosstalk control?” 13th Symposium on Integrated Circuits and Systems Design, 2000, pp. 315-320.

[5] C. J. Alpert, A. Devgan, S. T. Quay, “Buffer Insertion for Noise and Delay Optimization”, IEEE Transactions on Computer-Aided Design, pp. 1633-1645, November 1999.

[6] Hui Chen, L.; Marek-Sadowska, M., “Aggressor alignment for worst-case crosstalk noise”, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume: 20 Issue: 5 , May 2001, pp. 612-621.

[7] Shepard, K.L.; Chou, K. “Cell characterization for noise stability”, Custom Integrated Circuits Conference, 2000, pp. 91-94.

[8] CeltIC User Guide, Cadence Design Systems, 2002.

[9] PacifIC User Guide, Cadence Design Systems, 2002.

[10] Levy, R.; Blaauw, D.; Braca, G.; Dasgupta, A.; Grinshpon, A.; Chanhee Oh; Orshav, B.; Sirichotiyakul, S.; Zolotov, V. “ClariNet: a noise analysis tool for deep submicron design”, Design Automation Conference, 2000, pp. 233-238

[11] Zolotov, V.; Blaauw, D.; Sirichotiyakul, S.; Becer, M.; Oh, C; Panda, R; Grinshpon, A.; Levy, R.; "Noise Propagation and Failure Criteria for VLSI Designs", International Conference on Computer Aided Design, pp. 587-594, 2002.

[12] Bryant, R. E., "Extraction of gate level models from transistor circuits by four-valued symbolic analysis", International Conference on Computer Aided Design, pp. 350-353, 1991.

[13] Kuehlmann, A., Srinivasan, A., LaPotin, D. P., "Verity - A formal verification program for custom CMOS circuits", IBM J. Res. Development, Vol. 39, No. 1/2, pp. 149-165, Jan/Mar 1995.

[14] Bryant, R. E., "Graph-based algorithm for boolean function manipulation", IEEE Trans. Computers, Vol. 35, No. 8, pp. 677-691, Aug. 1986.