Signal Integrity Management in an SoC Physical Design Flow

Murat Becer Ravi Vaidyanathan Chanhee Oh Rajendran Panda
Motorola Inc., Austin, TX
MuratBecer, Ravi.Vaidyanathan, Chanhee.Oh, Rajendran.Panda@motorola.com

ABSTRACT

Signal integrity closure is one of the key challenges in DSM (Deep- SubMicron) physical design. In this paper, we propose a physical design methodology which includes signal integrity management through noise analysis and repair at multiple phases of the design so that a quick noise convergence can be achieved. The methodology addresses both functional and delay noise problems in the design and is targeted for block, platform, and chip level physical design of SoC (System-On-Chip) designs. A number of case studies are presented to illustrate the effectiveness of the proposed methodology and to provide valuable insights useful for successful signal integrity management.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Design, Experimentation

Keywords

Signal integrity, crosstalk noise, noise avoidance, noise repair

1. INTRODUCTION

Crosstalk noise is defined as the change in the voltage waveform of a net in an undesired way due to the signal activity in its neighboring nets which are capacitively coupled to it. Crosstalk noise has become a critical design and verification issue for large, high-performance designs. In noise analysis, the nets on which crosstalk noise is injected by one or more of its neighbors are called the *victim* nets whereas the nets that inject this noise are called the *aggressor* nets. Crosstalk noise can manifest itself in two ways. *Functional noise* refers to noise that occurs on a victim net which is being held quiet by a driver. Crosstalk noise on such a victim causes a glitch (Figure 1) which may propagate to a dynamic node or a latch, changing the circuit state and causing a functional failure [1][2]. On the other hand, *delay noise* refers to noise that occurs when two capacitively coupled nets switch simultaneously (Figure 1). Depending on the direction of these transitions, the delays on both nets are affected[3][4], giving rise to potential setup or hold time failures.

ISPD'03, April 6-9, 2003, Monterey, California, USA.

Although crosstalk noise has always existed in integrated circuits, it has become a critical issue due to the following reasons. Finer geometries and increasing interconnect density along with more metal layers have resulted in greater wire and via resistances. Narrow wires have also become thicker to cope with increased resistivity, thus resulting in the increased ratio of crosstalk capacitance to total capacitance. On the other hand, the usage of more aggressive and less noise immune circuit structures such as dynamic logic has increased due to performance reasons. Lower device lengths have resulted in faster but low vt gates. Together with lower supply voltages, the noise margins of these high-performance gates have been significantly lowered. Faster slews have resulted in increased injected noise whereas smaller clock cycles dictate much less tolerance to delay variations.

The above mentioned effects have rendered crosstalk noise a critical design and verification challenge for large, high-performance SoC designs.

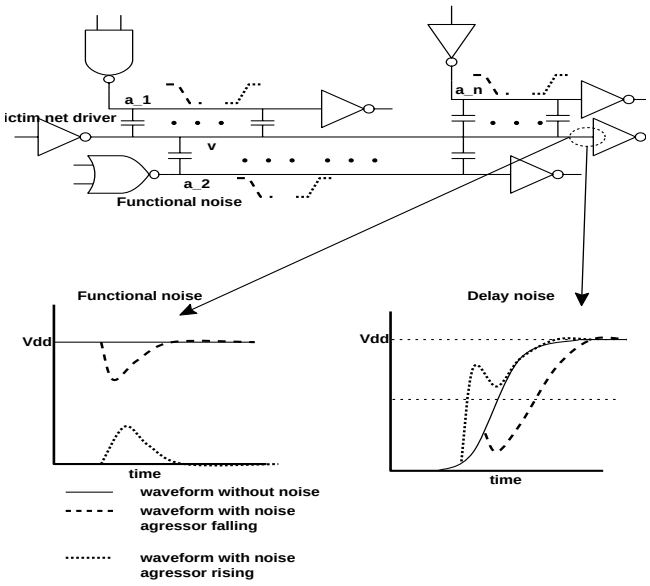

Figure 1. Functional and delay noise due to crosstalk

Analysis and verification challenges are due to the tremendous amount of victim-aggressor pairs in most designs, along with millions of distributed parasitic devices. While pessimistic assumptions result in an impractical number of false failures to deal with, detailed analysis of all coupled nets in the design is impossible. Several noise analysis techniques have been developed by employing electrical, logical and temporal isolation techniques[1][2] to reduce the problem space into the significant and realizable cases. Such methods also use reduced order modeling techniques[11] to

reduce the interconnect complexity and increase speed. Several challenges in functional[1] and delay noise[12][3] analysis have been addressed in recent literature.

As the above mentioned trends in interconnect and device technology have continued, efforts have been made to tackle crosstalk noise problems in early stages of the design cycle[13] and to address crosstalk noise problems during and after routing. These efforts include routing and interconnect optimization (wire spacing, wire widening, controlling coupling length and position)[5][6], buffer insertion[7][8] and gate sizing[9][10].

Although the above mentioned work has been significant in shedding light on the prevention, analysis and correction of crosstalk noise related problems, the design and EDA communities are still in search of a best-in-practice design methodology to address the crosstalk noise problem in several stages of the design cycle, especially in the context of SoC designs.

In this paper, we propose a crosstalk noise management methodology in an SoC physical design flow and present several case-studies using real, high-performance, low-power SoC designs. Our proposed flow aims to address both functional and delay noise problems. In our discussions, we investigate several techniques in pre-routing, routing and post-routing design stages and analyze the effects on both functional and delay noise problems. We look at the block, platform, and chip level problems in an SoC context and discuss the effectiveness and feasibility of several prevention and repair techniques in these different hierarchical levels.

The paper is organized as follows. In Section 2, we present the correlation between functional and delay noise problems, as well as our proposed signal integrity management methodology. Details on the several designs, used as test cases in this paper, are also presented in this section. Section 3 discusses noise prevention methodology and presents examples. Functional noise analysis and repair is explained in Section 4 along with numerous test cases. In Section 5, delay noise analysis and the repair of resulting delay problems are discussed. Section 6 contains closing remarks.

2. PROPOSED METHODOLOGY AND CASE STUDIES

A design is considered to have sound signal integrity only when there is no coupling noise large enough to cause a functional or timing error. Therefore, remedying every functional and timing violation due to cross coupling noise is stipulated as a criterion for the design to tape out. The proposed methodology, for this reason, is designed for addressing both kinds of noise problems in a concerted manner. When there are severe constraints on the design cycle time or the design resources, a design team may have to ill-afford a lengthy iterative exercise to analyze and fix every noise problem that is uncovered. Therefore, the steps in this SI management methodology are carefully chosen to realize quick design convergence wherein a noise-free design is obtained while meeting the timing, area, power, and other constraints.

It may first seem that fixing both functional and delay noise problems would take twice as much effort as fixing one of them. However, since the underlying causes for both problems are the same, viz. weak victim drivers, strong aggressors, large coupling, light loading, etc., there is a very high degree of correlation between occurrence of these two problems for the same net. Likewise, there is a strong correlation between the magnitude of functional noise glitch and the change in delay due to noise for a net. These correlations suggest that fixing one problem should often alleviate the other problem as well, if not completely eliminate. Figures 2 and 3 confirm the above observations.

Figure 2 shows the correlation between functional noise and delay noise. The top graph in Figure 2 is the distribution of delay noise on those nets which fail functional noise analysis (glitches above 200mV at the output of the receiver gate), whereas the bottom graph in Figure 2 shows the delay noise distribution on those nets which pass functional noise analysis among approximately 40,000 top level nets in a SoC design. The average and maximum delay noise in the top graph are 1381ps and 9.8ns respectively. Same values for the bottom graph are 279 ps and 4.2 ns. Figure 3 shows the delay noise distributions (delay noise above 400ps is shown for clarity) at two stages of the same design, one very early before noise fixes were done (top graph in Figure 3), and one quite later when a substantial number of functional noise violations had been fixed (bottom graph in Figure 3). The correlation between functional and delay noise violations is amply clear from these figures. These observations confirm that the remedial measures effective for one problem could be effective in addressing the other problem as well.

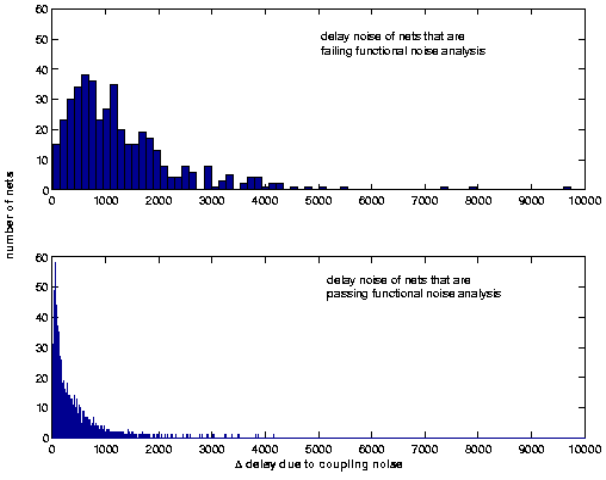

Figure 2. Functional noise and delay noise correlation

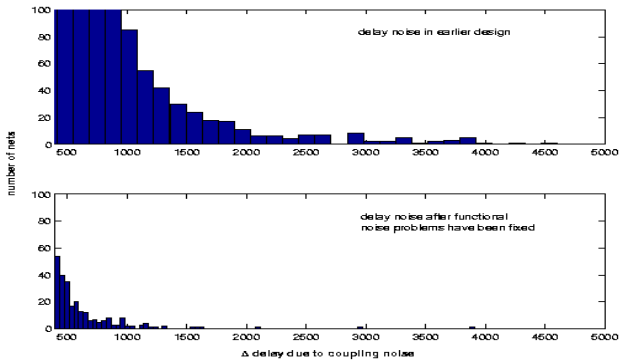

Figure 3. Improvement in delay noise after functional noise problems are addressed

Finally, Figure 4 demonstrates that the aforesaid expectation is valid by showing the amount of functional and delay noise on a net before and after buffer insertion. The buffering was done in order to bring the noise glitch below the specified limit. Top graph in Figure 4 shows the magnitude of glitch (functional noise) at the output of the receiver for an arbitrarily chosen net, before and after inserting a buffer. Bottom graph in Figure 4 shows the waveform for a fall transition at the receiver output for the same net before and after buffer insertion. It can be seen that the change in delay due to noise dissipated when the net was buffered to fix functional noise.

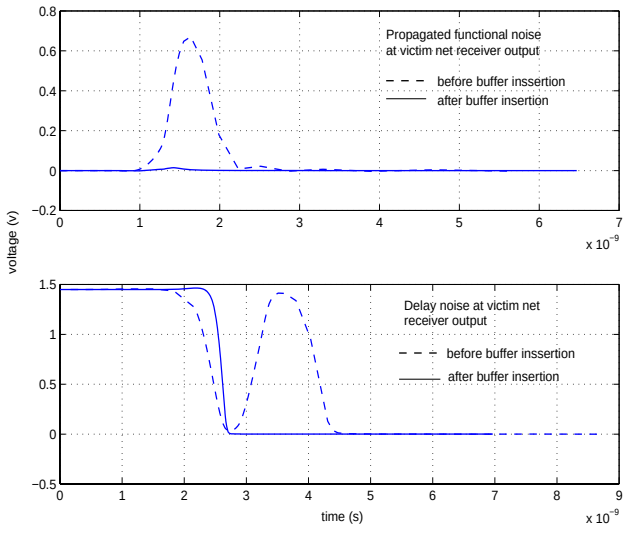

Figure 4. Effects of buffer insertion on functional and delay noise

Though a small amount of noise may not be harmful functionally, the resulting, although small, delay variation can still contribute to large timing errors since net delays contribute cumulatively to path delays. The consequence is that a design that is free of functional noise can yet have timing issues due to noise and apparently the number of delay noise problems one has to tackle is much larger than the number of functional noise problems. However, note that much of the delay noise problems that remain after all the functional noise problems (and the accompanying delay noise problems) are fixed, come from fairly small delay noise violations accumulating to a significant quantity on critical and near-critical paths. The positive side of this is that a significantly large number of small delay noise violations that are occurring on non-critical paths need not be tackled. This observation suggests an effective SI management strategy wherein the delay noise problems are addressed *only* after all the functional noise problems are addressed. Although one can make some arguments in favor of addressing functional noise after delay noise, we consider the following, in addition to the above discussion, as compelling reasons to tackle functional noise problems first.

- At early design stages (before functional noise problems are addressed), a large number of nets will have noise and a significant number of them also will have large amounts of noise. Since nets are shared by many paths, a large number of delay violations (based on path delays) will be reported and it will not be easy to manage such a large number of violations. It is rather convenient to address the problems initially for the shorter list of violators in the functional noise category. Since fixing each net fixes also the delay noise on many paths, the number of delay noise violations will drastically decrease after the functional failures are corrected.

- The relative noise magnitudes from a functional noise analysis can be utilized to arrive at an efficient order of applying certain noise repair measures such as victim/aggressor driver sizing so that faster noise convergence is achieved. Ordering the repair measures based on a delay noise report is somewhat convoluted since the severity of noise violation is determined in part by the criticality of the path on which the net lies. Moreover, the glitch voltage metric, available from functional noise analysis, is more convenient to use in determining repair measures, rather than the delta delay metric available from a delay noise analysis.

In line with the above arguments, we propose a methodology wherein functional failures are analyzed and repaired first at pre-route and post-route stages, and the delay noise analysis and repair are done only in the post-routing stage and after all the functional noise violations have been repaired. Figure 5 shows the flow of the proposed methodology for SI management.

As is clear from the flow diagram in Figure 5, noise analysis and repair is done in 3 phases: an early prevention phase, a post-route functional repair phase, and a post-route noise aware timing analysis and repair phase; the last two following detailed routing. Section 3 discusses the details of the early phase. The functional noise analysis and repair are discussed in sections 4.1 and 4.2. The delay noise analysis and delay optimization are discussed in sections 5.1 and 5.2.

The methodology presented in these sections was developed over a period of time based on actual design experiences from several designs of SoC blocks, platforms, and chips. From these experiences, we have chosen the following 5 designs, all of them in 130nm technology, to serve as illustrative case studies:

1. A wireless communication chip (referred as SoC_Chip): This is a top level integration example comprising over 20 large SoC blocks and platforms, with some of the blocks allowing over-the-

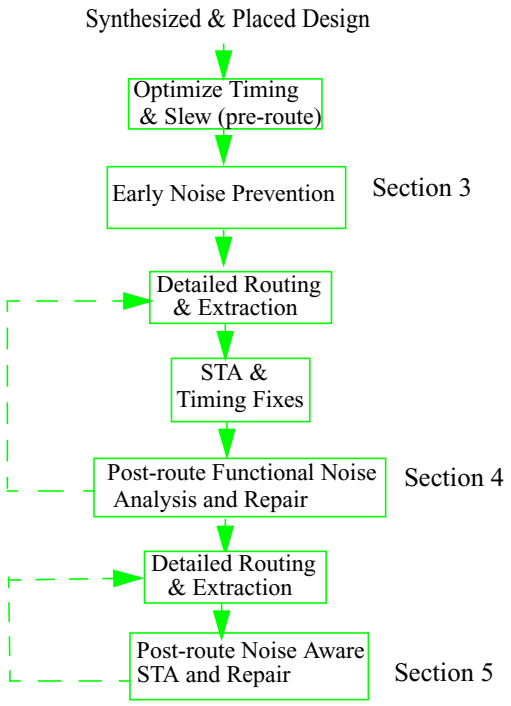

Figure 5. SI management methodology for SoC designs - Block level, platform level, and chip level flow

block routing. The top level also has a large sea-of-gates block which is routed along with the global nets. There are approximately 90,000 nets in this example. Each of the SoC blocks integrated at the top level are delivered in a timing-clean and SI-clean state.

2. A low-power IP platform (referred as SoC_Platform) which is integrated onto a wireless SoC. The platform consists of an IP core which is synthesized, 11 synthesized modules and peripherals and 24 compiled memories. The design has approximately 150,000 placed instances/cells and 160,000 nets. The platform is placed and routed flat at the top level as a sea-of-gates.

3. Two functional blocks in a wireless chip, SoC_Block_1 with approximately 45,000 nets and SoC_Block_2 with approximately 165,000 nets.

4. A high performance microprocessor core, SoC_Core with approximately 227,000 nets.

In the following sections, we use these designs to demonstrate various stages of our proposed flow in Figure 5.

3. NOISE PREVENTION METHODOLOGY

As the complexity of designs increases, it is becoming riskier to address noise problems with repairs alone. It is desirable that some design resources be invested early in the design cycle to prevent potential noise problems. There are a number of design techniques which can be applied for this purpose. These techniques can be applied to all nets in a pre-emptive manner or to selected noise-prone nets. Early noise analysis may be used to identify such nets based on floorplan/placement data and estimated routing [13].

Our noise prevention methodology adopts the following four techniques; the first three being related to physical design and the last to pre-route circuit optimization.

- Limiting the distance neighboring wires can travel in parallel: This prevents long parallel runs which create large coupling to dominant aggressors.
- Shielding: This technique is applied for structured routing topology, such as bus nets that are likely to experience noise problems from long parallel neighboring nets. Depending on the capability of routing tools, full or partial shielding is considered.
- Routing with extra spacing: This technique is also applied for structured routing topology, such as bus nets.
- Pre-route slew optimization: Slow slew rate at the receiving ends of a net indicates that the net is weakly driven and/or highly resistive, which makes the net susceptible to noise. With driver sizing or buffer (i.e. repeater) insertion, slew rate at the receiver inputs is improved. Although applying slew optimization globally results in stronger aggressor drivers, its benefit on overall noise due to the prevention of unacceptably weak victim drivers is greater.

While the parallel wire length and slew optimization techniques are applied globally to all the nets, shielding and extra spacing are applied only to nets with structured routing topology which are likely to experience noise problems. Determining specific settings for these techniques requires a few trials of routing and noise analysis since routing characteristics, such as routability and congestion, vary from design to design. For example, limiting parallel wire length is an effective way to reduce overall noise level. When set too aggressive, however, it results in problems in meeting timing requirement and routability by forcing wire routes to go through excessive turns and layer changes.

To demonstrate the effectiveness of some of the techniques described above, experimental data from the SoC_Platform and SoC_Block_2 is presented: Table 1 shows the number of delay noise violations for routing runs with different parallel wire length limits, in SoC_Platform. In SoC_Block_2, the number of functional violations are reduced to 500 when a parallel wire length limit of 500um is used compared to 1000 functional noise violations with no parallel wire length limit. The metric for choosing the parallel wire length limit is the reduction in number of noise violations while not causing routability problems and drc violations. Few trial routes are required to obtain a reasonable number, as it varies in each design.

Table 1: Delay noise violations for different parallel wire length limits

	Limit on parallel wire length	
	150 um	300 um
# violations	1936	3,142
worst delay slack	-0.22 ns	-0.57 ns

Table 2 shows noise violations for different slew rate targets in SoC_Platform. As the data shows, a faster slew constraint produces a better design for noise; both in number of functional and delay noise violations as well as the severity of worst path delay slack. Achieving faster slew may increase the layout area and the power consumption of the design as buffers are inserted to meet the target slew rate. Last row in Table 4 shows the number of inserted buffers, relative to the 1.0ns max. slew design, to achieve the respective slew targets. It has been observed that the increase in power consumption due to the inserted buffers is minimal as a result of improved slew rates which help reduce the short circuit power. These effects are taken into consideration as constraints during design decision process.

Table 2: Delay and functional noise violations for different slew constrains

	Slowest transition time		
	0.4 ns	0.7 ns	1.0 ns
# delay noise violations	2,818	3,258	5,771
worst delay slack	-0.22 ns	-0.54 ns	-0.70 ns
# functional noise violations	91	171	177
# inserted buffers	3559	510	-

Table 3 shows delay noise variation on a sample memory bus net in SoC_Platform and the change in the timing slack of the path that this net is part of, when the memory bus nets are routed with two different wire spacings. As can be seen in Table 3, additional delay due to crosstalk noise decreases by 95% from 0.22ns. to 0.01ns. when the memory bus net is routed with double spacing instead of single spacing. As a result of this decrease in delay noise, the timing slack of the path is improved by 97% from -0.68 ns to -0.02 ns.

Since these wires are expected to run long distances in parallel, it is good design practice to increase their separation, i.e. decrease crosstalk capacitance among them, in an early design stage.

Table 3: Delay noise violations for different wire spacing for memory bus

	Spacing	
	1x	2x
delay noise on memory bus net	0.22ns	0.01ns
timing slack on path	-0.68ns	-0.02ns

4.1. Functional Noise Analysis

In this section, we examine some practical issues in functional noise analysis which concerns the magnitude of glitches induced on victim nets.

Typical industrial designs may include more than millions of nets that need to be considered for noise. Since noise analysis needs to be done not only as design sign-off but following each noise avoidance/repair iteration, the turn-around time of noise analysis is of concern. In most practical settings, run time of no more than a few hours is expected for a design with about a million nets. To achieve such high speed during analysis, it is necessary to quickly filter out non-problematic nets which often constitute majority of the nets in the design. Such filtering is achieved through the use of a simplified analysis model, such as linearized driver and reduced-order interconnect model. At the same time, filtering is performed in a conservative manner to prevent missing real noise problems. It is shown that a well-designed noise filter can screen out more than 80% of the nets [1].

For the nets which are identified as potential noise violations during the filtering step, an accurate detailed noise analysis is performed. Strict noise criteria in today's high performance design often requires SPICE-level accuracy, including the effect of non-linearity of driver and receivers under noise conditions.

One of the major concerns in noise analysis is eliminating false violations, which are produced by conservative analysis model. For example, most full-chip noise analysis solutions use static approach which does not require input vectors. For a victim net, noise analysis identifies aggressors which produce worst case noise. Due to logical and timing correlation among the victim and aggressors, however, such worst case noise may not occur. It is critical for successful noise closure to eliminate these false noise violations from consideration. Noise analysis, in practice, should be able to utilize logic and timing correlation which are either automatically identified or specified by the designer. Some examples of logic constraints used for noise analysis in our methodology are:

- invert, same, imply, set_high/low/stable
- one-hot/cold, one-switching (among a group of signals)

In addition to localized logical constraints, global logic correlation is also effective. For example, a design may operate either in functional mode or in test mode. In such cases, signal nets which are active only during test mode should not be considered as aggressors for nets in functional mode, and vice versa.

Timing correlation information can be obtained from timing simulation or static timing analysis. For the proposed methodology, we used timing window information obtained from a static timing analysis tool. Depending on whether a net is being considered as a victim or an aggressor, different type of timing window may be used:

- **Activity window** specifies intervals during which a net can make transitions. This is useful to find a set of aggressors which can affect a victim net.
- **Sensitivity window** specifies intervals during which noise on a net is sensitive to specific type of noise.

Table 4 shows the number of functional noise violations in SoC_Core, where logical and timing correlations are used to eliminate false violations.

Table 4: Eliminating false violations with logical and timing correlation

	# violations
no constraint	595
with logical constraints only	555
with activity windows only	187
with activity+sensitivity windows	79
with logical and timing constraints	50

4.2. Functional Noise Repair Methodology

The challenge in repairing noise problems is, in so doing without causing problem for timing closure and physical design closure (routability and design rules conformance). This problem is compounded when the tool implementing a chosen noise repair action (e.g. buffer insertion, gate resizing, wide spacing, etc.) is not timing aware. That is when the repair tool does not have the required timing capability to determine the effect of noise repair on circuit's timing. A loose integration between the noise repair tool and a STA tool may partially offset this problem, but the run time of iterations would become prohibitively expensive. This is so because the changes in the circuit need to be communicated to the STA tool and likewise the revised timing information to the repair tool. Moreover, incremental timing may not be possible if the topology of the circuit is changed, as is the case with buffer insertion. In this case an expensive full timing will be necessary.

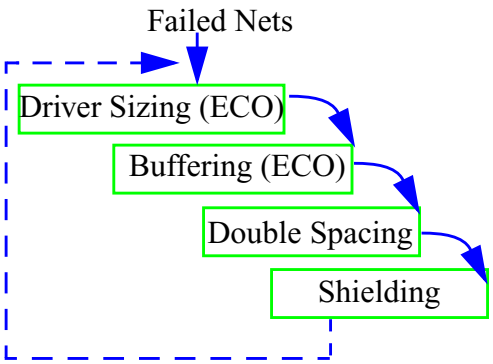

Figure 6. Post-route Functional Noise Repair

The above constraints may limit the number of options easily usable for correcting a given set of noisy nets in a given design situation. Figure 6 shows a flexible methodology for functional noise repair. Although the proposed methodology supports four repair methods, viz. driver sizing, buffer insertion, double spacing of nets, and shielding of nets; it does not enforce any particular order in which these options are to be used or even using every one of them. The following guidelines dictate which of the available actions will be used in a given design situation:

- Routing changes are to be preferred over sizing and buffering for fixing noise at the SoC chip integration stage. This assumes that all SoC blocks are timing-clean, and long global nets are already buffered in the previous timing optimization phase.

- While both driver sizing and buffering can be used for block level noise fixing, driver sizing is not to be preferred at the chip level since the drivers reside in the SoC blocks that are being integrated. However, gates in the sea-of-gates can be resized at the chip level, since they are legalized and routed at the chip level.

Table 5 demonstrates that most of the functional noise violations in the chip level global interconnects can be repaired using router directives. The table shows the progress in noise repair over 4 routing iterations for the SoC_Chip case. Column 2 shows the number of violations remaining to be addressed before rerouting. Column 3 and 4 show the number of nets double spaced and the number of violations fixed, respectively. It is interesting to note that the effectiveness (column 5) of double spacing diminishes with more iterations. In the early iterations, more number of noise violations are fixed than the number of nets double spaced. This is due to reduction of noise on some other nets for which the spaced nets are aggressors, in addition to noise reduction in the spaced nets themselves. As can be seen in Table 5, at the end of four iterations, the

number of noise violations was reduced 95%, from 885 to 47 by double spacing 1078 nets, proving the effectiveness of this method.

Table 5: Effectiveness of Double Spacing of Selected Nets in SoC_Chip

Itera -tion	#Violations Remaining	#Nets Spaced	#Violations Fixed	Effectiveness (#Fixed/ #Spaced)
1	885	240	356	1.48
2	529	176	198	1.13
3	331	254	220	0.87
4	111	308	64	0.21
Total	47	1078	838	0.78

Table 6: Effectiveness of Driver Sizing and Buffer Insertion in Soc Block

Iteration	#of Violations	# of Gates Sized	# of Buffers Inserted	# of Violations fixed
1	1380	1170	100	1180
2	500	400	80	450
3	280	200	50	250
4	170	120	45	160
5	90	75	40	85
6	70	50	15	65

Table 6 presents the results of applying gate sizing and buffer insertion on SoC_Block_1. In each iteration, gate sizing is the first choice and buffer insertion is applied to the nets that are not fixed with gate sizing. As can be seen, the number of functional noise violations are reduced by 95% after 6 iterations. A noteworthy observation is that new violations appear after each iteration. This is due to the change in the strength of gate sized or buffer inserted nets and also due to the change in the routes of several nets in each iteration.

5.1 Delay Noise Analysis

Noise-induced delay perturbation occurs when the victim and the aggressor nets switch simultaneously. Switching of victim and aggressors in the same direction speeds up the path through the victim and may lead to a hold time violation. Likewise, switching in the opposite directions can cause a setup time failure.

In design methodologies that do not explicitly analyze and address delay noise issues, it is a common practice to indirectly account for the noise effects on timing, by applying a multiplier (usually 1.5X to 2.0X) to the coupling capacitances and grounding them. The data in Table 7 shows the extent to which such a timing analysis can be pessimistic (as is the case with multiplier 2.0), or miss some violations (as is the case with multiplier 1.5) in SoC_Platform.

Table 7: Timing Analysis with Direct and Indirect Consideration of Noise Effect

STA Run with 1.5X Coupling Multiplier		STA Run with 2.0X Coupling Multiplier		STA Run Considering Noise Delay	
# Violat ions	Slack nS	# Violat ions	Slack nS	# Violatio ns	Slack nS
1138	-0.51	4865	-1.3	2936	-0.62

Supporting accurate delay noise analysis and repair in the physical design methodology thus helps deal with only realistic timing problems.

One of the difficulties in doing STA with noise effects is the need to iterate the timing and noise analysis until the timing windows (windows within which the signals can switch) converge [3][4]. We minimize the required time to reach an accurate noise aware static timing analysis, by considering infinite timing windows on the aggressor nets in the first iteration and then analyzing the critical paths only in the subsequent iterations, using updated timing windows. The proposed methodology also alleviates this problem to a large extent by postponing the delay noise analysis and repair to the very end, relying on elimination of a majority of delay violations when functional violations are fixed.

5.2. Delay Noise Repair

In this section, we present a methodology for delay optimization under crosstalk noise on SoC_Platform. Note that before noise aware static timing analysis is done, the design already passed through the previous stages of our proposed flow in Figure 5. As a result, SoC_Platform is timing clean based on “no-coupling STA” and functional noise free, at the starting point of this exercise.
We present a crosstalk noise aware static timing analysis and optimization algorithm for setup violations in Figure 7.

The algorithm presented in Figure 7 was applied on SoC_Platform, which required 2 STA iterations for the timing windows to converge. At this point, there were 88 setup violations and 63 hold violations. The worst setup violation was by 100ps whereas the worst hold violation was 20ps. Setup violations were fixed using the algorithm in Figure 7, where 76 driver gates of victim nets were sized up and 12 victim nets were double spaced from their worst aggressors.

Note that gate sizing and spacing of the victim nets on the data path will have competing effects on the hold violations which occur when the data signal is accelerated due to victim and aggressor nets switching in the same direction. For example, when the victim gate of the net with significant delay noise (decrease in delay) contribution to the path with hold violation, is sized up, this has a slow down effect through the reduction of delay noise as well as a speed up effect due to the increased size of the driver gate. Therefore, the hold violations were addressed by adding delay (with buffer sizing) to the clock path.

First STA Iteration with coupling and infinite windows

Subsequent STA iterations on critical paths with updated timing windows

Foreach failing path ***p***
s(p) = Slack(p)

Select net ***n*** which contributes most delta delay noise to path ***p***

(*) Resize the driver gate ***d*** of net ***n*** to the next higher power level

Incremental STA

If (new timing violations are created on other paths)
Resize ***d*** back to original size

Create router constraint to space the victim net ***n*** from its worst aggressors

Else If (timing on path ***p*** improves and no new timing violations are introduced)
Legalize placement of ***d***

If (s(p) >= 0)
Go to next path
Else
Select next net ***n*** with highest delta delay contribution and go to (*)

Figure 7.Crosstalk noise aware static timing analysis and optimization for setup violations

6. CONCLUSION

In this paper, we presented a crosstalk noise management methodology in an SoC physical design flow along with several case-studies using real, high-performance, low-power SoC designs. As part of the proposed methodology, techniques and issues have been discussed on prevention, analysis and repair of both functional and delay noise problems at block, platform, and chip levels in an SoC context. The proposed methodology is constructed carefully, detailing the specific techniques, chosen among several available, based on the nature and state of the design under consideration; thus favorable to address the signal integrity convergence problem in SoC designs.

References

[1] S. Alwar, D. Blaauw, A. Dasgupta, A. Grinshpon, R. Levy, C. Oh, B. Orshaw, S. Sirichotiyakul, and V. Zolotov, “Clarinet: A Noise Analysis Tool for Deep Submicron Design,” in Proceedings of Design Automation Conference DAC, June 2000, pp. 233-238.

[2] K. L. Shepard and V. Narayanan, “Noise in Deep submicron Digital Design,” in Proceedings of the IEEE International COnference on Computer Aided Design, November 1996, pp. 524-531.

[3] S. Sirichotiyakul, D. Blaauw, C. Oh, R. Levy, V. Zolotov, and J. Zuo, “Driver Modeling and Alignment for Worst-case delay noise,” in Proceedings of Design Automation Conference DAC, June 2001, pp. 720-725

[4] P. D. Gross, R. Arunachalam, K. Rajagopal, and L. T. Pileggi, “Determination of Worst-case Aggressor Alignment for Delay Calculation,” in Proceedings of IEEE International Conference on Computer Aided Design, ICCAD-98

[5] H. Zhou and D. F. Wong, “Global Routing with Crosstalk Constraints” in Proceedings of Design Automation Conference DAC, June 1998, pp. 374-377

[6] P. Saxena and C. L. Liu, “Crosstalk Minimization using Wire Perturbations”, in Proceedings of Design Automation Conference DAC, June 1999, pp. 100-103

[7] C. P. Chen and N. Menezes, “Noise Aware Repeater Insertion and Wire Sizing for on-chip Interconnect Using Hierarchical Moment Matching”, in Proceedings of Design Automation Conference DAC, June 1999, pp. 502-506

[8] C. J. Alpert, A. Devgan, and S. T. Quay, “Buffer Insertion for Noise and Delay Optimization”, in Proceedings of Design Automation Conference, June 1998, pp. 362-367

[9] I. H. R. Jiang, Y. W. Chang, and J. Y. Lou, “Crosstalk Driven Interconnect Optimization by Simultaneous Gate and Wire Sizing”, IEEE Transactions on Computer Aided Design, v. 19, pp. 999-1010, September 2000

[10] M. Hashimoto, M. Takahashi, and H. Onodera, “Crosstalk Noise Optimization by Post-layout Transistor Sizing”, in Proceedings of ISPD, April 2002, pp. 126-130

[11] A. Odabasioglu, M. Celik, and L. T. Pileggi, “PRIMA: Passive Reduced-order Interconnect Macromeodeling Algorithm”, IEEE Transactions on Computer Aided Design, v. 17, pp. 645-653, August 1998

[12] R. Arunachalam, K. Rajagopal, and L. T. Pileggi, “TACO: Timing Analysis with Coupling”, In Proceedings of ICCAD, Nov. 1998, pp. 1-8

[13] M. Becer, D. Blaauw, R. Panda, and I. N. Hajj, “Early Probabilistic Noise Estimation in Capacitively Coupled Interconnects,” in Proceedings SLIP, April 2002, pp. 77-84