Statistical Timing for Parametric Yield Prediction of Digital Integrated Circuits

J. A. G. Jess[†], K. Kalafala[‡], S. R. Naidu[†1], R. H. J. M. Otten[†], C. Visweswariah[*]

[*] Senior Member, IEEE

[†]Eindhoven University of Technology [‡]IBM Microelectronics Division IBM Thomas J. Watson Research Center

Eindhoven, The Netherlands East Fishkill, NY, USA Yorktown Heights, NY, USA

1

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.

[1]Corresponding author, is now with Magma Design Automation Pvt Ltd, Bangalore, India. This work was performed while he was a graduate student at Eindhoven University of Technology, The Netherlands. He may be contacted at srinath@magma-da.com.

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 (ITRS) [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 key to 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. While the present work does not directly handle statistical intra-chip variations, the last section of the paper describes how these could be accommodated in the future.

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 PERT where the objective was to calculate the probability distribution curve of the project completion time given that the sub-tasks 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 a paper of Nadas [2], and the theoretical complexity of the problem was established by Hagstrom [3]. Bounds on the project completion time were proposed by Kleindorfer [4] and Dodin [5].

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 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 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 [6] and [7] while path-based methods are more accurate in that they better take into account correlations due to reconvergent fanout and spatial correlation.

Monte Carlo and modified Monte Carlo methods have often been used as in [8] where yield is estimated by means of a surface integral of the feasible region. In the context of digital circuits, [9] considers the probability of each path meeting its timing requirement, but ignores

correlations between paths. An extremely efficient discrete probability approach in performance-space was proposed in [10], [11], but path reconvergence is handled with difficulty and global correlations are ignored. A good source of information about statistical design is [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. 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 correlation 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 desirable, therefore, 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 wekanesses 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 pre-processing 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 [15], [9], [16], for example. However, the nominal delays and slews and the sensitivity coefficients can be location-dependent to accommodate deterministic intra-chip 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

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

where s_i is the slack of the i^{th} 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 j^{th} global parameter from its nominal value. Delay, slew and loading effects are taken into account in the coefficients of A in our implementation usin 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 η, we can write the following:

$$F = \{\Delta z | s_i^{nom} + \sum_{j=1}^{n} A_{ij} \Delta z_j \geq \eta, i = 1, 2, \cdots, P\}. \tag{2}$$

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 sources of variation in the feasible region. This procedure is repeated for a range of η values to produce the entire slack vs. yield curve.

We shall begin by describing the most intuitive method of the three proposed in this paper 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 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 of the three, called the binding probability method, 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 non-linear, it must still be convex. Since slack is the difference between required time and arrival time, it is difficult for slack to be convex even if both required time and arrival time are convex.

A. The 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 pseudo-code 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);
```

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. 1 graphically illustrates the method in two dimensions. The yellow (in color copies of this paper) or grey (in black-and-white copies) 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 feasibility of each path constraint, we end up with a worst-case complexity of $O(Pn2^{(n \times \texttt{maxDepth})})$. 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

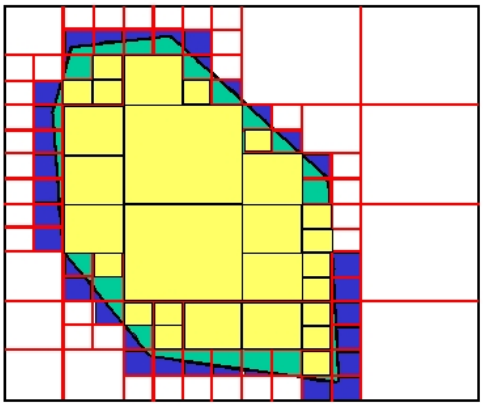

Fig. 1. Illustration of the parallelepiped method in 2 dimensions to a recursion depth of 4.

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 yellow/grey region of Fig. 1. At the lowest level of recursion, if at least one vertex is feasible and at least one infeasible, the upper bound gets the entire weighted volume of the parallelepiped (represented by the blue/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 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 the JPDF can be integrated over the volume of a parallelepiped. If one or more sources of variation form a multi-variate normal distribution, that part of the integral can be expressed as the product of differences of error functions in that sub-space. 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 efficiency, but have not yet been implemented.

1) Once a decision is made to recurse, only the *internal* vertices of the sub-parallelepipeds need to be visited, since 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 towards 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-vs.-slack curve at once instead of one performance point at a time. As each parallelepiped is processed, the contributions of the parallelepiped towards 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}^{parent} be the smallest slack at any of the vertices of the *parent* parallelepiped. Then for all slack below s_{min}^{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}^{parent} within this volume. For each sub-parallelepiped at the present level of recursion, yield credit corresponding to the smaller parallelepiped is granted for all slack from s_{min}^{parent} to s_{min} (lowest slack amongst the vertices of the sub-parallelepiped). 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 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 subject to linear matrix inequalities (LMIs). We start with a set of linear inequality constraints that define a feasible region (2), and which can be expressed in matrix form as

$$F = \{\Delta z \mid R_i^T \Delta z \leq t_i, i = 1, 2, \cdots P\}, \tag{3}$$

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

$$E = \{By + d \mid \|y\| \leq 1\}, \tag{4}$$

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

$$R_i^T(By + d) \leq t_i \text{ for all } \|y\| \leq 1, i = 1, 2, \cdots, P. \tag{5}$$

This in turn means that

$$sup_{\|y\| \leq 1}(R_i^T By + R_i^T d) \leq t_i, i = 1, 2, \cdots, P \tag{6}$$

or

$$\|BR_i\| + R_i^T d \leq t_i, i = 1, 2, \cdots, P. \tag{7}$$

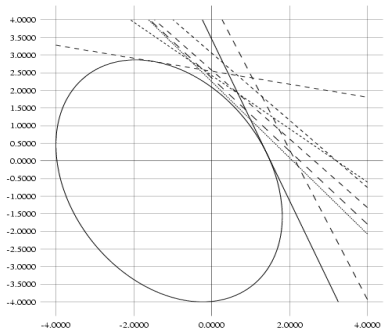

Fig. 2. Largest ellipse in 2 dimensions bounded by hyperplane constraints and the 4σ box.

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

$$\begin{aligned} &\text{maximize} && \log\det B \\ &\text{subject to} && B = B^T > 0 \\ &\text{subject to} && \|BR_i\| + R_i^T d \leq t_i, i = 1, 2, \cdots, P. \end{aligned} \tag{8}$$

Zhang and Gao [20] recently proposed an efficient, structure-exploiting primal-dual optimization algorithm to solve (8), 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 of [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 2 dimensions is shown in Fig. 2. 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 doubtless 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.

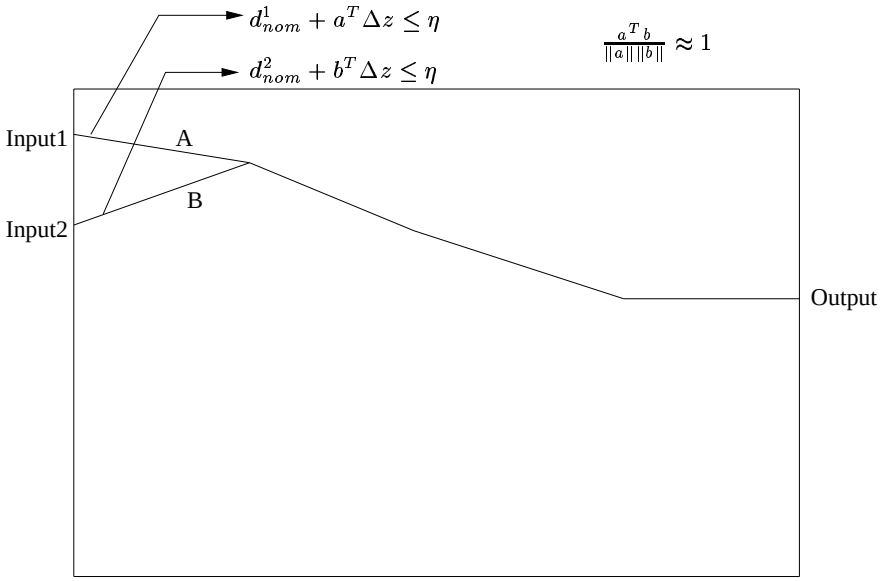

Fig. 3. Paths A and B with path vectors a and b share almost all their gates. Therefore the angle between them is almost zero.

For real circuits, there are many paths that have very similar characteristics owing to a large amount of path sharing, as illustrated in Figure 3. Paths marked A and B in Figure 3 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 set upto that point. This path would represent behavior not represented by any of the other paths in the set of paths selected upto that point. The cosine of the angle that a path with path vector a makes with a path b can be expressed as:

$$\cos\gamma = \frac{a^T b}{\|a\|\|b\|}. \tag{9}$$

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 > CurrentMaximumCosineArray[i]){
            CurrentMaximumCosineArray[i] = CurrentCosine;
         }
         if(CurrentMaximumCosineArray[i] <= GlobalMinimumCosine){
            GlobalMinimumCosine = CurrentMaximumCosineArray[i];
            PathIndex = i;
         }
       }
       j++;
       DirectionSet[j] = SensitivityMatrix[PathIndex];
   }
}
```

The idea encapsulated in the above pseudo-code is simple: at each iteration of the outermost *while* loop, we simply pick a path that makes the largest angle with respect to all paths in *DirectionSet*. The innermost *for* loop finds the smallest angle that any path makes with any path in *DirectionSet*. This operation is done efficiently by maintaining an array called *CurrentMaximumCosineArray* whose *ith* component maintains the cosine of the smallest angle that the *ith* path makes with all the paths in *DirectionSet* until the previous iteration of the outer *while* loop. When a new path is added to *DirectionSet* it is only necessary to update, for

each path in the original system, its entry in *CurrentMaximumCosineArray* with respect to the new path. The variable *GlobalMinimumCosine* is updated if the smallest angle that the current candidate path makes with respect to all the paths in *DirectionSet* is larger than the smallest angle that any other candidate path upto that point makes with the paths in *DirectionSet*. The value of $CurrentMaximumCosine$ at the end of the inner *for* loop measures the "degree of agreement" that the given candidate path has with all the paths selected so far. If this value is 1.0, it means that the direction represented by the candidate path is already represented in the set of paths selected so far. When the outer *for* loop is completed, *GlobalMinimumCosine* records the maximum "degree of disagreement" that any candidate path has with the set of paths in *CurrentSelectedSet*, and *PathIndex* records the identity of the candidate path that has this "degree of disagreement." As such it is natural that this path be included in the set of paths selected so far. If C is the number of paths required to be selected and m is the total number of paths in the system, then the procedure above can be seen to take $O(mCn)$ time. The particular direction selected by the procedure above may not correspond to the most critical path. 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 pseudo-code above, the direction makes an angle of at least $\cos^{-1}(GlobalMinimumCosine)$ 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 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

$$a_i^T x \leq b_i$$
$$a_j^T x \leq b_j \tag{10}$$

are the hyperplane equations corresponding to the two paths. Let the two normal vectors a_i and a_j point in the same direction. Let the origin be on the feasible side of both these hyperplanes, meaning that b_i and b_j are both positive. The path that is closest to the origin in terms of the absolute value distance will form the feasible region boundary. If $\frac{b_i}{\|a_i\|} < \frac{b_j}{\|a_j\|}$ then the distance from the origin to path i is less than the distance to path j. Therefore path i forms the feasible region boundary. If the origin lies on the infeasible side of both hyperplanes, then b_i and b_j will both be negative, and the path which is furthest away in absolute distance from the origin will form the feasible region boundary. In this case the feasible region boundary is once again decided by $min(\frac{b_i}{\|a_i\|}, \frac{b_j}{\|a_j\|})$. Minimizing the signed distance to each hyperplane will ensure the selection of the right path for the feasible region boundary. This step is actually more sophisticated than the "delay criterion" of taking the path with the lowest nominal slack as the critical path. Here we also take into account the sum of the squares of the sensitivity coefficients of the path as it has a direct bearing on its criticality. The algorithm below computes an array $FinalPathFilteredSet$ of size equal to the number of paths in the original system, where the ith entry is 1 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 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 Figure 4. 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

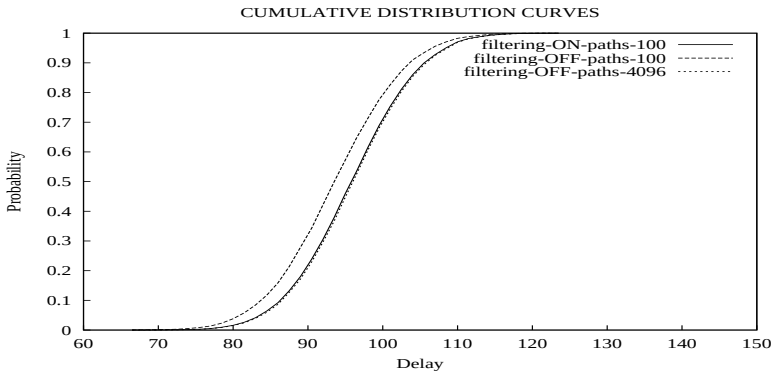

Fig. 4. Cumulative Distribution curves obtained by considering the top 100 nominally critical paths, 100 critical paths obtained by using path filtering and considering all paths in the circuit. In this case, the curve obtained by considering 100 paths obtained through path filtering nearly coincides with the real curve considering all 4096 paths.

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 on 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

$$\int\int\ldots\int_{R^*} JPDF(z_1, z_2, \ldots, z_n) dz_n dz_{n-1} \ldots dz_1. \tag{11}$$

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,

$$\begin{aligned}\delta z_1 \delta z_2 \dots \delta z_n &= \begin{vmatrix} \frac{\partial z_1}{\partial y_1} & \frac{\partial z_1}{\partial y_2} & \dots & \frac{\partial z_1}{\partial y_n} \\ \frac{\partial z_2}{\partial y_1} & \frac{\partial z_2}{\partial y_2} & \dots & \frac{\partial z_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_n}{\partial y_1} & \frac{\partial z_n}{\partial y_2} & \dots & \frac{\partial z_n}{\partial y_n} \end{vmatrix} \delta y_1 \delta y_2 \dots \delta y_n \\ &= |B| \delta y_1 \delta y_2 \dots \delta y_n, \end{aligned} \tag{12}$$

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

$$\begin{aligned} Y &= \int\int \dots \int_{R^*} JPDF(z) dz \\ &= \int\int \dots \int_{R} JPDF(By + d)|B| dy. \end{aligned} \tag{13}$$

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 characteristic 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:

$$\int\int \dots \int f(y_1, y_2, \dots y_n) dy_n \dots dy_1 = \sum_{i=1}^{i=N} w_i f(v_{i1}, v_{i2}, \dots v_{in}) \tag{14}$$

Here the w_is represent the weights of the integration rule, and $(v_{i1}, v_{i2}, \dots v_{in})$ represents the i-th point of the integration rule. Numerically stable integration formulae 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:

$$Y = \sum_{i=1}^{i=n} \frac{V}{2n}(f(e_i) + f(-e_i)) \tag{15}$$

where e_i is the i-th 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.

Higher degree formulas for integration over the unit sphere exist but they suffer from either one or both of two drawbacks: (a) they require an exponential number of points or (b) some of their coefficients are negative. The drawbacks impact the stability of the formulae 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 formulation following [21].

Let R_n be a bounded n-dimensional region which contains the origin ϕ and let Y_n be the surface of R_n. We must assume that R_n is starlike with respect to the origin ϕ [21]. This means

that every ray that starts at the origin intersects Y_n in exactly one point. Let us assume that we have an integration formula for a surface integral:

$$\int\int\ldots\int_{Y_n} f(y_1, y_2 \ldots y_n) d\sigma = \sum_{j=1}^{j=N} B_j f(v_{j,1}, v_{j,2} \ldots v_{j,n}). \tag{16}$$

The B_js are 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

$$rY_n = \{r\nu : \nu \in Y_n\}. \tag{17}$$

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:

$$\frac{d\sigma'}{d\sigma} = r^{n-1}. \tag{18}$$

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

$$\begin{aligned} \int\int\ldots\int_{rY_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma' &= \int\int\ldots\int_{Y_n} r^{n-1} (ry_1)^{\alpha_1} (ry_2)^{\alpha_2} \ldots (ry_n)^{\alpha_n} d\sigma \\ &= r^{n-1+\alpha} \int\int\ldots\int_{Y_n} y_1^{\alpha_1} y_2^{\alpha_2} \ldots y_n^{\alpha_n} d\sigma. \end{aligned} \tag{19}$$

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

as

$$\begin{aligned}\int\int\ldots\int_{R_n} y_1^{\alpha_1}y_2^{\alpha_2}\ldots y_n^{\alpha_n}dy_1dy_2\ldots dy_n &= \int_0^1\left(\int\int\ldots\int_{rY_n} y_1^{\alpha_1}y_2^{\alpha_2}\ldots y_n^{\alpha_n}d\sigma'\right)dr \\ &= \int_0^1 r^{n-1+\alpha}dr\int\int\ldots\int_{Y_n} y_1^{\alpha_1}y_2^{\alpha_2}\ldots y_n^{\alpha_n}d\sigma. \quad (20)\end{aligned}$$

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

$$\int_0^1 r^{n-1}f(r)dr \approx \Sigma_{i=1}^{M}A_if(r_i) \quad (21)$$

Then the points $r_i\nu_j$ and the coefficients A_i, $B_ji = 1, 2\ldots M, j = 1, 2\ldots N_0$ 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 randomised 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:

$$I(f) = \sum_{i=1}^{i=N} w_jf(v_j) \quad (22)$$

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

$$I(f) = \sum_{i=1}^{i=N} w_jf(Qv_j) \quad (23)$$

In (22) and (23) 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 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:

$$I(f) \approx (\frac{V}{2n}) \sum_{i=1}^{i=n} (f(e_i) + f(-e_i)) \tag{24}$$

where e_i is the unit vector along the i-th 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]:

$$I(f) \approx (\frac{V}{2n}) \sum_{i=1}^{i=n+1} (f(u_i) + f(-u_i)) \tag{25}$$

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:

$$\begin{aligned} I(f) \approx V\Big((\frac{(7-n)n}{2(m+1)^2(m+2)}) \sum_{i=1}^{i=n+1} (f(u_i) + f(-u_i)) + \\ (\frac{2(m-1)^2}{m(m+1)^2(m+2)}) \sum_{i=1}^{m(m+1)/2} (f(v_i) + f(-v_i)) \Big) \end{aligned} \tag{26}$$

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 (13) 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 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. Randomising the spherical-radial integration formula (randomized quadrature) allows us to make use of Monte-Carlo error theory to study the variance of the integral estimate.

D. Efficiency of randomised 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 offers no particular advantage. However, the randomised quadrature process outlined in the previous subsection has considerably lower variance than ordinary Monte-Carlo routines such as Gaussian sampling or uniform sampling over the ellipsoid. In Figure 5 we show the cumulative distribution curves obtained by using randomised quadrature over the ellipsoid ("ran-quad" in the 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 Figure 6 we show the standard deviation σ of the three integration procedures. As can clearly be seen, using randomised quadrature provides the lowest standard deviation of the three methods.

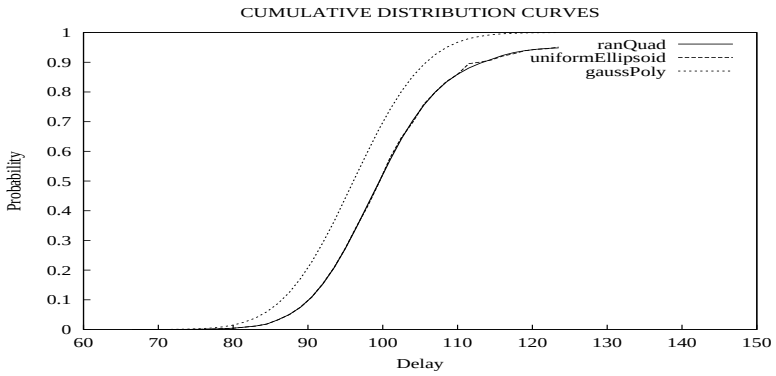

Fig. 5. Cumulative Distribution curves obtained by using Gaussian sampling, randomised quadrature over the sphere, and uniform sampling over the ellipsoid.

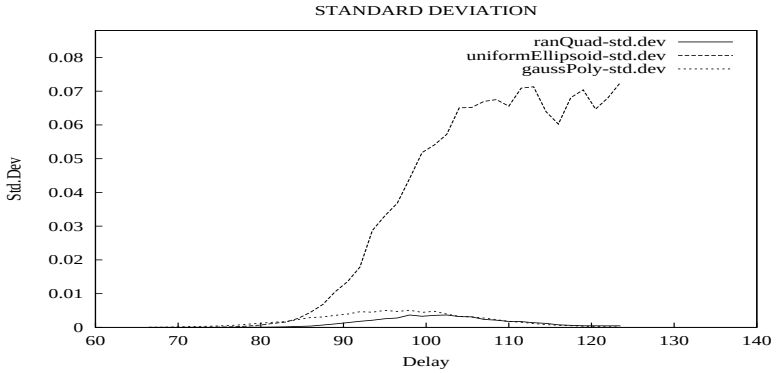

Fig. 6. Standard deviation of Gaussian sampling, randomised quadrature and uniform sampling over the ellipsoid. The randomised quadrature procedure has the lowest standard deviation at low values of yield.

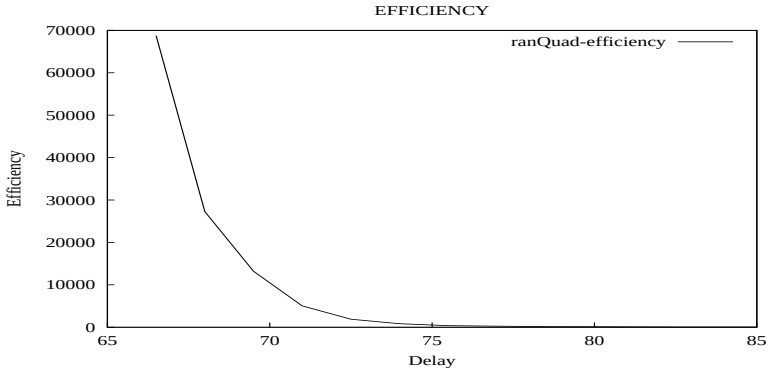

Fig. 7. Efficiency of Gaussian sampling and randomised quadrature. The randomised quadrature procedure is orders of magnitude more efficient than Gaussian sampling at low values of yield.

Following [25] let us define the efficiency

$$\epsilon = \frac{\sigma_1^2 \tau_1}{\sigma_2^2 \tau_2} \tag{27}$$

Here τ_1 and τ_2 represent the computation times for N trials of Gaussian sampling over the yield polytope and performing randomised quadrature, 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 randomised quadrature over the ellipsoid as compared to Gaussian sampling over the whole polytope is shown in Figure 7. This figure clearly brings out the quality of the randomised quadrature procedure. For low yields it is orders of magnitude 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 requiring 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

$$\Phi = \begin{bmatrix} (\frac{\partial s_1}{\partial z})^T \\ (\frac{\partial s_2}{\partial z})^T \end{bmatrix} [V] \begin{bmatrix} \frac{\partial s_1}{\partial z} \frac{\partial s_2}{\partial z} \end{bmatrix} = \begin{bmatrix} A_1^T \\ A_2^T \end{bmatrix} [V] [A_1 A_2] \tag{28}$$

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

$$\Phi = \begin{bmatrix} \sigma_1^2 & \sigma_1 \sigma_2 \rho \\ \sigma_1 \sigma_2 \rho & \sigma_2^2 \end{bmatrix} \tag{29}$$

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]. 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 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 are Gaussian random variables with means a_0 and b_0 repectively. 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

$$T_A = \phi\left(\frac{a_0 - b_0}{\theta}\right) \tag{30}$$

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

$$E[max(A,B)] = a_0 T_A + b_0(1 - T_A) + \theta\phi\left[\frac{a_0 - b_0}{\theta}\right] \tag{31}$$

and

$$Var(A,B) = (\sigma_A^2 + a_0^2)T_A + (\sigma_B^2 + b_0^2)(1 - T_A) + (a_0 + b_0)\theta\phi\left(\frac{a_0 - b_0}{\theta}\right) - E[max(A,B)]^2 \tag{32}$$

B. The 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 (28) 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 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.

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. 8 shows statistical timing results on the 200K gate ASIC circuit with simultaneous temperature and voltage variations. The Y axis represents yield and the X axis represents slack. Superposed on the same plot are results obtained by running EinsTimer 1,200 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 non-linear dependence on the number of paths. This shows that we must use path filtering as a pre-processing step before we use the ellipsoid method. Although results are shown here with only two sources of environmental variation, the anticipated

TABLE I
CPU TIMES ON 200K 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

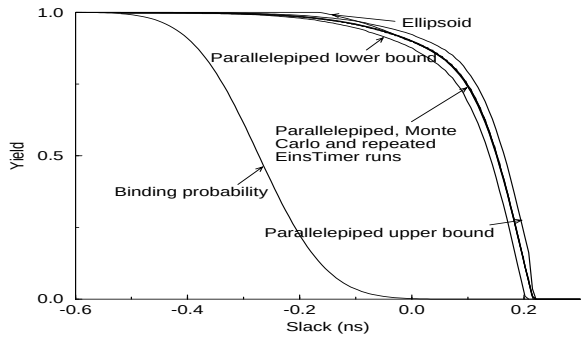

Fig. 8. Statistical timing results on 200K gate ASIC.

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 are from randomly generated problems with a large number of nominally equally critical paths. Fig. 9a 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.9b 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 1,000). With the exception of the ellipsoid method, all the methods are insensitive to first order to the number of data points requested. Finally, Fig.9c shows the growth of CPU

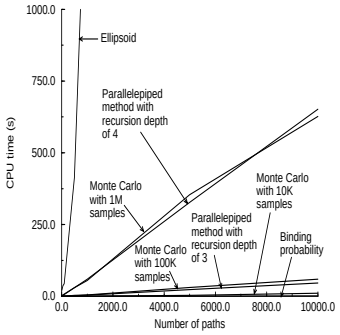

Figure 9a. CPU time vs. of paths.

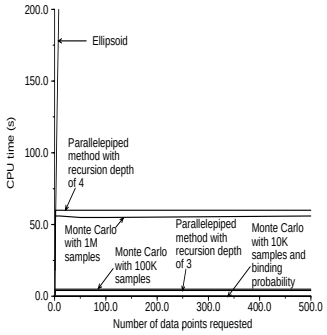

Figure 9b. CPU time vs. number of data points requested

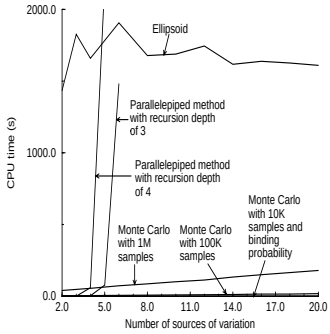

Figure 9c. CPU time vs. number number of sources of variation

time with the number of sources of variation (paths fixed at 1,000 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 6 dimensions. The use of 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 intergation 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. Comparisons

The parallelepiped method is very fast and accurate at low-dimensionality, but has exponential growth of CPU time with the number of sources of variation. The CPU time is independent to 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 accurate but 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 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.

XI. Conclusions 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 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 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 yield. The extreme efficiency of the binding probability method is motivating some new research 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 pre-processing step for both the parallelepiped as well as the binding probability method. Using path filtering might help in speeding up the determination of point infeasiblity. 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 spatial correlation along the lines of [6] 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, statistical intra-chip 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," tech. rep., Semiconductor Industry Association, 2001. Available at `http://public.itrs.net/Files/2001ITRS/Home.htm`.

[2] A. Nadas, "Probabilistic pert," *IBM Journal of Research and Development*, vol. 23, pp. 339–347, May 1979.

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

[4] G. Kleindorfer, "Bounding distributions for stochastic acyclic networks," *Operations Research*, vol. 19, pp. 1586–1601, 1971.

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

[6] H. Chang and S. Sapatnekar, "Statistical timing analysis considering spatial correlation in a pert-like traversal," *Proc. IEEE International Conference on Computer-Aided Design of Integrated Circuits and Systems*, pp. 621–625, Nov 2003.

[7] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First order incremental block-based statistical timing analysis," *Proc. Design Automation Conference*, pp. 331–336, June 2004.

[8] P. Feldmann and S. W. Director, "Integrated circuit quality optimization using surface integrals," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 12, pp. 1868–1879, December 1993.

[9] A. Gattiker, S. Nassif, R. Dinakar, and C. Long, "Timing yield estimation from static timing analysis," *Proc. IEEE International Symposium on Quality Electronic Design*, pp. 437–442, 2001.

[10] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," *Proc. 2001 Design Automation Conference*, pp. 661–666, June 2001.

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

[12] S. W. Director and W. Maly, eds., *Statistical approach to VLSI*, vol. 8 of *Advances in CAD for VLSI*. North-Holland, 1994.

[13] M. Orshansky and K. Keutzer, "A general probabilistic framework for worst case timing analysis," *Proc. 2002 Design Automation Conference*, pp. 556–561, June 2002.

[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 digtial integrated circuits," *Proc. 2003 Design Automation Conference*, June 2003. Anaheim, CA.

[15] D. E. Hocevar, P. F. Cox, and P. Yang, "Parametric yield optimization for MOS circuit blocks," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 7, pp. 645–658, June 1988.

[16] K. Krishna and S. W. Director, "The linearized performance penalty (LPP) method for optimization of parametric yield

and its reliability," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 14, pp. 1557–1568, December 1995.

[17] J. Cohen and T. Hickey, "Two algorithms for determining volumes of convex polyhedra," *Journal of the ACM*, vol. 26, pp. 401–414, July 1979.

[18] J. M. Wojciechowski and J. Vlach, "Ellipsoidal method for design centering and yield estimation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 12, pp. 1570–1579, October 1993.

[19] L. Vandenberghe, S. Boyd, and S.-P. Wu, "Determinant maximization with linear matrix inequality constraints," tech. rep., Information Systems Laboratory, Electrical Engineering Department, Stanford University, April 1996. Available from `http://www.stanford.edu/~boyd/maxdet.html`.

[20] Y. Zhang and L. Gao, "On numerical solution of the maximum volume ellipsoid problem," Tech. Rep. CAAM technical report TR01-15, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005, August 2001.

[21] A. H. Stroud, *Approximate calculation of multiple integrals*. 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), pp. 217–243, AP, 1980.

[23] I. P. Mysovskikh, *Interpolatory cubature formulas*. Moscow-Leningrad: Izd 'Nauka', 1981. Russian text.

[24] A. Genz and J. Monahan, "A stochastic algorithm for high-dimensional multiple integrals over unbounded regions with gaussian weight," *Journal of Computational Applied Mathematics*, no. 112, pp. 71–81, 1999.

[25] J. Hammersley and D. Handscomb, *Monte-Carlo methods*. Metheun and Co. limited, London, 1964.

[26] C. E. Clark, "The greatest of a finite set of random variables," *Operations Research*, pp. 145–162, March-April 1961.

[27] "Ieee standard for integrated circuit (IC) delay and power calculation system," no. IEEE Standard 1481-1999, pp. 1–390, 1999. available at `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," *Proc. IEEE International Conference on Computer-Aided Design of Integrated Circuits and Systems*, pp. 900–907, Nov 2003.

[29] B. Büeler and A. Enge and K. Fukuda, "Exact volume computation for polytopes: a practical study," in *Polytopes - combinatorics and computation* (G. Kalai and e. G. M. Ziegler, eds.), vol. DMV-Seminars vol. 29, Birkhäuser Verlag, Basel, Germany, 2000. Available from `http://www.lix.polytechnique.fr/Labo/Andreas.Enge/Volumen_en.html`.