The Count of Monte Carlo

Lou Scheffer
Cadence

ABSTRACT

In the famous Dumas novel *The Count of Monte Cristo*[4], the (future) count is falsely accused of treason. Only after a long and difficult fight does he achieve respectability and reveal his adversaries as imposters. Similarly, Monte Carlo techniques are falsely accused of several crimes, including wasting computational resources, missing errors, and inadequate feedback. However, a more detailed analysis shows that the real problem involves many more variables than commonly thought, that proposed solutions do poorly in these cases, and that Monte Carlo is not as expensive as has been thought. Furthermore the drawbacks are either easy to overcome or shared by the other possible solutions. In short, the stage is set for Monte Carlo to return to respectability, if not triumph.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer Aided Engineering

General Terms

Algorithms, Performance, Design, Verification

Keywords

Static timing, Process variation, Statistical timing, Yield

1. INTRODUCTION

Monte Carlo simulation is widely viewed as accurate and theoretically sound, but is accused of several problems:

- Monte Carlo is too CPU intensive for practical use.
- Monte Carlo is run on paths, but there are too many paths for full coverage. Therefore there is chance that some relevent net will be omitted.
- Monte Carlo, even if correct, provides no useful feedback to improve the circuit.

TAU'04, February 2–3, 2004, Austin, Texas, USA.

There is an element of truth to each of these objections, and therefore considerable research has gone into alternative methods[7, 9, 10, 12, 14].

Why is Monte Carlo thought to be slower than the alternatives, perhaps so slow as to be impractical? Most of the other proposed approaches are based on a modified static timing analysis (STA), propagating some information more complex than simple arrival times (PDFs, CDFs, or sensitivities). Static timing is one of the most efficient CAD algorithms, and one of the few that are $O(N)$, since it visits each node and each cell exactly once. Monte Carlo techniques, by their very nature, visit each node hundreds, if not thousands, of times. Furthermore, many researchers have at least experimented with Monte Carlo (typically to get "golden" results for comparison with their trickier methods) and found that a straightforward implementations of Monte Carlo is indeed very slow.

However, the efficiency disadvantage of Monte Carlo may be overstated, since the real problems of chip analysis occur in a space of (very) many dimensions. This gives considerable, and perhaps fatal, problems to the proposed new techniques but only minor complications to Monte Carlo methods. Furthermore, Monte Carlo techniques can easily be improved to reduce their run times, and are very easy to parallelize. Therefore they may be in practice among the fastest of the practical techniques.

Another knock about Monte Carlo is that it cannot consider all paths, unlike techniques that operate on the timing graph in an STA like fashion. This can be solved by using more Monte Carlo rather than less, using Monte Carlo analysis to help pick the paths to be simulated.

A final complaint against Monte Carlo is that without an explicit propagation through the timing graph, earlier design stages will not have any handle on optimization. We note that this is exactly the same problem faced by the lack of knowledge of detailed routing in synthesis, and propose similar solutions. Furthermore, with parallel processing, a Monte Carlo analysis suitable for these applications (faster but lower quality) can indeed be done incrementally.

2. PROCESS VARIATION HAS MANY DIMENSIONS

Process variation occurs in a highly multi-dimensional space[15]. Even in a simple case, where only interconnect is considered and all intra-chip variation is ignored, if there are N routing layers, then there are at least $4N$ dimensions. This is because for each routing layer, there are four main variables - metal thickness, metal width, inter-

layer dielectric thickness, and via resistance - all of which may vary independently. (Note that the metal width and metal spacing do *not* vary independently - their sum, the pitch, is extremely well controlled, so they are precisely anti-correlated.) Cell delays add at least three more dimensions, historically P (process), V (voltage), and T (temperature). P, intended to represent the cell speed, is a composite of more fundamental variations such as threshold voltage and oxide thickness. V and T are operating conditions and not manufacturing variances, but share many of the same characteristics and can benefit from the same analyses.

So even in this simplified case where all intra-chip variation is ignored, and cell speed has an (over) simplified model, there are still tens of variables, and this number cannot easily be reduced - and handling this case is one of the main requirements of statistical analysis. For example, a recent customer[1] built a chip that failed whenever the metal-4 vias were unusually low resistance and the rest of the metals were nominal[1] Note that this kind of problem is almost always uncaught by corner analysis, since it's impractical to use enough corners. Even if just slow/fast for each layer is considered, 10 layers lead to $2^{10} = 1024$ runs to account for all interconnect corners. If all four main variables for each layer are considered (which would have been required in the case above, for example), then then 10 layers will take $2^{40} \approx 10^{12}$ corners. Even this would not catch all errors since there is no guarantee that the worst case happens at any of these corners[14].

Next, one of the main motivations for statistical timing is to handle on-chip variations. This adds many more dimensions, as the next section shows.

3. REALISTIC TREATMENT OF ON-CHIP VARIATION

In addition to 'global' variations, where metal 1 (for example) is thick or thin across the whole chip, it is also true that metal 1 thickness varies from place to place on a single chip. Some of this is deterministic, determined by nearby features and environment. This should be compensated for before statistical analysis, a point whose importance is difficult to overstate (though Nassif[13] is trying).

The simplest form of on-chip variation assumes a linear trend across the chip. This creates 2 more variables for each existing variation (these define the plane of cross chip variation). Now we have at the very least 120 variables for a 10 layer process. More realistic models of on-chip variation introduce even more dimensions. Any statistical technique must look at all the paths that are critical, or may become critical, under any likely set of process conditions. Often this is several thousand paths, as shown in [8]. Figure 1, from this presentation, show how the yield curve varies as more paths are included. An accurate result (at least on this one design) requires considering thousands of paths. Each of these paths contains several gates, and an equal number of nets. To correspond with the physical situation, each gate, and each net, should be assigned several random variables, since each gate or interconnect piece depends on several variables, and even gates (or nets) very close to each other are not completely correlated. Thus the true (physics based) treatment involves tens of thousands, if not hundreds of thousands, of variables.

[1]One branch of the clock tree had more vias than the other. Though they had comparable delays under nominal conditions, if the M4 vias were less resistive then usual, the one branch was faster then the other, and hold errors resulted.

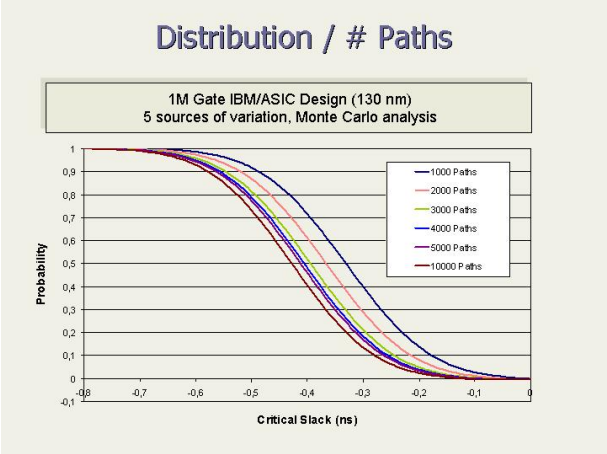

Figure 1: Yield curve vs. number of paths considered

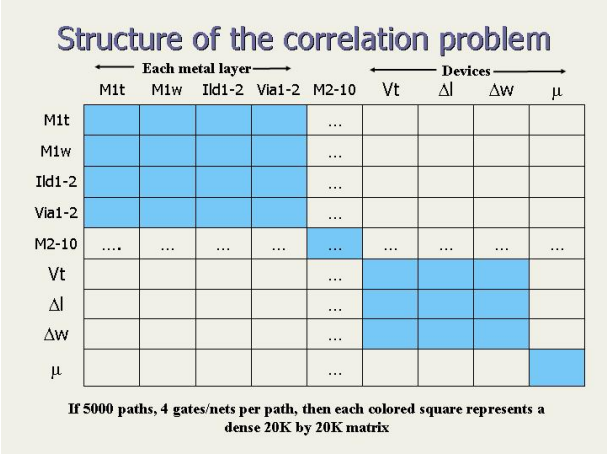

Figure 2: Structure of the correlation matrix

3.1 Correlation methods

Statistically, there is no conceptual problem to using scrillions of variables. We simply define a random variable (say Vt) for each transistor in each gate in all the possibly critical paths. We then define a correlation matrix that shows how all these Vts are correlated. Note that this matrix will be completely dense since all the Vts on a chip are correlated, and none of the correlations will be very close to 1 since even adjacent devices have significant random variations. This puts us in ugly mathematical territory where no correlations can be dropped since they are all significant, but no variables can be eliminated since none are correlated completely.

A thought experiment shows this correlation problem is real. If the variables were truly correlated, then only one path (the longest) would need to be considered. No matter what the variation it would always be the longest. On the other hand, if the paths were completely uncorrelated, then

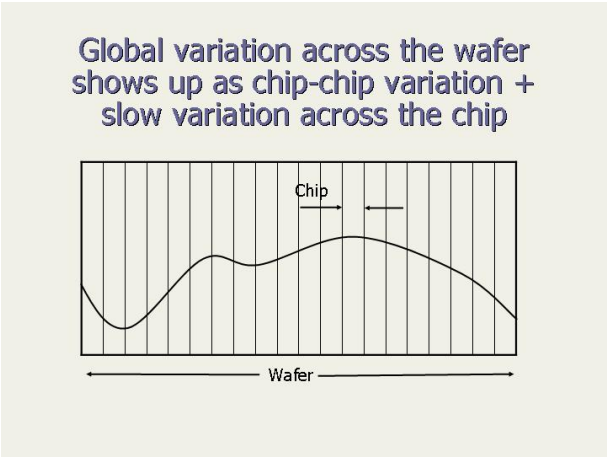

Figure 3: Global wafer variation

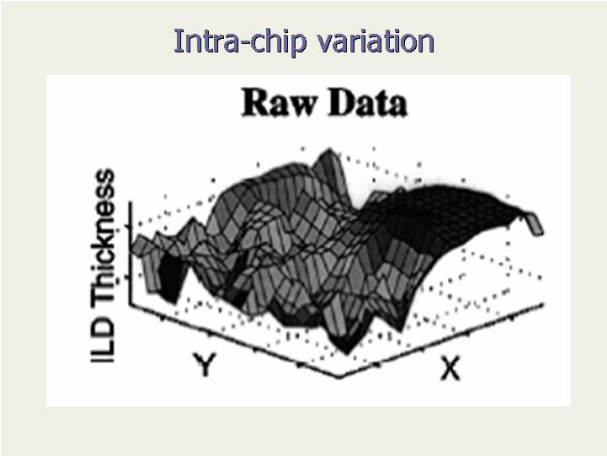

Figure 4: Intra-chip

only 40 or so paths would need to be considered. This is because, in this case, to make the Nth path the worst, if needs to be worse than path 1 (probability ¡ 1/2) AND worse than path 2 (prob ¡ 1/2) AND worse than path 3 and so on. By the time you get to the 40th path there is only a chance in a trillion 10^{12} of the path becoming critical. In real chips, though, the fact that including more paths changes the results, up through at least several thousand paths, shows the correlation problem is real and cannot be ignored.

Of course there is not just one process variable (Vt above) but in practice many variables for each interconnect layer and each cell parameter. For example, suppose we have 5000 paths, each with 4 gates and 4 nets (this is a bare minimum if we want to include the clocks from the point of divergence as well). Say we have 4 interconnect variables for each layer and 4 cell variables. This leads to the structure shown in figure 2. Overall, the structure is a banded 44 by 44 matrix. Each entry in this matrix is all zeros, or a 20K by 20K dense matrix.

3.2 Additive methods

Perhaps instead of utilizing the correlation structure of the problem, we can attempt to construct the distribution in such a way that the desired correlations result. We might express the Vt of a given transistor as a sum of a global Vt for the chip, a correlated component, and a purely random component. For the correlated component, we will need to pick basis functions and sum them up. These will have to be present at a wide range of scales, since some affects are gradients across the entire wafer(such as illustrated in figure 3), where others are much more localized as shown in figure 4 taken from[15].

How many basis functions will need to be added to recreate a realistic on-chip variation? Clearly reproducing a cross-chip variation such as shown in figure 4 will require a considerable number. Agarwal *et al.*[2] tried this approach and used a 6 level hierarchy with each level decomposing a square into 4 smaller squares. Therefore their basis functions were 1 full chip square, 4 smaller squares, 16 still smaller squares, and so on down to 1024 leaf squares as the smallest representable element, for a total of 1365 variables. Another way of estimating the variable count is to note that an industrial process manual[6] states different numbers for correlation if the transistors are no more than 200 microns apart. A chip 20 mm on a side contains 10,000 regions that are 200 by 200 microns, and hence would require about that many more variables.

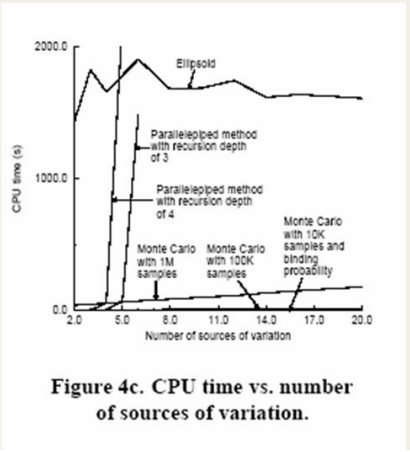

Figure 4c. CPU time vs. number of sources of variation.

Figure 5: Performance of different methods as dimensions increase

4. PERFORMANCE OF PDF METHODS

Many of the proposed methods for dealing with statistical variation[7, 9, 10, 12, 14] do not deal gracefully with large number of parameters, or require the parameters to be uncorrelated, or of specific distributions, or both. Figure 5, for example, from the paper[8], show how the run times of different methods depends on the number of variables. Clearly the parallelepided method, for example, does not cope well with large numbers of variables. Note, in particular, that the Monte Carlo methods are *not* the slowest of those tried - for any realistic number of parameters, they are the fastest! One of the few methods in the literature that does cope with spatial variation, that of Agarwal *et al.*[2], compares their proposed algorithm to Monte Carlo in terms of accuracy, but not in terms of run time. This data would be very interesting.

Another way of dealing with correlated variables is to transform them to uncorrelated variables by a change in basis[3]. This is the idea bahind principal component analysis. However such methods are based on a SVD analysis, which takes $O(N^3)$ time for N components. This is impractical with tens of thousands of variables.

5. PERFORMANCE OF MONTE CARLO

While Monte Carlo analysis is regarded as theoretically sound, there are many worries about its performance. These worries come largely from the large number of trials required. As a stochastic technique, the error scales as $1/\sqrt{(M)}$, where M is the number of trials. Hence to get 1% accuracy about 10,000 trials will be required, and this number will be assumed in the estimates below.

The simplest way to do Monte Carlo analysis would be to run the whole extract (including characterization), delay calculation, and timing flow many times. This would take about 10,000 times as long as the current signoff calculations, which would indeed be impractically slow. However, there are many ways to improve this time.

Extraction can include explicit variation of the extracted values with process variations[14]. Then extraction needs only be done once. It will be slower, and take more storage to hold the results, but by a small factor rather than a factor of 10,000. Likewise, delay calculation can incorporate this information, and in turn produce delays that are an explicit function of process variables[11]. Once again this only needs to be done once, and is only somewhat slower than conventional delay calculation.

Next, not all paths need to be simulated with Monte Carlo. Only those that are critical, or have a non-negligable change of becoming critical, need to be considered. Although this number is design dependent, it appears from looking at Figure 1 (and other similar designs) that the number of paths that need to be considered is about 1% of the total number of cells. These paths can be selected with a conventional static timing tool, or such a tool with relatively minor extensions (this is covered in more detail in the next section). Although selection of the paths that are, or might be, critical, is an interesting problem in its own right, this must be solved for many other techniques as well. It works just as well with Monte Carlo as other techniques. If 1% of the paths need be considered, but each evaluated 10,000 times, then we might expect the Monte Carlo process to take 100 times as long as conventional static timing. This is much better but still not good.

However, 10,000 evaluations of the same path can be much quicker than the evaluation of 10,000 different paths. For example, relatively few runs of a path can be used to generate a response surface model, and then further analysis can use this model, which is much faster. For example, if a linear response surface is adequate (and there is some evidence that this is the case for most parameters[14]), then if a path depends on 100 parameters, then 100 runs is enough to generate the response surface. Then the 10,000 Monte Carlo runs reduce to 10,000 vector dot products, taking a negligable time in comparison. 100 runs each on 1% of the paths will merely double the required CPU time, which is relatively acceptable. Higher order response surfaces,if needed, are still more efficient than independent evaluations.

Next, even the 100 or so runs required to establish the response surface are very similar. Incremental techniques can be used to speed up these evaluations with no loss of accuracy, since only a few parameters vary.

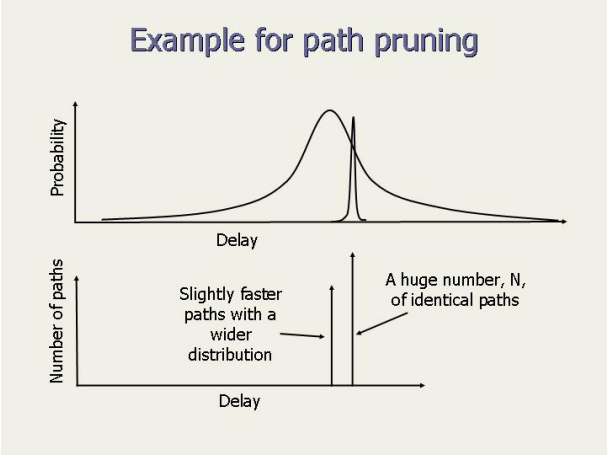

Figure 6: Example where worst nominal paths are not worst overall

Finally, Monte Carlo methods parallelize well, even upon the cheapest possible multi-processing setups (perhaps large clusters of Linux workstations, connected only by a network.)

At a very large number of dimensions, however, some surprises can emerge. Take, for example, the problem of generating the cases to be simulated. In the correlation model, from section 3.1, this is surprisingly tricky. A quick look at the literature shows that the way to generate a sample from a correlated data set of size N is to generate N independent random variables, then 'mix' them with a N by N matrix to generate the N correlated random variables. The NxN matrix is the Cholesky decomposition (basically the square root) of the correlation matrix. This decomposition always exists, since a correlation matrix must be positive definite, so there is no problem in theory. In practice, though, computing this decomposition for a dense matrix requires $O(N^3)$ time and $O(N^2)$ space. For $N = 100$ or so this is reasonable, but it $N > 100,000$ or so it becomes prohibitive. One possible alternative would be to just include the correlations to the nearest few hundred items (which makes the matrix sparse). Alternatively, the additive method of generating correlations[2] does not suffer from this problem, and can parallelize well.

6. MONTE CARLO REQUIRES PATH PRUNING

Another potential problem with Monte Carlo is that it works (at least when implemented in the obvious way) on paths, and there are too many paths to enumerate. Therefore normally the few apparently worst paths are selected and analyzed. But what if some other path (with a smaller nominal delay) could become worse under some condition? How can we guard against this?

First, let's see how this could happen. Figure 6 shows a thought experiment where the paths that determine yield never show up in any of the worst N paths under nominal conditions. Imagine a huge number N of paths with identical nominal delay, with a very small spread. Next, imagine M paths with delay very slightly less, but a wide spread,

with these M paths being completely independent. Each of these M paths could have up to a 1/2 chance of being more critical than any of the N nominally worse paths. By increasing M, we can ensure, to any desired probability, that the true limiting path is not one of the N nominally worst.

To avoid this problem, one approach might be to do the static analysis at several corners, then take the union of all paths found. A typical set of corners might be min and max interconnect delay (R times C), min and max cell delay (total C), and nominal. This would certainly help in practice, but is not guaranteed since the spread in figure 6 might be due to differences between layers, which does not appear in any of the corners above.

A better solution is to use Monte Carlo to pick the paths. This means picking some sets of parameters, finding the worst paths in each case, and taking the union of these paths. It's easy to see why this works - if you have any effect, no matter what the nominal delay, that limits the speed in (say) 10% of the cases, then it should also show up in 10% of the Monte Carlo runs. Using the approximation that $(1 - a/N)^N \approx e^{-a}$, we can see (for example) that if we use 100 Monte Carlo runs to select the paths, we have a probability of e^{-1} of missing a path that shows up in 1% of the cases, an e^{-2} probability of missing a path that shows up 2% of the time, and so on. There is basically no chance (e^{-10}) of missing a path that shows up 10% of the time.

7. FEEDBACK TO EARLIER STEPS

A very legitimate question with Monte Carlo analysis is how to generate feedback to earlier tools from the result. Part of the reason is that a pure Monte Carlo might not tell you *why* a path fails. However, with only a little more work, the delays computed under the various Monte Carlo cases can be used to compute the sensitivities and hence tell under which conditions the path fails. This is not difficult if designed into the Monte Carlo analysis.

A more serious objection is that you can make an incremental version of the timing graph methods, but an incremental version of Monte Carlo is difficult. Therefore although Monte Carlo might work for signoff, the effect of incremental changes will be hard to evaluate. Although this is a real concern, there are two answers to this objection. First, it may be not be a big disadvantage in practice, and second it may be possible to do Monte Carlo (at least a limited version) through the use of parallel processing, and do so almost as quickly (and incrementally) as conventional STA.

Accurate incremental analysis may not be very useful in practice. When a path is modified in synthesis or placement, in general the new routing is not known precisely. Therefore any net properties are only rough estimates. It's true that an STA like method could propagate these accurately, but it's not clear yet what is gained by an accurate propagation of crude estimates. It may well be that a crude-but-fast propagator is a better choice, and this might work equally well with an STA method or a Monte Carlo method. This area is ripe for lots of research into the best way to fix problems found by a statistical tool.

Second, it may well be possible to make an incremental Monte Carlo analysis. This might involve doing N conventional incremental timing analysis runs in parallel, each with slightly different data. Then the launching process can collect the changed arrival times (or slacks, or any other product of timing analysis) and estimate roughly the distributions (and correlations if need be) of any of the results. This will not be very high quality, since the number of Monte Carlo samples is limited to the number of parallel processors available, but it should be more than enough quality to evaluate the result of design changes.

Paths with high likelihoods of becoming too short pose no problem, since these MUST be fixed, and the fix is in general straightforward. They must be fixed since this error does not go away if the chip is run at reduced speed. Fixing these is relatively straightforward, since the fixes can be made with relatively large margins so extremely detailed analysis is not required. The fixes will be the same whether the problem is found by Monte Carlo or conventiional STA. Suppose, for example, that a hold error results whenever metal 3 is thin but metal 4 is thick. The solutions will resemble those for a hold time violation found for any other reason - balance the clock paths better, delay the clock to the launching flip-flop, or insert more delay in the $Q- > D$ path.

Predictions of long paths are harder to feed back since they are probabalistic. However, these can be handled the same way as routing effects. Synthesis does not understand detailed routing, either, but feedback from post detailed routing has been fed back for years. Typically, existing paths get the back annotated delays (and now probabilities will be added), and these are assumed to be correct as long as the path is not modified. As the logic structure is changed, new paths are created, which are assigned delays based upon estimates. These will also need to be assigned probabilities. Note that even if the probabalistic timing is computed directly upon the synthesis timing graph, this problem still exists (due to routing uncertainty) unless the synthesis tool also does signoff quality routing and extraction as it runs, which seems very unlikely. An open and very interesting question is how the uncertainty induced by statistical timing compares with other known sources, such as the uncertainty induced by the lack of detailed routing during synthesis.

In either event, whether computed upon the timing graph directly, or fed back from post processing analysis, synthesis must no longer have the view that only the most critical path counts. Instead, each path will have some chance of being critical, and there will be some benefit to improving paths that are slightly faster than the worst. Perhaps the biggest effect will be on optimizations such as leakage reduction, where slower cells are substituted until the delay becomes just less than that of the critical path. Instead, this process should probably stop when the chance that the path become critical rises above a certain percentage.

8. OTHER ADVANTAGES OF MONTE CARLO

Monte Carlo also has a number of practical advantages. It is insensitive to the distributions of the variations. It can be run at either the gate level or at SPICE level (which avoids the need for cell characterization). The effects of crosstalk are easy to add in (since a crosstalk analysis tool can generate paths with the aggressors included). IR drop can be included if the cells are characterized for it, or if the analysis is done at the SPICE level.

9. CONCLUSIONS

Monte Carlo analysis has many advantages. It is clearly technically correct, and can easily cope with complications

such as correlated variables, high dimensionality, and non-standard distributions. However, there are serious potential drawbacks: it is potentially slow, path selection could be incorrect, and feedback to other tools may be difficult because it does not work directly on the timing graph. However, in realistic cases with high numbers of dimensions, the competing methods are also slow, perhaps even more so than Monte Carlo, while Monte Carlo offers many possibilities for optimization and parallel processing. Path selection does indeed require care, but by using Monte Carlo this can be done to reasonable standards. Finally, feedback to tools earlier in the design chain looks straightfoward, similar to the feedback from other signoff tools that is performed today. Therefore, in the words of Faulkner[5], Monte Carlo will not merely endure, it will prevail.

10. REFERENCES

[1] You actually thought I'd reveal the name of the customer??

[2] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Statistical delay computation considering spatial correlations. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 271–276, 2003.

[3] H. Chang and S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of the 2003 ICCAD*, 2003.

[4] A. Dumas. *The Count of Monte Cristo*. An English translation is available from Penguin, USA, among others, 1846.

[5] W. Falkner. 1949 nobel prize acceptance speech. http://www.nobel.se/literature/laureates/1949/faulkner-speech.html.

[6] I. foundry. I've asked permission to refer to this manual, but not heard back yet.

[7] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. In *International Symposium on Quality Electronic Design*, pages 437 – 442, 2001.

[8] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. H. J. M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *Proc. 40th Design Automation Conference*, pages 932–937, 2003.

[9] H.-F. Jyu, S. Malik, S. Devadas, and K. Keutzer. Statistical timing analysis of combinational logic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1(2):126 – 137, June 1993.

[10] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *proceedings of the 2001 Design Automation Conference*, pages 661–666, 2001.

[11] Y. Liu, L. T. Pileggi, and A. J. Strojwas. Model order-reduction of rc(l) interconnect including variational analysis. In *Proc. 36th Design Automation Conference*, pages 201–206, 1999.

[12] V. Mehrotra. *Modeling the Effects of Systematic Process Variation on Circuit Performance*. PhD thesis, MIT, 2001.

[13] S. Nassif. Within-chip variability analysis. In *International Electron Devices Meeting Technical Digest*, pages 283–286, 1998.

[14] L. Scheffer. Explicit computation of performance as a function of process variation. In *Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems*, pages 1–8. ACM Press, 2002.

[15] B. Stine, D. Boning, and J. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):24–41, Feb 1997.