Handling Intra-Die Variations in PSTA

Luis Guerra e Silva
INESC-ID
IST / TU Lisbon
Lisbon, Portugal
lgs@inesc-id.pt

L. Miguel Silveira
Cadence Res. Labs/INESC-ID
IST / TU Lisbon
Lisbon, Portugal
lms@inesc-id.pt

ABSTRACT

For integrated circuit (IC) fabrication technologies of $45nm$ and below, the impact of process variability in circuit performance is extremely relevant. Parametric static timing analysis (PSTA) techniques, whereby delays are modeled as affine functions of process parameters, were thus introduced to enable the computation of accurate timing estimates, accounting for process variability. Most often, only variations that occur between fabricated ICs (inter-die) are modeled, as the number of variables necessary to model such effects is manageable. Variations that occur across the same IC (intra-die) are usually neglected, as modeling them can add significant complexity to the model. This paper evaluates the impact of modeling intra-die variations in the context of PSTA and proposes effective techniques for handling them.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated Circuits—*Design Aids*

General Terms

Algorithms, Design, Theory, Verification

Keywords

Intra-die variations, Parametric static timing analysis

1. INTRODUCTION

The need to adequately model the process variations that occur in modern IC technologies, has motivated the introduction of parametric delay models, where cell and interconnect delays are given by functions of the parameters of the fabrication process [7], rather than fixed nominal values. Several compatible PSTA techniques have likewise been proposed. The most well known of such techniques is statistical static timing analysis (SSTA), whereby process parameters are assumed to follow a given statistical distribution [7].

The perception that it might be abusive to assume the process parameters to follow a specific statistical distribution, has motivated the development of new PSTA methodologies, where process parameter variations are only specified by ranges. Such methodologies have the advantage of being a natural extension to traditional corner-based approaches. In this context, [6] proposed a linear-time approach for computing an affine function of the process parameters, which is an upper bound on the worst delay of a circuit, covering all process corners. While this method is quite efficient, in certain cases the computed upper bounds may not be tight and it does not incorporate a practical way of identifying the critical paths and corners. Addressing these issues, [5] proposed an efficient algorithm for computing the exact critical path of a given circuit, as well as their associated process parameter settings (corners). This approach has a worst-case exponential run-time, yet it performs fairly well in practice.

Process variations are classified as either inter-die or intra-die. *Inter-die* variations occur between dies. For example, metal width may vary between ICs produced in the same wafer, in different wafers, or in different runs. Common sources of inter-die variations are fluctuations in global process parameters, such as temperature. *Intra-die* variations occur within the same die. For example, two transistors within the same die may exhibit distinct gate lengths. Intra-die variations are mostly due to non-idealities in the fabrication equipment, such as CMP and optical proximity effects. While both types of variation have a systematic and a random component, only the former can be modeled, given enough knowledge of the fabrication process.

This paper evaluates the complexity of performing PSTA, while using delay models that account for both inter-die and intra-die variations [1]. Additionally, it proposes novel techniques to enable efficient PSTA of such models.

This paper is organized as follows. Section 2 details the parametric delay model to be used throughout the paper. Section 3 briefly reviews three PSTA algorithms that will subsequently be evaluated for the analysis of intra-die delay models. Section 4 discusses techniques for improving the performance of PSTA algorithms, when dealing with intra-die delay models. Finally, Sections 5 and 6 present experimental results and concluding remarks, respectively.

2. PARAMETRIC DELAY MODEL

This section details a parametric delay model, that accounts for both inter-die and intra-die variations, which is closely related to the ones proposed in [1] and [3].

2.1 Delays as Affine Functions of Parameters

A parametric delay model [7] is assumed, where delays are described by affine functions of process and operational parameter variations, corresponding to a first-order linearization of every delay, d, around a nominal point, λ_0, in a parameter space of size p. Assuming $\Delta\lambda = \lambda - \lambda_0$, we obtain

$$d(\Delta\lambda) = d_0 + \sum_{i=1}^{p} d_i \Delta\lambda_i \qquad (1)$$

where d_0 is the nominal value of d and d_i is the sensitivity to parameter λ_i, $i = 1, 2, \ldots, p$, at the nominal point λ_0.

GLSVLSI'11, May 2–4, 2011, Lausanne, Switzerland.

When delays are in the form of Eqn. (1), arrival times are known to be *convex* [2], implying that the smallest/largest delay or arrival time is obtained by setting each parameter to one of its extreme values. Assuming $\Delta\lambda_i \in \left[\Delta\lambda_i^{min}, \Delta\lambda_i^{max}\right]$, the maximum delay value is given by

$$\max_{\Delta\lambda}[d(\Delta\lambda)] = d(\Delta\lambda^*) = d_0 + \sum_{i=1}^{p} d_i \Delta\lambda_i^* \quad (2)$$

where the maximizing parameter variation assignment is

$$\Delta\lambda_i^* = \begin{cases} \Delta\lambda_i^{min} & \text{if } d_i \leq 0 \\ \Delta\lambda_i^{max} & \text{if } d_i > 0 \end{cases}, \; i = 1, 2, ..., p \quad (3)$$

The min can be computed by symmetry.

2.2 Intra-die Delay Model

The number of parameters of inter-die delay models is usually small, as each parameter is used to model an effect that occurs uniformly across the entire die. However, intra-die variations exhibit a strong spatial correlation, and accurately modeling them would require complex higher-order models, hard to incorporate into existing timing verification flows. An approach [1, 3] to deal with this problem is to discretize the die area into several regions, and assign a parameter to each region. The delays of the circuit elements located within a given region are assumed to vary linearly with the parameters assigned to that region. By appropriately setting parameter ranges and sensitivities, it is therefore possible to approximate a complex global non-linear model by combining several local linear models, at the expense of a much larger number of parameters. The objective of this work is to evaluate the capability of existing PSTA approaches to deal with such large models.

This work considers three types of parameter variations: inter-die, global intra-die and local intra-die. Inter-die variations are modeled by a single parameter, for the entire die area. Modeling global and local intra-die variations requires the discretization of the die area into several regions [1]. For the sake of simplicity, we will assume that the grids for global and local intra-die variations split the die area in $r_g \times r_g$ and $r_l \times r_l$ regions of equal size, respectively, and that r_g is a multiple of r_l and $r_g << r_l$.

Since we do not have the data necessary to build a real intra-die model, we will extrapolate from an inter-die model, for which data is readily available. This is not a problem in our case because, while intra-die delay data is not real, the complexity (size) of the model should be close to that of real instances. Therefore, we will split each original inter-die parameter, $\Delta\lambda_i$, into an inter-die component, $\Delta\hat{\lambda}_i$, a global intra-die component $\Delta\hat{\lambda}'_{i,g(x,y)}$ and a local intra-die component, $\Delta\hat{\lambda}''_{i,l(x,y)}$. We will consider that the new parameters will assume the same range of variation as the original parameter. The sensitivities of the new parameters will be computed by weighting the original sensitivity with constant coefficients α, β, and γ, such that $\alpha+\beta+\gamma = 1$. The coefficients will determine the relevance that will be given to each type of variation. The delay of a circuit element located at position (x, y) will thus be given by,

$$\hat{d}(x,y) = d_0 + \sum_{i=1}^{p} d_i \left[\alpha\Delta\hat{\lambda}_i + \beta\Delta\hat{\lambda}'_{i,g(x,y)} + \gamma\Delta\hat{\lambda}''_{i,l(x,y)}\right] \quad (4)$$

The parameters to be considered for each element depend on its position, and therefore $1 \leq g \leq r_g$ and $1 \leq l \leq r_l$ indicate the region of global and local variations where the element is located. Hence, we will have a different parameter for each original parameter and for each region. Having p original parameters and $r_g \times r_g$ plus $r_l \times r_l$ regions, we will end up with a total of $p \times (1 + r_g \times r_g + r_l \times r_l)$ parameters in the extrapolated model.

3. PARAMETRIC STA

In this section we review three algorithmic approaches for performing PSTA, which will be evaluated in Section 5.

3.1 Statistical Static Timing Analysis

The PSTA method that produced the most impact was proposed in [7], and performs what is known as SSTA. In this method, all timing quantities are expressed as a canonical formula, similar to Eqn. (1), but parameters are considered to assume gaussian distributions and possibly be correlated. Through a linear-time traversal of the circuit graph, all timing quantities (arrival times, slacks, etc) are computed, by performing sum/max operations between canonical formulas. The max operation is computed by matching the mean and variance of the output formula with the estimated mean and variance of the max of the input formulas [4]. Since the max of two gaussian distributions is not a gaussian distribution, there is an error associated with this operation. While this method is efficient and, in general, produces fairly accurate estimates, it does not report critical paths and corners, which is essential in circuit optimization.

3.2 Hyperplane Bounding

Another approach, proposed in [6], computes an upper bound on the worst delay of a circuit, in the form of Eqn. (1), covering all process corners. As in [7], the affine arrival time functions, designated by hyperplanes, are pushed through the timing graph, in a linear-time traversal, and when necessary a conservative approximation to the max between two input hyperplanes is computed. Such approximation, designated by output hyperplane, is an upper bound on the max of both input hyperplanes, valid for every corner. The computed output hyperplane is guaranteed to be a tight bound at the worst corner among the two input hyperplanes. Given the delay hyperplanes for the primary outputs, it is trivial to compute their worst value, using Eqns. (2) and (3), and therefore obtain an upper bound on the delay of the circuit. Since the computed max hyperplane is only guaranteed to be tight at the worst corner, among the two input hyperplanes, if such corner varies frequently across the circuit, large bounding errors can be accumulated and the estimates at the primary outputs may not be tight. Additionally, it is not possible to reliably infer the worst-delay corner/path from the computed upper bound hyperplane.

3.3 Automated WDC Computation

Consider a combinational block with n inputs and m outputs. When delays are given by Eqn. (1), any input/ouput delay $d_{i,j}(\Delta\lambda)$ can be accurately represented by a piecewise-affine function [5]. The *worst-delay corner* (WDC) problem, is concerned with computing an assignment, $\Delta\lambda^*$, to the parameter variation vector $\Delta\lambda$, that produces the worst delay, $d_{i,j}(\Delta\lambda)$, from any input $i = 1, \ldots, n$ to any output $j = 1, \ldots, m$. Assuming the worst delay to be the largest one, then the WDC problem can be formulated as

$$\max_{\Delta\lambda}\left\{\max_{j=1,\ldots,m}\left[\max_{i=1,\ldots,n} d_{i,j}(\Delta\lambda)\right]\right\} \quad (5)$$

The simplest algorithm for computing the WDC consists of calculating the affine delay function for each path and subsequently, using Eqns. (2) and (3), compute the corner that produces the worst delay, over all paths. Since this procedure requires all paths to be analyzed, and their number can grow exponentially with the number of vertices, it can become overly expensive, even for moderately sized circuits.

In order to avoid enumerating all the paths, [5] proposes the use of branch-and-bound techniques to prune the timing graph, thus significantly reducing the number of paths that require detailed analysis. Considering a primary output at a time, the method proposed in [5] performs an *explicit* or *implicit* analysis of all the input/output paths, ending at

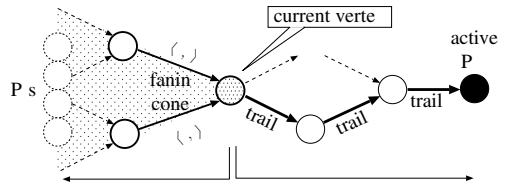

Figure 1: Illustration of delay estimates.

that primary output, designated as the *active* primary output. The timing graph is traversed in a backward fashion, starting at the active primary output, going through the internal vertices, and eventually ending at the primary inputs (if no pruning is performed). The vertex being visited in a given step is designated by *current vertex*. The path taken to reach that vertex from the active primary output is designated by *trail*. The worst delay, d^*, found among the complete paths already analyzed is continuously updated, as well as the corresponding corner, $\Delta\lambda^*$. For each current vertex of the timing graph, v, the algorithm relies on three parametric delay estimates, illustrated in Figure 1: d_v^{in} is an upper bound on the delay from any primary input to vertex v (e.g. in the fanin cone of v), and can be calculated beforehand through a forward levelized analysis of the timing graph; d_v^{out} is the delay of the trail; $d_v^{path} = d_v^{in} + d_v^{out}$, which represents an upper bound on the delay of any complete path going through v, that contains the trail. The fanin cone of v is pruned when the following condition holds

$$\max_{\Delta\lambda}[d_v^{path}(\Delta\lambda)] \leq d^* \tag{6}$$

In this case the delay of *any* complete path going through v and containing the trail can never be larger than the worst delay, already computed for some other complete path, d^*, therefore it is useless to further explore the fanin cone of v, as the worst delay, d^*, cannot be improved by such action. When the current vertex is a primary input, the trail is an input/output path, and $d_v^{path} = d_v^{out}$. In this case, if $\max_{\Delta\lambda}[d_v^{path}(\Delta\lambda)] > d^*$, then both d^* and $\Delta\lambda^*$ are updated. The procedure terminates when every path of the circuit is either explicitly visited or pruned. Upon termination, d^* is the worst delay and $\Delta\lambda^*$ is the worst-delay corner. The worst-delay path can also be returned if the corresponding trail is stored on every update of d^* and $\Delta\lambda^*$.

4. INTRA-DIE VARIATIONS AND PSTA

This section discusses techniques for efficiently performing PSTA on instances of the parametric delay model described in Section 2, which are adequate for modeling intra-die variations. Such models can contain a large number of parameters, and therefore may not be handled by most straightforward implementations of the PSTA algorithms available.

4.1 Model Structure

The delay model discussed in Section 2.2, can lead to a large number of parameters. However, given their spatial distribution across the die area, only a few of them will impact the delay of any given circuit element. Therefore, a no-nonsense improvement is the utilization of sparse data structures, since in any given affine delay formula most of the sensitivities will be 0. In general, this will only slightly improve performance, but it will make the problem manageable in terms of memory footprint.

The delay formulas of closely located elements are likely to share several parameters, as each parameter is allocated to a given die region. Moreover, it is likely that circuit elements that are close in the die may also be close in the netlist (i.e. within a few logic levels), since placement tools tend to place strongly connected elements close to each other, as a by-product of optimizing wire length and interconnect delay.

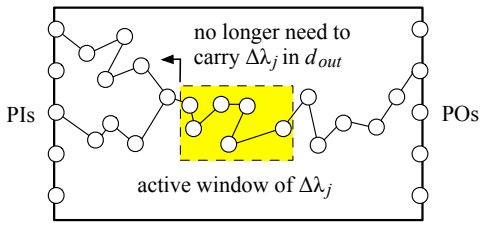

Figure 2: Illustration of local bounding.

Experimental evidence confirms this speculation since, for several benchmark circuits generated by commercial CAD tools, we have observed that the elements in the same region of the layout were isolated in a few clusters of the netlist. This means that the parameters that model specific regions of the die area will also be confined to a specific range of logic levels in the netlist.

4.2 Local Bounding

As observed, parameters allocated to particular regions of the die will, most likely, only exist within a limited window of logic levels in the circuit netlist, which we will designate by *active window* of the parameter. The computation of such window, for each parameter, can be performed by a forward/backward levelized traversal of the circuit graph. In each level, all the affine delay formulas associated with the elements in that level are analyzed and if, in any of such formulas a new parameter is detected, the active window of that parameter is marked as starting/stopping in that level.

For propagating canonical delay formulas [7] or hyperplanes [6], knowing the active window of each parameter is not useful, as we are interested in obtaining an explicit approximated formula for the delay at each primary output. This means that all the parameters must be carried across the circuit, until a primary output is reached.

When our objective is to compute the worst-delay path or corner [5], then knowing the active window of each parameter can be quite useful. Recalling the algorithm reviewed in Section 3, we can easily conclude that carrying back a parameter in d^{out} is only necessary if it is possible that any given d^{in}, that we will find further back, will also contain that same parameter. The information on whether that will happen is given by the the active window of the parameter, defined earlier. When we are outside of the active window of a parameter we can simply eliminate it from the formula, because we know it will no longer be relevant (see Figure 2). That elimination must be done by bounding, i.e. by adding to the nominal value the sensitivity to the parameter multiplied by the value obtained from Eqn. (3). Therefore, the elimination of a parameter $\Delta\lambda_j$ from a given delay $d^{out}(\Delta\lambda)$ produces an upper bound on $d^{out}(\Delta\lambda)$. Formally,

$$d^{out}(\Delta\lambda) \leq (d_0 + d_j\Delta\lambda_j^*) + \sum_{i=1,i\neq j}^{p} d_i\Delta\lambda_i \tag{7}$$

This procedure can enable great savings, since it can significantly reduce the size of the formulas to be carried across the circuit graph, particularly when the number of parameters is large and the size of the active windows is small.

4.3 Improving Bounds in WDC Computation

As discussed in Section 3, unlike the algorithm proposed in [5], the algorithm proposed in [6], does not provide a method for computing the worst-delay path and corner. However, it does provide a guaranteed upper bound on the delay, which is tight in most cases. Therefore, we propose to use such algorithm to compute the delay upper bound estimates, d^{in}, in the algorithm proposed in [5]. By having tight upper bounds to start with in [5], we would expect more pruning to be achieved and, consequently, running time to decrease.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setting

The benchmark circuits were synthesized and mapped to the Nangate OCL $45nm$ technology. As inter-die process parameters, we have considered the widths and thicknesses of the 10 metal routing layers, resulting in a total of 20 parameters. Variational delay computation was subsequently performed, and the resulting affine delay formulas were annotated in the corresponding timing graphs. Table 1 presents a brief characterization of the resulting benchmark circuits, where "#PI" and "#PO" columns report the number of primary inputs and outputs, "#Logic" and "#Net" columns report the number of combinational cells and the number of nets, and "#Vertex" and "#Edge" report the number of vertices and edges in the corresponding timing graph.

Intra-die effects have been modeled as described in Section 2.2, assuming $\alpha = 0.5$, $\beta = 0.25$, $\gamma = 0.25$, $r_g = 32$ and $r_l = 512$, resulting in a total of 5263380 parameters.

All the results presented in this section were obtained in a machine with an Intel Core 2 Duo @ 3GHz and 4GB of RAM. For all the runs a single processor was used. All the algorithms were implemented in C++ using sparse data structures to represent affine delays.

5.2 WDC Computation

Table 2 presents results for WDC [5], reviewed in Section 3, and improved versions of this algorithm. Columns "#P" and "CPU" report the the average size (i.e. number of non-zero elements) of the d^{out} formula and the CPU time in seconds, respectively. Column "[5]" reports the results for the original algorithm. Column "[5]+[6]" reports the results when the algorithm proposed in [6] is used to perform the initial upper bound computation, as discussed in Section 4. Column "[5]+LB" reports the results when the local bounding technique, proposed in Section 4, is employed. Finally, column "[5]+[6]+LB" reports the results combining all previous techniques. All the algorithms produced the exact same results, since none of the proposed techniques introduces any error. Clearly, computing the initial upper bound using the algorithm proposed in [6] yields enormous savings in WDC[5]. The local bounding technique also seems to be quite effective in reducing the number of parameters that need to be carried across the circuit, as observed by the average size of the d^{out} formula. However, due to its overhead it is only noticeable for larger circuits like C6288. We foresee that local bounding may be extremely useful for handling very large instances.

5.3 Comparison of PSTA Algorithms

Table 3 compares the PSTA algorithms reviewed in Section 3. All the PSTA algorithms are able to analyze the benchmark circuits in a fairly small amount of time. Therefore, we foresee their application to the analysis of larger industrial instances, considering intra-die variations.

6. CONCLUSIONS

In this paper we evaluate the capability of existing PSTA algorithms to handle intra-die delay models. We conclude that, even though the number of parameters in such models can be extremely large, existing PSTA algorithms are able to adequately handle them. Additionally, we propose two techniques built on top of [5]. One of such techniques, consists in using the estimates computed by [6] to aid [5] in the computation of the worst-delay path and corner, which is not directly provided by [7] or [6] and is of great interest in design validation and optimization. This technique enables significant savings in terms of run-time. Finally, we also propose a local bounding technique, that relies on the fact that in intra-die models variables are clustered between particular logic levels. This technique is also particularly effective in reducing the size of delay formulas carried across the circuit. Even though its impact in terms of run-time was only moderate, we expect it to be more effective for larger instances than the ones available in our experiments.

Design	#PI	#PO	#Logic	#Net	#Vertex	#Edge
C432	36	7	88	124	356	457
C499	41	32	170	211	595	736
C880	60	26	169	232	697	910
C1355	41	32	170	211	595	737
C1908	33	25	202	235	708	921
C2670	157	64	278	511	1204	1474
C3540	55	22	469	781	2551	3429
C5315	178	123	597	781	2551	3429
C6288	32	32	1005	1470	3556	5006

Table 1: Benchmark characterization.

Design	[5]		[5]+[6]		[5]+LB		[5]+[6]+LB	
	#P	CPU	#P	CPU	#P	CPU	#P	CPU
C432	325	0.34	334	0.10	126	0.33	128	0.08
C499	140	0.51	181	0.04	61	0.52	61	0.06
C880	333	0.37	342	0.09	145	0.37	147	0.10
C1355	131	0.59	180	0.04	61	0.61	62	0.07
C1908	308	0.74	340	0.09	112	0.72	113	0.08
C2670	218	0.33	222	0.08	89	0.36	89	0.10
C3540	411	3.17	501	0.25	158	3.03	170	0.22
C5315	211	1.01	210	0.17	112	1.04	104	0.19
C6288	780	139.46	916	2.96	180	112.54	217	1.02

Table 2: WDC with improved versions.

Design	Canonical[7]	Hyperplanes[6]	WDC[5]+[6]+LB
C432	<0.01	0.01	0.08
C499	<0.01	0.02	0.07
C880	0.25	0.03	0.10
C1355	0.19	0.02	0.07
C1908	0.27	0.02	0.08
C2670	0.26	0.04	0.11
C3540	0.72	0.10	0.22
C5315	0.49	0.11	0.19
C6288	2.92	0.13	1.02

Table 3: CPU time for PSTA algorithms.

7. REFERENCES

[1] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations. In *Proceedings of ICCAD*, pages 900–907, San Jose, CA, November 2003.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] H. Chang and S. Sapatnekar. Statistical Timing Analysis Under Spatial Correlations. *IEEE Transactions on CAD*, 24:1467–1482, September 2005.

[4] C. E. Clark. The Greatest Finite Set of Random Variables. *Operations Research*, 9:85–91, 1961.

[5] L. G. e Silva, J. Phillips, and L. M. Silveira. Effective Corner-Based Techniques for Variation-Aware IC Timing Verification. *IEEE Transactions on CAD*, 29:157–162, January 2010.

[6] S. Onaissi and F. N. Najm. A Linear-Time Approach for Static Timing Analysis Covering All Process Corners. *IEEE Transactions on CAD*, 27:1291–1304, July 2008.

[7] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett. First-Order Incremental Block-Based Statistical Timing Analysis. *IEEE Transactions on CAD*, 25(10):2170–2180, October 2006.