First-Order Incremental Block-Based Statistical Timing Analysis

Chandu Visweswariah, *Fellow, IEEE*, Kaushik Ravindran, *Student Member, IEEE*, Kerim Kalafala, Steven G. Walker, Sambasivan Narayan, Daniel K. Beece, Jeff Piaget, Natesan Venkateswaran, and Jeffrey G. Hemmett

***Abstract*—Variability in digital integrated circuits makes timing verification an extremely challenging task. In this paper, a canonical first-order delay model that takes into account both correlated and independent randomness is proposed. A novel linear-time block-based statistical timing algorithm is employed to propagate timing quantities like arrival times and required arrival times through the timing graph in this canonical form. At the end of the statistical timing, the sensitivity of all timing quantities to each of the sources of variation is available. Excessive sensitivities can then be targeted by manual or automatic optimization methods to improve the robustness of the design. This paper also reports the first incremental statistical timer in the literature, which is suitable for use in the inner loop of physical synthesis or other optimization programs. The third novel contribution of this paper is the computation of local and global criticality probabilities. For a very small cost in computer time, the probability of each edge or node of the timing graph being critical is computed. Numerical results are presented on industrial application-specified integrated circuit (ASIC) chips with over two million logic gates, and statistical timing results are compared to exhaustive corner analysis on a chip design whose hardware showed early mode timing violations.**

***Index Terms*—Criticality probability, incremental timing, statistical static timing, variability.**

I. Introduction

The TIMING characteristics of gates and wires that make up a digital integrated circuit show many types of variability. There can be variability due to manufacturing, due to environmental factors such as V_{dd} and temperature, and due to device fatigue phenomena such as electromigration, hot electron effects, and negative bias temperature instability (NBTI). The variability makes it extremely difficult to verify the timing of a design before committing it to manufacturing. Nominally subcritical paths or timing points may become critical in some regions of the space of variations due to excessive sensitivity to one or more sources of variation. The goal of robust design, to first order, is to minimize such sensitivities.

Traditional static timing methodology is corner based or case based, e.g., best case, worst case, and nominal. Unfortunately, such a methodology may require an exponential number of timing runs as the number of independent and significant sources of variation increases. Further, as described in [1], the analysis may be both pessimistic and risky at the same time. At corners that are timed, worst case assumptions are made, which are pessimistic; whereas, since it is intractable to analyze all possible corners, the missing corners may lead to failures detected after the manufacturing of the chip. Statistical timing analysis is a solution to these problems.

Statistical timing algorithms fall into two broad classes. The first is path-based algorithms wherein a selected set of paths is submitted to the statistical timer for detailed analysis. This set of methods can be thought of as "depth-first" traversal of the timing graph. In [2], the maximum of a set of path delays is computed, but correlations between the path delays are ignored. In [3], some theoretical results are derived on bounds on the maximum of a set of path delays under certain restrictions. In [4], these assumptions are relaxed, and correlations both due to dependence on global sources of variation and due to reconvergent fan-out (or path sharing) are taken into account.

Path-based statistical timing is accurate and has the ability to realistically capture correlations, but suffers from other weaknesses. First, it is not clear how to select paths for the detailed analysis since one of the paths that is omitted may be critical in some part of the process space. Second, path-based statistical timing often does not provide the diagnostics necessary to improve the robustness of the design. Third, path-based timing does not lend itself to incremental processing, whereby the calling program makes a change to the circuit and the timer answers the timing query incrementally and efficiently [5]. Finally, path-based algorithms are good at taking into account global correlations but do not handle independent randomness in individual delays. Doping effects and gate oxide imperfections are usually modeled as uncorrelated random phenomena. In fact, few, if any, statistical timing attempts in the literature include support for both correlated and independent randomness.

The statistical timer described in this paper belongs to the second class of statistical timers, namely block-based statistical timers. This set of methods traverses the timing graph in a levelized "breadth-first" manner. In [6], probability distributions are assumed to be trains of discrete impulses, which are propagated through the timing graph. However, correlations both due

Manuscript received June 16, 2004; revised October 27, 2004, January 31, 2005, and June 24, 2005. This paper was recommended by Associate Editor D. Blaauw.

C. Visweswariah, S. G. Walker, and D. K. Beece are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA.
K. Ravindran was with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA. He is now with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA.
K. Kalafala, J. Piaget, and N. Venkateswaran are with IBM Microelectronics, Hopewell Junction, NY 12533 USA.
S. Narayan and J. G. Hemmett are with IBM Microelectronics, Essex Junction, VT 05452 USA.

Digital Object Identifier 10.1109/TCAD.2005.862751

to global dependencies on the sources of variation and due to path sharing are ignored, as is the case with [7]. In this same general framework, [8] describes how correlations due to reconvergent fan-out can be taken into account, but not dependence on global sources of variation. In [9], an approximate block-based statistical timing analysis algorithm is described to reduce pessimism in worst case static timing analysis. The concept of parameterized delay models is proposed. Recently, [10] and [11] focused on handling spatial correlations due to intradie variability. While the timer in this paper shares some key similarities with previous efforts (such as the use of a general canonical delay model), these also suffer from some weaknesses. First, they do not provide diagnostics that can be used by a human designer or synthesis program to make the circuit more robust. Second, there is no report of any incremental statistical timing approach in the literature. Third, with the exception of [11], they do not provide for a general enough timing model to accommodate correlation due to dependence on common global sources of variation, independent randomness, and correlation due to path sharing or reconvergent fan-out. This paper describes a statistical timing algorithm that possesses the following strengths.

1) A canonical first-order delay model is employed for all timing quantities. The model allows for both global correlations and independent randomness (spatially correlated sources of variability are currently handled by means of derating factors, and their statistical treatment will be a subject of future work). Thus, timing results such as arrival times and slacks are also available in this canonical form, thereby providing first-order sensitivities to each of the sources of variation. These diagnostics can be used to locate excessive sensitivity to sources of variation and to target robust circuit designs by reducing these sensitivities.
2) The statistical timing algorithm is approximate but has linear complexity in the size of the circuit and the number of global sources of variation. The speed of the algorithm and its block-based nature allow the tool to time very large circuits and incrementally respond to timing queries after changes to a circuit are made. To the best of the authors' knowledge, this is the first incremental statistical timer in the literature or industry.
3) The algorithm computes, with a very small computer time overhead, local and global criticality probabilities, which are useful diagnostics in improving the performance and robustness of a design.

II. Canonical Delay Model

All gate and wire delays, arrival times, required arrival times, slacks, and slews (rise/fall times) are expressed in the standard or canonical first-order form as

$$a_0 + \sum_{i=1}^{n} a_i \Delta X_i + a_{n+1} \Delta R_a \tag{1}$$

where a_0 is the mean or nominal value, $\Delta X_i,\ i = 1, 2, \ldots, n$, represent the variation of n global sources of variation X_i, $i = 1, 2, \ldots, n$, from their nominal values, $a_i, i = 1, 2, \ldots, n$, give the sensitivities to each of the global sources of variation, ΔR_a is the variation of an independent random variable R_a from its mean value and, a_{n+1} is the sensitivity of the timing quantity to R_a. By scaling the sensitivity coefficients, we can assume that X_i and R_a are unit normal or Gaussian distributions $N(0, 1)$. Not all timing quantities depend on all global sources of variation; in fact, [10] and [11] suggest methods of modeling across-chip linewidth variation (ACLV) by having delays of gates and wires in physically different regions of the chip depend on different sets of random variables. In chips with voltage islands, the delay of an individual gate will depend only on the variability of the power supply of the island in which it is physically located.

III. Concept of Tightness Probability

Given any two random variables X and Y, the tightness probability T_X of X is the probability that it is larger than (or dominates) Y. Given n random variables, the tightness probability of each is the probability that it is larger than all the others. Tightness probability is called binding probability in [4] and [12]. The tightness probability of Y, T_Y is $(1 - T_X)$. In the following, we show how to compute the max of two timing quantities in canonical form and how to determine their tightness probabilities. Given two timing quantities

$$A = a_0 + \sum_{i=1}^{n} a_i \Delta X_i + a_{n+1} \Delta R_a \tag{2}$$

and

$$B = b_0 + \sum_{i=1}^{n} b_i \Delta X_i + b_{n+1} \Delta R_b \tag{3}$$

their 2×2 covariance matrix can be written as

$$\operatorname{cov}(A, B) = \begin{bmatrix} a_1 & a_2 & \cdots & a_n & a_{n+1} & 0 \\ b_1 & b_2 & \cdots & b_n & 0 & b_{n+1} \end{bmatrix} [V] \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \vdots & \vdots \\ a_n & b_n \\ a_{n+1} & 0 \\ 0 & b_{n+1} \end{bmatrix} \tag{4}$$

where V is the covariance matrix of the sources of variation. Assuming that the X_i are independent random variables for the purposes of illustration, V is the unity matrix, and, thus

$$\operatorname{cov}(A, B) = \begin{bmatrix} \sum_{i=1}^{n+1} a_i^2 & \sum_{i=1}^{n} a_i b_i \\ \sum_{i=1}^{n} a_i b_i & \sum_{i=1}^{n+1} b_i^2 \end{bmatrix} = \begin{bmatrix} \sigma_A^2 & \rho\sigma_A\sigma_B \\ \rho\sigma_A\sigma_B & \sigma_B^2 \end{bmatrix}. \tag{5}$$

By comparing the terms in (5), σ_A, σ_B, and the correlation coefficient ρ can be computed in linear time. Now, we seek to determine the distribution of $\max(A, B)$ and the tightness probabilities of A and B. We appeal to [13] and [14] for analytic expressions to solve this problem. Define

$$\phi(x) \equiv \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{6}$$

$$\Phi(y) \equiv \int_{-\infty}^{y} \phi(x)\, dx \tag{7}$$

$$\theta \equiv \left(\sigma_A^2 + \sigma_B^2 - 2\rho\sigma_A\sigma_B\right)^{\frac{1}{2}}. \tag{8}$$

Then, the probability that A is larger than B is

$$\begin{aligned} T_A &= \int_{-\infty}^{\infty} \frac{1}{\sigma_A}\phi\left(\frac{x-a_0}{\sigma_A}\right)\Phi\left(\frac{\left(\frac{x-b_0}{\sigma_B}\right) - \rho\left(\frac{x-a_0}{\sigma_A}\right)}{\sqrt{1-\rho^2}}\right) dx \\ &= \Phi\left(\frac{a_0 - b_0}{\theta}\right). \end{aligned} \tag{9}$$

The mean and variance of $\max(A, B)$ can also be analytically expressed as

$$\begin{aligned} E\left[\max(A,B)\right] &= a_0 T_A + b_0(1-T_A) + \theta\phi\left[\frac{a_0-b_0}{\theta}\right] \\ \mathrm{var}[\max(A,B)] &= \left(\sigma_A^2 + a_0^2\right) T_A + \left(\sigma_B^2 + b_0^2\right)(1-T_A) \\ &\quad + (a_0+b_0)\theta\phi\left(\frac{a_0-b_0}{\theta}\right) \\ &\quad - \left\{E\left[\max(A,B)\right]\right\}^2. \end{aligned} \tag{10}$$

Thus, the tightness probabilities, expected value, and variance of $\max(A, B)$ can be computed analytically and efficiently. Similar formulas can be developed for $\min(A, B)$. The CPU time of this operation increases only linearly with the number of sources of variation.

Tightness probabilities have an interpretation in the space of the sources of variation. If one random variable has a 0.3 tightness probability, then in 30% of the weighted volume of the process space, it is larger than the other variable, and in the other 70%, the other variable is larger. The weighting factor is the joint probability density function (JPDF) of the underlying sources of variation.

IV. Block-Based Statistical Timing: Key Idea

To apply these ideas to static timing, we need probabilistic equivalents of the "max," "min," "add," and "subtract" operations. The difficult part of block-based statistical timing is to reexpress the result of a min or max operation in canonical form for further correlated propagation in the timing graph. The concept of tightness probability helps us in this difficult step. The intuition behind this step is explained below in reference to a snippet of the timing graph shown in Fig. 1, assuming late mode computations for illustration purposes.

Let $C = c_0 + \sum_{i=1}^{n} c_i \Delta X_i + c_{n+1}\Delta R_c$ be the late-mode arrival time at node C, $D = d_0 + \sum_{i=1}^{n} d_i \Delta X_i + d_{n+1}\Delta R_d$ be the late-mode arrival time at node D, and the late-mode delays of the two edges of the timing graph be $d_{CG} = e_0 + \sum_{i=1}^{n} e_i \Delta X_i + e_{n+1}\Delta R_e$ and $d_{DG} = f_0 + \sum_{i=1}^{n} f_i \Delta X_i + f_{n+1}\Delta R_f$. We would like to compute the late-mode arrival time at timing point G which is

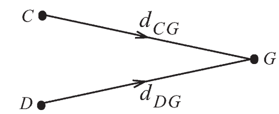

Fig. 1. Part of timing graph.

$$\begin{aligned} G = \max &\left[\left\{c_0 + \sum_{i=1}^{n} c_i\Delta X_i + c_{n+1}\Delta R_c \right.\right. \\ &\left. + e_0 + \sum_{i=1}^{n} e_i\Delta X_i + e_{n+1}\Delta R_e\right\} \\ &\left\{d_0 + \sum_{i=1}^{n} d_i\Delta X_i + d_{n+1}\Delta R_d \right. \\ &\left.\left. + f_0 + \sum_{i=1}^{n} f_i\Delta X_i + f_{n+1}\Delta R_f\right\}\right] \\ = \max &\left[\left\{(c_0+e_0) + \sum_{i=1}^{n}(c_i+e_i)\Delta X_i \right.\right. \\ &\left. + \left(\sqrt{c_{n+1}^2 + e_{n+1}^2}\right)\Delta R_a\right\} \\ &\left\{(d_0+f_0) + \sum_{i=1}^{n}(d_i+f_i)\Delta X_i \right. \\ &\left.\left. + \left(\sqrt{d_{n+1}^2 + f_{n+1}^2}\right)\Delta R_b\right\}\right] \\ = \max &\left[\left\{a_0 + \sum_{i=1}^{n} a_i\Delta X_i + a_{n+1}\Delta R_a\right\}\right. \\ &\left.\left\{b_0 + \sum_{i=1}^{n} b_i\Delta X_i + b_{n+1}\Delta R_b\right\}\right] \end{aligned} \tag{11}$$

where the coefficients of A and B (the two quantities whose max we seek to compute) are computed from the equations above. Thus, independent randomness is treated in a root of the sum of the squares (RSS) fashion, which reduces the spread of delay of a long path consisting of many stages.

Using the formulas of the previous section, we seek to express the max of the two potential arrival times (A and B) back into canonical form for further correlated propagation through the timing graph. From (10), we know the mean and variance of G. In traditional static timing, G would take the value of the larger of A and B, and for all downstream purposes, the characteristics of the dominant potential arrival time that determined the arrival time G are preserved, and the other potential arrival time is ignored. This is like having a tightness probability of 100% and 0%. In the probabilistic domain, the characteristics of G are determined from A and B in the proportion of their tightness probabilities. Thus, if the probabilities were 0.75 and 0.25, the sensitivities of A and B would be linearly combined in a 3:1 ratio to obtain the sensitivities of G. Mathematically

$$g_i = T_A a_i + (1 - T_A) b_i, \qquad i = 1, 2, \ldots, n \tag{12}$$

where T_A is the tightness probability of A.

The mean of the distribution of $\max(A, B)$ is preserved when converting it to canonical form. The only remaining quantity to be computed is the independently random part of the result. This is done by matching the variance of the canonical form to the variance computed analytically from (10). Thus, the first two moments of the real distribution are always matched in the canonical form.

Interestingly, the coefficients computed in this manner preserve the correct covariance to the global sources of variation as derived in [13] and are similar to the coefficients computed in [10]. According to the theorem from [13], the covariance between $G = \max(A, B)$ and any random variable Y can be expressed in terms of covariance between A and Y and B and Y as

$$\text{cov}(G, Y) = \text{cov}(A, Y) T_A + \text{cov}(B, Y)(1 - T_A). \tag{13}$$

Choose $Y = \Delta X_i$, one of the global sources of variation. By observing that $\text{cov}(A, \Delta X_i) = a_i$ and $\text{cov}(B, \Delta X_i) = bi$, we obtain

$$\text{cov}(G, \Delta X_i) = a_i T_A + b_i (1 - T_A). \tag{14}$$

Now, by applying the assumption that G is normally distributed, we get $g_i = a_i TA + b_i(1 - TA)$, confirming the previous intuition. It should be noted that the covariance to the independent sources of variation ΔR_a and ΔR_b is not preserved in our method.

The max of two Gaussians is not a Gaussian, but we reexpress it in the canonical Gaussian form and incur an accuracy penalty for doing so. However, this step allows us to keep alive and propagate correlations due to dependence on the global sources of variation, which is absolutely key to performing timing in a realistic fashion. Monte Carlo results will be shown in the results section to assess the accuracy of this method.

When more than two edges of the graph converge at a node, the max or min operation is conducted one pair at a time, just as with deterministic quantities. The tightness probabilities are treated as conditional probabilities and postprocessed to compute the final tightness probability of each arc incident on the node whose arrival time is being computed. For example, suppose there are three arcs P, Q, and R, incident at a node. Suppose the tightness probabilities when maxing P and Q are 0.6 and 0.4, respectively. The max of these two quantities is then maxed with R, and suppose the tightness probabilities are 0.8 and 0.2, respectively. Then, the final tightness probabilities are $T_P = 0.6 \times 0.8 = 0.48$, $T_Q = 0.4 \times 0.8 = 0.32$, and $T_R = 0.2$. As more equally critical signals are maxed, accuracy degrades slightly, since the asymmetry in the resulting probability distribution increases, making it harder to approximate in canonical form.

Slews (rise/fall times) are propagated in much the same manner. If the policy is to propagate the worst slew, then a separate tightness probability is computed for the slews and applied to represent the bigger slew in canonical form. If the policy is to propagate the latest arriving slew, then the same arrival tightness probabilities are applied to combine the incoming slews to obtain the output slew.

In this manner, by replacing the "plus," "minus," "max," and "min" operations with probabilistic equivalents, and by reexpressing the result in a canonical form after each operation, regular static timing can be carried out by a standard forward and backward propagation through the timing graph [15]. Early and late mode, separate rise and fall delays, sequential circuits, and timing tests are, therefore, easily accommodated just as in traditional timing analysis.

V. Criticality Computation

The methods presented in the previous section enable statistical timing analysis, during which the concept of tightness probability is leveraged to propagate arrival and required arrival times in a parametric canonical form. In this section, the use of tightness probabilities in computing criticality probabilities [16] is presented. One of the important outcomes of deterministic timing is the ability to find the most critical path. In the statistical domain, the concept of the most critical path is probabilistic. The criticality probability of a path is the probability that the path is critical; the criticality probability of an edge is the probability that the edge lies along a critical path; and the criticality probability of a node is the probability that a critical path passes through that node. Computing these probabilities will obviously have important benefits in enumerating critical paths, enabling robust optimization and generating test vectors for at-speed test.

The method of computing criticality probabilities in this section assumes independence between the various tightness probabilities in a timing graph. While we believe this is a reasonable assumption in practice, it is nonetheless a theoretical limitation of the approach.

A. *Forward Propagation*

The ideas behind criticality computations are described by means of an example. Consider the combinational circuit of Fig. 2. In this example, separate rising and falling delays and slew effects are ignored for simplicity, but the ideas can be extended in a straightforward manner. Likewise, sequential circuits pose no special problem. The example assumes late-mode timing, but early-mode timing follows the same reasoning.

The timing graph of the circuit is shown in Fig. 3. During the forward propagation phase of timing analysis, each edge of the timing graph is annotated with an arrival tightness probability (ATP), which is the probability that the edge determines the arrival time of its output node. The ATPs in this example have

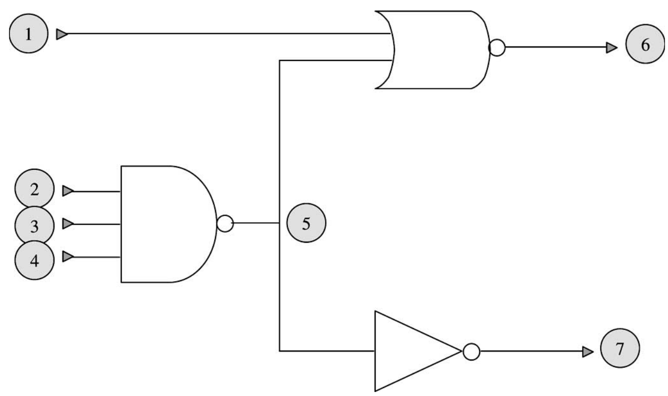

Fig. 2. Sample circuit.

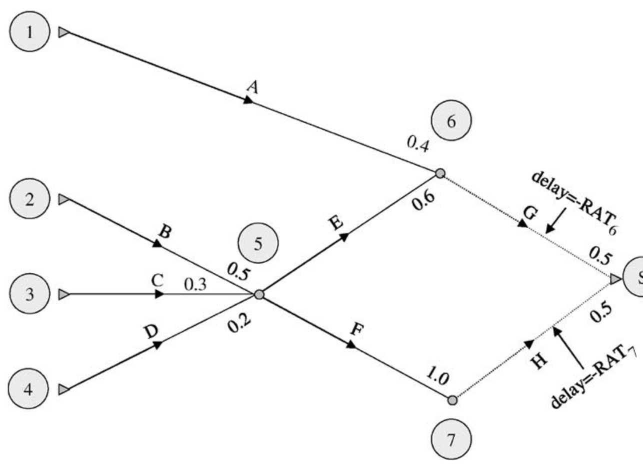

Fig. 3. Timing graph of sample circuit.

been chosen arbitrarily and are shown at the tail of each edge of the timing graph. Once the primary outputs are reached, a virtual output edge is added from each primary output to a sink node, shown as edges G and H in Fig. 3. Each such edge is considered to have a delay equal to the negative of the asserted required arrival time at the corresponding primary output. In the presence of timing tests (such as setup, hold, or clock pulsewidth tests), a virtual edge is added to the sink node whose delay is the negative of the computed statistical required arrival time. Then the standard forward propagation procedure is continued to compute the "arrival time" of the sink of the graph, and the ATPs of the virtual output edges. In this case, for illustration purposes, the ATP of each of the virtual output edges is chosen to be 0.5.

Property 1: The sum of the ATPs of all edges incident on any node of the timing graph is 1.0.

Property 2: The criticality of a path is the product of the ATPs of all edges along the path. For path 2B5E6GS to be critical, for example, edge B has to determine the arrival time of node 5 (probability $= 0.5$), edge E has to determine the arrival time of node 6 (probability $= 0.6$), and edge G has to determine the arrival time of node S (probability $= 0.5$), for a total probability of 0.15, assuming independence between these events.

Property 3: The sum of the criticality of all paths in a timing graph is 1.0.

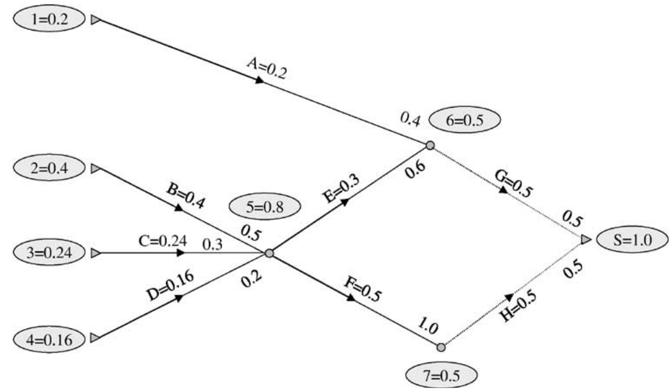

Fig. 4. Backward traversal of timing graph.

B. Backward Propagation

Fig. 4 shows the criticality calculations during the backward propagation phase of timing analysis. During the backward propagation, we will compute the global criticality of each edge and each node of the timing graph, and the required arrival tightness probability (RATP) of each edge of the timing graph, which is the probability that the edge determines the required arrival time of its source node.

Property 4: The sink node has a node criticality probability of 1.0. This property is obvious, since all paths must pass through the sink node. The sum of the ATPs of the virtual output edges is, therefore, also 1.0.

Starting with the sink node S, the backward propagation first considers edges G and H. They each have a 0.5 edge criticality, since they each determine the arrival time of S with 0.5 probability. The criticality of nodes 6 and 7 are, likewise, 0.5 each.

Property 5: The criticality of an edge is the product of its ATP and the criticality probability of its sink node. Clearly, an edge is globally critical only to the extent the sink node is critical and it determines the arrival time of that sink node.

Property 6: The criticality of a node in the timing graph is the sum of the criticality of all edges leaving that node. Using the above two properties, the criticalities of edges and nodes are easily computed during a levelized backward traversal of the timing graph and are shown in Fig. 4. The criticality computations can piggy-back on top of the usual required arrival time calculations. Note that the criticality of edge A, for example, is the product of the criticality of node 6 (0.5) and the ATP of edge A (0.4). The criticality of node 5, for example, is the sum of the edge criticalities of edges E and F.

Corollary 6.1: The criticality of any node in the timing graph is the sum of the path criticalities of all paths in its fan-out cone. For example, node 5 has two paths in its fan-out cone, path 5E6GS with a path criticality of 0.3 and path 5F7HS with a path criticality of 0.5, totaling to a node criticality of 0.8 for node 5.

Property 7: The sum of the node criticalities of all the primary outputs is 1.0. For general sequential circuits, this property would apply to all slack-determining endpoints (primary output and timing test points).

As the backward propagation progresses, RATPs are computed and annotated on to the timing graph. These probabilities are shown close to the source node of each edge in Fig. 5.

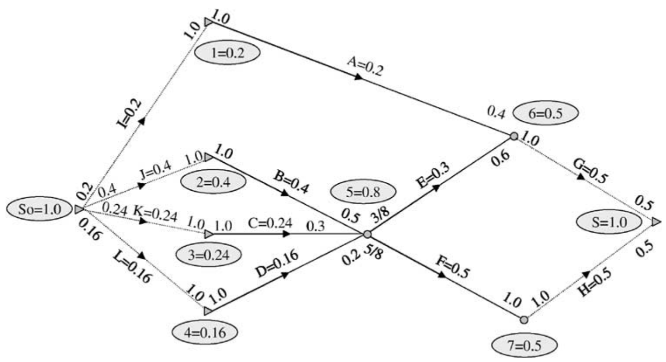

Fig. 5. Source node of timing graph.

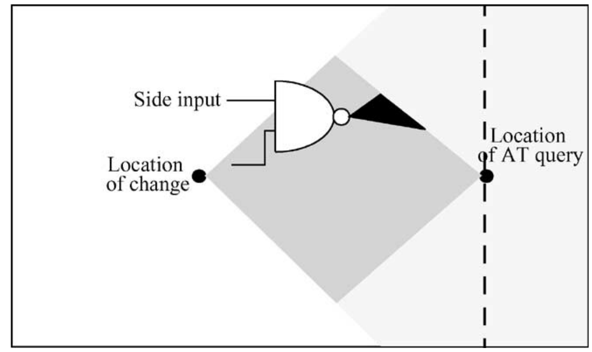

Fig. 6. Incremental timing analysis.

Property 8 (Dual of Property 1): The sum of the RATPs of all edges originating at any node of the timing graph is 1.0. At a node such as 5, where there are multiple fan-out edges, the RATPs will be in the proportion of the edge criticality probabilities of the downstream edges. When the primary inputs are reached during backward traversal, a new node of the timing graph called the source node is postulated, with virtual input edges from the source node to each of the primary inputs, shown as edges I, J, K, and L in Fig. 5. Each virtual input edge is considered to have a delay equal to the arrival time of the corresponding primary input, and the required arrival time of the source node is computed. During this computation, the RATPs of the virtual edges are also determined.

Property 9: The ATP of each of the virtual input edges is 1.0.

Property 10 (Dual of Property 4): The criticality of the source node is 1.0. This property is obvious, since every path passes through the source node.

Property 11 (Dual of Property 7): The sum of the node criticalities of all the primary inputs is 1.0.

Property 12 (Dual of Property 9): The sum of the edge criticalities of the virtual input edges is 1.0 as is the sum of their RATPs.

Property 13 (Dual of Property 2): The criticality of any path is the product of the RATP of all edges of the path. Thus, the criticality of path SoJ2B5E6GS is $0.4 \times 1.0 \times 3/8 \times 1.0 = 0.15$.

Property 14: The criticality of an edge is the sum of the criticality of all paths through that edge.

Property 15: The product of the ATPs along any path of the graph is equal to the product of the RATPs.

Property 16: The sum of the edge criticalities of any cutset of the timing graph that separates the source from the sink node is 1.0. In other words, any cut through the graph that leaves the source node on one side and the sink node on the other will cut edges whose criticality probabilities sum to 1.0. This must be the case since every critical path will have to pass through exactly one edge of the cutset.

It is important to note that the edge and node criticalities can be computed on a global basis, or on a per-endpoint basis, where an endpoint is a slack-determining node of the graph (a primary output or either end of a timing test segment). The application will dictate which type of computation is more efficient and suitable.

C. Path Enumeration

Enumeration of paths in order of criticality probability is useful in a number of different contexts, such as producing reports, providing diagnostics to the user or a synthesis program, listing paths for test purposes, listing paths for common path pessimism removal (CPPR) purposes [17], and enumerating paths for analysis by a path-based statistical timer [4]. One straightforward manner of enumerating paths is by means of a breadth-first visiting of the nodes of an augmented graph as shown in Fig. 5, while following the unvisited node with the highest criticality probability at each juncture. A running total of the criticality probability of the listed paths is maintained, and the path enumeration stops when the set of critical paths has been covered with a certain confidence.

During the path enumeration, the following properties are useful.

Property 17: The ATP of an edge is an upper bound on the criticality of any path that passes through that edge.

Property 18: The RATP of an edge is an upper bound on the criticality of any path that passes through that edge.

Property 19: The criticality probability of an edge is an upper bound on the criticality of any path that passes through that edge.

Property 20: The criticality probability of a node is an upper bound on the criticality of any path that passes through that node.

VI. Incremental Statistical Timing

Optimization or physical synthesis programs often call an incremental timer millions of times in their inner loop. To suit this purpose, a statistical timer needs to incrementally and efficiently answer timing queries after one or more changes to the circuit has been made.

Consider the situation shown in Fig. 6. Assume a single change has been made to the circuit at the location shown. The change could be the addition of a buffer, the resizing of a gate, the removal of a latch, and so on. Assume that the calling program queries the timer for the arrival time at the "Location of AT query" point. Clearly, only the arrival times in the yellow cone of logic change (on black-and-white hardcopies, the lightest grey region). Further, only arrival time changes in the fan-in cone of the query point can have an effect on the query. The intersection of these regions of logic is shown in green (or the darker grey region). Theoretically, by purely topological reasoning, the portion of the circuit that must be retimed to answer this query can be limited to the intersection of these two cones of logic. This kind of limiting is called level limiting and is accomplished by storing arrival time (AT), required arrival time (RAT), and AT–RAT levels for each gate [5]. In practice, all arrival times in the fan-out cone of the change point and to the left of the query point (i.e., up to the vertical dashed line shown in Fig. 6) are updated. The levelization and limiting procedures are identical for the statistical timing situation, and the implementation can easily ride on top of an existing deterministic incremental capability.

In addition to level limiting, the amount of recomputation can be further reduced by dominance limiting. Consider the NAND gate shown in Fig. 6. One input of the NAND gate is from the "changed" cone of logic and the other from an unchanged region. If the arrival time at the output of the NAND gate is unchanged, because it was determined both before and after the change by the side input, then the fan-out cone of the NAND gate (shown in dark black in Fig. 6) can potentially be skipped in answering the query. This type of limiting is called dominance limiting. In our statistical timer, the notion of "change" is treated probabilistically by examining the tightness probabilities. If the ATP of the side input is sufficiently close to 1.0 both before and after the change, then the arrival time of the output of the NAND gate need not be recomputed, and its fan-out cone can potentially be skipped until some other input of that fan-out cone is known to have materially changed. Similar concepts are applicable during backward propagation of required arrival times.

Of course, there are several complications that must be faced in a real application such as latches, multiple clock phases and phase changes, and the dynamic adaptation of data structures to such changes. These details are omitted due to lack of space, but our implementation takes into account all of these considerations.

VII. Implementation

The above ideas have been implemented in a prototype called EinsStat. EinsStat is implemented on top of the static timing analysis program EinsTimer in C++ with Tcl scripting under Nutshell. Multiple clock phases, phase renaming, rule tests (such as setup and hold tests), automatic tests (such as clock gating, clock pulsewidth, and clock inactive tests), loop cut checks, same-mode constraints (comparing late versus late or early versus early, instead of the usual late/early comparison), arbitrary timing assertions, timing adjusts anywhere in the timing graph, and clock overrides are supported as in EinsTimer. The timer works permanently in incremental mode [18], even if a complete timing report is requested.

Each timing assertion, gate delay, wire delay, and timing test guard time must be modeled in canonical form, i.e., with a mean part, a dependence on global sources of variation and an independent random portion. Backward compatibility with deterministic timing is preserved by setting the mean value of an adjust or assertion to the deterministic value, and the randomness to zero or to a user-specified proportional variability. The EinsStat implementation allows each gate and each wire to have its own customized variability model, provided the model can be expressed in the canonical form. Furthermore, the EinsStat implementation utilizes a general-purpose three-tier sensitivity modeling approach, whereby delay dependencies to underlying sources of variation can be obtained either by: 1) analytic means (i.e., appealing to either technology models or an underlying simulator); 2) finite differencing of corner-based delays; or 3) using user-specified global assertions (e.g., EinsStat supports Tcl commands to express a situation in which, for example, "all normal Vt gates have a 1% independent randomness and a 4% correlated variability, and similarly all low Vt gates have a 2% independent randomness and 5% correlated variability; and, furthermore, the two sets of variations mistrack with respect to each other"). To enable efficient memory use, each source of variation may be categorized as either "sparse" (maintained in a linked-list data structure, avoiding the need to explicitly store zero sensitivity values) or "dense" (in a compact array structure, using fixed variable indices, explicitly storing zero sensitivity values). As an example, lower levels of metal that are used frequently throughout a design are preferably represented densely, while less frequently used higher levels of metal are better off being treated in a sparse manner.

EinsStat supports a multitude of process variables, including individual metal layers, N-type field effect transistor/P-type field effect transistor (NFET/PFET) mistracking, mistracking between different Vt device families, and product-reliability factors. For initial testing purposes, three global sources of variation were studied. The first is gate versus wire delays. Each of these sets of delays can have an independent and correlated variability, and a mistrack coefficient. In the case of gate versus wire delays, mistrack implies that when gates get faster, wires get slower, and vice versa, and in general expresses correlations between the two sets of delays. The second supported global source of variation is rise versus fall delays of gates (to model N/P mistrack due to manufacturing variations or fatigue effects). Again, each of these can have a random and correlated part and a mistrack coefficient. The third supported source of variation is meant similarly to study mistrack between normal Vt and low Vt gates. In the benchmark results presented in the next section, sensitivities to these three global sources of variation were provided in a blanket fashion as a percentage of the nominal delay.

VIII. Numerical Results

EinsStat was first run on industrial application-specific integrated circuit (ASIC) chips of various sizes with zero random variability and no global sources of variation. The arrival time, required arrival time, and slack were compared between EinsTimer and EinsStat at every node of the circuit, for every clock phase, both rising and falling, and for both early mode and late mode. This was a good test to detect certain kinds of software bugs in the EinsStat implementation, since the two sets of results must be identical in the absence of any variability.

TABLE I
CPU AND MEMORY RESULTS

Name	Gates	Clock domains	Propagate segments	CPU time (secs.)			Memory (MB)		
				Load	EinsTimer	EinsTimer + EinsStat	Base	EinsTimer	EinsTimer + EinsStat
A	3,042	2	17,579	5.1	2.8	3.8	111	53	60
B	183,186	79	959,709	140.5	121.3	187.6	423	177	723
C	1,085,034	182	5,799,545	5131.5	809.9	1233.1	3200	600	4300
D	1,213,361	18	6,969,860	783.5	1079.3	1485.7	2990	1160	4380
E	2,095,176	51	13,460,759	1494.9	1316.9	2724.3	4590	3320	11330

A set of industrial ASIC designs was timed with three global sources of variation as well as independent randomness built into every edge of the timing graph. The benchmark results are shown in Table I, in which the chips are code named A, B, etc., to preserve confidentiality. The column "Propagate segments" represents the number of edges in the timing graph with unique source–sink pairs of nodes. The "Load" column lists the CPU time to load the netlist, timing models, and assertions. The "EinsTimer + EinsStat" column is the CPU time of the deterministic base timer, while the "EinsStat" column shows the CPU time taken when the statistical timer runs alongside (and in addition to) the deterministic timer. All CPU times were measured on an IBM Risc/System 6000 model 43P-S85 on a single processor. All timing runs included forward propagation of early and late arrival times and reverse propagation of early and late required arrival times. Similarly, the memory consumption to load each design, assertions and delay models (Base), run deterministic timing (EinsTimer), and statistical timing alongside (and in addition to) deterministic timing (EinsTimer + EinsStat) are shown in subsequent columns of Table I. The CPU and memory overhead of statistical timing are very reasonable, considering the wealth of additional data being generated. In the small test case A, memory consumption was dominated by the delay models, so the overhead due to statistical timing was dwarfed. In test case E, the larger overhead was due to nodes in the timing graph having extremely high incidence due to silicon-on-chip (SoC) timing macromodels.

The statistical experiments were performed both with and without criticality computations, and the CPU time and memory overhead were observed to be nearly identical (within 1%), lending credence to the efficiency of the criticality computations.

Test chip "A" (3042 logic gates) was used to demonstrate the importance of global correlations. The critical path in this chip is a long combinational path passing through about 60 stages of logic, with a nominal delay of 23.06 ns, including wire delay. With 5% correlated variability (i.e., assuming all delays move in concert with respect to a source of variability) on every gate and wire delay, the longest path delay is 23.01 ns with a σ of 0.9 ns. With 5% independent variability (i.e., assuming each circuit delay may vary independently) on every gate and wire delay, the longest path delay is 23.62 ns with a σ of 0.13 ns. Clearly, with more independent randomness, there is more cancellation of variability along a long path, yielding a tighter distribution but with a more pessimistic mean. The correlated case produces a more optimistic mean path delay but with a much bigger spread. EinsStat allows the modeling of these extreme situations and anything in-between.

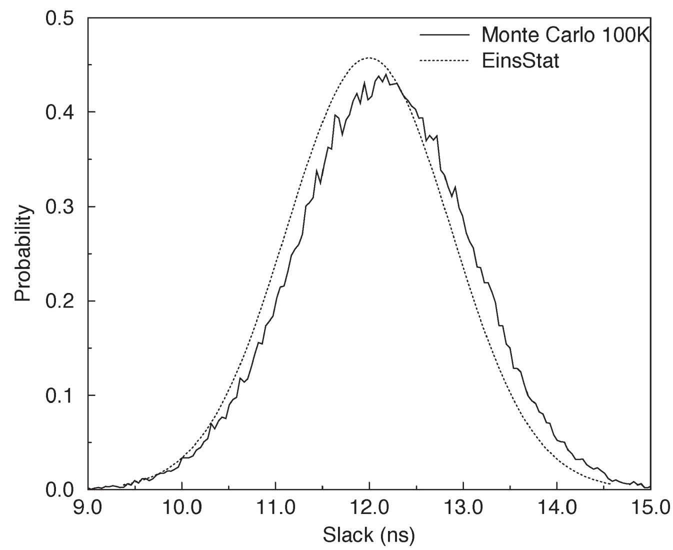

Fig. 7. EinsStat versus Monte Carlo analysis case "Monte Carlo 1."

The primary goal of EinsStat is to produce timing results in a parameterized form and, therefore, to give the designer information regarding the robustness of the design. However, EinsStat produces these timing results as random variables, and the correctness of the mean and spread of these random variables can be verified by Monte Carlo analysis. To render the analysis tractable, EinsStat makes a number of assumptions that prevent it from obtaining the exact result. Inaccuracy creeps in every time the probability distribution resulting from a max or min operation is reexpressed in canonical form. Specifically, the max or min of two Gaussians is not Gaussian, but EinsStat forces it back into a Gaussian form. The extent of these inaccuracies is revealed by Monte Carlo analysis.

In order to validate the timing results obtained from EinsStat, a comparison of EinsStat with Monte Carlo simulation on four small- to medium-sized benchmarks was performed. For each case, one representative slack, that of the nominally critical endpoint, was selected for comparison purposes. Figs. 7–10 show the slack distribution of both EinsStat and Monte Carlo analysis for the four test cases. It can be seen from these figures that EinsStat predicts the mean value, spread, and tails with reasonable accuracy.

The runtime comparison of the EinsStat runs with that of Monte Carlo analysis appears in Table II. The runs were performed on the same computer. From Table II, it can be seen

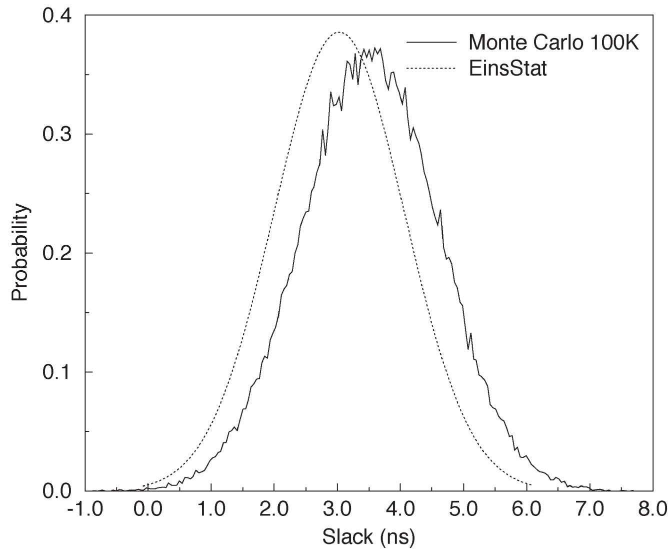

Fig. 8. EinsStat versus Monte Carlo analysis case “Monte Carlo 2.”

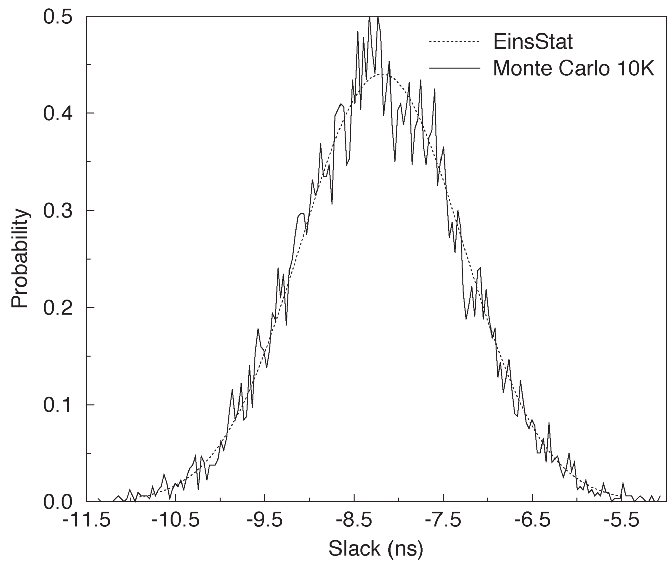

Fig. 9. EinsStat versus Monte Carlo analysis case “Monte Carlo 3.”

that EinsStat is significantly faster than both sequential and parallel (utilizing up to 45 processors) Monte Carlo analysis.

Early on in the verification process, it became obvious that the runtimes required for serial Monte Carlo would quickly become prohibitive. Therefore, significant development effort was invested to create a high-performance Monte Carlo capability. This tool uses a client/server approach to perform parallel timing runs on different host machines, controlled by a central Monte Carlo process, with all data transfer occurring via transmission control protocol (TCP). While this effort made Monte Carlo verification a viable option on the larger designs, note that runtimes are still several magnitudes of order larger than those of EinsStat (see column 6 of Table II).

A repowering experiment on chip “A” was used to evaluate incremental operation of EinsStat. For each of 493 gates with negative slack, the gate power level (size) was modified, and EinsStat was queried for the new slack on each pin of the modified gate. Incremental EinsStat was six times faster than

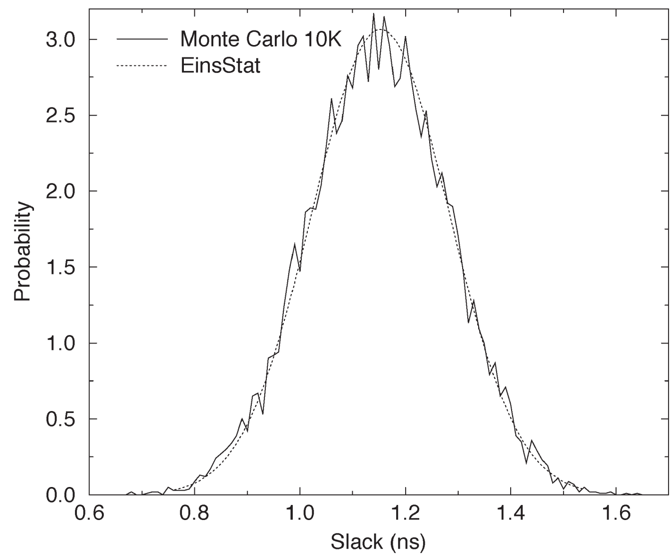

Fig. 10. EinsStat versus Monte Carlo analysis case “Monte Carlo 4.”

TABLE II
MONTE CARLO VERSUS EinsStat COMPARISON

Test case	Gates	EinsStat CPU	Monte Carlo		
			Samples	Sequential CPU dd:hh:mm:ss	Parallel CPU dd:hh:mm:ss
1	18	1 sec.	100000	5:57	N/A
2	3042	2 sec.	100000	2:01:15:10	2:46:55
3	11937	7 sec.	10000	0:20:33:40	51:05
4	70216	59 sec.	10000	N/A	4:36:12

nonincremental EinsStat with identical results. For large designs and for different types of changes and queries, we expect the runtime improvement obtained by incremental processing to be quite dramatic.

An EinsStat analysis of an industrial ASIC design whose hardware was known to have hold violations was performed to consider the effects of back-end-of-the-line variability on circuit performance. This design utilized seven wiring planes, each of which was modeled by an independent random variable to represent metal variability. The results of this analysis were compared to a traditional exhaustive corner-based study (i.e., to determine the combination of fast/slow metal layer assignments that produces the worst possible slack). As indicated in Fig. 11, a statistical treatment of parameter variation results in a 3σ early mode slack of -162 ps representing a pessimism reduction of 63 ps over the traditional exhaustive corner-based analysis.

IX. FUTURE WORK AND CONCLUSION

This paper presents a novel incremental statistical timing algorithm that propagates first-order sensitivities to global sources of variation through a timing graph. Each edge of the timing graph is modeled by a canonical delay model that permits global dependence as well as independent randomness. The timing results are presented in a parametric form, which can help a designer or optimization program target robustness in the design. A novel theoretical framework for computing local and global criticality probabilities is presented, thus providing detailed timing diagnostics at a very small cost in runtime.

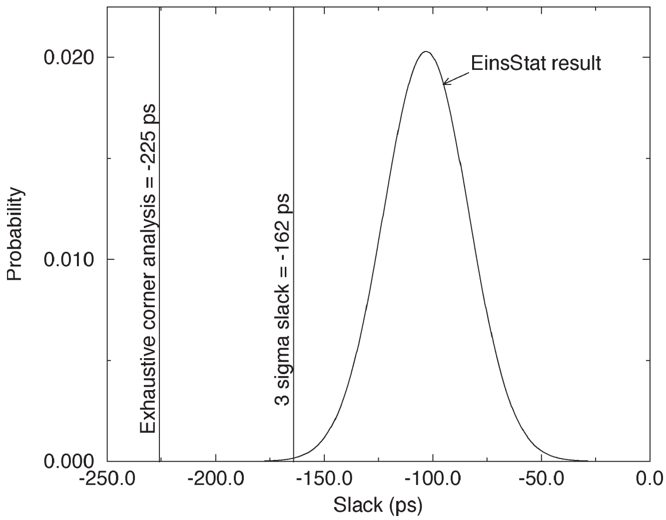

Fig. 11. EinsStat result on industrial ASIC design for early mode slacks.

The following avenues of future work suggest themselves. The assumption of linear dependence of delay on each source of variation is valid only for small variations from nominal behavior. Extending the theory to handle general nonlinear models and asymmetric distributions would be a big step forward. Second, the impact of variability of input slews and output loads on the delay of timing graph edges can be chain ruled into the canonical delay model as suggested by [9]. Third, the criticality computations in this paper assume independence between the criticality probabilities of any two paths, an assumption, but not quite correct. Extending the theory to remove dependence on this assumption is a challenging task that we hope to address in the future. Finally, extending EinsStat to account for spatially correlated variability is another challenging task that we hope to address in future work.

ACKNOWLEDGMENT

The authors would like to thank J. D. Hayes, P. A. Habitz, J. Narasimhan, M. R. Guthaus, D. S. Kung, D. J. Hathaway, V. B. Rao, A. J. Suess, and J. P. Soreff for their useful discussions and contributions.

REFERENCES

[1] C. Visweswariah, “Death, taxes and failing chips,” in *Proc. Design Automation Conf.*, Anaheim, CA, Jun. 2003, pp. 343–347.
[2] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, “Timing yield estimation from static timing analysis,” in *Proc. IEEE Int. Symp. Quality Electronic Design*, San Jose, CA, 2001, pp. 437–442.
[3] M. Orshansky and K. Keutzer, “A general probabilistic framework for worst case timing analysis,” in *Proc. Design Automation Conf.*, New Orleans, LA, Jun. 2002, pp. 556–561.
[4] 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. Design Automation Conf.*, Anaheim, CA, Jun. 2003, pp. 932–937.
[5] R. P. Abato, A. D. Drumm, D. J. Hathaway, and L. P. P. P. van Ginneken, “Incremental timing analysis,” U.S. Patent 5 508 937, Apr. 16, 1996.
[6] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, “Fast statistical timing analysis by probabilistic event propagation,” in *Proc. Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 661–666.
[7] M. R. C. M. Berkelaar, “Statistical delay calculation: A linear time method,” in *Proc. ACM/IEEE Workshop Timing Issues Specification and Synthesis Digital Systems (TAU)*, Austin, TX, Dec. 1997, pp. 15–24.
[8] A. B. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula, “Computation and refinement of statistical bounds on circuit delay,” in *Proc. Design Automation Conf.*, Anaheim, CA, Jun. 2003, pp. 348–353.
[9] L. Scheffer, “Explicit computation of performance as a function of process variation,” in *Proc. ACM/IEEE Workshop Timing Issues Specification Synthesis Digital Systems (TAU)*, Monterey, CA, Dec. 2002, pp. 1–8.
[10] H. Chang and S. S. Sapatnekar, “Statistical timing analysis considering spatial correlations using a single PERT-like traversal,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2003, pp. 621–625.
[11] A. Agarwal, D. Blaauw, and V. Zolotov, “Statistical timing analysis for intra-die process variations with spatial correlations,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2003, pp. 900–907.
[12] J. Jess, “DFM in synthesis,” IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Dec. 2001. Res. Rep.
[13] C. E. Clark, “The greatest of a finite set of random variables,” *Oper. Res.*, vol. 9, no. 2, pp. 145–162, Mar./Apr. 1961.
[14] M. Cain, “The moment-generating function of the minimum of bivariate normal random variables,” *Amer. Stat.*, vol. 48, no. 2, pp. 124–125, May 1994.
[15] C. Visweswariah, “System and method for statistical timing analysis of digital circuits,” Docket YOR9-2003-401, Aug. 2003, filed with the U.S. Patent office.
[16] ——, “System and method for probabilistic criticality prediction of digital circuits,” Docket YOR9-2003-402, Aug. 2003, filed with the U.S. Patent office.
[17] D. J. Hathaway, J. P. Alvarez, and K. P. Belkhale, “Network timing analysis method which eliminates timing variations between signals traversing a common circuit path,” U.S. Patent 5 636 372, Jun. 3, 1997.
[18] C. Visweswariah, “System and method for incremental statistical timing analysis of digital circuits,” Docket YOR9-2003-403, Aug. 2003, filed with the U.S. Patent office.

Chandu Visweswariah (S’82–M’89–SM’96–F’05) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Chennai, India, in 1985, and the M.S. and Ph.D. degrees in computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1986 and 1989, respectively.

He has been a Research Staff Member at IBM’s Thomas J. Watson Research Center, Yorktown Heights, NY since 1990. He presently manages a circuit and interconnect analysis group. He has developed various circuit simulation, circuit optimization, and timing software tools that are in production use in IBM. He is the author or coauthor of one book and technical papers. He holds U.S. patents with more pending. His research interests include modeling, analysis, timing, optimization, and manufacturability of integrated circuits. In 2002, he was a Visiting Assistant Professor at the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands.

Dr. Visweswariah has won one IBM Corporate Award, two IBM Outstanding Technical Achievement Awards, and two IBM Best Paper Awards. He has served on the Technical Program Committee of DAC, ICCAD, ICCD, and CICC. Two of his papers were selected for the “Best of ICCAD” volume of 40 of the best papers published in 20 years of ICCAD. He won a Best Paper Award at DAC 2004.

Kaushik Ravindran (S’03) is currently working toward the Ph.D. degree in electrical engineering from the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA.

Kerim Kalafala received the M.Sc. degree in computer and systems engineering from Rensselaer Polytechnic Institute (RPI), in 1998.

He has worked for IBM/EDA since 1998. He is a Senior Software Engineer in the IBM Electronic Design Automation Group. He has developed various timing analysis algorithms that are in use at IBM.

Mr. Kalafala, along with several coauthors, he has received two IBM best paper awards as well as a best paper award at DAC 2004.

Steven G. Walker received the B.S. degree in applied and engineering physics from Cornell University, Ithaca, NY, in 1983.

He joined the IBM Research Division, Yorktown Heights, NY, in 1989, where he initially worked on the development of automated GaAs process characterization and optoelectronic wafer-level test systems. Since 1995, he has worked on the development of a common design methodology employed by several S/390 and PowerPC complementary metal–oxide–semiconductor microprocessor chips. He has concentrated in the areas of static noise and timing analysis, power analysis, silicon-on-insulator body voltage initialization, and schematic device-level libraries.

Sambasivan Narayan, photograph and biography not available at the time of publication.

Daniel K. Beece received the Ph.D. degree in physics from the University of Illinois, Urbana-Champaign, in 1983.

He has worked in IBM's Watson Research Division since 1982. He has worked in several areas in very large scale integration design automation, including verification and simulation. He is currently working with the System Design Verification Group.

Jeff Piaget received the B.S. degree in computer engineering from the University of Massachusetts, Amherst, in 1988.

He is currently an Advisory Engineer with the IBM Electronic Design Automation Group, Hopewell Junction, NY.

Natesan Venkateswaran received the B.Sc. degree in physics from the University of Madras, India, the M.E. degree in electrical engineering from the Indian Institute of Science, Bangalore, in 1991, and the Ph.D. degree in computer engineering from the University of Cincinnati, Cincinnati, OH, in 1997.

He has since been a member of the Electronic Design Automation Group in the IBM Server and Technology Group, Hopewell Junction, NY. He has been involved with development of placement and floor planning tools in IBM. His current focus is on researching/developing statistical timing analysis tools.

Jeffrey G. Hemmett received the B.S. and M.Sc. degrees in mechanical engineering and the Ph.D. degree in engineering with focus on systems modeling and analysis from the University of New Hampshire, Durham, in 1992, 1994, and 2001, respectively.

He spent two years developing computer-aided design tools at Ford Motor Company. In 2000, he joined the Electronic Design and Automation group at IBM, Essex Junction, VT, where he initially contributed to the development of fast transient simulation and formal sensitivity analysis tools before transitioning into Monte Carlo development, which led to his current focus in the area of statistical static timing.