Methodology From Chaos in IC Implementation

Kwangok Jeong[1] and Andrew B. Kahng[1,2]
[1]ECE and [2]CSE Departments, University of California at San Diego, La Jolla, CA, USA
kjeong@vlsicad.ucsd.edu, abk@cs.ucsd.edu

***Abstract*— Algorithms and tools used for IC implementation do not show deterministic and predictable behaviors with input parameter changes. Due to suboptimality and inaccuracy of underlying heuristics and models in EDA tools, overdesign using tighter constraints does not always result in better final design quality. Moreover, negligibly small input parameter changes can result in substantially different design outcomes.**

In this paper, we assess the nature of 'chaotic' behavior in IC implementation tools via experimental analyses, and we determine a methodology to exploit such behavior based on the 'multi-run' and 'multi-start' sampling concepts proposed in [2]. We also suggest the number of sampling trials that yields more predictably good solutions; this allows us to improve quality of design without any manual analysis or manipulation, without changing any existing tool flows, and without unnecessary expenditure of valuable computing resources.

I. Introduction

With the rapid scaling of design complexity and quality requirements for system-on-chip products, electronic design automation (EDA) tools are confronted with ever-increasing problem difficulty and instance size. Virtually all underlying problem formulations within the IC implementation flow are NP-hard, and only heuristics - typically with unknown suboptimality - are usable in practice. In addition, due to the shrinking scale and large variability of nanometer process technologies, the modeling and analysis of physical problems are often fraught with accuracy challenges.

Commercial EDA tools and methodologies are used in production design because they improve turnaround times and design productivity. However, due to the suboptimality of underlying heuristics and the inaccuracy of underlying models and analyses, designers typically expect to spend significant time and effort after the tool flow is completed, to analyze remaining problems and fix them manually. To trade off quality of results versus implementation time and effort, designers must be able to *predict* the output quality of heuristic tools and methodologies. Based on their predictions, designers may target different objectives (e.g., minimizing area rather than performance), or different values of objectives (e.g., higher clock frequency, or lower leakage power).

With the above in mind, "predictability" is one of the most important attributes of IC design automation algorithms, tools and methodologies [1]. However, EDA heuristic approaches do not always behave according to users' intentions. Yakowitz et al. [5] study random search in machine learning under noise, and Kushner [4] analyzes the effect of noise in machine learning as stochastic approximation. Hartoog [3] observes the existence of noise in a VLSI algorithm and proposes exploitation of the noise to produce various benchmark circuits. At the first Physical Design Symposium in 1997, Ward Vercruysse of Sun Microsystems referred to the back-end implementation flow as a "chaos machine". The implication is that a very small change in inputs could lead to a very large change in outputs. In 2001, Kahng and Mantik [2] examined 'inherent noise' in IC implementation tools, i.e., how equivalent inputs could lead to different outputs.

In the present work, we assess the nature of 'chaotic' behavior in IC implementation tools, with an aim to establishing a beneficial methodology from such chaos. To assess the chaotic behavior, we experimentally determine answers to four questions:

- How strongly correlated are post-synthesis netlist quality and post-routing design quality?
- How strongly correlated are design quality as evaluated by vendor place-and route tools and design quality as evaluated by signoff tools?
- What chaotic behavior is associated with input parameters of vendor synthesis tools?
- What chaotic behavior is associated with input parameters of vendor place-and-route tools?

Our experimental analyses confirm substantial "chaos" in vendor tools, e.g., worst timing slack can vary by up to hundreds of picoseconds, and area can vary by up to 16.4%, in a given (65nm) block implementation. Furthermore, there is little correlation between major design stages.

Based on our experimental results, we establish a methodology to exploit the unavoidable noise and chaos in back-end optimization tools. Our methodology is based on 'multi-run' and/or 'multi-start' execution of the design flow with small, intentional input parameter perturbations. This recalls the work of [2], which also proposed the exploitation of 'inherent tool noise' to achieve more predictable and stable tool outcomes. A key difference between [2] and our work is that the 'noise' sources in [2] - renaming cell instances, perturbing the design hierarchy, etc. - are not practically usable. On the other hand, our proposed 'chaos' levers, such as changing clock uncertainty by 1 picosecond, are trivially implemented and transparent to the design flow. With the increasing availability of multiple and parallel computing platforms - multiple workstations in server farms, multiprocessor workstations, multicore processors, and multithreaded cores - our multi-run approach can be deployed with negligible impact on the overall design cycle time.[1] We also provide a method to find which input parameters are most sensitive to the perturbation, and optimal number of runs to obtain reasonably good and predictable solutions.

The remainder of this paper is organized as follows. In Section II, we examine our four motivational questions regarding the unpredictability and 'chaotic' nature of commercial tools.

[1]In fact, since better-quality block implementations typically close faster, our approach can potentially reduce overall design cycle time.

We describe our proposed method for exploiting chaotic tool behavior in Section III. Finally, Section IV gives conclusions.

II. Chaos in Implementation Tools Due to Intentional Input Disturbance

There are two basic types of knobs that affect the quality of optimization.

- **Tool-specific options -** specifically, command options to turn on/off specific optimization heuristics. (We do not explore this, but use the same default tool options in all of our experiments.)
- **Design-specific constraints -** notably timing-related constraints (clock cycle time, clock uncertainty, input/output minimum/maximum delay, etc.) and floorplan-related constraints (utilization, aspect ratio, primary pin locations, etc.).

In our study, we only examine the impact of intentional perturbation of design-specific constraints. For synthesis, we explore impact of timing-related constraints, and for placement and routing (P&R), we explore the impact of both timing- and floorplan-related constraints. We vary design-specific parameters by small amounts, and measure design quality metrics such as worst negative slack (WNS), total negative slack (TNS), and total standard-cell area.

We implement four testcases using a *TSMC* 65nm *GPLUS* library: two open-source cores, *AES* and *JPEG*, obtained as RTL from *opencores.org* [6], and two subblocks *LSU* (load and store unit) and *EXU* (execution unit) of the *OpenSparcT1* design, obtained from the *Sun OpenSPARC Projects* site [7]. We use a traditional timing-driven synthesis, placement and routing flow, and analyze final timing quality using a signoff RC extraction (*Synopsys STAR-RCXT* [13]) and a signoff static timing analyzer (*Synopsys PrimeTime* [10]). All specific tool versions are given in the references section below. Table I shows one of the implementation results for each of our testcases at the signoff stage. The remainder of this section describes our experimental investigation of the four motivating questions above - on the correlation between design stages, and on the chaotic behavior of implementation tools.

TABLE I

Testcase information at signoff with nominal parameter values. Implementation uses *Synopsys Design Compiler* and *Synopsys Astro*. 'Skew' is the clock uncertainty constraint given at synthesis stage to account for clock skew occurring at placement and routing stages.

Design	Cycle time (*ns*)	Skew (*ns*)	IO Delay (*ns*)	WNS (*ns*)	TNS (*ns*)	#Cells	Area (μm^2)
AES	1.7	0	0	-0.095	-0.210	22438	48957
JPEG	2.2	0	0	-0.224	-10.937	69845	178696
LSU	1.2	0	0	-0.327	-170.835	24945	113479
EXU	1.2	0	0	-0.760	-423.869	20382	69780

A. Correlation of Quality between Design Stages

Motivating Question 1: How strongly correlated are post-synthesis netlist quality and post-routing design quality?

It is by no means certain that a better-quality synthesized netlist will eventually lead to a better-quality design after placement and routing (P&R). Our first experiments examine the impact of the quality of input netlists on final P&R outcomes. We synthesize the *AES* core with various clock cycle times to have different timing quality at a target clock cycle time in synthesis. For each synthesized netlist, we perform placement and routing at the given target clock cycle time. We use *Cadence RTL Compiler* for synthesis and *Cadence SOC Encounter* for P&R.

Table II summarizes the worst negative slack values after synthesis and after P&R, respectively. The first column shows the clock cycle time applied at synthesis stage, and the second column shows the worst negative slack (WNS) with the target clock cycle time, i.e., 2*ns*. The third and fourth columns show the WNS values after placement and routing, obtained from *Cadence SOC Encounter* (*SOCE*) [9] and *Synopsys PrimeTime* (*PT*) [10], respectively.

TABLE II

Timing quality of synthesized netlists versus timing quality after placement and routing, and signoff.

Clock used synthesis (*ns*)	WNS (*ns*) with 2.0*ns* clock after synthesis	Clock used P&R (*ns*)	WNS (*ns*) from *SOCE*	WNS (*ns*) from *PT*
1.60	0.400	2.0	0.171	-0.249
1.80	0.200	2.0	0.088	-0.196
1.90	0.101	2.0	0.112	-0.195
1.95	0.051	2.0	0.074	-0.449
2.00	0.001	2.0	0.088	-0.252
2.10	-0.097	2.0	0.088	-0.214
2.20	-0.196	2.0	0.120	-0.281
2.40	-0.395	2.0	0.162	**-0.081**

From the data, we observe that an input netlist with better timing slack netlist does not always result in better timing slack after placement and routing. Furthermore, due to the timing miscorrelation between P&R and signoff tools, the worst-slack netlist from synthesis stage, obtained using largest clock cycle time (i.e., 2.4*ns*), actually results in the best timing at signoff. How this can occur is suggested by Figure 1, which gives a rank-correlation plot of endpoint slack values of timing paths in the *AES* netlist, between post-synthesis and post-placement stages. The correlation coefficient is only 0.421. Due to this miscorrelation between synthesis and P&R stages, the eventual benefit from maximizing the quality of the post-synthesis netlist is unclear. This gives us some intuition that any incremental tool runs should be directed to the P&R stage rather than the synthesis stage.

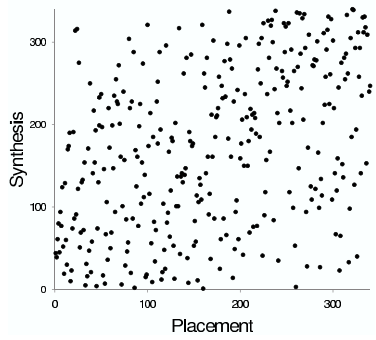

Fig. 1. Rank correlation of timing slack at all endpoints, i.e., input pins of all registers, between post-synthesis and post-placement stages for the *AES* designs.

Motivating Question 2: How strongly correlated are design quality as evaluated by vendor place-and route tools and design quality as evaluated by signoff tools?

Aside from the suboptimal nature of underlying optimization algorithms, there is another source of noise in the traditional implementation flow. Timing optimization is always based on (incremental) timing analysis. As is well known, such timing analysis requires models for gates and interconnect: timing and power models for gates are precharacterized in lookup tables using SPICE, and interconnect RC models are extracted from layout with precharacterized capacitance tables. Using the gate and interconnect models, effective load capacitance, slew degradation, interconnect delay are calculated using, e.g., asymptotic waveform estimation. The effective load capacitance and slew values thus obtained are then used as table indices to find corresponding gate delay values from delay tables. However, optimization tools use simplified models and embedded calculators to evaluate timing with the least possible computational expense. As a result, timing results seen by optimization tools can differ from those seen by signoff timing analysis tools.

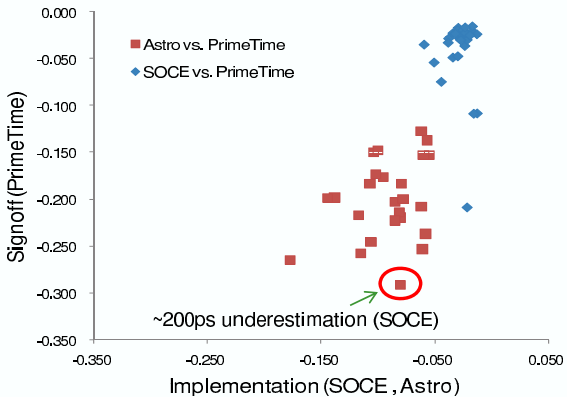

Fig. 2. Worst negative slack correlation between a signoff timing analysis tool *Synopsys PrimeTime* on the y-axis, and implementation tools *Synopsys ASTRO* (rectangles) and *Cadence SOC Encounter* (diamonds) on the x-axis.

Figure 2 shows the timing correlation between a signoff timing analyzer *Synopsys PrimeTime*, equipped with a signoff RC extractor *Synopsys Star-RCXT*, and two place-and-route tools, *Synopsys Astro* (*ASTRO*) [11] and *Cadence SOC Encounter* (*SOCE*) [9], for 29 different implementations of the *AES* core. We observe that more than 200*ps* of timing slack difference can occur.

To understand methodology implications of P&R vs. signoff discrepancies, we make a brief excursion into root causes of such discrepancies. Typical root causes are as follows.

- **RC-extraction.** We extract RC values from a signoff extractor *Synopsys Star-RCXT* (*STAR*) and a place-and-route tool *Cadence SOC Encounter* (*SOCE*). We compare the extracted capacitance values using the *Cadence Ostrich* program. Figure 3 compares the extracted capacitances from *STAR* with those from *SOCE*. We observe that *SOCE* underestimates capacitance by 18.6%.[2] This significant difference may explain why SOCE so consistently sees optimistic timing slacks when compared to the signoff tool (Figure 2).

[2]These data do not speak to the accuracy of extraction within a place-and-route tool. E.g., such a discrepancy may arise from a simplified two-dimensional capacitance table given to the place-and-route tool.

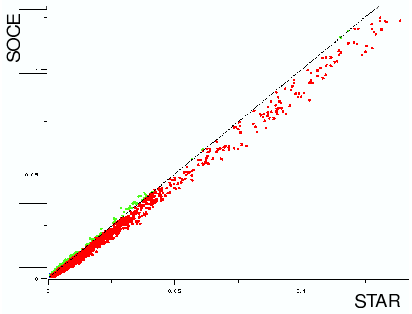

Fig. 3. Normalized capacitance correlation between a signoff RC extractor *Synopsys Star-RCXT* (*STAR*) on the x-axis, and a place-and-route tool Cadence SOC Encounter (*SOCE*) on the y-axis.

- **Delay calculation.** We compare timing between *SOCE* and *PT* with the same RC parasitic file from *SOCE*, to eliminate the impact of the discrepancy in RC extraction. Table IV shows the WNS and TNS calculated from both tools for our four testcases with default input parameters. The data suggests that delay calculation in *SOCE* and *PT* is well-correlated, as is usually the case. (N.B.: Viable implementation and signoff tools will calculate delay on a given extracted path to within at most a couple of tens of picoseconds difference from the 'golden' tool.)

TABLE III
WNS AND TNS FROM SIGNOFF TIMING ANALYZER *Synopsys PrimeTime* (*PT*) AND P&R TOOL *Cadence SOC Encounter* (*SOCE*), USING THE SAME RC PARASITICS FROM *SOCE*.

Design	*PT*		*SOCE*	
	WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)
AES	0.144	0	0.146	0
JPEG	0.129	0	0.095	0
LSU	-0.002	-0.004	-0.005	-0.040
EXU	-0.171	-11.483	-0.183	-12.669

- **Other: Path-tracing and signal integrity in timing analysis.** (1) Endpoint slack differences greater than several hundred picoseconds are typically due to discrepancies in path-tracing - i.e., the interpretation of timing constraints, timing exceptions, generated clocks, cycle-breaking in the timing graph, etc. - in the static timer. However, the testcases we use have only simple timing constraints consisting of one clock definition along with input/output delay constraints. Thus, path-tracing is not a factor in the timing miscorrelation between P&R and signoff in our experiments. (2) Endpoint slack differences on the order of 100*ps* are often due to discrepancies in signal integrity (e.g., crosstalk-induced delay variation) analyses. However, all timing analyses in our experiments are performed with signal integrity options turned off. Thus, signal integrity is not a factor in observed timing miscorrelations, either.

Although there is a discrepancy between place-and-route tools and signoff tools, we believe that this discrepancy (in all experimental data we report here) is systematic and attributable to internal RC extraction and path delay calculation. As shown in Figure 2, *SOCE* consistently underestimates the

timing slack. This correlated discrepancy can be predicted and compensated. We may infer that design quality at the P&R stage is a stronger lever on final signoff timing than design quality at the synthesis stage.

B. Chaotic Behavior in Optimization Tools

In this subsection, we assess the impact of intentional perturbations applied to input parameters of synthesis and place-and-route tools. (As noted above, the perturbations that we study differ from the netlist manipulations studied in [2], and are transparently applied without changing the design flow.)

Motivating Question 3: What chaotic behavior is associated with input parameters of vendor synthesis tools?

We analyze the impact of perturbation of timing-related parameters, such as clock cycle time, input/output delay, and clock uncertainty, at the synthesis stage. We use two commercial gate-level synthesis tools, *Synopsys Design Compiler* (*DC*) [12] and *Cadence RTL Compiler* (*RC*) [8]. We vary each parameter by an amount ranging from -3*ps* to 3*ps* with 1*ps* increments, and measure the changes in netlist quality metrics. Table II-B summarizes WNS and total standard-cell area of the resulting synthesized netlists.

Ideally, small perturbations of, e.g., a few picoseconds in input parameters should not change output quality, or should have predictable consequences. For example, a reduction of clock cycle time by 1 picosecond can be reasonably expected to result in a reduction of timing slack by the same 1 picosecond, since the difficulty of design optimization is virtually unchanged. However, due to the unpredictability of optimization tools, the resulting design quality appears to be random. We observe up to 53*ps* and 34*ps* of WNS variations in *DC* and in *RC*, respectively. Among the results, we observe that some input perturbations result in better timing quality than the original (without perturbations) design optimization. This improvement can be regarded as a benefit from 'chaotic behavior' of the design optimization tools. However, as discussed in Section II-A, input parameter perturbations at synthesis may not have a great effect on final signoff design quality, due to the unpredictable miscorrelation between synthesis and place-and-route tools.

Motivating Question 4: What chaotic behavior is associated with input parameters of vendor place-and-route tools?

We also analyze the impact of perturbations of both timing-related parameters and floorplan-related parameters, in place-and-route tools. Since our second experiment above suggested that the quality of an input netlist is not preserved during placement and routing, we take one synthesized netlist arbitrarily from the synthesis results in the Table IV results, and perform traditional timing-driven placement and routing. We use two place-and-route tools, *Synopsys Astro* (*ASTRO*) and *Cadence SOC Encounter* (*SOCE*). The nominal clock cycle time, input/output delay, and clock uncertainty values are shown in Table I, and nominal utilization and aspect ratio are 70% and 1.0, respectively.

Table V summarizes worst negative slack (WNS) and total negative slack (TNS) calculated using a signoff RC extraction and static timing analysis. We do not include the final area of placed and routed designs due to space limitations in the table, but the variation of area is observed to be as high as 16.4% in the *AES* core when utilization is increased by just 1%.[3]

From the table, we again observe that small input parameter perturbations give rise to large timing slack changes, e.g., WNS varies by up to 190*ps* (from -96*ps* to -287*ps*) in *EXU*, and TNS varies by more than 69*ns* (from -152*ns* to -83*ns*) in *JPEG*.

C. Summary of observations

Our experimental study provides the following evidence for 'chaotic behavior'.

- Input parameter perturbation in synthesis results in up to 53*ps* of WNS variation, and sometimes produces better-quality netlists than the original design constraints. However, because there is little correlation between post-synthesis netlist quality and post-routing design quality, improved synthesis results will not necessarily improve results after placement and routing.
- There is a large discrepancy in timing slack between optimization tools and signoff tools, which in our studies arises mainly from (i) the difference between optimization-internal and signoff RC extractors, and (ii) small discrepancies in delay calculation between optimization and signoff tools. Timing slack with the same RC parasitics can vary up to 34*ps* solely due to delay calculation discrepancies. However, the discrepancy may be predictable and hence compensatable.
- Input parameter perturbation in place-and-route tools results in up to 190*ps* of WNS variation, up to 46*ns* of TNS variation, and up to 16.4% variation in total standard-cell area. In contrast to chaos in synthesis outcomes, the chaos in place-and-route outcomes appears more exploitable to improve final signoff quality.
- This chaotic behavior is unpredictable. In particular, for different design types or domains, it may not be possible to assess differences in nature of the chaotic behavior. However, from Table V, we can observe that more timing-critical designs (e.g., designs with more negative timing slack, more violating paths, or smaller clock cycle time) tend to show more sensitivity to the input parameter perturbation.

III. Predictability From Chaos

In the previous section, we observed 'chaos': large variation in final implementation quality arising from small (negligible) input parameter changes. Frequently, results after small input perturbations are better than those obtained using nominal input parameter values. From the results, we expect that we can achieve better design quality (and, potentially, improved design cycle time) without additional human effort or flow modifications. The key idea: run multiple times with small input perturbations, and return the best-quality solution.

If only one CPU or tool license is available, then we can run multiple times on one CPU ('*multi-run*'), trading design quality for runtime. But when there are idle CPUs and licenses

[3]Because area and power are correlated, as the area changes, the design power is also affected by the input parameter perturbation. However, the degree of "chaos" for power will be less: WNS is a "max" attribute of the design, while power (as well as TNS) is a "sum" or "average" attribute.

TABLE IV

IMPACT OF INTENTIONAL PERTURBATIONS OF INPUT PARAMETERS FOR SYNTHESIS TOOLS. *RC* AND *DC* INDICATE RESULTS FROM *Cadence RTL Compiler* AND *Synopsys Design Compiler*, RESPECTIVELY. **BOLD** ENTRIES INDICATE MIN/MAX IN EACH EXPERIMENT.

Parameter	Noise (Δ)	*AES*				*JPEG*			
		DC		*RC*		*DC*		*RC*	
		WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)
Clock Cycle	-3ps	-0.243	42261	**-0.051**	50947	-0.154	300731	-0.120	220030
	-2ps	-0.222	43986	-0.050	51036	-0.159	294333	-0.117	223779
	-1ps	-0.218	44058	-0.047	50752	-0.169	299315	**-0.122**	220312
	0ps	**-0.245**	41735	-0.046	50673	-0.165	299270	-0.110	224222
	1ps	-0.229	42985	-0.048	50216	**-0.177**	298607	-0.118	219223
	2ps	-0.232	42645	**-0.041**	51154	-0.174	298420	-0.112	216779
	3ps	**-0.216**	44168	-0.041	51033	**-0.148**	299912	**-0.110**	222282
Clock Uncertainty	-3ps	**-0.245**	41735	**-0.046**	50673	**-0.165**	299270	**-0.111**	223544
	-2ps	-0.245	41735	**-0.046**	50673	**-0.165**	299270	-0.110	222552
	-1ps	-0.245	41735	-0.046	50673	-0.165	299270	-0.109	223065
	0ps	-0.245	41735	-0.046	50673	-0.165	299270	-0.110	224222
	1ps	**-0.227**	43286	-0.046	50673	-0.165	299270	-0.108	223945
	2ps	-0.245	41735	-0.046	50673	-0.165	299270	**-0.107**	222697
	3ps	-0.245	41735	-0.046	50673	-0.165	299270	-0.112	221580
IO Delay	-3ps	**-0.216**	44168	**-0.041**	51033	-0.174	298123	**-0.110**	222282
	-2ps	**-0.252**	41892	-0.041	51154	**-0.155**	299840	-0.116	216878
	-1ps	-0.234	42482	-0.048	50216	**-0.177**	298607	-0.118	219360
	0ps	-0.245	41735	-0.046	50673	-0.165	299270	-0.110	224222
	1ps	-0.233	42860	-0.047	50752	-0.164	299244	-0.121	220106
	2ps	-0.222	43986	-0.050	51036	-0.164	299725	-0.115	223658
	3ps	-0.216	44314	**-0.051**	50947	-0.169	299461	**-0.123**	221147
Best	-	**-0.216**	-	**-0.041**	-	**-0.148**	-	**-0.107**	-
Worst	-	**-0.252**	-	**-0.051**	-	**-0.177**	-	**-0.123**	-
Delta	-	**0.036**	-	**0.010**	-	**0.029**	-	**0.016**	-

Parameter	Noise (Δ)	*LSU*				*EXU*			
		DC		*RC*		*DC*		*RC*	
		WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)	WNS (*ns*)	Area (μm^2)
Clock Cycle	-3ps	-0.110	84943	**-0.042**	94275	-0.133	50685	-0.040	58481
	-2ps	-0.110	84983	**-0.047**	93995	**-0.095**	51661	-0.052	58315
	-1ps	-0.102	85166	-0.042	94450	-0.098	51571	-0.049	58215
	0ps	-0.104	85052	-0.044	94119	-0.128	50442	-0.042	58287
	1ps	-0.105	84747	-0.046	93509	-0.108	50858	**-0.057**	58330
	2ps	**-0.101**	84915	-0.044	92629	**-0.145**	50314	-0.036	58217
	3ps	**-0.108**	84904	-0.046	92028	-0.099	51221	**-0.032**	58044
Clock Uncertainty	-3ps	**-0.104**	85086	**-0.048**	93000	-0.115	50763	-0.026	58264
	-2ps	-0.108	84928	-0.045	93594	-0.124	50638	**-0.046**	58463
	-1ps	-0.105	85007	-0.048	94350	**-0.147**	49995	-0.031	58508
	0ps	-0.104	85052	-0.044	94119	-0.128	50442	-0.042	58287
	1ps	-0.109	84840	-0.045	93983	-0.117	50575	-0.029	58458
	2ps	**-0.111**	84744	-0.044	93729	**-0.105**	50804	**-0.023**	58543
	3ps	-0.105	85097	**-0.042**	93743	-0.134	50329	-0.035	58659
IO Delay	-3ps	-0.106	84834	**-0.054**	92105	-0.099	51221	-0.042	57967
	-2ps	-0.107	84757	**-0.040**	93817	-0.118	50995	**-0.036**	58217
	-1ps	-0.110	84699	-0.042	94372	-0.125	50483	-0.040	58434
	0ps	**-0.104**	85052	-0.044	94119	-0.128	50442	-0.042	58287
	1ps	-0.109	84863	-0.047	94218	-0.098	51571	**-0.049**	58215
	2ps	-0.110	85101	-0.042	94128	**-0.095**	51661	-0.049	58249
	3ps	**-0.112**	84833	-0.044	94010	**-0.133**	50685	-0.040	58481
Best	-	**-0.101**	-	**-0.040**	-	**-0.095**	-	**-0.023**	-
Worst	-	**-0.112**	-	**-0.054**	-	**-0.147**	-	**-0.057**	-
Delta	-	**0.011**	-	**0.014**	-	**0.052**	-	**0.034**	-

in a large computing farm, we may be able to use as many CPUs as possible in parallel ('*multi-start*') without affecting the design cycle time. In either scenario, we obtain a "best-of-*k*" methodology: (i) run *k* times on a CPU while varying input parameter values, or (ii) start *k* runs with different input parameter values, and take the best results out of the *k* different runs.

If we wish to experimentally determine the best number *k* of runs - in a statistically meaningful manner - it at first seems necessary to execute many trials for each possible value of *k*. For example, we could conduct *N* trials each with a set of *k* runs, then record the best solution out of each set of *k* runs, and then find the average ('expected') best solution for the given value of *k*. If we know the average best-of-*k* solution value for each value of *k*, then we can determine which *k* gives reasonably good solutions compared to the cost of resources. The challenge is that the above-described procedure requires far too many runs. Naively, if we run *N* trials of "best-of-*k*" runs, we may require $N \times k$ separate runs. And if we test six different values of *k* numbers - e.g., 1, 2, 3, 4, 5, and 10 - through 100 trials, we would have to perform $100 \times (1 + 2 + 3 + 4 + 5 + 10) = 2500$ separate runs.

To reduce the number of test runs needed to determine the best *k* value, we use the following sampling approach, which was originally presented in [2].

1) Run a smaller number of different runs, e.g., 50 times with different inputs, instead of 2500 runs as in the previous example.
2) Record a quality metric, e.g., WNS, for each run. Then, assume that the set of solutions for the 50 runs is the 'virtual' solution space.
3) Randomly sample *k* solutions out of the 'virtual' solution space *N*=100 times, and record the best results for each choice of *k* solutions. This process replaces the actual *N* trials of *k* runs each.
4) Find minimum, maximum, and average values of the best results recorded from the *N* sampling trials.

For our experiments, we use this "best-of-*k*" method out of

TABLE V

IMPACT OF INTENTIONAL PERTURBATION OF INPUT PARAMETERS IN PLACEMENT AND ROUTING TOOLS. *ASTRO* AND *SOCE* REPRESENT THE RESULTS FROM *Synopsys Astro* AND *Cadence SOC Encounter*, RESPECTIVELY. **BOLD** ENTRIES INDICATE MIN/MAX IN EACH EXPERIMENT.

Parameter	Noise (Δ)	*AES*				*JPEG*			
		ASTRO		*SOCE*		*ASTRO*		*SOCE*	
		WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)
Clock Cycle	-3ps	-0.105	-1.388	-0.039	-0.183	-0.170	-40.785	-0.188	-127.129
	-2ps	-0.105	-1.398	**-0.033**	**-0.117**	**-0.160**	-29.732	**-0.205**	**-91.147**
	-1ps	**-0.097**	-1.135	-0.035	-0.189	-0.183	**-28.936**	-0.186	**-147.223**
	0ps	-0.102	**-1.109**	**-0.084**	-0.287	-0.234	**-58.009**	-0.198	-140.570
	1ps	**-0.107**	-1.500	-0.061	-0.375	-0.209	-42.422	-0.193	-105.972
	2ps	-0.106	**-1.511**	-0.058	**-0.435**	**-0.248**	-53.286	**-0.180**	-138.579
	3ps	-0.106	-1.503	-0.053	-0.220	-0.196	-34.022	-0.183	-135.538
Clock Uncertainty	-3ps	-0.107	**-1.500**	**-0.084**	**-0.287**	**-0.136**	-32.186	-0.192	-119.593
	-2ps	-0.107	-1.500	**-0.084**	**-0.287**	-0.204	-34.042	-0.191	-104.662
	-1ps	-0.107	-1.500	-0.084	-0.287	-0.187	-52.320	**-0.208**	-129.670
	0ps	**-0.102**	**-1.109**	-0.084	-0.287	**-0.234**	**-58.009**	-0.198	-140.570
	1ps	**-0.114**	-1.119	-0.084	-0.287	-0.172	-32.838	**-0.177**	**-99.811**
	2ps	-0.114	-1.119	-0.084	-0.287	-0.141	-28.878	-0.188	**-140.688**
	3ps	-0.114	-1.119	-0.084	-0.287	-0.157	**-25.764**	-0.197	-103.762
IO Delay	-3ps	-0.106	**-1.503**	-0.053	-0.220	-0.159	**-29.123**	-0.183	-135.538
	-2ps	-0.106	-1.511	-0.058	**-0.435**	-0.227	-54.296	**-0.180**	-138.579
	-1ps	**-0.107**	-1.500	-0.061	-0.375	-0.207	-43.397	-0.193	-105.972
	0ps	-0.102	**-1.109**	**-0.084**	-0.287	-0.234	-58.009	-0.198	-140.570
	1ps	**-0.097**	-1.135	-0.035	-0.189	**-0.249**	**-65.507**	-0.186	**-147.223**
	2ps	-0.105	-1.398	**-0.033**	**-0.117**	**-0.157**	-31.410	**-0.205**	**-91.147**
	3ps	-0.105	-1.388	-0.039	-0.183	-0.198	-42.760	-0.188	-127.129
Aspect Ratio	-0.03	-0.104	-1.480	**-0.036**	**-0.214**	**-0.132**	-20.743	-0.200	**-152.493**
	-0.02	-0.114	-1.312	-0.036	-0.214	-0.183	-42.322	**-0.177**	-115.859
	-0.01	-0.120	-1.543	**-0.090**	**-0.507**	-0.199	-34.466	-0.183	-139.986
	0.00	**-0.102**	**-1.109**	-0.084	-0.287	**-0.234**	**-58.009**	-0.198	-140.570
	0.01	-0.112	-1.520	-0.041	-0.269	-0.145	-38.358	-0.195	-123.821
	0.02	**-0.136**	**-1.974**	-0.041	-0.269	-0.193	-44.742	-0.203	-120.917
	0.03	-0.120	-1.868	-0.061	-0.439	-0.166	**-20.162**	**-0.222**	**-102.049**
Placement Utilization	-3%	**-0.149**	**-3.106**	**-0.046**	-0.339	-0.186	-42.323	**-0.173**	**-82.535**
	-2%	-0.127	-1.602	-0.062	-0.274	-0.190	-44.052	**-0.209**	-102.873
	-1%	-0.121	-2.057	-0.047	**-0.113**	-0.209	**-60.275**	-0.187	-133.042
	0%	**-0.102**	**-1.109**	-0.084	-0.287	**-0.234**	-58.009	-0.198	**-140.570**
	1%	-0.133	-1.235	**-0.112**	**-5.392**	**-0.133**	-27.029	-0.187	-107.559
	2%	-0.132	-2.126	-0.046	-0.543	-0.139	-24.120	-0.179	-96.438
	3%	-0.129	-1.390	-0.046	-0.679	-0.140	**-19.303**	-0.190	-105.871
Best	-	**-0.097**	**-1.109**	**-0.033**	**-0.113**	**-0.132**	**-19.303**	**-0.173**	**-82.535**
Worst	-	**-0.149**	**-3.106**	**-0.112**	**-5.392**	**-0.249**	**-65.507**	**-0.222**	**-152.493**
Delta	-	**0.051**	**1.997**	**0.080**	**5.279**	**0.117**	**46.204**	**0.049**	**69.958**

Parameter	Noise (Δ)	*LSU*				*EXU*			
		ASTRO		*SOCE*		*ASTRO*		*SOCE*	
		WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)	WNS (*ns*)	TNS (*ns*)
Clock Cycle	-3ps	-0.104	-2.061	-0.146	-26.218	**-0.307**	**-1.945**	-0.174	-2.828
	-2ps	**-0.082**	**-1.298**	-0.164	-27.249	-0.262	-2.472	-0.161	-2.940
	-1ps	-0.142	-3.539	-0.171	-27.489	-0.271	-3.869	**-0.287**	**-8.692**
	0ps	-0.127	**-5.096**	-0.140	-21.140	**-0.160**	-2.740	-0.164	-3.314
	1ps	-0.116	-1.735	**-0.130**	**-20.008**	-0.237	-2.084	-0.177	-2.403
	2ps	**-0.144**	-3.611	-0.167	**-28.938**	-0.267	-3.992	-0.178	-2.693
	3ps	-0.090	-1.508	**-0.178**	-25.584	-0.274	**-7.430**	**-0.096**	**-1.191**
Clock Uncertainty	-3ps	-0.125	-4.246	**-0.182**	-27.546	-0.267	-1.510	**-0.176**	-2.681
	-2ps	**-0.108**	-0.507	-0.159	**-28.859**	**-0.186**	**-1.424**	-0.157	-2.554
	-1ps	**-0.209**	**-1.674**	-0.158	-21.579	-0.257	-4.424	**-0.133**	**-2.086**
	0ps	-0.127	-5.096	-0.140	-21.140	-0.160	-2.740	-0.164	**-3.314**
	1ps	-0.140	-3.462	-0.156	-24.432	**-0.300**	-3.832	-0.159	-2.354
	2ps	-0.120	-2.813	**-0.130**	-23.977	-0.200	-2.292	-0.163	-2.817
	3ps	-0.153	**-6.908**	-0.137	**-18.991**	-0.241	**-6.533**	-0.168	-2.256
IO Delay	-3ps	**-0.113**	-2.114	-0.150	-26.238	-0.245	**-6.694**	**-0.096**	**-1.191**
	-2ps	**-0.160**	-1.401	-0.156	-30.726	-0.277	-4.769	-0.178	-2.693
	-1ps	-0.136	**-0.728**	**-0.223**	**-48.136**	-0.169	-2.905	-0.177	-2.403
	0ps	-0.127	**-5.096**	**-0.140**	-21.140	**-0.160**	-2.740	-0.164	-3.314
	1ps	-0.125	-2.970	-0.143	-25.409	-0.173	-3.459	**-0.287**	**-8.692**
	2ps	-0.135	-1.177	-0.158	**-18.943**	-0.207	-5.755	-0.161	-2.940
	3ps	-0.114	-2.006	-0.152	-23.232	**-0.302**	**-1.595**	-0.174	-2.828
Aspect Ratio	-0.03	-0.139	-1.721	-0.146	-20.466	-0.299	-3.425	**-0.233**	-4.862
	-0.02	-0.142	**-0.625**	-0.160	**-27.070**	-0.220	-6.548	-0.174	-3.066
	-0.01	**-0.230**	-3.382	**-0.174**	**-19.188**	**-0.152**	-2.934	-0.195	-5.307
	0.00	-0.127	**-5.096**	-0.140	-21.140	-0.160	**-2.740**	-0.164	-3.314
	0.01	**-0.065**	-0.304	**-0.128**	-22.411	**-0.306**	-2.845	**-0.129**	**-1.549**
	0.02	-0.167	-4.990	-0.144	-22.588	-0.217	-5.895	-0.129	-1.549
	0.03	-0.129	-3.994	-0.144	-23.951	-0.216	**-6.801**	-0.228	**-5.932**
Placement Utilization	-3%	**-0.149**	-1.817	-0.136	-19.187	**-0.185**	-2.172	-0.200	-3.432
	-2%	-0.139	-2.908	**-0.184**	**-31.209**	**-0.261**	-3.382	-0.239	-5.484
	-1%	**-0.083**	-0.965	**-0.133**	**-16.787**	-0.243	-2.361	-0.201	-4.398
	0%	-0.127	**-5.096**	-0.140	-21.140	-0.160	-2.740	**-0.164**	-3.314
	1%	-0.096	-3.213	-0.139	-20.818	-0.189	**-1.091**	-0.188	**-2.106**
	2%	-0.123	-4.113	-0.164	-27.267	-0.218	**-6.299**	-0.209	**-6.203**
	3%	-0.100	**-0.897**	-0.134	-25.518	-0.221	-5.300	**-0.240**	-6.156
Best	-	**-0.065**	**2.813**	**-0.128**	**-16.787**	**-0.152**	**-1.091**	**-0.096**	**-1.191**
Worst	-	**-0.230**	**-6.908**	**-0.223**	**-48.136**	**-0.307**	**-7.430**	**-0.287**	**-8.692**
Delta	-	**0.165**	**9.721**	**0.095**	**31.349**	**0.154**	**6.339**	**0.190**	**7.501**

TABLE VI

RANK ORDER (WITH RESPECT TO EXPECTED BEST-OF-k SOLUTION QUALITY FROM 100 TRIALS, FOR FIVE INPUT PARAMETERS OF PERTURBATIONS, FOR k = 1,..., 10 RUNS. THIS TABLE IS DERIVED FROM *AES* RESULTS IMPLEMENTED USING *ASTRO*.

AES						*JPEG*					
k	T	S	B	A	U	k	T	S	B	A	U
1	1	3	2	4	5	1	4	2	5	3	1
2	2	3	1	4	5	2	4	2	5	3	1
3	1	3	2	4	5	3	4	2	5	3	1
4	2	3	1	4	5	4	4	2	5	3	1
5	2	3	1	4	5	5	4	2	5	3	1
6	1	3	2	4	5	6	5	2	4	3	1
7	1	3	2	4	5	7	5	2	4	3	1
8	2	3	1	4	5	8	5	2	4	3	1
9	1	3	2	4	5	9	5	2	4	3	1
10	1	3	2	4	5	10	5	3	4	2	1
LSU						*EXU*					
k	T	S	B	A	U	k	T	S	B	A	U
1	2	4	3	5	1	1	5	4	1	2	3
2	1	3	4	5	2	2	5	4	2	1	3
3	1	4	5	3	2	3	5	3	1	2	4
4	2	4	5	3	1	4	5	4	2	1	3
5	1	4	5	3	2	5	5	3	2	1	4
6	3	4	5	1	2	6	5	3	2	1	4
7	3	4	5	1	2	7	5	4	2	1	3
8	3	4	5	1	2	8	5	4	2	1	3
9	2	4	5	1	3	9	5	4	2	1	3
10	2	4	5	1	3	10	5	3	2	1	4

all simulation results that were summarized in Table V.

First, we find which input parameter is the most useful to perturb, with respect to $k = 1, 2, 3, ..., 10$. We randomly choose k solutions in each of 100 trials, out of our seven different runs for each perturbed input parameter: clock cycle time (T), clock uncertainty (S), input/output delay (B), aspect ratio (A), and utilization (U). We then find worst, best, and average of the best WNS from 100 trials for each k value.

Table VI shows the "quality" ranks of each input parameter for our testcases implemented using *ASTRO*. From the table, we observe that clock cycle time (T) or input/output delay (B) perturbations may be the best for the *AES* design, utilization (U) perturbation may be the best for the *JPEG* design, and input/output delay (B) or aspect ratio (A) perturbations may be the best for the *EXU* design. For *LSU*, when k is small, clock cycle time (T) or utilization (U) perturbations give the best design quality, but when k is larger than six, aspect ratio (A) perturbations give the best design quality.

Second, we find the best k value if both the input parameters and the parameter values are randomly chosen, since the best knob is not common for different testcases. For each testcase, we randomly choose k solutions in each of 100 trials, out of the 35 different solutions available (five different input parameters times seven different (perturbed) values of each parameter) for the testcase. Figures 4, 6, 8, and 10 (one figure per testcase) show the worst, best and average WNS values out of 100 trials of best-of-k sampling when *ASTRO* is used for implementation. Figures 5, 7, 9, and 11 show the worst, best and average WNS values out of 100 trials of best-of-k sampling when *SOCE* is used for implementation.

From Figures 4-11, we observe that the average WNS from 100 trials improves rapidly with increasing k. In most cases, when $k = 3$, the average expected quality is within 20ps of the best solution quality. We also observe that the worst-case WNS from 100 trials can be improved significantly with small k. For example, when $k = 3$, multi-run or multi-start using *SOCE* is expected to improves WNS of *EXU* by more than 100ps.[4]

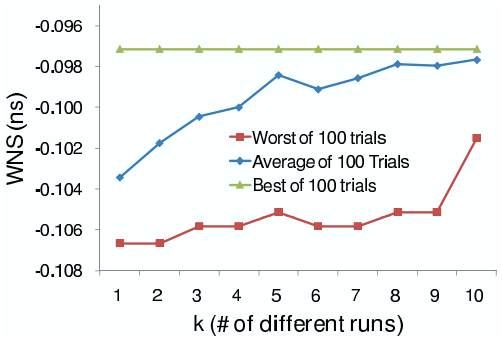

Fig. 4. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *AES* implemented using *ASTRO*.

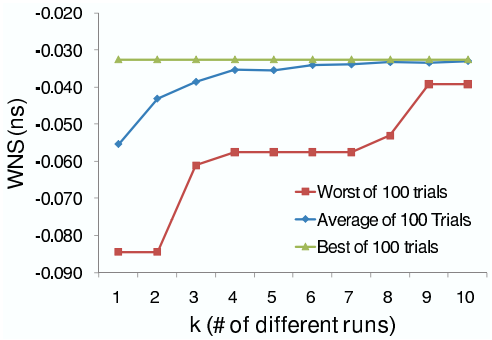

Fig. 5. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *AES* implemented using *SOCE*.

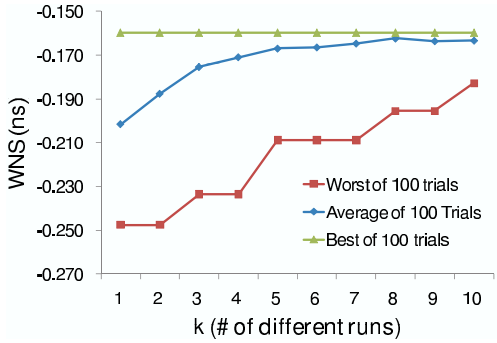

Fig. 6. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *JPEG* implemented using *ASTRO*.

IV. CONCLUSION

In this work, we have experimentally assessed the nature of 'chaotic' behavior in commercial IC implementation tools. 'Chaos' is attributable to miscorrelations of performance analyses between synthesis and P&R, and between P&R

[4]With the multi-run scenario, the runtime overhead of $k = 3$ can be naively viewed as three times larger than that of the traditional design methodology. However, if we consider the effort and time required to analyze and manually improve timing quality by more than 100ps, the overhead from the pure tool runtime can be substantially compensated. Furthermore, since better quality of the initial optimization typically results in faster timing closure, the overall design cycle time can be potentially reduced. With the multi-start scenario, it is clear that there will be negligible design time overhead, since our methodology can be trivially implemented and does not require any change to the existing design flow.

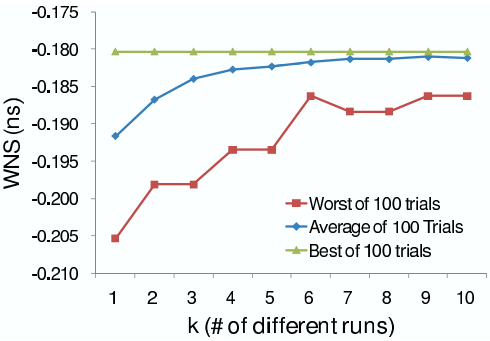

Fig. 7. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *JPEG* implemented using *SOCE*.

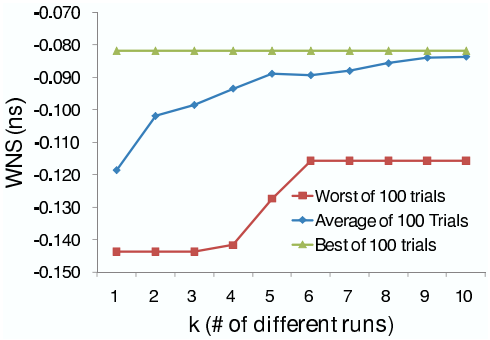

Fig. 8. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *LSU* implemented using *ASTRO*.

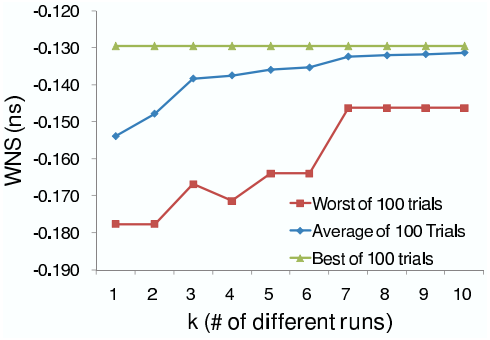

Fig. 9. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *LSU* implemented using *SOCE*.

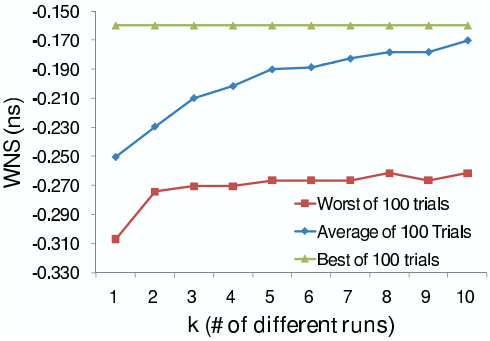

Fig. 10. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *EXU* implemented using *ASTRO*.

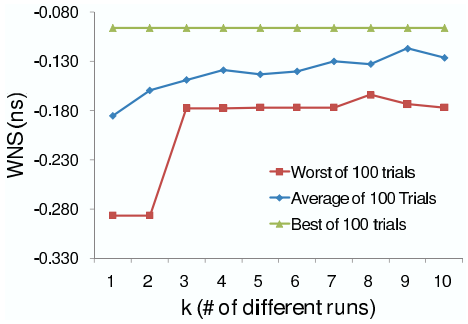

Fig. 11. Worst, best and average WNS values out of 100 trials with respect to expected best-of-k solution quality: *EXU* implemented using *SOCE*.

and signoff. We also characterize the effects of intentional, negligible perturbations of input parameters on output quality of commercial tools and flows.

Based on our experimental results, we propose a methodology to exploit the chaotic tool behavior using 'multi-run' (1 license or 1 CPU scenario) or 'multi-start' (multiple licenses and CPUs scenario) with intentional perturbations of the input parameters. We also describe an efficient method to determine the best number k of multiple runs that will yield predictably high-quality solutions without any additional manual analysis or manipulation, without changing any design flows, and without wasting valuable computing resources.

The deployment of new implementation and signoff tool capabilities opens up new directions for ongoing and future work, including the following. (1) We seek to analyze the potential advantages of the inherent "chaos" in advanced *physical synthesis* tools that exploit physical information at the synthesis stage to reduce synthesis-placement miscorrelations. (2) We seek to evaluate the benefits of chaos in conjunction with more advanced signoff methodologies (e.g., signal integrity-enabled STA), as well as more advanced signoff analyses (e.g., path-based analysis), which may exhibit even more chaotic behavior than today's standard flows.

REFERENCES

[1] *International Technology Roadmap for Semiconductors*, 2007. http://public.itrs.net/ .

[2] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. International Symposium on Quality in Electronic Design*, 2002, pp. 206-211.

[3] M. R. Hartoog, "Analysis of Placement Procedures for VLSI Standard Cell Layout", *Proc. Design Automation Conference*, 1998, pp. 646-652.

[4] H. J. Kushner, "Asymptotic Global Behavior for Stochastic Approximation and Diffusions with Slowly Decreasing Noise Effects: Global Minimization via Monte Carlo", *SIAM Journal on Applied Mathematics* 47(1) (1987), pp. 169-185.

[5] S. Yakowitz and E. Lugosi, "Random Search in the Presence of Noise, With Application to Machine Learning", *SIAM Journal of Scientific and Statistical Computing* 11(4) (1990), pp. 702-712.

[6] *OPENCORES.ORG*, http://www.opencores.org/ .

[7] *Sun OpenSPARC Project*, http://www.sun.com/processors/opensparc/ .

[8] *Cadence RTL Compiler version 7.1*. http://www.cadence.com/products/digital_ic/rtl_compiler/index.aspx .

[9] *Cadence SOC Encounter version 7.1*. http://www.cadence.com/products/di/soc_encounter/Pages/ .

[10] *Synopsys PrimeTime version B-2008.12*. http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx .

[11] *Synopsys Astro version Y-2006.06*. http://www.synopsys.com/products/products.html .

[12] *Synopsys Design Compiler version Y-2006.06*. http://www.synopsys.com/products/logic/design_compiler.html .

[13] *Synopsys Star-RCXT version Z-2007.06*. http://www.synopsys.com/products/products.html .