A Linear-Time Approach for Static Timing Analysis Covering All Process Corners*

Sari Onaissi
Department of ECE
University of Toronto
Toronto, Ontario, Canada
sari@eecg.utoronto.ca

Farid N. Najm
Department of ECE
University of Toronto
Toronto, Ontario, Canada
f.najm@utoronto.ca

ABSTRACT

Manufacturing process variations lead to circuit timing variability and a corresponding timing yield loss. Traditional corner analysis consists of checking all process corners (combinations of process parameter extremes) to make sure that circuit timing constraints are met at all corners, typically by running static timing analysis (STA) at every corner. This approach is becoming too expensive due to the exponential increase in the number of corners with modern processes. As an alternative, we propose a linear-time approach for STA which covers all process corners in a single pass. Our technique assumes a linear dependence of delay on process parameters and provides tight bounds on the worst-case circuit delay. It exhibits high accuracy (within 1-3%) in practice and, if the circuit has m gates and n relevant process parameters, the complexity of the algorithm is $\mathcal{O}(mn)$.

1. INTRODUCTION

The continuous scaling of VLSI technology has led to an increase in the impact that manufacturing process variations can have on circuit delays. These process variations can include die-to-die and within-die process variations, and more generally they can include supply voltage and temperature variations.

One traditional approach to timing verification, at least for ASIC's, is to make sure that a circuit passes its timing requirements at every process corner, using static timing analysis (STA). We will refer to this as *traditional corner analysis.* A loose definition of a "process corner" is that it is a vector of extreme values of all process parameters under consideration. However, such techniques, which involve performing STA over all corners, can be time consuming as the number of corners can be exponential in the number of process parameters under study. Moreover, such methods usually do not allow for the incorporation of within-die variations into the timing analysis of a circuit.

With the increase in the number of interesting process variables in modern processes, the increased cost of traditional corner analysis has become a concern. One alternative approach has been statistical static timing analysis (SSTA) [5, 4, 7, 1, 10, 6]. In SSTA, process parameters are considered to be random variables (RV's), and they lead to other RV's that model cell delays and signal arrival times. However, SSTA has certain problems of its own. For one thing, it depends on knowledge of correlations among within-die features, which are not easily available. Also, it is not necessarily very cheap, especially when one needs to use principal components analysis to resolve the within-die correlations issue.

In this paper, we propose a novel technique for all-corner analysis in a single-shot, with an approach that looks very much like a single run of STA. This is achieved by using linear models of delay (in terms of underlying parameters) and propagating sensitivities through the circuit. The computational complexity of the algorithm will be seen to be $\mathcal{O}(mn)$, where m is the number of gates or cells in the circuit and n is the number of process parameters under consideration. Compare this with the cost of traditional corner analysis, which is $\mathcal{O}(m2^n)$. Our approach is not ideal, it does incur some over-estimation of the worst case delay, for example, but the over-estimation is negligible, in the 1-3% range for the circuits we have tested.

The rest of this paper is organized as follows. An overview is given in section 2, which conveys the salient features of our technique including the delay model and the scope of the work. A description is then given in section 3, which is the core of the paper, of the propagation technique through a single logic gate, cell, or stage. Section 4 is a brief description of circuit level propagation and section 5 describes implementation issues. Finally, section 6 presents empirical validation results and concluding remarks are given in section 7.

2. OVERVIEW

The general idea of our approach is very similar to traditional STA, except that instead of using specific values of arrival times and delays, we represent them as *affine linear functions* of the underlying process parameters.

2.1 Linearity

Linearity is not too strong an assumption, as one may easily verify by circuit simulation on a modern process, and it has been widely adopted recently in the context of SSTA (e.g., in [10]). At face value, propagating linear functions in a timing graph, in the context of STA, would seem problematic because, while the summation of two linear functions is also a linear function, this is not true when one considers the "max" operation which must be applied at every node of the timing graph. For instance, in the simplest case when delay depends linearly on a single parameter, such as in Fig. 1, the max of two intersecting straight line segments **ab** and **cd** is a broken line **aed**. In our work, instead of the true maximum of two planes, which is not a plane, we will use a new plane which is an upper bound on the delay at all points. Thus, in Fig. 1, we would use the dashed line **ad** in place of the true maximum **aed**. We will only be concerned with the accuracy (tightness of this upper bound) at the process corners and not at any nominal mid-range points. The trick is to do this efficiently, and with good accuracy. It looks easy in the 1-D case, but it is

*This work was supported by Intel Corp. and by the Natural Sciences and Engineering Research Council of Canada (NSERC).

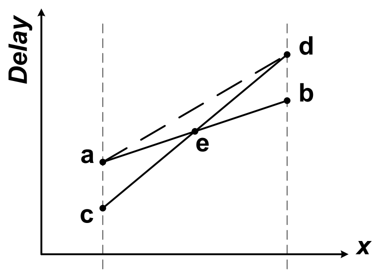

Figure 1: A simple 1-D case

not so simple in general.

As a final comment on the linearity question, we should mention that, even if the linear dependence of gate delay on process parameters is not strictly valid, one can still apply the proposed technique by first constructing a linear expression which is an upper bound on whatever non-linear surface one may have for describing the true dependence of delay on these parameters, and then use that linear expression in place of the true delay, in our algorithm.

2.2 Delay Model

All delays and all arrival times in the logic circuit will be captured as affine linear functions of *normalized* process parameters, whose values range between -1 and $+1$. We will refer to these functions as **delay hyperplanes**; if there are n process parameters under consideration, these functions represent planes in $(n+1)$-dimensional space. Normalizing a process parameter is a trivial operation, which can be illustrated with a simple example. Suppose that the delay of a logic gate depends linearly on one process parameter, say V_t, according to $D = \alpha_0 + s\Delta V_t$, where $-0.05V \leq \Delta V_t \leq 0.05V$ and s has units of sec/Volt. This process parameter can be normalized, i.e., made to vary between -1 and $+1$, by simply multiplying s by $0.05V$, leading to a new sensitivity coefficient $\alpha = (0.05V)s$ whose units are sec. If a unitless variable X, which varies between -1 and $+1$, is used to represent the normalized threshold voltage, then the gate delay can now be written as $D = \alpha_0 + \alpha X$.

In general, having normalized all process parameters, a delay hyperplane is captured by a linear expression, as follows:

$$D = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n \tag{1}$$

where the X_i are the *normalized* process parameters, α_0 is the nominal delay, and α_i are the sensitivities of delay D to the different normalized process parameter variations. A **corner** C is defined as a set of values of all the normalized process parameters, where each of the parameters takes a value of either -1 or $+1$. Therefore $C = (X_1, X_2, \cdots, X_i, \ldots, X_n)$, where $X_i = \pm 1$ for $1 \leq i \leq n$. If the total number of process parameters under consideration is n, then the total number of corners is 2^n.

The **delay of a hyperplane at a corner** is the value obtained by substituting values of the coordinates of the corner in the equation of the hyperplane. Thus, if for a certain hyperplane the delay at corner C is D_C, then the point (C, D_C) belongs to this plane in $(n+1)$-dimensional space. We also use the notation $D(C)$ to refer to the delay of hyperplane D at corner C. Finally, we will use the terminology "**height** of a point" in a hyperplane to refer to the delay of that point. This will help provide some intuition for the various operations that we will perform on hyperplanes.

2.3 Scope of the Work

In traditional corner analysis, one is typically concerned with global die-to-die variations, not with within-die variations. The true reason for this is the exponential complexity of traditional corner analysis. It becomes too expensive to enumerate combinations of within-die variations. However, due to the linear time complexity of our approach, as will be seen below, it is actually possible to apply it to within-die variations as well. Indeed, the only requirement on our variables X_i in (1) is that their various combinations be meaningful process corners that one cares about. Some of them may be physical, some may be voltage or temperature, some may be global, some local, etc. In the paper, we will simply refer to the X_i as process parameters and to their combinations as process corners, without regard to exactly what type of parameters they are.

Due to space limitations, and for clarity of the presentation, the description of the technique in this paper will be somewhat limited. For one thing, we will focus on combinational circuits, which are the crux of the problem, and we will only discuss the problem of estimating the largest circuit delay. In other words, we focus on set-up constraints. However, the work is applicable as-is to hold constraints, and we will show a couple of charts at the end that illustrate our results on estimation of the minimum circuit delay (required to check for hold time violations). Furthermore, if one includes within-die variables, then the technique becomes useful for checking the margins that one needs to leave for clock skew and other mismatch related effects.

3. METHOD AT LOGIC STAGE LEVEL

In this section, we present a method for finding the delay hyperplane at the output of a single logic stage, given the delay hyperplanes for all arrival times at its inputs. A **logic stage** is defined as a cell and its output interconnect structure. The output hyperplane becomes an input for the analysis of downstream stages. We assume that the logic cell and its output interconnect have already been characterized, so that the delay hyperplanes for the timing arcs of the cell delay itself are also available. Given a cell with u inputs, let its input arrival time hyperplanes be:

$$\begin{aligned} D_1 &= s_0^{(1)} + s_1^{(1)} X_1 + s_2^{(1)} X_2 + \cdots + s_n^{(1)} X_n \\ D_2 &= s_0^{(2)} + s_1^{(2)} X_1 + s_2^{(2)} X_2 + \cdots + s_n^{(2)} X_n \\ &\;\;\vdots \\ D_u &= s_0^{(u)} + s_1^{(u)} X_1 + s_2^{(u)} X_2 + \cdots + s_n^{(u)} X_n \end{aligned} \tag{2}$$

Considering the additional delay and variability introduced by the cell itself, each cell can have two delay hyperplanes associated with each of its timing arcs (a rising delay hyperplane and a falling delay hyperplane). The manner in which these hyperplanes are used to account for cell delay can vary. If one is interested in, say the output arrival time of the cell for a rising output, then for every cell input pin, the timing arc rise delay hyperplane is added to the input arrival time hyperplane, leading to a new set of u delay hyperplanes. If one simply wants the worst-case output arrival time irrespective of signal direction, then we create *two* delay hyperplanes for each input, obtained by simply adding each input arrival time hyperplane to, respectively, the arc rise delay hyperplane and the arc fall delay hyperplane, leading to a total of $2u$ delay hyperplanes. In any case, and in order to show a generic analysis, we will assume that one has obtained k input hyperplanes, where k could either be u or $2u$, as follows:

$$\begin{aligned} D_1 &= \alpha_0^{(1)} + \alpha_1^{(1)} X_1 + \alpha_2^{(1)} X_2 + \cdots + \alpha_n^{(1)} X_n \\ D_2 &= \alpha_0^{(2)} + \alpha_1^{(2)} X_1 + \alpha_2^{(2)} X_2 + \cdots + \alpha_n^{(2)} X_n \\ &\;\;\vdots \\ D_k &= \alpha_0^{(k)} + \alpha_1^{(k)} X_1 + \alpha_2^{(k)} X_2 + \cdots + \alpha_n^{(k)} X_n \end{aligned} \tag{3}$$

We refer to these hyperplanes as the **input delay hyperplanes**, keeping in mind of course that they are the result of adding the hyperplanes of the cell input signal arrival times to the hyperplanes of the cell timing arc delays.

Let P be the largest delay over all corners of these input delay hyperplanes, i.e.:

$$P = \max_{i=1}^{k} \left[\max_{j=1}^{2^n} \left(D_i(C_j) \right) \right] \tag{4}$$

We refer to P as the **peak delay** of the input delay hyperplanes. Let the peak delay occur at corner $C_p = (X_1^*, X_2^*, \cdots, X_n^*)$, on hyperplane D_p, where D_p is one of the k input delay hyperplanes, so that:

$$D_p(C_p) = P \tag{5}$$

We will call D_p the **peak plane**, C_p the **peak corner**, and the point (C_p, P) the **peak point** of the k input hyperplanes.

3.1 The Output Hyperplane

If we consider the largest delay at every corner (over all the input delay hyperplanes evaluated at that corner), then the resulting set of 2^n points obviously need not lie on a single hyperplane. For example, in Fig. 2, the four maximum delay points are $(-1, -1, 10)$, $(-1, +1, 10)$, $(+1, -1, 16)$, and $(+1, +1, 14)$, and they are not co-planar. Yet, in order to maintain computational efficiency, we will insist on modeling all delays and all arrival times with delay hyperplanes. Thus, we seek to find an output delay hyperplane D_F that acts as a *ceiling* to the k input hyperplanes at all corners, never under-estimating the delay at any corner but possibly over-estimating it at some. We are, indeed, interested in an output hyperplane that has minimal over-estimation. The problem of finding an *optimal* such hyperplane, which minimizes say the average over-estimation error at all corners, can be formulated as a linear program (LP). However, such an LP would be of exponential complexity, which is not acceptable.

Instead, in this work we propose a linear time algorithm for finding a "good" output delay hyperplane that is a ceiling on all the k input hyperplanes without being too pessimistic. In this respect, we will first specify certain criteria that the output delay hyperplane should satisfy and will then describe our algorithm. These criteria are meant to reduce the over-estimation error, and to make sure that the output hyperplane never under-estimates the maximum delay at any corner.

3.1.1 Output Hyperplane Criteria

We require the output delay hyperplane D_F to satisfy the following criteria:

1. For every corner C_j, for $1 \leq j \leq 2^n$, we require:

$$D_F(C_j) \geq \max_{i=1}^{k}(D_i(C_j)) \tag{6}$$

so that the output hyperplane should never under-estimate the value of the maximum delay at any corner.

2. For every corner C_j, for $1 \leq j \leq 2^n$, we require:

$$D_F(C_j) \leq P \tag{7}$$

where P is the peak delay defined above. The purpose of this criterion is to limit the over-estimation of delays.

3. Given that the peak point defined above is (C_p, P), we also require:

$$D_F(C_p) = P \tag{8}$$

so that the output hyperplane does *not* over-estimate the delay at the peak corner.

In order to find an output hyperplane that meets these criteria, our approach consists of *four* major tasks to be performed: 1) **finding the peak point** over all input hyperplanes, 2) **changing the origin**, 3) **raising the input hyperplanes**, and finally 4) **covering the raised hyperplanes** with the output hyperplane. In what follows, we explain each of these operations, and explain the procedures that we use to achieve these tasks. We also prove that these procedures indeed achieve the required tasks, and that the combination of these four tasks results in a hyperplane that satisfies the criteria specified above.

3.2 Finding the Peak Point

Recall that the peak point (C_p, P) is such that P is the highest value of delay in the k input hyperplanes at all the corners, and $C_p = (X_1^*, X_2^*, \cdots, X_n^*)$ is the corner at which this delay occurs. This point belongs to the peak plane D_p. An example of a peak point is the point $(+1, -1, 16)$ in Fig. 2.

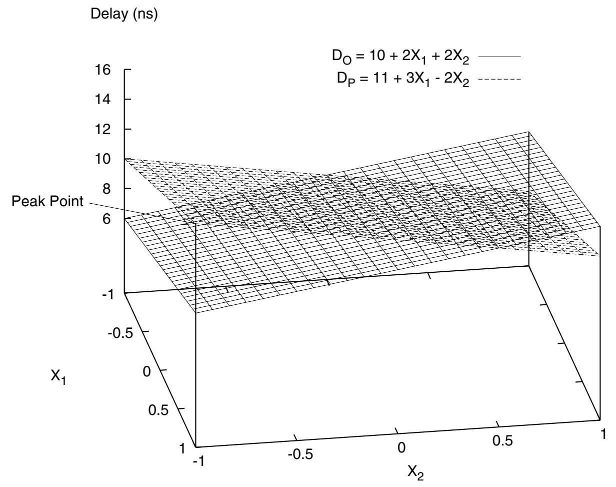

Figure 2: Peak Point of Two Hyperplanes

In order to find the peak point, the highest delay of every hyperplane and its corresponding corner are first found for each of the k input delay hyperplanes. For a given plane D_i, its highest delay p_i can be found using:

$$p_i = \alpha_0^{(i)} + \sum_{j=1}^{n} |\alpha_j^{(i)}| \tag{9}$$

The corner c_{pi} corresponding to this delay can be easily found by setting $X_j = +1$ if $\alpha_j^{(i)} > 0$, and $X_j = -1$ if $\alpha_j^{(i)} < 0$. We then find:

$$P = \max_{i=1}^{k}(p_i) \tag{10}$$

and the peak corner C_p is simply the corner corresponding to the highest delay among all the p_i values.

Finding the highest point of every hyperplane is of complexity $\mathcal{O}(n)$, and doing this for all k input delay hyperplanes is $\mathcal{O}(kn)$. Finding the maximum of all these points is $\mathcal{O}(k)$, so that the overall complexity of finding the peak point is $\mathcal{O}(kn)$.

3.3 Changing the Origin

The next step is to change the origin of the system of coordinates in $(n+1)$-dimensional space such that the new origin is at the point $(C_p, 0)$. We also want to change the directions of some of the coordinate axes so that the new normalized process parameters in this system (Y_i) vary between 0 and 2. This transformation of the coordinate system is not absolutely required, but is cheap and will make subsequent steps of the algorithm clearer and more understandable. Let us call the peak corner in the new system of coordinates C'_p, where $C'_p = (0, 0, \cdots, 0)$, thus, the peak point in the modified system of coordinates becomes (C'_p, P).

Let the transformed equations of the input delay hyperplanes, after modifying the system of coordinates, be as follows:

$$\begin{aligned}
D'_1 &= \beta_0^{(1)} + \beta_1^{(1)} Y_1 + \beta_2^{(1)} Y_2 + \cdots + \beta_n^{(1)} Y_n \\
D'_2 &= \beta_0^{(2)} + \beta_1^{(2)} Y_1 + \beta_2^{(2)} Y_2 + \cdots + \beta_n^{(2)} Y_n \\
&\;\cdot \\
&\;\cdot \\
&\;\cdot \\
D'_k &= \beta_0^{(k)} + \beta_1^{(k)} Y_1 + \beta_2^{(k)} Y_2 + \cdots + \beta_n^{(k)} Y_n
\end{aligned} \tag{11}$$

Modifying the system of coordinates is a simple exercise in analytical geometry and it can be shown that one can achieve it by replacing X_j with $-X_j^*(Y_j - 1)$ for $1 \leq j \leq n$, in each of the k input delay hyperplanes equations. It is also easily shown that substituting X_j with $-X_j^*(Y_j - 1)$ in the equation of a hyperplane

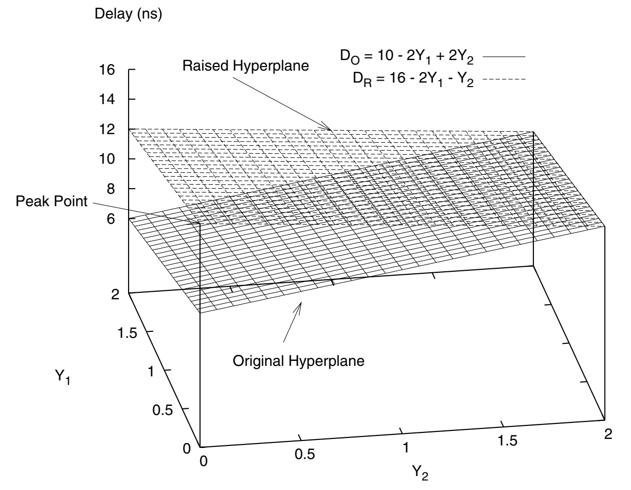

Figure 3: Raising a Hyperplane

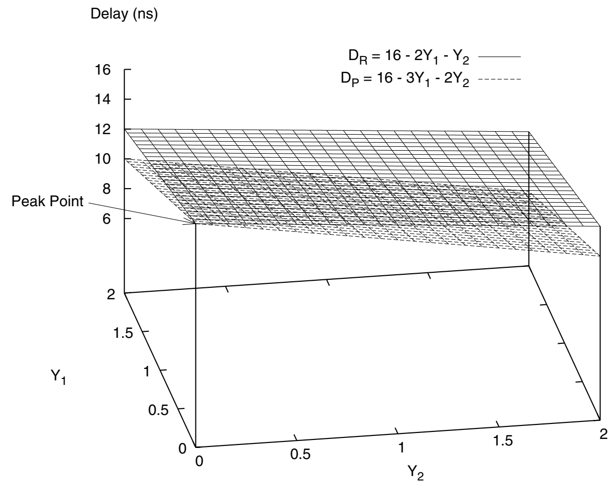

Figure 4: Raised Planes

D_i in (3) results in:

$$\beta_0^{(i)} = \alpha_0^{(i)} + \sum_{j=1}^{n} \alpha_j^{(i)} X_j^* \tag{12}$$

and, for $1 \leq j \leq n$, we have:

$$\beta_j^{(i)} = -\alpha_j^{(i)} X_j^* \tag{13}$$

These relations are used to find the expressions for the k input delay hyperplanes in the modified system of coordinates. For a single hyperplane, the complexity of this operation is $\mathcal{O}(n)$, and thus for all the k hyperplanes the complexity is $\mathcal{O}(kn)$.

3.3.1 Remarks

Without loss of generality, if the peak hyperplane is D'_k, then it is easily shown that $\beta_0^{(k)} = P$ and that $\beta_j^{(k)} \leq 0$ for $1 \leq j \leq n$. To see this, recall that C'_p is the corner at the origin in the new coordinate system, i.e., $C'_p = (0, 0, \cdots, 0)$, so that the value of the constant term of the peak plane must be P. Moreover, if any $\beta_j^{(k)} > 0$, for some $1 \leq j \leq n$, then we can always find a point higher than (C'_p, P) by setting the value of Y_j to 2. This is a contradiction since no point has a delay higher than P among all of the input hyperplanes.

Likewise, if for a hyperplane D'_i other than the peak plane the highest point also corresponds to the peak corner C'_p then, by the same reasoning, each $\beta_j^{(i)} \leq 0$ for $1 \leq j \leq n$. And in the case when the highest point of a hyperplane D'_i corresponds to a corner other than C'_p, then at least one $\beta_j^{(i)} \geq 0$. If this were not true, then the highest point in such a hyperplane would correspond to the peak corner C'_p.

3.4 Raising the Hyperplanes

We then perform an operation that we call **"raising hyperplanes"** on all of the input delay hyperplanes. Intuitively, the purpose of this step is to raise some corners of every hyperplane, by as little as possible, but by just enough to make it pass through the peak point at the peak corner. This will greatly facilitate the subsequent step of choosing an output hyperplane. As an example of this operation, one of the planes (D_O) of Fig. 2 is shown in both its original form and in its "raised" form in Fig. 3. Then, Fig. 4 shows both the peak plane D_P and the new raised plane D_R; both planes now pass through the peak point at the peak corner.

Raising a hyperplane is the most crucial and involved step in our algorithm. The three required criteria of section 3.1.1, lead to three related criteria that our "raised" hyperplanes must meet. Some aspects of these criteria are not unique, in the sense that a suitable output delay hyperplane may be found by using slightly different choices. However, there are good intuitive reasons behind the choices we have made, as will be pointed out below as they arise, and we have verified empirically that they lead to good results. For a hyperplane D'_i in (11), let the corresponding "raised" hyperplane be D''_i, given by:

$$D''_i = \gamma_0^{(i)} + \gamma_1^{(i)} Y_1 + \gamma_2^{(i)} Y_2 + \cdots + \gamma_n^{(i)} Y_n \tag{14}$$

3.4.1 Raised Hyperplane Criteria

For any hyperplane D'_i in (11), the criteria for its "raised" hyperplane D''_i in (14) are as follows:

1. For every corner C'_j, for $1 \leq j \leq 2^n$, we require that
$$D''_i(C'_j) \geq D'_i(C'_j) \tag{15}$$
so that a "raised" hyperplane never under-estimates the delay at any corner.
2. For every corner C'_j, for $1 \leq j \leq 2^n$, we require that
$$D''_i(C'_j) \leq P \tag{16}$$
where P is the peak delay of the k input hyperplanes. The purpose of this criterion is to limit the over-estimation of corner delays.
3. Given that the peak point in the modified system of coordinates is (C'_p, P), then we require that
$$D''_i(C'_p) = P \tag{17}$$
so that every "raised" hyperplane passes through the peak point.

It is easy to see that (17) leads to the requirement that the constant term in the equation of any "raised" hyperplane be P, since $C'_p = (0, 0, \cdots, 0)$, so that, for all $1 \leq i \leq k$, we have as a first result:

$$\gamma_0^{(i)} = P \tag{18}$$

3.4.2 Procedure

Given the remarks in section 3.3.1, we can classify our input delay hyperplanes into three classes. The first class contains only one hyperplane: the peak plane. The second class contains planes, other than the peak plane, whose highest points happen to be at the peak corner C'_p. The third class consists of those remaining hyperplanes whose highest points are at corners other than the peak corner. The steps taken to "raise" a hyperplane differ from one class to another, as considered in the following three **cases**. In each case, we will describe without proof the procedure applied in our algorithm. We then give a section in which the validity of all the steps is rigorously proven.

3.4.2.1 Case 1.

This is the case when the hyperplane belongs to the first class, i.e., it is the peak plane. In this case, we select:

$$\gamma_0^{(i)} = \beta_0^{(i)} = P \tag{19}$$

and, for $1 \leq j \leq n$,

$$\gamma_j^{(i)} = \beta_j^{(i)} \tag{20}$$

Thus, the hyperplane remains unchanged.

3.4.2.2 Case 2.

This is the case when the hyperplane belongs to the second class, i.e., the highest point in this plane is achieved at the peak corner C'_p. In this case, and as shown in section 3.3.1, $\beta_j^{(i)} \leq 0$, for $1 \leq j \leq n$. For this case, in order to "raise" hyperplane D'_i, our procedure is to only change the value of its constant term and of only *one* of its sensitivity coefficients $\beta_j^{(i)}$, $1 \leq j \leq n$. There is some flexibility in the choice of exactly which coefficient to change. Best empirical results are obtained by choosing the *largest* coefficient, i.e., the least negative one. Assume, without loss of generality, that $\beta_1^{(i)} = \max_{j=1}^{n}(\beta_j^{(i)})$. Then, the "raised" hyperplane is obtained according to the following construction:

$$\gamma_0^{(i)} = P \tag{21}$$

and

$$\gamma_1^{(i)} = \frac{-P + \beta_0^{(i)} + 2\beta_1^{(i)}}{2} \tag{22}$$

and, for $2 \leq j \leq n$,

$$\gamma_j^{(i)} = \beta_j^{(i)} \tag{23}$$

Since the original plane $D'_i \leq P$ at all corners, then the corner $(2, 0, 0, \ldots, 0)$ yields $\beta_0^{(i)} + 2\beta_1^{(i)} \leq P$ and, therefore, $\gamma_j^{(i)} \leq 0$, for $1 \leq j \leq n$.

3.4.2.3 Case 3.

This is the case when the hyperplane belongs to the third class, i.e., the highest point of this plane corresponds to a corner other than the peak corner C'_p. In this case, and as we saw in section 3.3.1, at least one $\beta_j^{(i)} \geq 0$, for $1 \leq j \leq n$. In this case, in order to "raise" a plane D'_i, we change the value of the constant term, and of all the sensitivities with positive values in the expression for that hyperplane. Assume, without loss of generality, that $\beta_j^{(i)} \geq 0$ for $1 \leq j \leq \hat{n}$, where $\hat{n}$ is the number of positive sensitivities of D'_i. In this case, we formulate the "raised" hyperplane according to:

$$\gamma_0^{(i)} = P \tag{24}$$

and, for $1 \leq j \leq \hat{n}$,

$$\gamma_j^{(i)} = \frac{-P + \beta_0^{(i)} + \sum_{l=1}^{\hat{n}} 2\beta_l^{(i)}}{2\hat{n}} \tag{25}$$

and, for $\hat{n} + 1 \leq j \leq n$,

$$\gamma_j^{(i)} = \beta_j^{(i)} \tag{26}$$

As in the previous case, since $D'_i \leq P$ at all corners, then by a judicious choice of a specific corner we easily find that $-P + \beta_0^{(i)} + \sum_{j=1}^{\hat{n}} 2\beta_j^{(i)} \leq 0$. Thus, in this case as well, $\gamma_j^{(i)} \leq 0$, for $1 \leq j \leq n$.

3.4.3 Proof of Correctness

We will now prove that, for each of the three cases under consideration, the "raised" hyperplane meets the criteria specified in section 3.4.1. Notice that, in all three cases, the "raised" hyperplane D''_i has a constant term $\gamma_0^{(i)} = P$, and $\gamma_j^{(i)} \leq 0$, for $1 \leq j \leq n$. Recall also that the peak corner is at the origin $C'_p = (0, 0, \cdots, 0)$, thus for each of these cases:

$$D''_i(C'_p) = \gamma_0^{(i)} = P \tag{27}$$

Therefore, the "raised" hyperplane D''_i satisfies the third criterion of section 3.4.1 for all the three cases.

Moreover, given that for all three cases, $\gamma_j^{(i)} \leq 0$, and since $0 \leq Y_j \leq 2$, for $1 \leq j \leq n$, then it is also straightforward to see, from (14), that for all corners C_t, $1 \leq t \leq 2^n$:

$$D''_i(C_t) \leq \gamma_0^{(i)} = P \tag{28}$$

Thus, the "raised" hyperplane D''_i satisfies the second criterion of section 3.4.1 for all the three cases in section 3.4.2.

It remains to prove that, for all three cases of section 3.4.2, the "raised" hyperplane D''_i satisfies the *first* criterion of section 3.4.1. Recall that this criterion is the requirement that the delay of the "raised" hyperplane D''_i, at any corner, be no less than the delay of the original hyperplane D'_i at the same corner.

We start with case 1. In this case, the hyperplane D'_i is the peak plane and it is not changed. Thus, the first criterion of section 3.4.1 is trivially satisfied for this case.

We now consider case 2. In this case, only one of the sensitivities of D'_i is changed to find D''_i and we assumed, without loss of generality, that $\beta_1^{(i)}$ is the sensitivity term to be changed. Also, the constant term of the hyperplane, $\beta_0^{(i)}$, was changed, by *increasing* its value to $\gamma_0^{(i)} = P$. Notice that the original value $\beta_0^{(i)} \leq P$ because this is not the peak plane. Thus, it is impossible for the "raised" plane to under-estimate the delay at a corner C'_t which has $Y_1 = 0$. Therefore it is enough to prove that D''_i does not under-estimate the delay at any corner C'_t, where $Y_1 = 2$. Given such a corner $C'_t = (2, Y_2, \cdots, Y_n)$, we have:

$$D''_i(C'_t) = P + 2\gamma_1^{(i)} + \gamma_2^{(i)} Y_2 + \cdots + \gamma_n^{(i)} Y_n \tag{29}$$

which, using (22) and (23), can be written as:

$$D''_i(C'_t) = P + 2\left(\frac{-P + \beta_0^{(i)} + 2\beta_1^{(i)}}{2}\right) + \sum_{j=2}^{n} \beta_j^{(i)} Y_j \tag{30}$$

which easily reduces to:

$$D''_i(C'_t) = \beta_0^{(i)} + 2\beta_1^{(i)} + \sum_{j=2}^{n} \beta_j^{(i)} Y_j = D'_i(C'_t) \tag{31}$$

Therefore, the first criterion of section 3.4.1 is satisfied for every corner C'_t in this case.

We finally consider case 3. In this case, all the positive sensitivities of D'_i are changed in order to arrive at D''_i and we assumed, without loss of generality, that the only positive sensitivities are $\beta_j^{(i)} \geq 0$ for $1 \leq j \leq \hat{n}$. In addition, the constant term of the hyperplane expression ($\beta_0^{(i)}$) is changed and its value is *increased* to P. Thus, it is impossible for the "raised" hyperplane to under-estimate the delay at a corner C'_t which has $Y_j = 0$, for $1 \leq j \leq \hat{n}$. Therefore, it is enough to prove that D''_i does not under-estimate the delay at any corner C'_t for which at least one $Y_j = 2$, for $1 \leq j \leq \hat{n}$.

Actually, it would suffice to prove that D''_i does not underestimate the delay at corners C'_t, where all $Y_j = 2$ for $1 \leq j \leq \hat{n}$. This is because, with $\gamma_j^{(i)} \leq 0$ and $\beta_j^{(i)} \geq 0$ for $1 \leq j \leq \hat{n}$, if for such a corner C'_t, $D''_i(C'_t) \geq D'_i(C'_t)$, then changing any Y_j for $1 \leq j \leq \hat{n}$ from 2 to 0 would only increase the value of $D''_i(C'_t)$

and decrease the value of $D'_i(C'_t)$, thus maintaining the inequality. Now, given such a corner C'_t, where $Y_j = 2$ for $1 \leq j \leq \hat{n}$, we have:

$$D''_i(C'_t) = P + 2\sum_{j=1}^{\hat{n}} \gamma_j^{(i)} + \sum_{j=\hat{n}+1}^{n} \gamma_j^{(i)} Y_j \tag{32}$$

which, using (25) and (26), can be written as:

$$\begin{aligned} D''_i(C'_t) &= P + 2\sum_{j=1}^{\hat{n}} \left(\frac{-P + \beta_0^{(i)} + 2\sum_{l=1}^{\hat{n}} \beta_l^{(i)}}{2\hat{n}} \right) \\ &\quad + \sum_{j=\hat{n}+1}^{n} \beta_j^{(i)} Y_j \end{aligned} \tag{33}$$

which easily reduces to:

$$D''_i(C'_t) = \beta_0^{(i)} + 2\sum_{j=1}^{\hat{n}} \beta_j^{(i)} + \sum_{j=\hat{n}+1}^{n} \beta_j^{(i)} Y_j = D'_i(C'_t) \tag{34}$$

Therefore, the first criterion in section 3.4.1 is satisfied for any corner C'_t in this case 3.

3.4.4 Complexity

"Raising" one hyperplane requires examining each of its sensitives and might involve the modification of these sensitivities according to pre-specified equations. Thus "raising" one hyperplane is of complexity $\mathcal{O}(n)$, and performing this operation for all the k input delay hyperplanes is $\mathcal{O}(kn)$.

3.5 Covering the Raised Hyperplanes

We now have a set of raised hyperplanes shown in (35). These hyperplanes satisfy the three criteria for raised hyperplanes in section 3.4.1 and our goal now is to find an output hyperplane that satisfies the criteria of section 3.1.1.

$$\begin{aligned} D''_1 &= P + \gamma_1^{(1)} Y_1 + \gamma_2^{(1)} Y_2 + \cdots + \gamma_n^{(1)} Y_n \\ D''_2 &= P + \gamma_1^{(2)} Y_1 + \gamma_2^{(2)} Y_2 + \cdots + \gamma_n^{(2)} Y_n \\ &\cdot \\ &\cdot \\ &\cdot \\ D''_k &= P + \gamma_1^{(k)} Y_1 + \gamma_2^{(k)} Y_2 + \cdots + \gamma_n^{(k)} Y_n \end{aligned} \tag{35}$$

Let the expression for the desired output delay hyperplane be as shown:

$$D'_F = \lambda_0 + \lambda_1 Y_1 + \lambda_2 Y_2 + \cdots + \lambda_n Y_n \tag{36}$$

Our algorithm finds the output plane based on:

$$\lambda_0 = P \tag{37}$$

and, for $1 \leq j \leq n$,

$$\lambda_j = \max_{i=1}^{k}(\gamma_j^{(i)}) \tag{38}$$

3.5.1 Proof of Correctness

We will prove that the output delay hyperplane found above satisfies the criteria set in section 3.1.1. In our discussion, D'_F refers to the equation of the output delay hyperplane in the modified system of coordinates, while D_F refers to the equation of this hyperplane in the original system.

First of all, since $\lambda_0 = P$, it follows that $D'_F(C'_p) = P$, and equivalently $D_F(C_p) = P$, thus the third criterion of section 3.1.1 is satisfied. Since $\gamma_j^{(i)} \leq 0$, $1 \leq j \leq n$, for all raised hyperplanes, then $\lambda_j \leq 0$ for $1 \leq j \leq n$. Given that $0 \leq Y_j \leq 2$, for $1 \leq j \leq n$ it is easy to see that for any corner C'_t in the modified system of coordinates, $1 \leq t \leq 2^n$, $D'_F(C'_t) \leq P$ and equivalently $D_F(C_t) \leq P$ for any corner C_t in the original system of coordinates. Thus, the output delay hyperplane satisfies the second criterion of section 3.1.1.

It remains to prove that the output delay hyperplane satisfies the first criterion of section 3.1.1. Recall that this criterion is that the output delay hyperplane should not under-estimate the maximum delay at any corner. From the first criterion of section 3.4.1, we know that for any "raised" plane D''_i and at any corner C'_t in the modified system of coordinates we have $D''_i(C'_t) \geq D'_i(C'_t)$, where D'_i is the hyperplane that was "raised" in order to create D''_i. From (37), (38), and the fact that $0 \leq Y_j \leq 2$, for $1 \leq j \leq n$, we can deduce that for all corners (C'_t), $1 \leq t \leq 2^n$, in the modified system of coordinates:

$$D'_F(C'_t) \geq \max_{i=1}^{k}(D''_i(C'_t)) \tag{39}$$

thus, by using the first criterion for "raised hyperplanes" of section 3.4.1, we can easily deduce that:

$$D'_F(C'_t) \geq \max_{i=1}^{k}(D'_i(C'_t)) \tag{40}$$

and equivalently that for all corners (C_t), $1 \leq t \leq 2^n$, in the original system of coordinates:

$$D_F(C_t) \geq \max_{i=1}^{k}(D_i(C_t)) \tag{41}$$

Therefore, the output delay hyperplane also satisfies the first criterion of section 3.1.1.

3.5.2 Complexity

Finding each value of λ takes a time linear in k, and thus finding all n values is of complexity $\mathcal{O}(nk)$.

3.6 Interconnect Handling

After finding the equation of the output delay hyperplane in the modified system of coordinates, we change our system back to the original one. This can be easily done by reversing the transformations performed above. This yields the equation of the delay hyperplane at the output of the cell of the logic stage. In order to be able to propagate a delay hyperplane to subsequent logic stages, we must first account for the delay and variability introduced by the interconnect structure of the current logic stage. This can be done by simply adding the delay hyperplane of the interconnect structure to the delay hyperplane at the output of the cell, to get the hyperplane at the output of the logic stage. In the case of multiple fanout nets, the interconnect structure may have a distinct delay hyperplane for each of the fanouts. In this case, the interconnect delay hyperplane corresponding to each fanout is added to the delay hyperplane at the output of the cell to get the delay hyperplane at this output of the logic stage. This is a simple operation of complexity $\mathcal{O}(fn)$, where f is the number of fanout points of the logic stage.

3.7 Overall Complexity for a Logic Stage

An investigation of all the operations that our method performs at a logic stage shows that the total complexity for a single logic stage is $\mathcal{O}(kn + fn)$, where as we stated before, k could either be the equal to the number of inputs of the cell u, or twice that number, f is the number of fanout nets, and n is the number of process parameters being considered. Thus the complexity of our method at the logic stage can be written as $\mathcal{O}(un+fn)$. Since, for obvious technology reasons, both u and f are bounded by some small fixed number, such as perhaps 10, and since they do not scale with the size of the problem, then the overall complexity for analysis of one logic stage is $\mathcal{O}(n)$.

4. METHOD AT CIRCUIT LEVEL

As stated earlier, our approach is quite similar to traditional STA at the circuit level. Thus, in order to find the maximum delay of the circuit under process variations, we apply our method to logic stages whose input delay information (hyperplanes) is available and then propagate the resulting delay hyperplanes to subsequent logic stages. This process is repeated until we get the delay hyperplanes at the primary output nodes of the circuit. The

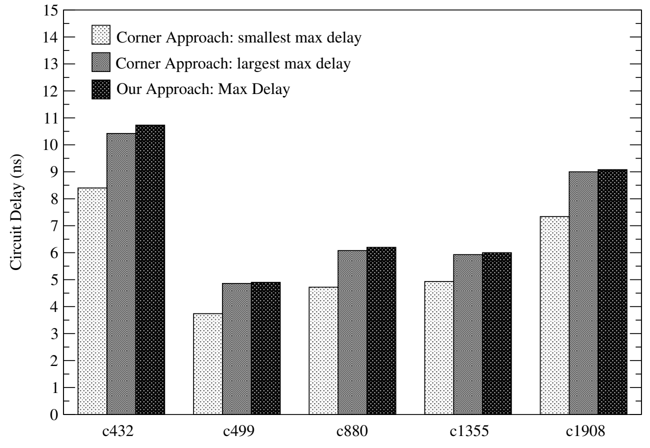

Figure 5: Max. Delays for Smaller Circuits

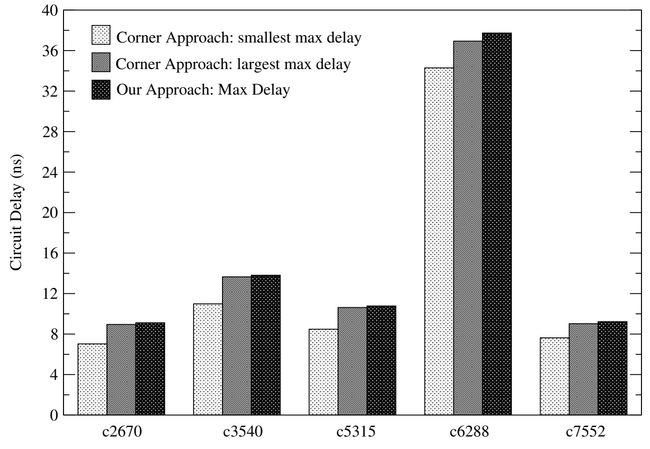

Figure 6: Max. Delays for Larger Circuits

circuit delay for each corner is the maximum of the various output node delays at that corner. This "max" operation is identical to the "max" operation for a single stage, and our algorithm is applicable one final time in that case to give a single hyperplane for the overall circuit delay. It is then straightforward to find the maximum value of delay that this plane can produce. The complexity of this step is $\mathcal{O}(zn)$, where z is the number of primary output nodes of the circuit.

It should be mentioned that, once the delay hyperplane is available at the circuit primary outputs, it provides more information than simply the worst-case corner and delay. It provides information on sensitivity of overall circuit delay to the various parameters. This can be useful to diagnose problems with circuits that may be too sensitive to specific sources of variations.

If a circuit has m stages in total, and since the analysis of a single stage is $\mathcal{O}(n)$, and since $z \leq m$, then the overall complexity of STA for the whole circuit, covering *all* 2^n process corners, is only $\mathcal{O}(mn)$. Thus, for a given fixed circuit size m, the complexity grows *linearly* with the number of variable process parameters n.

5. IMPLEMENTATION

As is to be expected, our algorithm requires a pre-characterized cell library and some variational model of interconnect delay.

5.1 Logic Cells Characterization

We constructed and characterized a small CMOS cell library in 90nm technology, and these cells were used as a standard library for our testing. For every input arc in every logic cell, the rise and fall delays of the cell were computed, using HSPICE, for a number of input slew (slope) rates and load capacitances. The results were stored in tables called nominal delay tables, where, given an input and its slew rate (slope), and the load capacitance on the cell, the delay can be computed through interpolation between appropriate values from the delay tables. This is the standard modern approach for modeling the delay of logic cells [9].

Then, the sensitivities of the delays were found for variations in four process parameters: the NMOS threshold voltage (ΔV_{tn}), the NMOS channel length (ΔL_n), the PMOS threshold voltage (ΔV_{tp}), and the PMOS channel length (ΔL_p). The sensitivities to every process parameter were found by fixing all other process parameters at their nominal values, choosing a number of values for this parameter, and finding the delay of the cell in each case using HSPICE. After that, linear regression was performed on the resulting delays to find the un-normalized sensitivities of cell delays to the process parameter. The sensitivities were then normalized so that the parameters vary between -1 and 1.

For every input arc, this characterization is performed for a transition in the input which causes a rise in the output signal, and a transition which causes a fall in the output signal. Thus, every input arc has a rise delay hyperplane that uses rise sensitivities and a fall delay hyperplane that uses fall sensitivities. It is worth noting that median values were chosen for the input slew rates and the load capacitances in the characterization process, and that we keep a record of the nominal delay of the cell for these slews and load capacitance values. The significance of this point will become clear shortly.

Table 1: Our Approach vs. Corner Approach.

ISCAS-85 Circuit	Corner Approach Max. Delay	Our Approach Max. Delay	Percentage Error (%)
c432	10.42 ns	10.73 ns	3.04%
c499	4.86 ns	4.90 ns	0.78%
c880	6.08 ns	6.20 ns	1.89%
c1355	5.93 ns	6.00 ns	1.24%
c1908	9.00 ns	9.08 ns	0.88%
c2670	8.95 ns	9.11 ns	1.74%
c3540	13.65 ns	13.81 ns	1.18%
c5315	10.62 ns	10.77 ns	1.39%
c6288	36.93 ns	37.73 ns	2.15%
c7552	9.03 ns	9.23 ns	2.30%

5.2 Interconnect Characterization

For the characterization of interconnect, the technique described in [2] was applied. An interconnect fanout structure is described as a tree of lumped resistances and capacitances, i.e., an *RC*-tree. One is also given the sensitivities of every resistor and capacitor to the relevant process parameters. From this, one finds the nominal delay at every node of the tree, as well as the sensitivities of the delay at every node to the same process parameters. The process parameters of interest in our work, as in [2], are the metal width (ΔW), metal thickness (ΔT), and the interlayer dielectric thickness (ΔH). For the test cases considered in this paper, we limited the effect that a process parameter can have on the value of a resistor or a capacitor to no more than 10%.

5.3 Circuit Modeling

It is worth noting that, for every logic cell, we use the method in [8] to compute the *effective capacitance* seen at the output of this cell. This is a standard method that allows us to use the precharacterized nominal delay tables to find the nominal delays of the hyperplanes of a particular cell in the circuit [9]. Notice that the sensitivities to physical parameters which were measured during cell characterization (with some median input slope and output load applied) need to be modified according to the context of the cell in the given circuit, i.e., according to the actual slope and load presented to the cell in the circuit. This is a subtle point, which we have found can be overcome with good accuracy by *scaling* the characterized sensitivities by the ratio of the nominal delay of the cell in the context of the circuit to its nominal delay during characterization.

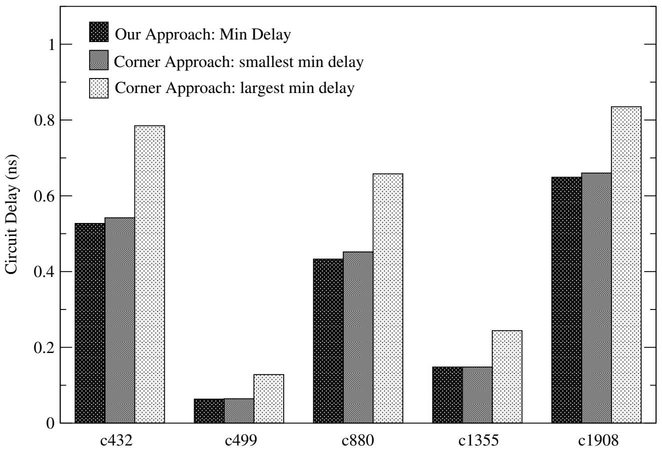

Figure 7: Min. Delays for Smaller Circuits

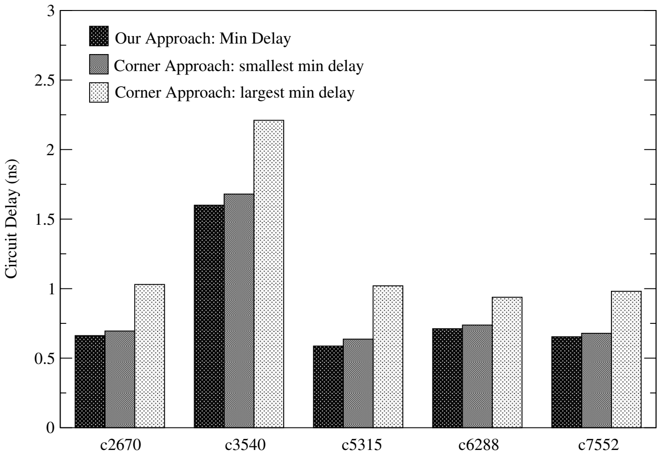

Figure 8: Min. Delays for Larger Circuits

6. RESULTS

We ran our algorithm on all circuits of the ISCAS-85 benchmark suite [3], mapped to our 90nm cell library. In order to test the accuracy of our approach, we compared the maximum delay of every circuit as computed by our algorithm to the true maximum delay computed by finding the delay of the circuit at every process corner. The results and comparison between the two approaches are shown in Table 1, which shows that our approach predicts the maximum delay of these circuits quite accurately. The bar diagrams in Fig. 5 and Fig. 6 show these same results, with three bars for every circuit. The first bar is found by running traditional STA for every process corner and recording the smallest value found (over all corners) for the maximum delay of the circuit. The second bar shows the largest value found (over all corners) for the maximum delay of the circuit. The third shows the worst-case delay reported by our approach. It is noteworthy that *i)* there is a significant difference between the first and second bars, indicating that process variations cause a significant delay spread for these circuits, and *ii)* our approach finds, in linear time, a tight upper-bound on the worst-case delay. Finally, as mentioned earlier, our approach is applicable to the min delay case as well, for checking hold time violations. In this respect, Figs. 7 and 8 show similar data for the min delay case.

We also recorded the run-times of our approach for these circuits and compared it with the run-times of the corner approach. As expected, the results in Table 2 show that our approach achieves a speed-up of approximately 20 times when compared to the corner approach. While our approach is $\mathcal{O}(mn)$ (for a circuit with m gates and n relevant process parameters), the computational complexity of the corner approach is $\mathcal{O}(m2^n)$, hence the observed speed-up. Moreover, as the number of process parameters being considered increases with future technology, this speed-up will become even more dramatic.

Table 2: Run-time Comparison.

ISCAS-85 Circuit	Corner Approach Run Time	Our Approach Run Time	Speed-up (×)
c432	4.23 s	0.22 s	19.23
c499	10.38 s	0.53 s	19.58
c880	18.39 s	0.82 s	22.43
c1355	34.02 s	1.57 s	21.67
c1908	45.76 s	2.24 s	20.43
c2670	6.99 s	0.36 s	19.42
c3540	23.9 s	1.09 s	21.92
c5315	3.5 s	0.18 s	19.44
c6288	53.59 s	2.4 s	22.33
c7552	71.22 s	3.19 s	22.33

7. CONCLUSION

All-corner timing analysis in linear time has been the "holy grail" in timing analysis for some time. In this paper, a linear-time approach has been presented which does this with negligible error (1-3%). The error is always conservative, so that one will never under-estimate the maximum circuit delay and never over-estimate the minimum circuit delay. This technique uses standard/traditional timing models for cells and interconnect and hence is very easy to integrate into today's design methodology. It is hoped that this work will lead to practical techniques for handling variability in VLSI design.

8. ACKNOWLEDGEMENTS

We would like to thank Mr. Khaled Heloue for his helpful feedback and contributions to this work.

9. REFERENCES

[1] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. Computation and refinement of statistical bounds on circuit delay. In *Design Automation Conference*, pages 348–353, June 2-6 2003.

[2] K. Agarwal, D. Sylvester, D. Blaauw, F. Liu, S. Nassif, and S. Vrudhula. Variational delay metrics for interconnect timing analysis. In *DAC*, pages 381–384, June 7-11 2004.

[3] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *IEEE International Symposium on Circuits and Systems (ISCAS-85)*, pages 663–698,, June 1985.

[4] H. Chang and S. Sapatnekar. Statistical timing alaysis considering spatial correlations using a single PERT-line traversal. In *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 621–625, November 9-13 2003.

[5] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. In *Int'l Symp. on Quality Electronic Design*, pages 437–442, March 2001.

[6] K. R. Heloue and F. N. Najm. Statistical timing analysis with two-sided constraints. In *Int. Conf. on Computer Aided Design (ICCAD)*, pages 829–863, November 2005.

[7] J. A. G. Jess, K. Kalafala, W. R. Naidu, R. H. J. M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *Design Automation Conference*, pages 932–937, June 2-6 2003.

[8] J. Quain, S. Pullela, and L. Pillage. Modeling the "effective capacitance" for the rc interconnect of cmos gates. *IEEE Trans. on Computer-Aided Design*, 13(12):1526–1535, 1994.

[9] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, Norwell, MA, 1st edition, 2004.

[10] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Waler, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Design Automation Conference*, pages 331–336, June 2004.