Statistical timing for parametric yield prediction of digital integrated circuits

Please check the document version of this publication:

Statistical Timing for Parametric Yield Prediction of Digital Integrated Circuits

Jochen A. G. Jess, *Associate Member, IEEE,* Kerim Kalafala, Srinath R. Naidu, Ralph H. J. M. Otten,
and Chandu Visweswariah, *Fellow, IEEE*

Abstract—Uncertainty in circuit performance due to manufacturing and environmental variations is increasing with each new generation of technology. It is therefore important to predict the performance of a chip as a probabilistic quantity. This paper proposes three novel path-based algorithms for statistical timing analysis and parametric yield prediction of digital integrated circuits. The methods have been implemented in the context of the EinsTimer static timing analyzer. The three methods are complementary in that they are designed to target different process variation conditions that occur in practice. Numerical results are presented to study the strengths and weaknesses of these complementary approaches. Timing analysis results in the face of statistical temperature and V_{dd} variations are presented on an industrial ASIC part on which a bounded timing methodology leads to surprisingly wrong results.

Index Terms—Digital integrated circuits, timing.

I. INTRODUCTION

YIELD LOSS is broadly categorized into catastrophic yield loss (due to contamination and dust particles, for example) and parametric or circuit-limited yield loss, which impacts the spread of performance of functional parts. This paper presents three algorithms for statistical timing analysis and parametric yield prediction of digital integrated circuits due to both manufacturing and environmental variations.

With each new generation of technology, variability in chip performance is increasing. The increased variability renders existing timing analysis methodology unnecessarily pessimistic and unrealistic. The traditional "bounded" or "corner-based" static timing approach further breaks down in the case of multiple voltage islands. The International Technology Roadmap for Semiconductors [1] has identified a clear need for statistical timing analysis.

The algorithms in this paper pay special attention to correlations. Capturing and taking into account inherent correlations are absolutely the key in obtaining a correct result. Correlations occur because different paths may share one or more gates, and because all gate delays depend on some global parameters such as junction depth or ambient temperature. All methods in this paper fully take into account both classes of correlations and are equipped to handle deterministic across-the-chip variations. This paper does not directly address spatial correlation across a chip, but our methods can be extended to handle it as well, using the same principle as suggested in [2].

II. PREVIOUS WORK

There is a wealth of literature on parametric statistical timing analysis and yield prediction. The problem was first proposed in the context of statistical program evaluation and review technique (PERT) where the objective was to calculate the probability distribution curve of the project completion time, given that the subtasks in a task graph were random variables drawn from some distribution. The problem was quickly recognized as falling in a difficult complexity class known as the #P-complete class. This meant that it was impossible to produce in polynomial time a constant-factor approximation of the true probability distribution curve of project completion time. The statistical PERT problem is covered in the paper of Nadas [3], and the theoretical complexity of the problem was established by Hagstrom [4]. Bounds on the project completion time were proposed by Kleindorfer [5] and Dodin [6].

In the context of integrated circuits, statistical timing methods may broadly be classified into performance-space methods that manipulate timing variables such as arrival times and slacks as statistical quantities, and parameter-space methods that perform manipulations in the space of the sources of variation. In the performance space, we are conceptually interested in integrating the joint probability density function (JPDF) of the delays of all paths over a cube of side equal to the required delay and of dimensionality equal to the number of paths. In other words, it amounts to the integration of a complicated JPDF over a simple integration region in high-dimensional space. In parameter space, on the other hand, we are interested in integrating the JPDF of the sources of parametric variation over a complex feasible region in relatively low-dimensional space.

Another broad classification of the statistical timing methods is to divide them into two categories: block-based methods and path-based methods. Block-based methods have linear complexity and are amenable to incremental processing, as noted by [2] and [7], while path-based methods are more accurate in that they better take into account the correlations due to reconvergent fan-out and spatial correlation.

Monte Carlo and modified Monte Carlo methods have often been used as in [8] where the yield is estimated by means of a surface integral of the feasible region. In the context of digital circuits, Gattiker *et al.* [9] consider the probability of each path meeting its timing requirement, but ignore correlations between paths. An extremely efficient discrete probability approach in the performance space was proposed in [10] and [11], but the path reconvergence is handled with difficulty and global correlations are ignored. A good source of information about the statistical design is in [12]. A recent performance-space probabilistic framework was proposed in [13] but has a restricted domain of application.

III. MOTIVATION

Unfortunately, most existing methods take into account one or other type of correlation mentioned in the previous section, but not both. An example of a work that takes both types of correlations into account is in [2]. Their modeling approach is similar to ours, but our algorithms are different. They also often neglect the dependence of slew (rise/fall time) and downstream load capacitance on the sources of variation. This paper proposes a unified framework to handle correlations due to path sharing as well as correlations due to the fact that the gates on the chip are affected by the same set of global parameters. It is crucial to take both types of correlations into account if we are to accurately predict yield. This paper builds on the ideas contained in [14].

Any methodology for statistical timing analysis must be able to handle different process conditions. It is possible that no single method will be able to accurately predict yield for all performances in all types of conditions. It is therefore desirable to develop a suite of methods, which can target different situations (low yield/high yield, few sources of global variation/many sources of global variation). This paper is an attempt to construct such a suite of methods. We propose two methods that operate in the space of manufacturing variations (parameter space) and one method that operates in the space of path delays. The three methods have complementary strengths and weaknesses as outlined below.

1) The first method proposed in this paper, the parallelepiped method, is best suited for a situation with a small number of sources of global variation. It provides a guaranteed lower bound on the true probability distribution curve of circuit delay and a "useful" upper bound on the true probability distribution curve. A "best-guess" estimate of the true curve can also be produced which in practice approximates the real curve fairly well.

2) The second of the methods proposed in this paper, the ellipsoid method, is less sensitive than the first method to the number of sources of variation and is highly effective at low yields. It also provides information that can be used to tune the circuit to improve yield. However, it cannot be directly used when there are many critical paths in the circuit. We propose a novel preprocessing step to reduce the number of paths that need to be considered.

3) The last method proposed in this paper is a performance-space method (which operates in the space of path arrival times) whose chief advantage is its extremely low time complexity. It is intended for use in situations where a quick estimate of yield is desired.

IV. MODELING

All three methods presented in this paper assume that the delay and slew (or rise/fall time) of each arc of the timing graph are linear functions of the sources of variation, similar to the assumptions in [9], [15], and [16], for example. However, the nominal delays and slews and the sensitivity coefficients can be location dependent to accommodate deterministic intrachip variability. The actual statistical timing analysis consists of two phases. In the first phase, a representative set of paths is gathered by the timing analysis program after a nominal timing analysis. The sensitivity coefficients of each "complete" path (including the launching and capturing clock paths if any) are computed and accumulated by path-tracing procedures. In the second phase, the statistical timing engine predicts the distribution of the minimum of all the path slacks. Path slack is defined as the difference between the required time and arrival time of the signal along the path. All methods work off of a common timing graph and path-tracing procedure.

The slack of each of P paths is modeled as

$$s_i = s_i^{\text{nom}} + \sum_{j=1}^{n} A_{ij}\Delta z_j \tag{1}$$

where s_i is the slack of the ith path (a statistical quantity), s_i^{nom} is the nominal slack, n is the number of global sources of variation, A_{ij} is a $P \times n$ matrix of path sensitivities, and δz_j is the variation of the jth global parameter from its nominal value. Delay, slew, and loading effects are taken into account in the coefficients of A in our implementation using the concept of chain ruling. The slew u can be expressed as $u = k_1 + k_2\Delta z$. Delay can be expressed as $d = k_3 + k_4 u + k_5\Delta z$. Substituting in this expression for the slew, we obtain $k_4 k_2 + k_5$ as the sensitivity coefficient of delay to the process variations Δz.

For a required slack ρ, we can write the following:

$$F = \left\{ \Delta z \middle| s_i^{\text{nom}} + \sum_{j=1}^{n} A_{ij}\Delta z_j \geq \rho, \qquad i = 1, 2, \cdots, P \right\}$$

$$F = \left\{ \Delta z \middle| d_i^{\text{nom}} + \sum_{j=1}^{n} -A_{ij}\Delta z_j \leq \eta_i, \qquad i = 1, 2, \cdots, P \right\}. \tag{2}$$

Here, d_i^{nom} is the nominal delay of the ith path and $\eta_i = \nu_i^{\text{nom}} - \rho$ where ν_i^{nom} is the nominal required time of the ith path. Each of the above P constraints represents a hyperplane in n-dimensional parameter space, on one side of which the path has sufficient slack and on the other side of which it is a failing path. The intersection of all the "good" half-spaces forms a convex polytope and is defined as the feasible region. The goal of the parameter-space methods is to integrate the JPDF of the
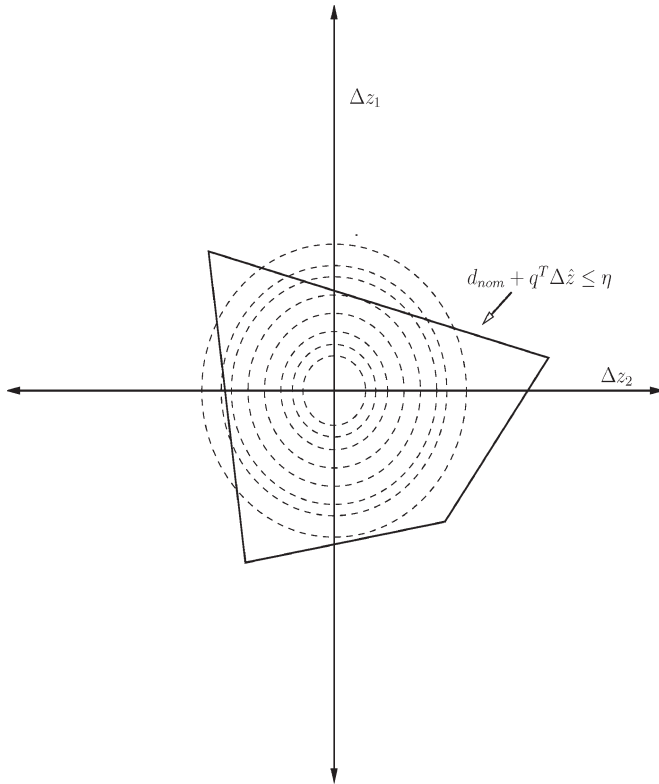
Fig. 1.   Feasible region defined by hyperplanes.

sources of variation in the feasible region. Mathematically, this integral can be expressed as

$$Y = \iint \cdots \int_F f(\Delta z_1, \Delta z_2, \ldots \Delta z_n) d\Delta z. \qquad (3)$$

This procedure is repeated for a range of η values to produce the entire slack versus the yield curve.

In Fig. 1, we show a feasible region in two dimensions. The dotted concentric circles represent a JPDF, which we wish to integrate over the feasible region. The value of the integral is the yield value for a particular slack value. When the slack value changes, the hyperplanes are shifted along their normal vectors to get a new feasible region.

The methods we propose in this paper do not directly handle spatial correlation. However, they can easily be extended to handle it. The method used in [2] is to divide the chip into subregions using a grid. Then, for any given process parameter, they assume different random variables for the individual grid buckets. A covariance matrix is imposed upon the random variables to describe the correlations between them. The same technique can be used in our case with an increase in the dimensionality of the problem due to the variables from the grid buckets. There would be no change in the linear model. However, for very fine grids, the number of variables can become very large. The parallelepiped and ellipsoid methods that we present below are both sensitive to the number of parameters, although the ellipsoid method is less so. In case of large dimensionality, we might first want to use principal component analysis to reduce the dimensionality. An alternative would be to use on-chip variation to handle the spatial correlation. In this method, sensitivity coefficients would be made location dependent. The method would be less accurate than the grid-based approach but would be much faster. In this paper, we treat the number of parameters to be a variable.

We shall begin by describing the most intuitive method among the three proposed in this paper, which is called the parallelepiped method. This method performs a "brute-force" integration of the JPDF of global sources of variation in the feasible region. The next method we present, the ellipsoid method, first approximates the feasible region by the maximum volume ellipsoid that can be inscribed in it, and then performs the integration over the ellipsoid. Both the parallelepiped and ellipsoid methods work in parameter space, i.e., the space of the sources of variation. The last method we present, the fastest among the three, is called the binding probability method, which is a performance-space method in that it works in the space of slacks of the paths.

V. PARALLELEPIPED METHOD

The basic idea of the parallelepiped method is to recursively divide the feasible region into the largest possible fully feasible parallelepipeds and integrate the JPDF of the underlying sources of variation over these parallelepipeds instead of the original feasible region. The approach does not require delays to have linear models, and allows for arbitrary distributions of the sources of variation. However, if the model is nonlinear, it must still be convex. Since slack is the difference between the required time and arrival time, it is difficult for slack to be convex even if both required time and arrival time are convex.

A. Algorithm

The basic reference on the parallelepiped approach is the second algorithm of Cohen and Hickey [17]. The method rests on the fact that if all vertices of an n-parallelepiped lie in any convex feasible region, then all points in the interior of the parallelepiped are feasible. With the above observation, the region of integration in the parameter space is recursively subdivided into progressively smaller parallelepipeds until we find parallelepipeds all of whose vertices are feasible. Then, we simply sum up the weighted volume of the feasible parallelepipeds to obtain a lower bound on the desired yield as shown in the pseudocode below for a single given performance requirement

```
procedure Vol (ll, recursionDepth){
   if (recursionDepth < maxDepth){
      if (all vertices of parallelepiped are feasible)
         add integral of region to yield;
      else{
         subdivide region into smaller parallelepipeds;
         for (each new parallelepiped p)
            Vol (lowerLeft (p), recursionDepth+1);
      }
   }
}
Vol (lowerLeft (boundingBox), 0).
```
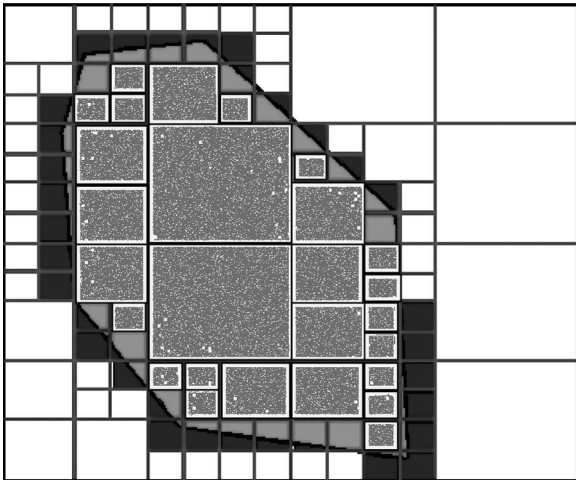
Fig. 2. Illustration of the parallelepiped method in two dimensions to a recursion depth of four. Light-gray regions contribute to lower bound of yield. For the best estimate, a portion of the weight of each parallelepiped on the boundary is also included.

The algorithm begins by choosing a boundingBox that is known to contain the feasible region. For statistical timing, the obvious choice is the $\pm 4\sigma$ or $\pm 3\sigma$ box in n-dimensions. In the algorithm, lowerLeft represents a function that returns the vertex of the parallelepiped that has the lowest coordinate in each dimension. Fig. 2 graphically illustrates the method in two dimensions. Gray regions contribute to the final yield computation and obviously provide a lower bound on the required probability integral. Note that descent to the lowest level of recursion is confined to the boundaries of the feasible region.

Since at worst, we visit every leaf node of a q-ary tree, where $q = 2^n$, and at each vertex, we check the feasibility of each path constraint, we end up with a worst case complexity of $O(Pn2^{(n \times \max \text{Depth})})$. Pn is the complexity of checking the feasibility of one vertex. In fact, if a static timer is employed, the feasibility of a vertex can be established more efficiently. In any case, the method is exponential in the product of the recursion depth and the dimensionality of the manufacturing space. However, several tricks can be applied to speed up this algorithm in practice.

1) If a particular path is infeasible at all vertices, the recursion can stop at once. No matter how deep the recursion, that particular path will not become feasible, so there is no good yield to be had.
2) If a particular path is feasible at all vertices, that path can be skipped as the recursion proceeds. This trick is implemented by simply maintaining a list of "skippable" paths that grows as the depth of the recursion increases.
3) The number of recursion levels can be drastically reduced by modifying the basic algorithm to additionally produce an upper bound and a best estimate answer. The (strict) lower bound is still the weighted volume of the gray region of Fig. 2. At the lowest level of recursion, if at least one vertex is feasible and at least one is infeasible, the upper bound gets the entire weighted volume of the

parallelepiped (represented by the black region in the figure). Although not a strict upper bound, in practice, this estimate always exceeds the exact yield. The "best estimate" result gets the yield credit proportional to the fraction of vertices that is feasible. With this mechanism, we have found that three to four levels of recursion are always sufficient for accurate results. The law of large numbers helps, since each parallelepiped at the lowest level of recursion contributes a signed error.

4) The parallelepiped method can handle any statistical distribution of the underlying sources of variation, provided that the JPDF can be integrated over the volume of a parallelepiped. If one or more sources of variation form a multivariate normal distribution, that part of the integral can be expressed as the product of differences of error functions in that subspace. The manufacturing space is first rotated and scaled so as to obtain circular symmetry. Then, the required error functions are precomputed and stored in a single array of size $2^n + 1$ to avoid repeated calls to the system erf function.

The following tricks will further improve the efficiency but have not yet been implemented.

1) Once a decision is made to recurse, only the internal vertices of the subparallelepipeds need to be visited, since the feasibility at the vertices of the parent parallelepiped has already been ascertained.
2) Since the bulk of the weighted volume is near the center of the JPDF, an adaptive grid scheme could be considered which uses a finer grid near the origin of the Δz space and a progressively coarser grid toward the boundary of the bounding box.
3) Recursion can be carried out by subdividing the parallelepiped in one dimension at a time, and if a path is infeasible at all vertices, for example, subdivision in the other dimensions is obviated.

B. Modified Algorithm

The above algorithm has been adapted to compute the entire yield-versus-slack curve at once instead of one performance point at a time. As each parallelepiped is processed, the contributions of the parallelepiped toward the yield for all slack values are simultaneously recorded before proceeding to the next parallelepiped or next level of recursion. All CPU time results in this paper use this modified method.

The basic idea is briefly explained here. Let $s_{\min}^{\text{parent}}$ be the smallest slack at any of the vertices of the parent parallelepiped. Then, for all slack below $s_{\min}^{\text{parent}}$, the entire parent parallelepiped is in the feasible region, and appropriate yield credit is given. As we recurse, we are only interested in slacks greater than $s_{\min}^{\text{parent}}$ within this volume. For each subparallelepiped at the present level of recursion, the yield credit corresponding to the smaller parallelepiped is granted for all slack from $s_{\min}^{\text{parent}}$ to $s_{\min}$ (lowest slack among the vertices of the subparallelepiped). The upper bound and best guess yields are similarly kept updated as the recursion proceeds.

Thus, the entire yield curve is produced by a single recursive loop. The modified algorithm computes identical results compared to the repeated invocation of the basic algorithm presented earlier. Modified versions of all of the tricks mentioned in conjunction with the basic algorithm continue to be applicable, and have been implemented in our prototype.

VI. MAXIMUM VOLUME ELLIPSOID METHOD

The basic idea is to compute the maximum volume n-dimensional ellipsoid that is entirely within the feasible region (a similar idea was explored in [18]), and integrate the JPDF of the sources of variation in the simpler ellipsoidal approximation rather than the original feasible region. The integral provides a lower bound on the yield. This method relies on a linear delay model but allows arbitrary distributions of the underlying sources of variation.

A. Ellipsoid Computation

There has been tremendous recent progress in solving the maximum volume ellipsoid problem. It is shown in [19] that ellipsoidal volume maximization subject to linear constraints is a special case of the MAXDET problem in which the determinant of a matrix is maximized and subject to linear matrix inequalities (LMIs). We start with a set of linear inequality constraints that defines a feasible region (2), and which can be expressed in matrix form as

$$F = \left\{ \Delta z \mid R_i^{\mathrm{T}} \Delta z \le t_i, \qquad i = 1, 2, \ldots P \right\} \qquad (4)$$

where R_i^{T} is the ith row of $-A$ and $t_i = s_i^{\mathrm{nom}} - \eta$. An arbitrary ellipsoid is expressed as a collection of points

$$E = \{ By + d \mid \|y\| \le 1 \} \qquad (5)$$

where B is a symmetric positive-definite matrix that linearly transforms all points in the unit sphere and d is the center of the ellipsoid. To find the largest ellipsoid in the feasible region, it is sufficient to maximize the determinant of the transformation matrix B since the volume of the ellipsoid is the volume of the unit sphere times the determinant of B. The requirement that $E \subset F$ means that

$$R_i^{\mathrm{T}}(By + d) \le t_i \quad \text{for all} \quad \|y\| \le 1, \qquad i = 1, 2, \ldots, P. \qquad (6)$$

This in turn means that

$$\sup_{\|y\| \le 1} \left(R_i^{\mathrm{T}} By + R_i^{\mathrm{T}} d \right) \le t_i, \qquad i = 1, 2, \ldots, P \qquad (7)$$

or

$$\|BR_i\| + R_i^{\mathrm{T}} d \le t_i, \qquad i = 1, 2, \ldots, P. \qquad (8)$$
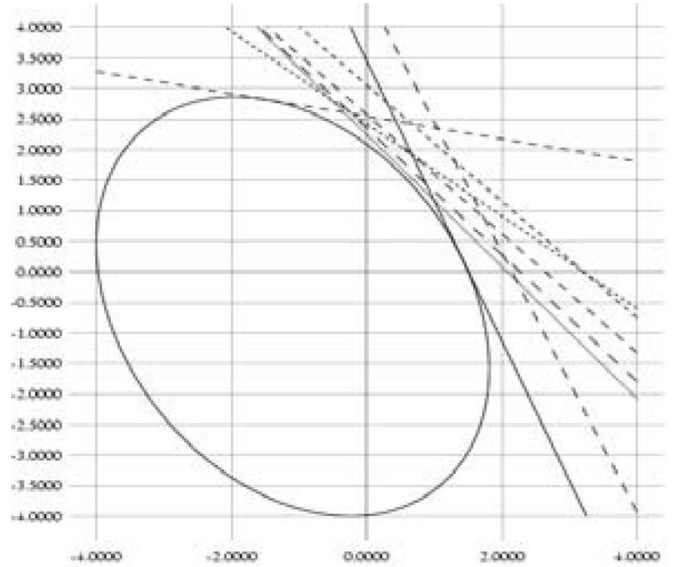

Fig. 3. Largest ellipse in two dimensions bounded by hyperplane constraints and the 4σ box.

To find the ellipsoid of largest volume inside the feasible region F, we solve the convex optimization problem

$$
\begin{aligned}
\text{maximize} \quad & \log \det B \\
\text{subject to} \quad & B = B^{\mathrm{T}} > 0 \\
\text{subject to} \quad & \|BR_i\| + R_i^{\mathrm{T}} d \le t_i, \qquad i = 1, 2, \ldots, P. \qquad (9)
\end{aligned}
$$

Zhang and Gao [20] recently proposed an efficient structure-exploiting primal-dual optimization algorithm to solve (9), and have made available public-domain MATLAB code. By manipulating the optimality conditions and taking advantage of the properties of transformation matrices of ellipsoids, this method solves for fewer variables, can handle problems of larger dimensionality, and is vastly more efficient than the original implementation in [19]. In addition to the linear slack constraints, bounding box constraints (e.g., the $\pm 4\sigma$ box) are applied. An example of the ellipsoid computed by the program in two dimensions is shown in Fig. 3. The lines in this figure represent the paths in the system, and the ellipsoid is the largest ellipsoid that can be inscribed in the feasible region defined by the lines (hyperplanes) and the 4σ bounding box.

B. Path Filtering

We must address the path explosion problem that will doubtlessly occur in large circuits. There are many millions of paths leading from latch to latch in even moderately sized circuits, and even if we were to look at only "critical" paths—or those with a delay, large enough to impact the delay of the whole circuit—we may arrive at several thousand paths. The algorithm to find the maximum volume ellipsoid is at least cubic in the number of paths [20]. Therefore, we will benefit by reducing the number of paths as much as possible.

For real circuits, there are many paths that have very similar characteristics, owing to a large amount of path sharing, as
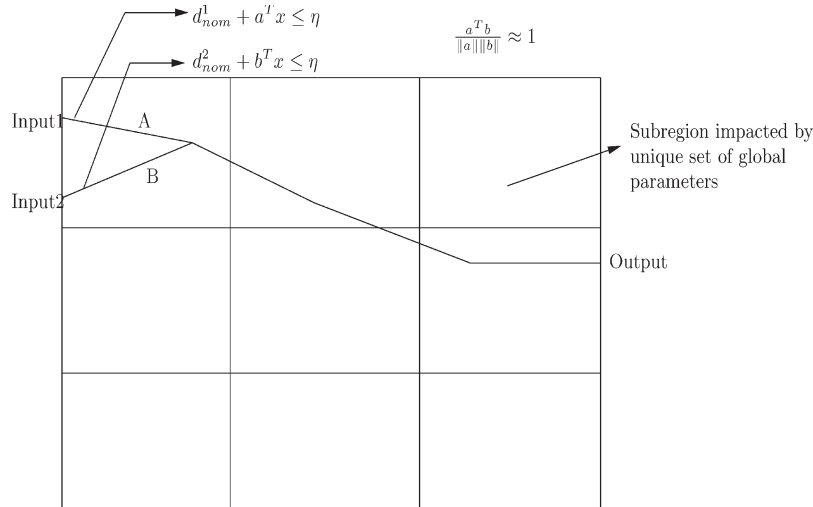
Fig. 4. Paths A and B with path vectors a and b share almost all their gates. Therefore, the angle between them is almost zero.

illustrated in Fig. 4. Paths marked A and B in Fig. 4 depend on the same set of parameters, and must also have nearly the same coefficients since they share so many gates. This suggests that we can club these two paths together, and thus reduce the number of paths. Thus, we can augment the path selection process so that, paths are selected not only according to high nominal delay (delay criterion) or low nominal slack (slack criterion). The other measure we suggest is checking to see if a candidate path vector is oriented at an angle not represented by any other paths already in the list (angle criterion). The idea behind both of these criteria is to select paths that, in some extreme circumstances, become limiting to circuit performance. In order to take care of the angle criterion, we can devise a path selection procedure that, at each step, selects a path whose sensitivity vector makes the largest possible angle with all the paths in the selected setup to that point. This path would represent a behavior not represented by any of the other paths in the set of paths selected up to that point. The cosine of the angle that a path with path vector a makes with a path b can be expressed as

$$\cos\gamma = \frac{a^{\mathrm{T}}b}{\|a\|\|b\|}. \tag{10}$$

Having taken care of the angle criterion, we can then turn our attention to selecting paths according to the slack criterion. For all paths whose path vectors point in nearly the same direction, we can pick a path which has the highest delay and is therefore most likely to be critical. The cosine of the angle between two path vectors is the same as the correlation coefficient between the paths. This allows us to provide a statistical interpretation to the "angle" and "delay" criteria: the angle criterion tries to pick the least correlated set of directions, and the delay criterion attempts to pick a path along a given direction that is most critical. Thus, the path-filtering procedure can be seen to approximate the polyhedron in a manner so as to retain most of the information contained in the original polyhedron.

Let us express the preceding ideas in terms of an algorithm which can be viewed as a preprocessing step that takes a large number of paths and produces a user-defined smaller number of "representative" paths. The basic algorithm can be expressed

as follows, where NumDirections is the user-defined number of "representative" directions

```
SelectRepresentativeDirections{
   for (i = 0; i < TotalNumberOfPathsInList; i++){
      CurrentMaximumCosineArray[i] = -1.0;
   }
   DirectionSet[0] = SensitivityMatrix[0];
   j = 0;
   while (j < NumDirections){
      GlobalMinimumCosine = 1.0;
      for (i = 0; i < TotalNumberOfPathsInList; i++){
         CurrentCosine = CosineValue(SensitivityMatrix[i],
            DirectionSet[j]);
         if (CurrentCosine > CurrentMaximum-
            CosineArray[i]){
            CurrentMaximumCosineArray[i] =
               CurrentCosine;
         }
         if (CurrentMaximumCosineArray[i]<=
            GlobalMinimumCosine){
            GlobalMinimumCosine = CurrentMaximum-
               CosineArray[i];
            PathIndex = i;
         }
      }
      j++;
      DirectionSet [j] = SensitivityMatrix[PathIndex];
   }
}.
```

Next, we will describe how to select the most critical of all paths whose sensitivity vectors point in nearly the same direction.

When the last direction is selected in the pseudocode above, the direction makes an angle of at least $\cos^{-1}$(Global-MinimumCosine) with all the other selected directions. Further, none of the paths outside the selected set makes an angle greater than $\cos^{-1}$(GlobalMinimumCosine) with the paths in the
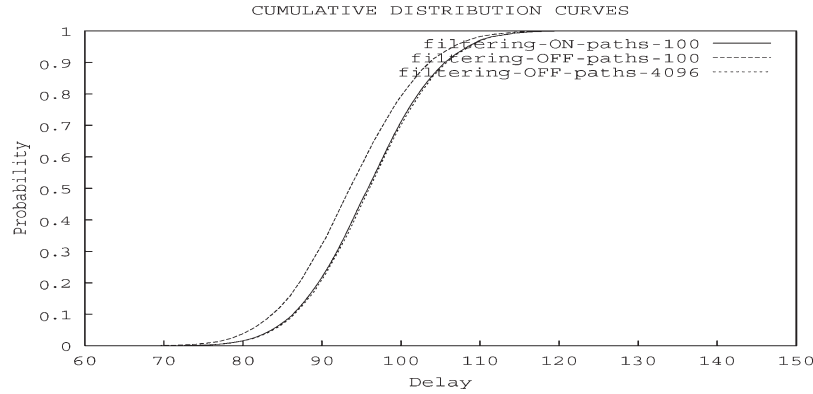
Fig. 5. Cumulative distribution curves obtained by considering the top 100 nominally critical paths, 100 critical paths obtained by using path filtering, and considering all paths in the circuit. In this case, the curve obtained by considering 100 paths obtained through path filtering nearly coincides with the real curve considering all 4096 paths.

selected set. Therefore, any given path outside the selected set lies within a "cone" of $\cos^{-1}$(GlobalMinimumCosine) of some selected path. It remains to define the feasible region boundary in each cone, i.e., select the path among all those lying in the cone that actually forms the boundary (this will also be the critical path in the cone). This can be done by computing the distances of the paths from the origin. When the origin is on the feasible side of a hyperplane, the distance is positive, and when it is on the infeasible side of a hyperplane, the distance is negative. Let us consider two paths with normal vectors a_i and a_j, such that

$$a_i^{\mathrm{T}} x \le b_i$$
$$a_j^{\mathrm{T}} x \le b_j \qquad (11)$$

are the hyperplane equations corresponding to the two paths. Then, the hyperplane selected in a cone is the one that has the minimum distance with respect to the origin. The algorithm below computes an array FinalPathFilteredSet of size equal to the number of paths in the original system, where the ith entry is one if that path is selected by the algorithm:

```
FindFeasibleRegionBoundary{
    Eta = RequiredPerformance;
    for (i = 0; i < SizeOfDirectionSet; i++){
        MinimumDistanceSoFar = Infinity;
        for (j = 0; j < TotalNumberOfPathsInList; j++){
            Direction = CosineValue(DirectionSet[i],
                SensitivityMatrix[j]);
            if (Direction >= GlobalMinimumCosine){
                Distance = (Eta - NominalDelay[j])/
                    Length(SensitivityMatrix[j]);
                if (Distance <= MinimumDistanceSoFar){
                    MinimumDistanceSoFar = Distance;
                    CurrentMinIndex = j;
                }
            }
        }
    }
    FinalPathFilteredSet [CurrentMinIndex] = 1;
    }
}.
```

The above discussion is intended for the case of computing the yield for a single performance. As the performance values change, the angles between the hyperplanes do not change since the hyperplanes continue to have the same normal vectors. However, the distances of the hyperplanes from the origin will change, and in any given cone, a different hyperplane may form the boundary of the feasible region as the required performance value changes. Therefore, it is necessary to run FindFeasibleRegionBoundary once for each performance, but it is sufficient to run SelectRepresentativeDirections only once. The results of applying the angle criterion to filter paths are shown in Fig. 5. Three curves are shown in this figure. The cumulative distribution curve, assuming the polyhedron is approximated by the first 100 paths in order of decreasing nominal delay, is shown as "filtering-OFF-paths-100." The cumulative distribution curve obtained by selecting 100 paths according to the path-filtering procedures outlined above is shown as "filtering-ON-paths-100." This curve corresponds fairly closely to the real cumulative distribution curve obtained by taking all 4096 paths in the system showing that path filtering is effective in approximating the feasible region. Our main purpose, however, is to approximate the feasible region, such that the maximum volume ellipsoid inscribed in the approximated feasible region is not much different from the maximum volume ellipsoid inscribed in the feasible region described by all paths. This purpose is served by the way in which we perform path filtering: all important directions are accounted for ensuring that the ellipsoid is "boxed" in all sides, and is therefore fairly representative of the real ellipsoid.

C. Numerical Integration

There remains the problem of integrating the JPDF of the sources of variation over the resulting ellipsoidal approximation of the feasible region. The yield is represented as

$$\iint \cdots \int_{R^*} \mathrm{JPDF}(z_1, z_2, \ldots, z_n) dz_n dz_{n-1} \ldots dz_1 \qquad (12)$$

where R^* is the ellipsoidal region. Instead of integrating over the ellipsoid, we perform a change of variables and integrate

over the unit sphere. This is because of the existence of numerical integration algorithms to perform integration over the unit sphere. Also

$$\delta z_1 \delta z_2 \ldots \delta z_n = \begin{vmatrix} \frac{\partial z_1}{\partial y_1} & \frac{\partial z_1}{\partial y_2} & \cdots & \frac{\partial z_1}{\partial y_n} \\ \frac{\partial z_2}{\partial y_1} & \frac{\partial z_2}{\partial y_2} & \cdots & \frac{\partial z_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_n}{\partial y_1} & \frac{\partial z_n}{\partial y_2} & \cdots & \frac{\partial z_n}{\partial y_n} \end{vmatrix} \delta y_1 \delta y_2 \ldots \delta y_n$$

$$= |B| \delta y_1 \delta y_2 \ldots \delta y_n \qquad (13)$$

where we make use of the fact that the Jacobian matrix is simply the transformation matrix B which is computed by the convex optimization problem discussed previously. Therefore, the yield Y can be written as

$$Y = \iint \cdots \int_{R^*} \text{JPDF}(z) dz$$

$$= \iint \cdots \int_{R} \text{JPDF}(By + d)|B| dy \qquad (14)$$

where R is the unit sphere.

The basic numerical integration rules for a sphere are those of [21]. An integration rule for a domain of integration consists of a weighted sum of functional values evaluated at specific points. The points and weights are the characteristics of the rule. Integration rules are designed so that they exactly integrate certain easy classes of functions, for example, polynomials of low degree. An integration rule can be expressed as follows:

$$\iint \cdots \int f(y_1, y_2, \ldots y_n) dy_n \ldots dy_1$$

$$= \sum_{i=1}^{i=N} w_i f(v_{i1}, v_{i2}, \ldots v_{in}). \qquad (15)$$

Here, the w_is represent the weights of the integration rule and $(v_{i1}, v_{i2}, \ldots v_{in})$ represents the ith point of the integration rule. N is the number of points in the integration rule. Numerically stable integration formulas are those whose points are within the region of integration and whose weights are positive. A degree-3 formula for integration over the unit sphere is the following:

$$Y = \sum_{i=1}^{i=n} \frac{V}{2n} (f(e_i) + f(-e_i)) \qquad (16)$$

where e_i is the ith unit vector and V is the volume of the unit sphere in n-dimensions. After extensive experimentation, we found that a degree-3 or degree-5 rule was insufficient to model the variation of a Gaussian integrand over the feasible region. At low yields, when the (small) feasible region is located far away from the mean of the JPDF, a degree-3 or degree-5 formula is sufficient to model the variation of the Gaussian integrand over the feasible region. At higher yields (larger feasible regions located closer to the mean of the JPDF), a degree-3 or degree-5 formula is incapable of modeling the

entire variation as both high yield regions and low yield regions are included in the feasible region.

Higher degree formulas for integration over the unit sphere exist but they suffer from either one or both of two drawbacks: 1) they require an exponential number of points or 2) some of their coefficients are negative. The drawbacks impact the stability of the formulas and/or runtime. We address this situation by reformulating the integral over the unit sphere as a spherical-radial integral. The radial integral is a one-dimensional integral and is computed by a Gaussian integration formula. The spherical integral is an integral of the JPDF integrand over the surface of the unit sphere. This transformation has the effect of reducing the variation of the original integrand, since the variation over the surface of the unit sphere is considerably less than over the entire sphere, especially for integrals obeying some form of radial symmetry. Below, we provide the mathematical details of the following formulation [21].

Let R_n be a bounded n-dimensional sphere which contains the origin ϕ and, let Y_n be the surface of R_n. Let us assume that we have an integration formula for a surface integral [21]

$$\iint \cdots \int_{Y_n} f(y_1, y_2 \ldots y_n) d\sigma = \sum_{j=1}^{j=N} B_j f(v_{j,1}, v_{j,2} \ldots v_{j,n}). \qquad (17)$$

The B_js are the weights of the integration formula, and $(v_{j,1}, v_{j,2} \ldots v_{j,n})$ is the jth point of the integration formula. Note that in the above equation, $d\sigma$ is a differential element on the surface of the unit sphere. We would like the points of this formula to lie on the surface Y_n. For a real number $r > 0$, let

$$rY_n = \{r\nu : \nu \in Y_n\}. \qquad (18)$$

Let us consider the integration of a monomial term over the surface of a sphere of radius r. Let $d\sigma'$ represent a differential element of area on the surface of the sphere of radius r. By expressing both $d\sigma$ and $d\sigma'$ in polar coordinates, we can see that the following holds true:

$$\frac{d\sigma'}{d\sigma} = r^{n-1}. \qquad (19)$$

Then, we can write the integral of a monomial term over the surface of a sphere of radius r as follows:

$$\iint \cdots \int_{rY_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma'$$

$$= \iint \cdots \int_{Y_n} r^{n-1} (ry_1)^{\alpha_1} (ry_2)^{\alpha_2} \ldots (ry_n)^{\alpha_n} d\sigma$$

$$= r^{n-1+\alpha} \iint \cdots \int_{Y_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma \qquad (20)$$

where $\alpha = \Sigma_{i=1}^n \alpha_i$.

The integral of a monomial over the entire unit sphere R_n can then be written as

$$
\iint \cdots \int_{R_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} dy_1 dy_2 \ldots dy_n
$$

$$
= \int_0^1 \left(\iint \cdots \int_{rY_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma' \right) dr
$$

$$
= \int_0^1 r^{n-1+\alpha} dr \iint \cdots \int_{Y_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma. \quad (21)
$$

Suppose we have the following one-dimensional integration formula of degree d for the radial integral:

$$
\int_0^1 r^{n-1} f(r) dr \approx \Sigma_{i=1}^{M} A_i f(r_i). \quad (22)
$$

Then, the points $r_i \nu_j$ and the coefficients A_i, $B_j i = 1, 2 \ldots M, j = 1, 2 \ldots N_0$ are an integration formula of degree d for R_n.

It is possible to use one additional trick to deal with the (still) substantial variation of the spherical integrand over the surface of the unit sphere. The general high accuracy of numerical methods and the dimensionality-independence of Monte Carlo methods can be combined to good effect in the form of randomized quadrature rules. The fundamental basis of these rules is the observation that if we were to rotate the points of a given integration rule, then we get another integration rule of the same degree. In other words, let the following be an integration rule of degree k for the surface of the unit sphere:

$$
I(f) = \sum_{i=1}^{i=N} w_j f(v_j). \quad (23)
$$

Then, the following is also an integration rule of the same degree:

$$
I(f) = \sum_{i=1}^{i=N} w_j f(Q v_j). \quad (24)
$$

In (23) and (24), we have denoted the points in vector form. Q is an orthogonal transformation.

The reformulation of the yield integral requires us to calculate a spherical integral over the surface of the unit sphere (in $n-1$ dimensions) and a radial integral in one dimension. Now, we investigate the techniques to calculate the spherical surface integral. Let us denote the spherical surface integral by $I(f)$. A degree-3 formula to compute the surface integral is the following:

$$
I(f) \approx \left(\frac{V}{2n} \right) \sum_{i=1}^{i=n} (f(e_i) + f(-e_i)) \quad (25)
$$

where e_i is the unit vector along the ith coordinate direction and V is the surface area of the unit sphere in n dimensions.

Another degree-3 formula that is slightly more expensive than the previous one is the following [22], [23]:

$$
I(f) \approx \left(\frac{V}{2n} \right) \sum_{i=1}^{i=n+1} (f(u_i) + f(-u_i)). \quad (26)
$$

The $n+1$ points u_i are the vertices of a regular n-simplex with the vertices located on the surface of the unit sphere. This rule can be extended to a degree-5 rule in a natural way

$$
I(f) \approx V \left(\left(\frac{(7-n)n}{2(m+1)^2(m+2)} \right) \sum_{i=1}^{i=n+1} (f(u_i) + f(-u_i)) \right.
$$

$$
+ \left(\frac{2(m-1)^2}{m(m+1)^2(m+2)} \right)
$$

$$
\left. \sum_{i=1}^{m(m+1)/2} (f(v_i) + f(-v_i)) \right). \quad (27)
$$

The v_is are obtained by taking the midpoints of the edges of the n-simplex whose vertices are on the unit sphere (and given by the u_is) and projecting them onto the surface of the unit sphere. The total number of points in the formula is $(n+1)(n+2)$.

In our implementation, the integral in (14) is computed by a modification of the stochastic integration method of Genz and Monahan [24]. The integration is split up into the product of a radial and spherical part as shown above. The spherical part is accomplished by applying Mysovskikh's rules [22], [23] to a series of spherical surfaces (or infinitesimal annulus shells), and the radial part is computed by Gaussian quadrature. Spherical integration is performed by randomizing deterministic integration rules to obtain a higher degree of accuracy than conventional Monte Carlo integration. Integration is performed by applying a degree-5 rule with $(n+1)(n+2)$ points, giving the method a polynomial complexity. The basic set of points is randomly rotated to get a new set of points. The quality of the radial integration can be improved by increasing the number of spherical surfaces, whereas the quality of the spherical integration can be improved by increasing the number of points sampled on each spherical surface. Randomizing the spherical-radial integration formula (randomized quadrature) allows us to make use of the Monte Carlo error theory to study the variance of the integral estimate.

D. Efficiency of Randomized Quadrature

Integrating over the ellipsoid provides us with a lower bound on the true yield since the ellipsoid does not cover the corners of the feasible region. Therefore, it might seem that approximating the feasible region by an ellipsoid and integrating over it offer no particular advantage. However, the randomized quadrature process outlined in the previous section has considerably lower variance than ordinary Monte Carlo routines such as Gaussian sampling or uniform sampling over the ellipsoid. In Fig. 6, we show the cumulative distribution curves obtained by using the randomized quadrature over the ellipsoid ("ran-quad" in the
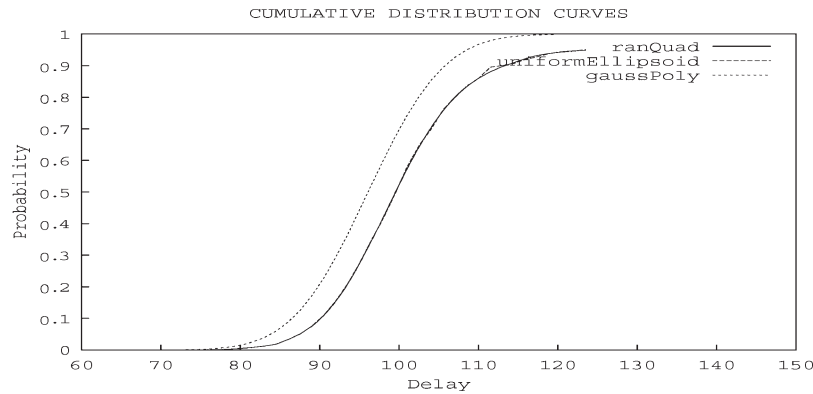
Fig. 6. Cumulative distribution curves obtained by using the Gaussian sampling, randomized quadrature over the sphere, and uniform sampling over the ellipsoid.
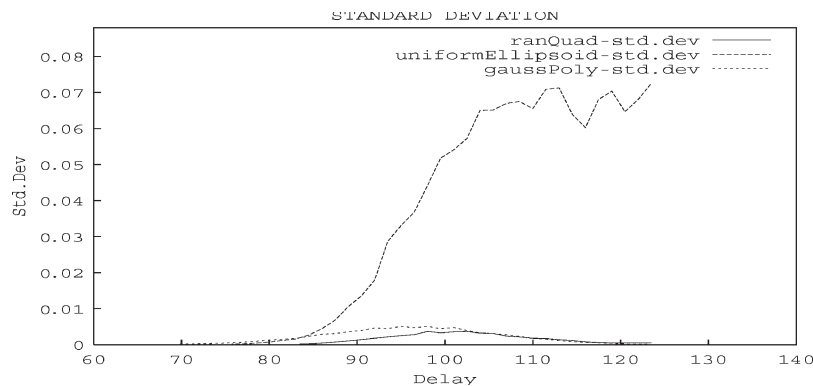

Fig. 7. Standard deviation of the Gaussian sampling, randomized quadrature, and uniform sampling over the ellipsoid. The randomized-quadrature procedure has the lowest standard deviation at low values of yield.
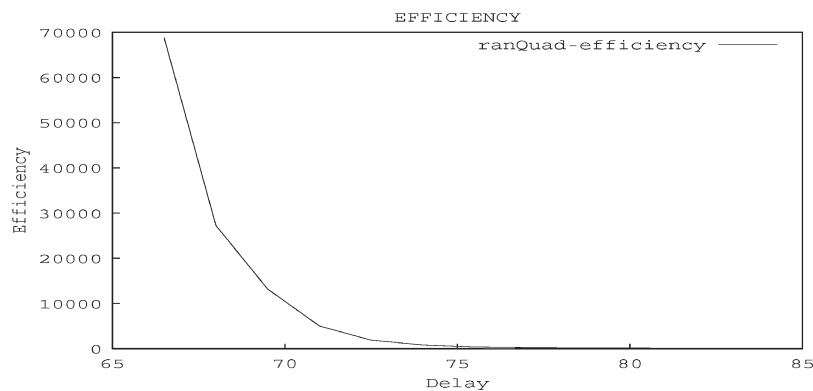

Fig. 8. Efficiency of the Gaussian sampling and randomized quadrature. The randomized-quadrature procedure is the orders of magnitude more efficient than Gaussian sampling at low values of yield.

figure), using uniform sampling over the ellipsoid ("uniform-ellipsoid") and the curve obtained by performing Gaussian sampling over the whole polytope ("gaussPoly"), which represents the real yield. As can clearly be seen, the integrals over the ellipsoid are lower bounds to the real yield curve. In Fig. 7, we show the standard deviation σ of the three integration procedures. As can clearly be seen, using the randomized quadrature provides the lowest standard deviation of the three methods.

Following the study in [25], let us define the efficiency

$$\epsilon = \frac{\sigma_1^2 \tau_1}{\sigma_2^2 \tau_2}. \qquad (28)$$

Here, τ_1 and τ_2 represent the computation times for N trials of Gaussian sampling over the yield polytope and performing randomized quadrature, and σ_1 and σ_2 represent the standard deviations of these two sampling procedures, respectively. The logic behind the above equation is simple: if a low standard deviation procedure has a high average time per trial value, then it may become equivalent to a high standard deviation procedure having a low average time/trial. In other words, one can afford to run a high standard deviation procedure for the extra trials needed to make up for the gap in accuracy without incurring any increase in computation time, because of the low time per trial value. The efficiency of using the randomized quadrature over the ellipsoid as compared to Gaussian sampling over the whole polytope is shown in Fig. 8. This figure clearly

brings out the quality of the randomized-quadrature procedure. For low yields, it is the order of magnitudes that is more efficient than Gaussian sampling (uniform sampling over the ellipsoid is simply not competitive). However, the yield value returned by exact integration over the ellipsoid is a lower bound on the true yield value. For the fastest performances, when yield is very low, it pays to use the ellipsoid method to get a quick lower bound on the true yield.

VII. Binding Probability Method

This section describes the last of the three novel statistical timing algorithms called the binding probability method, which is a performance-space method. The basic idea of this method is to compute the probability distribution of the minimum slack of the first two nominally most critical paths. Then, a recursion is set up to find the distribution of the minimum of this distribution and the third most critical path, and so on. The difficult part, of course, is to keep the correlations alive as the recursion proceeds. This method rests on both the linear delay model as well as requires the underlying parameter distributions to be Gaussian.

A. Computing pdf and Binding Probability

The slack of every path is a linear combination of the Gaussian sources of variation, and hence is Gaussian. The algorithm begins by taking the two nominally most critical paths and computing their 2×2 covariance matrix

$$\Phi = \begin{bmatrix} \left(\frac{\partial s_1}{\partial z}\right)^{\mathrm{T}} \\ \left(\frac{\partial s_2}{\partial z}\right)^{\mathrm{T}} \end{bmatrix} [V] \begin{bmatrix} \frac{\partial s_1}{\partial z} & \frac{\partial s_2}{\partial z} \end{bmatrix} = \begin{bmatrix} A_1^{\mathrm{T}} \\ A_2^{\mathrm{T}} \end{bmatrix} [V][A_1 A_2] \quad (29)$$

where s_1 is the slack of the first path, s_2 is the slack of the second path, A_i^{T} is the ith row of A, zs are the sources of variation, and V is the $n \times n$ covariance matrix of the sources of variation. Comparing to

$$\Phi = \begin{bmatrix} \sigma_1^2 & \sigma_1 \sigma_2 \rho \\ \sigma_1 \sigma_2 \rho & \sigma_2^2 \end{bmatrix} \quad (30)$$

the variances σ_1, σ_2 and the correlation coefficient ρ can easily be computed. Next, the distribution of $\min(s_1, s_2)$ is computed analytically using formulas from [26] after first rewriting $\min(s_1, s_2)$ as $-\max(-s_1, -s_2)$. These formulas express the mean and variance of the random variable corresponding to the minimum of two random variables A and B in terms of the so-called tightness probabilities. The tightness probability T_A of A with respect to B is the probability that A is larger than B. The term tightness probability has also been called binding probability in previous literature. Let A and B be the Gaussian random variables with means a_0 and b_0, respectively. Define $\theta = (\sigma_A^2 + \sigma_B^2 - 2\rho\sigma_A\sigma_B)^{1/2}$. Then, the tightness probability of A with respect to B can be written as

$$T_A = \Phi\left(\frac{a_0 - b_0}{\theta}\right) \quad (31)$$

where $\phi(t) = (1/\sqrt{2\pi})e^{-0.5t^2}$ and $\Phi(y) = \int_{-\infty}^{y} \phi(t)dt$. Having thus computed the tightness probabilities, we can then write

$$E\left[\max(A, B)\right] = a_0 T_A + b_0(1 - T_A) + \theta\phi\left[\frac{a_0 - b_0}{\theta}\right] \quad (32)$$

and

$$\mathrm{Var}\left(\max(A, B)\right) = \left(\sigma_A^2 + a_0^2\right) T_A + \left(\sigma_B^2 + b_0^2\right)(1 - T_A)$$
$$+ (a_0 + b_0)\theta\phi\left(\frac{a_0 - b_0}{\theta}\right) - E\left[\max(A, B)\right]^2. \quad (33)$$

B. Recursion

The next step is to create a fictitious path that captures the correlations of all the paths processed so far, with all the paths to be considered in the future. Let b_1 denote the tightness probability of path 1 with respect to path 2 and $b_2 = 1 - b_1$ the tightness probability of path 2 with respect to path 1. The fictitious path consists of a linear combination of all the timing graph edges along path 1 with probability b_1 and all the gates along path 2 with probability b_2; in other words, a vector consisting of a linear combination of the slack sensitivities of the two paths is created, to be plugged back into (29) for the covariance computation at the next step of recursion. If one or the other path is always dominating (binding probability of unity), its sensitivities are preserved as is. The distribution obtained above is then approximated to be Gaussian. The algorithm proceeds by finding the probability distribution of the minimum of this fictitious path slack and the third most critical path, and so on, until the probability distribution changes by a sufficiently small tolerance.

At the end of this procedure, the binding probabilities accumulated along the way give us the probability that any given path is critical, and in fact, the probability that any given branch of the timing graph is critical. Such diagnostics can be used to guide a yield-aware optimization.

VIII. Implementation

All three methods presented in this paper have been implemented in C++ as a prototype component called EinsStat in the EinsTimer static timing analysis environment. In the case of the ellipsoid method, the maximum volume ellipsoid MATLAB code of Zhang and Gao [20] was converted via the MATLAB compiler to C++.

The "front-end" collects the worst paths and computes their nominal slacks and sensitivities directly off the EinsTimer timing graph. Currently, a user-specified number of worst paths are collected throughout the entire design (which could easily be extended to collect all paths within a user-specified slack window). For each path, a corresponding downstream (setup) test slack is computed—this implicitly takes into account the most critical clock path leading to the test. "Full paths" are traced from the clock source, through the launching latch, through the data path, to the capturing latch, and back to the clock source. This way, common clock path correlations are fully captured.
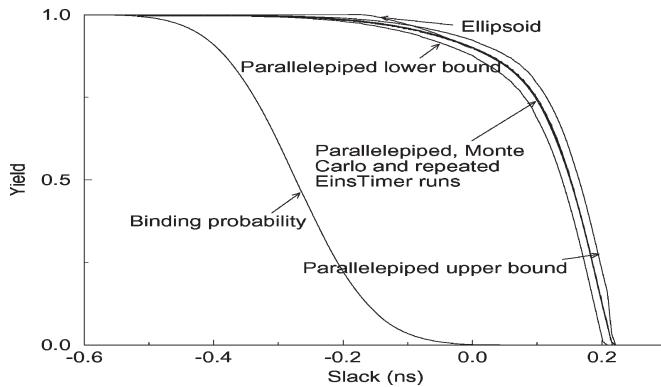
Fig. 9.   Statistical timing results on 200K gate ASIC.

TABLE I
CPU TIMES ON 200 K GATE ASIC

Method	1000 paths	15,000 paths
Repeated EinsTimer runs	68 hours	
Monte Carlo 1M samples	60 s	855 s
Parallelepipeds	20 s	141 s
Binding probability	20 s	152 s
Ellipsoid	3.41 hours	Out of memory

Sensitivities with respect to global environmental conditions are determined by finite differences as our analytical delay models [27] are characterized as functions of voltage and temperature. In addition, the underlying delay calculations fully account for input slew and downstream pin capacitance dependencies on the sources of variation.

IX. NUMERICAL RESULTS

Two flavors of numerical results will be presented in this section. The first is a set of results obtained from statistical analysis of a real-world 200K gate ASIC design with environmental variations, while leaving manufacturing parameters at their "slow chip" setting. The second set of results is from running artificially generated problems with a large number of nominally equally critical paths and random sensitivities.

A 200K gate ASIC circuit was first analyzed with individual temperature and voltage variations, leading to a surprising result. The EinsTimer best-case result was the worst slack of all, and the nominal result was the best slack! Further investigation revealed that this chip has a short primary-input-to-latch path, whose slack deteriorates rapidly with lower temperature and higher V_{dd}, because the clock is disproportionately sped up. At higher temperatures and lower V_{dd} values, latch-to-latch paths with more traditional slack sensitivities dominate. This type of surprising result is easily unearthed with statistical analysis.

Fig. 9 shows statistical timing results on the 200K gate ASIC circuit with simultaneous temperature and voltage variations. Even though temperature and voltage variations are mostly deterministic, we use these two parameters to illustrate our methodology in an industrial setting. Our methodology could easily be applied to real statistical variations such as those in channel length and thickness of the oxide. The Y-axis represents yield and the X-axis represents slack. Superposed on the same plot are the results obtained by running EinsTimer 1200 times at a regular grid of sample temperature/V_{dd} pairs and converting the sample points to probabilities. All methods are pretty accurate, except the binding probability method that is unable to accurately capture the highly skewed slack distribution in this case. CPU times on an IBM Risc/System 6000 model 43P-S85 are shown in Table I. The ellipsoid method takes too much memory and too much time because of its highly nonlinear dependence on the number of paths. This shows that we must use the path filtering as a preprocessing step

before we use the ellipsoid method. Although the results are shown here with only two sources of environmental variation, the anticipated applications of these methods are to solve the problem of timing circuits with multiple voltage islands and to take manufacturing variations into account.

The second set of results is from randomly generated problems with a large number of nominally equally critical paths. Fig. 10(a) shows the growth in CPU time as a function of the number of paths analyzed, with the number of sources of variation fixed at 4 and 100 points requested on the slack curve. The ellipsoid method has polynomial complexity in the number of paths, while the others are linear. Fig. 10(b) shows the growth in CPU time as a function of the number of points requested on the yield curve (number of variations fixed at 4, number of paths at 1000). With the exception of the ellipsoid method, all the methods are insensitive to the first order to the number of data points requested. Finally, Fig. 10(c) shows the growth of the CPU time with the number of sources of variation (paths fixed at 1000 and data points at 100). To first order, the Monte Carlo and binding probability methods are unaffected by the number of parameters, whereas the ellipsoid method has polynomial dependence and the parallelepiped method has exponential dependence which dominates the run time above six dimensions. The use of the path filtering as discussed in the section on path filtering can reduce the computational burden of finding the ellipsoid. Notice that once we have found a nearly optimal ellipsoid (using path filtering to reduce the number of constraints) that does not completely fit inside the original polytope, we can always shrink the ellipsoid to make it fit inside the true polytope. Then, our numerical integration method on the ellipsoid will continue to give a low-variance lower bound on true yield, which would be invaluable especially at low yields.

X. CRITICALITY INFORMATION FROM THE METHODS

There are two notions of criticality that we discuss in this section. The first relates to asking the following question for each path: what percentage of the time does the path in question contribute to the worst (minimum) slack? From the point of view of the yield optimization across all slacks, it is important to identify the paths which are critical most of the time, as these are the paths which are most yield limiting. This kind of information is easy to obtain using Monte Carlo simulation. Given N sample points in the process parameter space, we can count the number of times a given path contributed to worst slack and take its criticality as this number divided by N. The second flavor of the parallelepiped method essentially
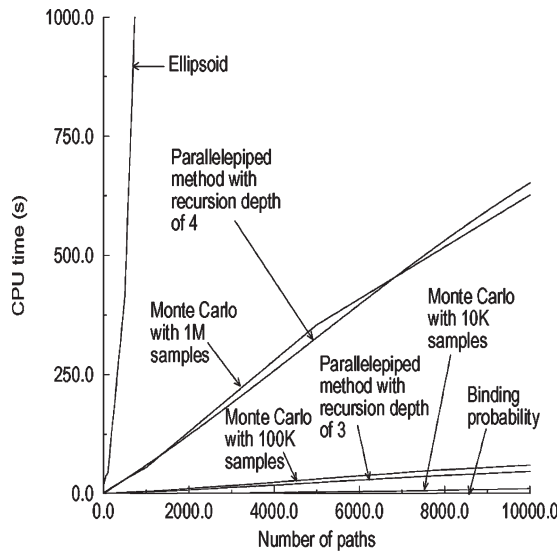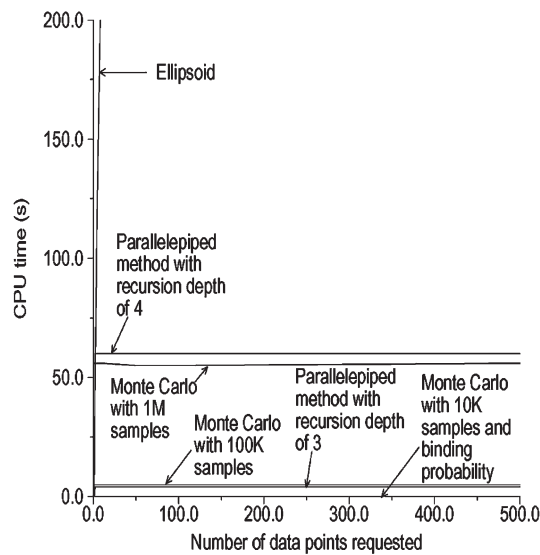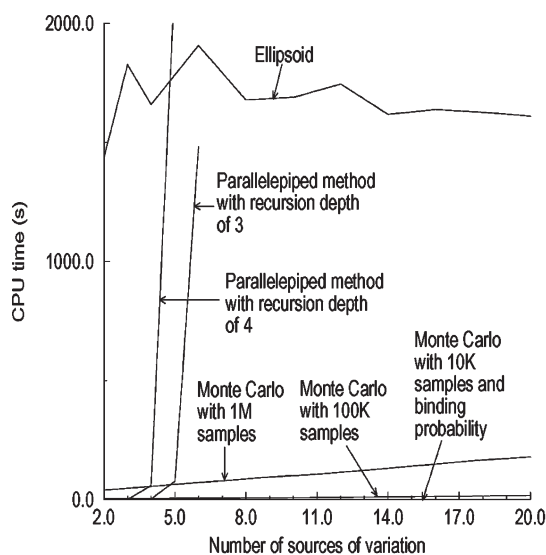
Fig. 10. (a) CPU time versus of paths. (b) CPU time versus the number of data points requested. (c) CPU time versus the number of sources of variation.

TABLE II
GLOBAL CRITICALITY OF TOP FIVE PATHS

Path No.	Nominal Slack (ps)	Parallepiped Global Criticality	Monte-Carlo Global Criticality
1	-49	0.229	0.234
2	-49	0.128	0.131
3	-49	0.088	0.090
4	-49	0.000	0.0000
5	-49	0.160	0.165

searches through the process parameter space exhaustively, and vertices of the parallelepipeds discovered along the way can be used to determine the path criticality. Whenever we encounter a parallelepiped at the lowest level of recursion, we keep count $count(i)$ for each path of the number of vertices of the parallelepiped where the given path is the worst path in terms of slack. Then, for each path, we augment its global criticality measure by a fraction of the yield integral for that parallelepiped, where the fraction is given by $count(i)/2^n$. The justification for this step is that for a small enough parallelepiped, the fraction $count(i)/2^n$ approximates the fraction of the time the given path is the worst slack path when points are sampled within the parallelepiped. In Table II below, we show how this method compares with the traditional Monte Carlo simulation. The experiment was performed for five parameters $(n = 5)$. The table below shows that the method compares well with Monte Carlo simulation.

It is possible to define another notion of criticality that provides an information relevant to optimizing yield for a given slack. In order to determine the criticality of a given path at a given slack, we essentially determine the change in the volume of the yield polytope as we move the given path slightly. In principle, Monte Carlo simulation can be used to determine this measure of criticality for each path, but it is not likely to be very accurate, as the sample points must be generated close to the boundary of the polytope.

We shall show below how both the first flavor of the parallelepiped method as well as the ellipsoid method can be used to obtain the criticality of any given path at a particular slack value. In the parallelepiped method, at the lowest level of recursion, we can determine for each parallelepiped for which paths the parallelepiped is infeasible. For each parallelepiped that is on the boundary, we determine the list of paths that violates some vertex of the parallelepiped. If for some parallelepiped there is exactly one path which is infeasible for that parallelepiped, we attribute the yield mass of that parallelepiped to the path in question. The logic behind this step is that if we were to slightly perturb a path that is infeasible for a parallelepiped, then it would become feasible. Note that if there are two or more paths which are infeasible for a given parallelepiped, we would not be justified in attributing the yield mass of the parallelepiped to any of the violating paths, because shifting only one of them will not make the parallelepiped feasible. The accuracy of this method depends upon the size of the smallest parallelepiped used in the recursion. When the recursion is completed, not only does the yield value for the given slack value become known, but also the incremental weighted volume for each path as it is slightly perturbed. We

Path No.	Nominal Slack (ps)	Parallelepiped Criticality	Monte-Carlo Performance Criticality
1	-49	0.015	0.018
2	-49	0.0004	0.001
3	-49	0.0000	0.0000
4	-49	0.0000	0.0000
5	-49	0.0000	0.0000

have shown in Table III the criticality results computed for the top five paths using the first version of the parallelepiped method and the Monte Carlo method.

Since the parallelepiped method becomes infeasible at higher dimensions, we cannot use the method above to determine the criticality for each path at a given performance. We show how the maximum volume ellipsoid can be used to obtain information regarding the criticality of each path at a given performance.

The main idea here is to note that the center of the maximum volume ellipsoid approximates the center of the polytope and the ellipsoid shape approximates the shape of the polytope. In order to increase the weighted volume of the polytope, we may try a variety of schemes such as moving the ellipsoid center closer to the center of the JPDF or expanding the ellipsoid along one or more axes to increase the weighted volume of the ellipsoid. Having expanded the ellipsoid and/or shifted it remains to shift the hyperplanes of the polytope to accommodate the new ellipsoid. By this, we mean shifting each hyperplane along the direction of its normal so as to make it a tangent to the new ellipsoid. Not all hyperplanes will need to move; only those which intersect the expanded/shifted ellipsoid. Also, the hyperplanes that block yield-rich regions of the feasible space will have to move more than hyperplanes which border yield-poor regions. The amount of movement for each hyperplane can easily be translated into a change in the nominal delay of the path corresponding to that hyperplane. We illustrate the procedure to obtain a delay information for each hyperplane.

The feasible region of circuit operation could be expressed as

$$F = \left\{ \Delta z | R_i^{\mathrm{T}} \Delta z \le t_i, \qquad i = 1, 2, \ldots P \right\} \quad (34)$$

where the ith hyperplane is given by

$$R_i^{\mathrm{T}} \Delta z \le t_i. \quad (35)$$

Replacing Δz with $By + d$, where B is the ellipsoid transformation matrix, we get the inequality

$$R_i^{\mathrm{T}} By \le t_i - R_i^{\mathrm{T}} d. \quad (36)$$

In the y-space, the above equation represents a hyperplane whose distance from the origin is given by

$$D = \frac{t_i - R_i^{\mathrm{T}} d}{\| R_i^{\mathrm{T}} B \|}. \quad (37)$$

Method	Nominal	0.95 * Nominal	1.05 * Nominal
Ellipsoid	0.010	0.0009	0.014
Top-path speedup	0.001	0.000028	0.001
Top 30 paths speedup	0.002	0.0002	0.003

When the ellipsoid expands and/or moves, the distance D gets affected. Suppose the ellipsoid transformation matrix B changes by an amount ΔB and the center of the ellipsoid moves by an amount Δd. Then, the new distance becomes

$$D_{\mathrm{new}} = \frac{t_i - R_i^{\mathrm{T}}(d + \Delta d)}{\| R_i^{\mathrm{T}}(B + \Delta B) \|}. \quad (38)$$

If $D_{\mathrm{new}} < 1$, then we can calculate an amount Δt_i to be subtracted from the nominal delay d_i^{nom} of path i, so that the path becomes a tangent to the changed ellipsoid. We have

$$\Delta t_i = \| R_i^{\mathrm{T}}(B + \Delta B) \|(1 - D_{\mathrm{new}}). \quad (39)$$

After trying out various approaches of shifting the ellipsoid center in conjunction with expanding the axes of the ellipsoid, we found that the simple scheme of expanding the smallest axis of the ellipsoid while retaining the same center works quite well in increasing the volume of the polytope. Increasing the length of the smallest axis of the ellipsoid provides the largest increase in the volume of the ellipsoid as compared to increasing the length of any other axis. We performed our experiments at several different values of required slack. In each case, we obtained tuning factors for each path describing how much each path had to move in order to become tangential to the expanded ellipsoid. It turned out that among paths that were of the same nominal slack, some were more critical to improving yield at a given performance. Let $S = \Sigma \Delta t_i$ denote the sum of all the speedups of the paths calculated by the above procedure for a given required slack. One other scheme to increase the volume of the polytope is to dedicate the total speedup S to the nominally most critical path. Still another scheme is to share the speedup among the top K paths by speeding up each path by an amount S/K. We show in Table IV the yield increases obtained when the tuning factors are applied as calculated by the procedure above, and we compare them with the yield increases obtained by applying the other two schemes. For the three performances of interest (nominal performance and 5% above and below, where nominal performance is the performance that the circuit was designed for), the ellipsoid method of expanding the smallest axis of the ellipsoid and then speeding up paths as necessary to accommodate the expanded ellipsoid is found to provide a larger increase in yield as compared to the other methods of distributing the total speedup. This suggests that the nominal criticality of a path alone does not determine how important the path is in increasing the weighted volume of the feasible region. Instead, the path's orientation with respect to the feasible region determines its importance.

It is important to emphasize that the tuning factors in themselves do not provide a means of increasing yield, as changing the nominal delay of a particular path may affect other paths, and it may not be possible to speed up all the paths as required

TABLE V
TUNING FACTORS VERSUS MONTE CARLO PERFORMANCE CRITICALITY

Path No.	Nominal Slack (ps)	Tuning Factors	Monte-Carlo performance Criticality
1	-49	0.014	0.018
2	-49	0.207	0.001
3	-49	0.000	0.000
4	-49	0.033	0.000
5	-49	0.029	0.000

by their tuning factors. Nevertheless, the tuning factors may provide a valuable guidance to a statistical yield optimization tool as with regard to the relative importance of the paths to the increasing yield. Table V shows the tuning factors for the top five paths in the design used for this section.

One interesting and useful feature of the ellipsoid algorithm is that it provides nonzero tuning factors even for paths which do not bound the feasible region, but are completely dominated by another path, so that when this other path is speeded-up slightly, the overshadowed path becomes critical. This phenomenon is reflected in the results shown above. Paths 4 and 5 are noncritical at the nominal performance (Table III). In fact, path 4 (Table II) is noncritical at all performances. However, this is because, these paths are overshadowed by other critical paths. The tuning factors for these paths are shown to be fairly high in Table V, suggesting that they are important from the point of view of increasing the volume of the polytope. To see if paths 4 and 5 are truly important, we first applied all the tuning factors to obtain path speedups and calculated the yield increase at the nominal performance. The yield increase was found to be 0.01. Next, we speeded-up all the paths according to their tuning factors except for path 4. The resulting change in the volume of the polytope was found to be only 0.006. A similar phenomenon could be observed with respect to path 5. This shows that although these paths are not actually critical at the performance of interest or perhaps even at any performance, they could still have an impact on polytope volume. Methods such as the parallelepiped method or even the Monte Carlo method would not be able to identify such paths. However, the ellipsoid method would be able to identify such paths. When the tuning factor for a path is 0.0, we cannot in general conclude that the path is not important for yield optimization; speeding-up the path may still increase the volume of the yield polytope but not by very much.

XI. COMPARISONS

The parallelepiped method is very fast and accurate at low dimensionality, but it has an exponential growth of CPU time with the number of sources of variation. The CPU time is independent to the first order of the number of points requested on the yield curve and linear with the number of paths selected. Hence, it is best suited to be accurate but at low-dimensionality analysis.

The ellipsoid method, on the other hand, handles high-dimensionality extremely well and successfully handled problems with over 20 sources of variation. However, it has a linear growth with the number of points requested on the

slack curve and polynomial growth with the number of paths. Thus, it is most effective when the dimensionality is high and the number of paths can be filtered down to a manageable number.

The binding probability method is extremely fast and consistently outperforms all the other methods. It is also the least accurate of the methods proposed, both because of the Gaussian approximation and because of the loss of accuracy in propagating correlations. The complexity is linear in the product of the number of paths and the number of data points requested on the yield curve, but is relatively insensitive to the number of sources of variation.

The three methods therefore provide a complementary arsenal of techniques depending on the situation at hand.

XII. CONCLUSION AND FUTURE WORK

This paper presented three algorithms for statistical timing analysis that pay a great deal of attention to the inherent correlation between the delays of gates and paths on a chip. Each method has strengths and weaknesses, and by implementing all three in a common infrastructure and with a common interface, the best features of each method can be exploited as the situation demands. Results of the statistical timing analysis on a 200K gate ASIC were presented.

There are several avenues of future work. Several measures to improve efficiency were suggested in the body of this manuscript. Various diagnostics can be inferred from these methods too. For example, in the ellipsoid method, the major and minor axes of the ellipsoid tell us the least and most important directions in which to nudge the circuit for improved parametric yield and the most important manufacturing parameters on which to improve the control if possible. The binding probability method gives us a rank-ordered set of gates, the improvement of whose delays will have the most impact on improving the yield. The extreme efficiency of the binding probability method is motivating some new researches into handling skewed distributions in this method. Extending the method to compute yield gradients will enable automated yield-aware optimization. The center of the ellipsoid can be taken to be the point at which the process parameters should be centered ideally.

One avenue of future work is to apply the path-filtering algorithm as a preprocessing step for both the parallelepiped as well as the binding probability method. Using the path filtering might help in speeding up the determination of point infeasibility. Applying the binding probability method to paths that are in the same "direction cone" might give better results than applying it to paths that point in very different directions.

One avenue of future work is to incorporate the spatial correlation along the lines of [2] and [28]. Two other extensions are intriguing. The first is to compute the so-called Löwner–John ellipsoid, which is the smallest ellipsoid that circumscribes the feasible region, so as to obtain an upper bound on the yield. The second is to obtain a simplicial decomposition of the feasible region (see [29] for an excellent survey) and then to integrate the JPDF of the sources of variation over the resulting simplices.

Finally, the statistical intrachip variation can be accommodated in a number of ways. One technique is to have a position-dependent random variable, upon which the delays of all gates depend. Another is to divide the chip into regions, with each region having a common set of random variables. The variables of nearby regions are tightly correlated, while those that are far apart are only loosely correlated.

REFERENCES

[1] "International technology roadmap for semiconductors 2001 edition," Semiconductor Industry Association, 2001. Tech. Rep. [Online]. Available: http://public.itrs.net/Files/2001ITRS/Home.htm

[2] H. Chang and S. Sapatnekar, "Statistical timing analysis considering spatial correlation in a pert-like traversal," in *Proc. IEEE Int. Conf. Comput.-Aided Design Integr. Circuits and Syst.*, Nov. 2003, pp. 621–625.

[3] A. Nadas, "Probabilistic PERT," *IBM J. Res. Develop.*, vol. 23, no. 3, pp. 339–347, May 1979.

[4] J. Hagstrom, "Computational complexity of pert problems," *Networks*, vol. 18, pp. 139–147, 1988.

[5] G. Kleindorfer, "Bounding distributions for stochastic acyclic networks," *Oper. Res.*, vol. 19, no. 5, pp. 1586–1601, 1971.

[6] B. Dodin, "Bounding the project completion time in pert networks," *Oper. Res.*, vol. 33, no. 4, pp. 862–881, 1985.

[7] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First order incremental block-based statistical timing analysis," in *Proc. Des. Autom. Conf.*, Jun. 2004, pp. 331–336.

[8] P. Feldmann and S. W. Director, "Integrated circuit quality optimization using surface integrals," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 12, pp. 1868–1879, Dec. 1993.

[9] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, "Timing yield estimation from static timing analysis," in *Proc. IEEE Int. Symp. Quality Electron. Des.*, 2001, pp. 437–442.

[10] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Proc. Des. Autom. Conf.*, Jun. 2001, pp. 661–666.

[11] S. Naidu, "Timing yield calculation using an impulse-train approach," in *Proc. VLSI Design*, Jan. 2002, pp. 219–224.

[12] S. W. Director and W. Maly, Eds. *Statistical Approach to VLSI*, vol. 8. North-Holland, The Netherlands: Elsevier, 1994

[13] M. Orshansky and K. Keutzer, "A general probabilistic framework for worst case timing analysis," in *Proc. Des. Autom. Conf.*, Jun. 2002, pp. 556–561.

[14] J. A. G. Jess, K. Kalafala, S. R. Naidu, C. Visweswariah, and R. H. J. M. Otten, "Statistical timing for parametric yield prediction of digital integrated circuits," in *Proc. Des. Autom. Conf.*, Anaheim, CA, Jun. 2003, pp. 932–937.

[15] D. E. Hocevar, P. F. Cox, and P. Yang, "Parametric yield optimization for MOS circuit blocks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 7, no. 6, pp. 645–658, Jun. 1988.

[16] K. Krishna and S. W. Director, "The linearized performance penalty (LPP) method for optimization of parametric yield and its reliability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 12, pp. 1557–1568, Dec. 1995.

[17] J. Cohen and T. Hickey, "Two algorithms for determining volumes of convex polyhedra," *J. ACM*, vol. 26, no. 3, pp. 401–414, Jul. 1979.

[18] J. M. Wojciechowski and J. Vlach, "Ellipsoidal method for design centering and yield estimation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 10, pp. 1570–1579, Oct. 1993.

[19] L. Vandenberghe, S. Boyd, and S.-P. Wuin "Determinant maximization with linear matrix inequality constraints," Inf. Syst. Lab. Electr. Eng. Dept., Stanford Univ., Stanford, CA, Apr. 1996. Tech. Rep. [Online]. Available: http://www.stanford.edu/~boyd/maxdet.html

[20] Y. Zhang and L. Gao, "On numerical solution of the maximum volume ellipsoid problem," Dept. Comput. Appl. Math., Rice Univ., Houston, TX, Tech. Rep. CAAM Tech. Rep. TR01-15, Aug. 2001.

[21] A. H. Stroud, *Approximate Calculation of Multiple Integrals*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[22] I. P. Mysovskikh, "The approximation of multiple integrals by using interpolatory cubature formulae," in *Qualitative Approximation*, R. A. DeVore and K. Scherer, Eds. New York: Academic, 1980, pp. 217–243.

[23] ——, *Interpolatory Cubature Formulas*. Moscow-Leningrad: Nauka, 1981. Izd, Russian text.

[24] A. Genz and J. Monahan, "A stochastic algorithm for high-dimensional multiple integrals over unbounded regions with gaussian weight," *J. Comput. Appl. Math.*, vol. 112, no. 1, pp. 71–81, Nov. 1999.

[25] J. Hammersley and D. Handscomb, *Monte Carlo Methods*. London, U.K.: Metheun and Co. Limited, 1964.

[26] C. E. Clark, "The greatest of a finite set of random variables," in *Oper. Res.*, vol. 9, Mar.–Apr. 1961, pp. 145–162.

[27] *IEEE Standard for Integrated Circuit (IC) Delay and Power Calculation System*, pp. 1–390, 1999. IEEE Standard 1481-1999. [Online]. Available: http://fp.ieeexplore.ieee.org/iel5/6837/18380/00846710.pdf?isNumber=18380&prod=standards&arnumber=00846710

[28] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical timing analysis for intra-die process variations considering spatial correlations," in *Proc. IEEE Int. Conf. Compu.-Aided Design Integr. Circuits and Syst.*, Nov. 2003, pp. 900–907.

[29] B. Büeler, A. Enge, and K. Fukuda, "Exact volume computation for polytopes: A practical study," in *Polytopes—Combinatorics and Computation*, vol. 29, G. Kalai and G. M. Ziegler, Eds. Basel, Germany: Birkhäuser Verlag, 2000. [Online]. Available: http://www.lix.polytechnique.fr/Labo/Andreas.Enge/Volumen_en.html

Jochen A. G. Jess (A'97) received the Masters and Ph.D. degrees in electrical engineering from Aachen University of Technology, Germany, in 1961 and 1963, respectively.

In 1971, he was a Full Professor with tenure to establish and run the Design Automation Section of Eindhoven University of Technology, The Netherlands. In almost 30 years with Eindhoven, he and his group contributed to almost any relevant subject in the area of VLSI chip design. Until his retirement from the university in the year 2000, the group published almost 380 papers. Under his supervision, 45 students graduated with their Ph.D. degrees. He is one of the founders of the "Design Automation and Test in Europe" (DATE) Conference. After his retirement, he started as a Consultant with IBM Thomas J. Watson Research Center and the Embedded Systems Institute of Eindhoven University. Currently, he is a Consultant with Philips Research at Eindhoven, The Netherlands. In addition, he teaches courses for professionals in chip architecture for the Center of Technical Training (CTT) of Philips and for the Embedded Systems Institute (ESI) of Eindhoven University.

Dr. Jess received the EDAA "Lifetime Achievement Award" in 2005.

Kerim Kalafala received the M.Sc. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY.

Since 1998, he has been with the IBM Electronic Design Automation Group, IBM Microelectronics Division, East Fishkill, Hopewell Junction, NY, where he is currently a Senior Software Engineer. He has developed various timing analysis algorithms that are in use at IBM.

Mr. Kalafala, along with several very talented colleagues and coauthors, has received two IBM Best Paper Awards, a Best Paper Award at DAC 2004, as well as Electronic Design News Innovator and Innovation of the Year honors.

Srinath R. Naidu received the B.Tech. degree (with Honors) in computer science and engineering from the Institute of Technology, Banaras Hindu University, Varanasi, India, in May 1996, the M.Sc.Eng. degree in computer science from the Indian Institute of Science, Bangalore, India, in 1998, and the Ph.D. degree in the area of statistical timing analysis for digital integrated circuits from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2004.

Since January 2004, he has been with Magma Design Automation, Bangalore, India. His main research interests are in the broad areas of statistical timing, power optimization, and logic synthesis.

Dr. Naidu was a corecipient of the IBM Pat Goldberg Memorial Best Paper Award in the fields of electrical engineering, computer science, and mathematics, in 2003.

Ralph H. J. M. Otten received the M.S. and Ph.D. degrees from Eindhoven University of Technology, Eindhoven, The Netherlands, in 1971 and 1976, respectively. His M.S. and Ph.D. theses were on the decomposition of automata and the automatic layout for analog integrated circuits, respectively.

Until 1981, he was an Associate Professor at Eindhoven University of Technology, where he taught courses on computer-aided design of electronic circuits and algorithms. His research during those years was concentrated on automatic layout design, particularly on floorplanning. From 1981 to 1987, he was a research staff member with the Mathematical Sciences Department, Thomas J. Watson Research Center, IBM Corporation. In 1985, he became the Manager of the Yorktown Silicon Compiler Project. From 1987 to 2000, he was a Professor with the Electrical Engineering Department, Delft University of Technology. In 1997–1998, he held the McKay Chair with the University of California, Berkeley. At present, he is a Full Professor in electronic systems and the Curriculum Director for the Faculty of Electrical Engineering, Eindhoven University of Technology. His main interests are in the implementation of large electronic circuits, particularly in the automation of that task. The goal is to provide a scientific basis for modern design trajectories that unify pre- and postsilicon designs.
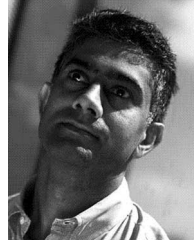
Chandu Visweswariah (S'84–M'84–SM'95–F'05) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Chennai, India, in 1985, and the M.S. and Ph.D. degrees in computer engineering from Carnegie Mellon University in Pittsburgh, PA, in 1986 and 1989, respectively.

In 2002, he was a Visiting Assistant Professor with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. He has been a research staff member with IBM's Thomas J. Watson Research Center in Yorktown Heights, NY, ever since, and currently manages a Circuit and Interconnect Analysis Group. He has developed various circuit simulations, circuit optimization, and timing software tools that are in production use with IBM. His research interests include modeling, analysis, timing, optimization, and manufacturability of integrated circuits. He has authored and coauthored one book and over 50 technical papers. He holds five U.S. patents with 12 more in the pipeline.

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. In December 2005, he was selected as an Innovator profiled by EE Times in its "Great Minds, Great Ideas" project and special issue.