Speedpath Analysis Under Parametric Timing Models

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

Joel R. Phillips
Cadence Research Labs
Cadence Design Systems
Berkeley, CA 94704
jrp@cadence.com

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

ABSTRACT

The clock frequency of a digital IC is limited by its slowest paths, designated by speedpaths. Given the extreme complexity involved in modeling modern IC technologies, often speedpath predictions provided by timing analysis tools are not correct. Therefore, several practical techniques have recently been proposed for design debugging, that combine silicon stepping of improved versions of a circuit with subsequent correlation between measured and predicted data. Addressing these issues, this paper proposes a set of techniques that enable the designer to obtain reduced subsets of paths, guaranteed to contain all the speedpaths of a given circuit or block. Such subsets can be computed either from timing models, prior to fabrication, or incorporating actual delay measurements from fabricated instances.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design, Theory, Verification

Keywords

Speedpath analysis, Parametric timing models

1. INTRODUCTION

The performance of a digital IC is limited by its most timing-critical paths, usually designated by *speedpaths*. Performance is most often related to the maximum frequency at which a given IC can be clocked, and can be measured by applying a set of functional tests to the fabricated IC, designed to exercise all the speedpaths. Above the limiting clock frequency, some of these tests will fail.

Most ASICs are designed to operate at a particular clock frequency, therefore all the fabricated ICs that meet that clock frequency threshold are accepted, and the remaining ones are rejected. Microprocessor designs are, in general, more flexible and able to operate on a relatively wide range of clock frequencies. Therefore, a binning scheme is usually adopted, whereby every IC is tested and, according to its maximum clock frequency, is assigned to a given bin, or rejected. Each bin corresponds to a speed rating, and ICs with higher speed ratings are sold at higher prices.

Ideally, speedpaths would be correctly predicted by static timing analysis (STA) tools, prior to fabrication. However, accurately modeling all the effects present in modern IC technologies is an overly complex task. Furthermore, such technologies are increasingly sensitive to process variability, which introduces additional uncertainty. Consequently, most often speedpath prediction may not be entirely accurate, some of the critical paths in the fabricated IC may actually be missed and, instead, other paths may be reported with a distinct, yet relatively close, predicted delay.

The importance of correctly identifying the set of speedpaths of a given circuit is two-fold. First, it enables timely design correction, by fixing the speedpaths prior to fabrication, such that the design meets the target clock frequency. Second, it improves the quality of test generation. Since it is usually impossible to test the behavior of a given IC for all possible test sequences, only the sequences that exercise the critical aspects of the circuit, such as speedpaths, are tested for the purpose of speed rating. By failing to exercise the speedpaths of a given IC, the testing procedure may be unable to correctly identify its limiting clock frequency.

Even though STA tools may fail to correctly identify all the speedpaths, due to limited accuracy in modeling modern IC technology effects, they should at least be able to provide a conservative set of paths, guaranteed to contain all the speedpaths. Therefore, by fixing all such paths prior to fabrication, designers could ensure that the speedpaths would also be fixed. Addressing this problem, and given a parametric timing model of a circuit, this paper proposes a technique, built on top of [3], that enables the efficient computation of its top k critical timing corners and paths. By choosing k appropriately, the designer should be able to obtain a set of paths that contains all the speedpaths.

Given the limitations of STA tools that have been discussed, several practical techniques [6, 1, 5, 8] have recently been proposed for design debugging, that combine silicon stepping (e.g. fabrication of increasingly optimized versions of the same circuit) with subsequent analysis and correlation of measured and predicted data. Such techniques target the improvement of timing models by feeding back such models with actual delay measurements from fabricated instances. Speedpath isolation techniques, discussed in [6], target the identification of speedpaths in fabricated IC instances by correlating silicon delay measurements with timing and functional models. In a slightly different approach, [1] employs learning techniques on a small set of examples for improving circuit-wide speedpath prediction. Another approach, recently proposed in [8], uses real delay measurements, obtained from fabricated IC instances, to rank the most likely combinations of speedpaths, from within a set of user-supplied paths. This approach assumes that the timing models are accurate, and therefore proposes that the most likely combinations of speedpaths are the ones

DAC 2010, June 13-18, 2010, Anaheim, California, USA.

where the sum of quadratic errors between the model and the measurements is the smallest.

This paper proposes a methodology that enables the extraction, for each primary output of a combinational IC block, of a reduced set of paths that is *guaranteed* to contain the corresponding silicon speedpath. Unlike [8], no tedious path selection is required by the user, since only the timing graph of the block and the silicon delay measurements [6] must be provided. Additionally, our approach takes the realistic assumption of inaccuracies in the timing model. The resulting set of candidate speedpaths can be used for detailed analysis or even be fed back to [8] for ranking.

The remainder of this paper is organized as follows. Section 2 introduces the parametric timing models underlying the work present in this paper. Section 3 proposes an extension to [3] that enables the efficient computation of the top k critical timing corners and paths of a given IC block. Section 4 proposes a methodology for, given silicon delay measurements of an IC block, computing a reduced subset of paths that is guaranteed to contain all its speedpaths. The experimental results are discussed in Section 5. Finally, Section 6 presents brief concluding remarks.

2. PARAMETRIC TIMING MODELING

The timing information of a circuit is modeled by a *timing graph* $G = (V, E)$, where *vertices*, $v \in V$, correspond to pins in the circuit, and directed *edges*, $e \in E$, correspond to pin-to-pin delays in cells or interconnect. The *primary inputs*, $u \in PI(G)$, are vertices with no incoming edges. All vertices with no outgoing edges are *primary outputs*, $w \in PO(G)$, but there may also be primary outputs with outgoing edges. A *complete path* is a sequence of edges, connecting a primary input to a primary output. A *path* is a sequence of edges connecting any two vertices

Each edge of the timing graph is annotated with the corresponding *delay*, resulting from a delay calculation procedure, whereby slews are forward propagated across the circuit and, using appropriate cell and interconnect models, the delays and output slews for each component are computed [4]. Since it is out of the scope of this paper to discuss the delay computation procedure, in the following, we will assume that the timing information of any circuit is already made available in the form of an annotated timing graph.

This work assumes a parametrized static timing analysis (PSTA) model [10, 8, 9], where delays are described by affine functions of process and operational parameter variations, corresponding to a first-order linearization of every delay, d, around a nominal point, λ_0, in the parameter space. Considering the parameter space to have size p, and representing d as a function of the incremental parameter variation vector, $\Delta\lambda = \lambda - \lambda_0$, around a nominal value λ_0, we obtain

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

where $d_0 = d(\lambda_0)$ is the nominal value of d and d_i is the sensitivity of d to parameter λ_i, $i = 1, 2, \ldots, p$, computed at the nominal point λ_0.

When delays are given in the form of Eqn. (1), arrival times can be exactly represented by piecewise-affine functions, since they are the result of a sequence of min/max and sum operations between piecewise-affine functions and affine functions. Affine functions are *convex* [2]. Further, since both min/max and sum operators produce convex functions when operating on convex functions, the resulting piecewise-affine arrival time functions are also convex. In the context of timing analysis, convexity implies that the smallest/largest delay or arrival time is obtained by setting each parameter to one of its extreme values. For delays, that are represented by affine functions, this value is fairly easy to compute. Assuming that $\Delta\lambda_i \in [\Delta\lambda_i^{min}, \Delta\lambda_i^{max}]$, if in Eqn. (1) we set to $\Delta\lambda^{max}$ the parameter variations with positive sensitivities, and to $\Delta\lambda^{min}$ the remaining ones, we are

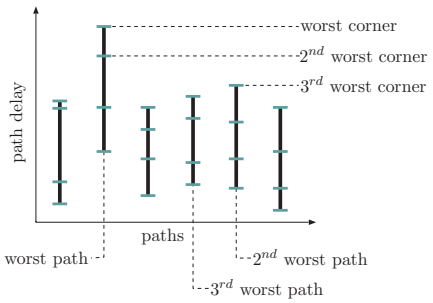

Figure 1: Example of worst-delay paths and corners.

maximizing the value of the affine delay function over the parameter space, therefore obtaining the maximum value

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

where the maximizing parameter variation assignment is

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

The min can be computed by symmetry. For affine functions this computation takes linear time in the number of parameters, however, for piecewise-affine functions this computation is quite expensive, since it requires an implicit or explicit enumeration of all the 2^p possible solutions (corners), which makes it exponential in the number of parameters.

3. K-MOST CRITICAL PATHS/CORNERS

Since path delays are affine functions, they can assume any value between their minimum and maximum values, that occur at the corners specified by Eqn. (3) and its symmetrical. Figure 1 illustrates the range of possible path delay values for 6 paths of a given circuit, where 2 parameters were considered (and therefore 4 corners exist). Vertical lines correspond to the delay range of each path. Small horizontal lines mark the path delay values at each of the 4 corners.

The *worst-delay path*, or speedpath, of a circuit is the path that exhibits the worst delay, among all paths, for every possible assignment of the process parameter variation vector. As discussed in Section 2, the worst delay of any given path, and also of the speedpath, is achieved in a given corner of the process parameter variation space. Therefore, *worst-delay corner* is the corner of the process parameter variation space that produces the worst delay in the worst-delay path, as can be observed in Figure 1. However, the second worst-delay corner can occur in the worst-delay path or in the second worst-delay path. In this illustration it occurs also in the worst-delay path, and the third worst-delay corner occurs in the second worst-delay path, and so on.

The brute-force method for computing the k-worst delay paths would consist in extracting the worst-delay from the parametric delay formula of every path, by applying Eqns. (2,3), and subsequently picking the paths that produced the k-worst values of the resulting path delay set. This procedure is clearly unfeasible since the number of paths can grow exponentially with the number of vertices. Computing the k-worst delay corners in a simplistic manner would involve timing the circuit for every corner and subsequently picking the corners that produced the k-worst values of the resulting circuit delay set. This approach would also be unfeasible, due to the exponential number of corners involved.

3.1 Worst-Delay Corner Computation

This section presents a brief overview of the automated methodology proposed in [3] for computing the worst-delay path/corner of a circuit, given a linear parametric characterization of cell and interconnect delays. This formulation will be subsequently extended to enable the exact computation of the k most critical paths of a circuit.

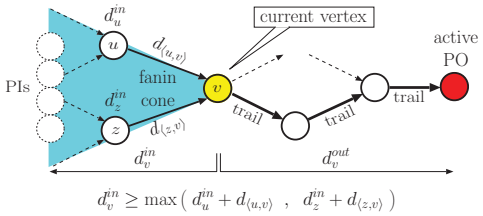

Figure 2: Illustration of delay estimates.

Consider the timing graph of a combinational block with n inputs and m outputs. Assuming that delays, annotated in edges, are affine functions of the process parameters, as in Eqn. (1), then any delay, $d_{i,j}(\Delta\lambda)$, from input i to output j can be accurately represented by a piecewise-affine function [3]. The *worst-delay corner* (WDC) problem, consists in computing an assignment, $\Delta\lambda^*$, to the parameter variation vector, $\Delta\lambda$, that produces the worst delay, $d_{i,j}(\Delta\lambda)$, from any input $i = 1,\ldots,n$ to any output $j = 1,\ldots,m$. In the following, and for the sake of clarity, we will assume *late* mode operation, meaning that the worst delay will be the *largest* delay. Assuming that $d_{i,j}(\Delta\lambda)$ is the piecewise-affine function of the delay from primary input i to primary output j, then the WDC problem can be formulated as

$$\max_{\Delta\lambda}\left\{\max_{j=1,\ldots,m}\left[\max_{i=1,\ldots,n} d_{i,j}(\Delta\lambda)\right]\right\} \tag{4}$$

The simplest exhaustive algorithm for computing the WDC consists in calculating the affine delay function of each path in the circuit, by adding the affine delay functions of its edges, and subsequently computing the corner that produces the worst delay for each path, using Eqns. (2) and (3). The WDC is the corner that produces the worst delay among all paths. This procedure exhibits linear run-time complexity in the number of paths and parameters. However, since the number of paths can grow exponentially with the number of vertices, this procedure can have exponential run-time complexity in the number of vertices, thus becoming overly expensive even for moderately sized circuits.

In order to avoid the detailed analysis of every path in the circuit, [3] proposes the use of branch-and-bound techniques to prune the timing graph, thus significantly reducing the number of paths requiring detailed analysis. Considering a primary output at a time, the technique proposed in [3] performs an *explicit* or *implicit* analysis of all the *complete paths* that end at that primary output, that we will designate as the *active* primary output. The timing graph is traversed in a backward fashion, starting at the active primary output, going through the internal vertices, and eventually ending at the primary inputs (if no pruning is performed). The vertex being visited in a given step is designated by *current vertex*. The path taken to reach that vertex from the active primary output is designated by *trail*. If reconvergent fanouts exist, the same vertex can be reached from the same primary output, through distinct trails. The worst delay, d^*, found among the complete paths already analyzed is continuously updated, as well as the corresponding corner, $\Delta\lambda^*$. For each current vertex of the timing graph, v, the algorithm relies on three parametric delay estimates:

- d_v^{in} is an upper bound on the delay from any primary input to vertex v (e.g. in the fanin cone of v);
- d_v^{out} is the delay of the trail;
- $d_v^{path} = d_v^{in} + d_v^{out}$, which represents an upper bound on the delay of any complete path going through v, that contains the trail.

The affine expression of d_v^{in} is calculated beforehand, through a block-based analysis of the timing graph, for which the bounding technique proposed in [9] can be used, as well as other bounding techniques. An illustration of these estimates is presented in Figure 2. The fanin cone of a vertex v is pruned when the following condition is verified.

$$\max_{\Delta\lambda}[d_v^{path}(\Delta\lambda)] \leq d^* \tag{5}$$

In this case the delay of *any* complete path going through v and containing the trail can never be larger than the worst delay, already computed for some other complete path, d^*, therefore it is useless to further explore the fanin cone of v, as the worst delay, d^*, cannot be improved by such action. It should be noted that this pruning is only valid for a specific trail. If v is subsequently visited through another trail its fanin cone may not be pruned. If a given current vertex is not pruned all its fanin vertices are scheduled to be visited. When the current vertex is a primary input, the trail is a complete path, from a primary input to a primary output, being d_v^{out} its affine delay function. If the worst delay of this path is worst than the worst delay d^* found so far, then both d^* and $\Delta\lambda^*$ are updated. The procedure terminates when every path of the circuit is either explicitly visited or pruned. On termination, the procedure returns the worst delay, d^*, as well as the worst-delay corner, $\Delta\lambda^*$. The worst-delay path can also be returned if the trail is stored on every update of d^* and $\Delta\lambda^*$. A similar set of branch-and-bound techniques can be applied for searching the parameter space [3].

3.2 Modified Computational Procedure

The procedure reviewed in Section 3.1 can be slightly modified to enable the extraction of the k-worst delay paths rather than just the most critical path.

First, it is necessary to have a data structure, possibly a list, of size k, for storing all the k paths, as well as their associated worst delays. For convenience, in this data structure the paths will be ordered by ascending order of their worst delays. This means that among the elements stored in the data structure, the least critical will be in the first position and the most critical will be in the last position.

Surprisingly, the modifications that need to be introduced in the algorithm described in Section 3.1 to enable the computation of the k most critical paths are very few. Initially, the data structure described in the previous paragraph is empty. During the execution of the algorithm, while the number of paths stored in the data structure is smaller than k, no pruning is performed, and therefore all complete paths visited are stored in the data structure. When the number of elements in the data structure reaches k, the pruning is activated. Afterwards, the pruning will be performed using the smallest worst delay present in the data structure as the threshold, d^*. When a given path exhibits a worst delay larger than the smallest worst delay present in the data structure, that path is inserted in the proper position in the data structure. Subsequently, the first path, which exhibits the smallest worst delay among all the paths in the data structure will be removed, in order to keep at most k elements at any time. On termination, the k most critical paths are stored in the data structure.

Clearly, the amount of pruning is correlated with the value of k. For larger values of k, more paths are stored in the data structure, and therefore the first path will have a smaller worst delay value. This means that the pruning threshold, d^*, will be a smaller number, and consequently less paths get pruned. It is therefore expected that larger values of k may lead to larger execution times.

The procedure proposed in [3] for computing the worst-delay corner can be modified in a similar manner to enable the computation of the k-worst delay corners.

4. SILICON SPEEDPATH SELECTION

This section proposes a methodology that, given a timing graph of a combinational IC block and silicon delay measurements [6] at the primary outputs of that block, extracts a reduced set of paths, for each primary output, that is guaranteed to contain the corresponding silicon speedpath.

4.1 Model Uncertainty

In the following we will assume that the delays computed by our timing model, detailed in Section 2, are not entirely accurate when compared to the "real" delays of the fabricated circuit. Therefore, if $d'(\Delta\lambda)$ is the delay of a given element of the fabricated circuit, at the parameter variation setting, $\Delta\lambda$, of the fabrication process, and $d(\Delta\lambda)$ is the corresponding delay given by the timing model, then we assume them to be related by the following expression,

$$d'(\Delta\lambda) = d(\Delta\lambda) + \varepsilon \tag{6}$$

where $\varepsilon \in [\varepsilon^{min}, \varepsilon^{max}]$. This range is intended to account for the uncertainty associated to the timing model. The values of ε^{min} and ε^{max} can be independently set for each edge delay in the model. This enables designers to set larger ranges for elements, or regions of the circuit, which they know to be modeled less accurately, and smaller ranges where model accuracy is known to be good. Clearly, the wider the range $[\varepsilon^{min}, \varepsilon^{max}]$, the less effective the speedpath selection techniques proposed in the following sections will be.

4.2 Delay Threshold Pruning

We start by computing for each vertex v its d_v^{in} estimate (see Section 3.1), which must now consider the errors associated with all the delays in the fanin cone of v. Afterwards, for every primary output with a measured silicon delay D, we perform a pruning procedure similar to the one reported in Section 3.1, but where the fanin cone of a vertex v gets pruned if the following condition holds,

$$\max_{\Delta\lambda}[d_v^{path}(\Delta\lambda)] + \varepsilon_{path}^{max} < D \tag{7}$$

This means that the delay of *any* complete path going through v and containing the trail can never reach D, even considering model uncertainty. Therefore, none of such paths can be the speedpath of the corresponding primary output.

This procedure leads to the elimination of a huge number of paths, thus becoming feasible to enumerate the remaining ones, that are stored in an efficient data structure, containing the candidate speedpaths for each primary output. The steps described in remainder of this section will target the elimination of as many candidate speedpaths as possible.

4.3 Domination

If the delay of a candidate speedpath b, $d'_b(\Delta\lambda)$, is always larger than the delay of another candidate speedpath a, $d'_a(\Delta\lambda)$, both related to the same primary output, then a cannot be the speedpath, and thus can be eliminated from the candidate list. In this case we say that $d'_b(\Delta\lambda)$ *dominates* $d'_a(\Delta\lambda)$. Formally,

$$d_b(\Delta\lambda) + \varepsilon_{d_b}^{min} > d_a(\Delta\lambda) + \varepsilon_{d_a}^{max} \tag{8}$$

This condition can be trivially verified by checking for,

$$\min_{\Delta\lambda}[d_b(\Delta\lambda) - d_a(\Delta\lambda)] + \varepsilon_{d_b}^{min} - \varepsilon_{d_a}^{max} > 0 \tag{9}$$

Since this domination check is cheap, all the candidate speedpaths, corresponding to the same primary output, are checked against each other for domination. The dominated candidates are then removed from the candidate list.

Even if the delay of a candidate speedpath a is not dominated by the delay of another candidate speedpath b, it can be dominated by the delays of all the other candidate speedpaths $b, c, \ldots$ together. When that occurs, that candidate speedpath can still be eliminated. This condition can be verified by detecting if there is no assignment for $\Delta\lambda$ such that $d'_a(\Delta\lambda)$ is simultaneously not smaller than the delays of all the other candidate speedpaths, $d'_b(\Delta\lambda), d'_c(\Delta\lambda), \ldots$, corresponding to the same primary output. This condition holds if the following LP is infeasible [7],

$$\begin{aligned} \text{s.t.} \quad & d_a(\Delta\lambda) + \varepsilon_{d_a}^{max} \geq d_b(\Delta\lambda) + \varepsilon_{d_b}^{min} \\ & d_a(\Delta\lambda) + \varepsilon_{d_a}^{max} \geq d_c(\Delta\lambda) + \varepsilon_{d_c}^{min} \\ & \ldots \\ & \Delta\lambda_i^{min} \leq \Delta\lambda_i \leq \Delta\lambda_i^{max}\ ,\ i = 1, 2, \ldots, p \end{aligned} \tag{10}$$

4.4 Parameter Range Adjustment

Given the list of candidate speedpaths for a primary output, and its silicon delay measurement, D, then the delay of any of the candidates cannot exceed D. Formally,

$$d(\Delta\lambda) + \varepsilon^{min} \leq D \tag{11}$$

This condition can be used to compute tighter bounds on the parameter variation ranges. The bound, α_k, for a given parameter variation, $\Delta\lambda_k$, is given by,

$$\alpha_k = \frac{D - d_0 - \sum_{i=1, i\neq k}^{p} d_i \overline{\Delta\lambda_i^*} - \varepsilon^{min}}{d_k} \tag{12}$$

where $\overline{\Delta\lambda_i^*}$ is the symmetrical of Eqn. (3) (e.g. the minimizing parameter assignment). If d_k is positive, then $\Delta\lambda_k \leq \alpha_k$. Conversely, if d_k is negative, then $\Delta\lambda_k \geq \alpha_k$. This simple check is performed for every candidate speedpath.

4.5 Consistency

For a given candidate path a to be a speedpath, there must be an assignment to $\Delta\lambda$, that makes $d'_a(\Delta\lambda)$ *exactly* equal to the measured delay in its corresponding primary output, D_a, and that makes the delays of all other candidate speedpaths, $d'_b(\Delta\lambda), d'_c(\Delta\lambda), \ldots$, not larger than the measured delay at their corresponding primary outputs, $D_b, D_c, \ldots$. This problem can be formulated as the following LP,

$$\begin{aligned} \text{s.t.} \quad & d_a(\Delta\lambda) + \varepsilon_a^{max} \geq D_a \\ & d_a(\Delta\lambda) + \varepsilon_a^{min} \leq D_a \\ & d_b(\Delta\lambda) + \varepsilon_b^{min} \leq D_b \\ & d_c(\Delta\lambda) + \varepsilon_c^{min} \leq D_c \\ & \ldots \\ & \Delta\lambda_i^{min} \leq \Delta\lambda_i \leq \Delta\lambda_i^{max}\ ,\ i = 1, 2, \ldots, p \end{aligned} \tag{13}$$

When the LP is feasible, the candidate speedpath a is said to be *consistent* with the other candidates, and is kept. Otherwise, it is eliminated from the candidate list.

4.6 Compatibility

A set of candidate speedpaths, $a, b, c, \ldots$, corresponding to different primary outputs, are said to be *compatible* if, for a some $\Delta\lambda$, they can all be the silicon speedpath of their corresponding primary output. Clearly, the true silicon speedpaths, for all the primary outputs, must be compatible. Compatibility check between candidate speedpaths $a, b, c, \ldots$ can be performed by finding a feasible solution for the following LP,

$$\begin{aligned} \text{s.t.} \quad & d_a(\Delta\lambda) + \varepsilon_a^{max} \geq D_a \\ & d_a(\Delta\lambda) + \varepsilon_a^{min} \leq D_a \\ & d_b(\Delta\lambda) + \varepsilon_b^{max} \geq D_b \\ & d_b(\Delta\lambda) + \varepsilon_b^{min} \leq D_b \\ & d_c(\Delta\lambda) + \varepsilon_c^{max} \geq D_c \\ & d_c(\Delta\lambda) + \varepsilon_c^{min} \leq D_c \\ & \ldots \\ & \Delta\lambda_i^{min} \leq \Delta\lambda_i \leq \Delta\lambda_i^{max}\ ,\ i = 1, 2, \ldots, p \end{aligned} \tag{14}$$

Checking for compatibility for all possible combinations of candidate speedpaths can be overly expensive. Therefore, relying on the fact that unconstrained versions of larger problems can be generated, a branch-and-bound procedure may be employed, similarly to the approach followed by [8].

5. EXPERIMENTAL RESULTS

The techniques proposed in Sections 3 and 4 were coded in C++. Benchmark circuits, from the ISCAS combinational suite were synthesized and mapped to a $90nm$ industrial technology. As process parameters, we considered the widths and thicknesses of the 8 metal routing layers, resulting in a total of 16 parameters. Variational cell and interconnect delay computation was performed using the procedure described in [4]. Finally, for each circuit a timing graph was generated and affine cell and interconnect delays were annotated as edge properties. Table 1 presents a brief characterization of the resulting benchmark circuits, where "#PI" and "#PO" columns report the number of primary inputs and outputs, "#Logic" and "#Net" columns report the number of combinational cells and the number of nets, and "#Vertex" and "#Edge" report the number of vertices and edges in the corresponding timing graph. "#Path" reports the number of complete paths, considering all Rise/Fall combinations.

All the results presented in this section were obtained in a machine with an Intel Xeon @ 2.33GHz and 28GB of RAM. For all the runs a single processor was used and none of them took more than 200MB of memory. For the solution of linear problems we have used LPSolve 5.5 API.

The left plot of Figure 3 presents the worst delay values for the top 1000 critical paths in c6288. As can be observed, there is a fair number of paths that exhibit a worst delay close to the worst delay of the most critical path. The right plot of Figure 3 presents the circuit delay values for the top 1000 critical corners in c6288. Clearly, there is a group of approximately 70 corners that exhibit a circuit delay very close to the circuit delay of the most critical corner. Both of these observations justify the necessity of identifying sets of speedpaths and critical corners, rather than just one speedpath and the most critical corner.

Table 2 presents results for the computation of the top k speedpaths and critical corners, as discussed in Section 3, where the column names with "1", "100" and "1000" indicate the number k of top speedpaths or critical corners requested. "#S" reports the amount of search (visited vertices or analyzed corners). "CPU" reports the total run-time in seconds. As can be observed, the top 100 and 1000 speedpaths can still be computed in a small amount of time, even though pruning is obviously reduced ("#S" is larger). Even though much less efficient, the computation of the top 100 and 1000 critical corners can still be performed in a fair amount of time. That seems to hold even for c6288, that exhibits an abnormally large number of paths.

Table 3, presents results for the silicon speedpath selection algorithm proposed in Section 4. Since we did not have easy access to silicon delay measurements, we have generated an "emulation" of such measurements by timing the circuits for an unbiased $\Delta\lambda$ assignment (e.g. including parameter variations with values above/below and far/close from their nominal value). Additionally, we have added to each edge delay a random value, following a uniform distribution with range $[\varepsilon^{min}, \varepsilon^{max}]$, in order simulate model inaccuracies. The arrival times at the primary outputs were used as silicon delay "measurements". The columns "#P", "#D", "#I" and "#R" report the number of delay threshold-pruned paths, the number of dominated paths, the number of inconsistent paths and the remaining paths, that were not pruned, and that are viable speedpath candidates. The "CPU" column reports the run-time in seconds.

Analyzing the results in Table 3 we can conclude that the procedure proposed in Section 4 is able to select a small percentage of the total number of paths of each circuit, except the huge c6288. While the delay threshold pruning strategy seems to be, by far, the most effective; domination and inconsistency checks also seem to be able to filter a non-negligible amount of paths. Parameter adjustments are not reported, as only the range of one parameter was adjusted, for c7552. As expected, performance is highly depend on the error range. CPU times are not too small, but given the complexity of the task, we believe that they are acceptable. An interesting feature of these techniques is that, by imposing a smaller timeout to the LP feasibility checks, the user is able to reduce run-time at the expense of obtaining a larger number of candidate paths. A large amount of CPU time is spent checking for LP feasibility. Since LPSolve is not particularly efficient in this task, there is opportunity for further performance improvements by using a commercial solver, such as CPLEX.

6. CONCLUSIONS

This paper proposes an exact, yet efficient, method for computing the top k critical paths (speedpaths) and the top k critical corners of a digital IC, considering variability. The computation of the top critical paths is of particular interest in speedpath debugging, as there are several paths with a worst delay close to the worst delay of the most critical path. Additionally, the computation of the top critical corners can provide an automated process for picking the set of corners that will be used during timing sign-off. Furthermore, this paper also proposes a set of techniques that, given a timing graph of a combinational IC block and silicon delay measurements at the primary outputs of that block, are able to extract a reduced set of paths, for each primary output, that is guaranteed to contain the corresponding silicon speedpath. Such techniques are easy to parallelize for modern multicore systems, thus enabling their application to larger and more complex industrial designs.

7. REFERENCES

[1] P. Bastani, K. Killpack, L.-C. Wang, and E. Chiprout. Speedpath prediction based on learning from a small set of examples. In *Proceedings of DAC*, pages 217–222, Anaheim, CA, June 2008.

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

[3] L. G. e Silva, L. M. Silveira, and J. R. Phillips. Efficient Computation of the Worst-Delay Corner. In *Proceedings of DATE*, pages 1617–1622, Nice, France, April 2007.

[4] L. G. e Silva, Z. Zhu, J. Phillips, and L. M. Silveira. Variation-Aware, Library Compatible Delay Modeling Strategy. In *Proceedings of the IFIP VLSI-SoC Conference*, October 2006.

[5] C. Kashyap, P. Bastani, K. Killpack, and C. Amin. Silicon feedback to improve frequency of high-performance microprocessors - an overview. In *Proceedings of ICCAD*, pages 778–782, San Jose, CA, November 2008.

[6] K. Killpack, C. Kashyap, , and E. Chiprout. Silicon Speedpath Measurement and Feedback into EDA Flows. In *Proceedings of DAC*, pages 390–395, San Diego, CA, June 2007.

[7] S. Kumar, C. Kashyap, and S. Sapatnekar. A Framework for Block-Based Timing Sensitivity Analysis. In *Proceedings of DAC*, pages 688–693, Anaheim, CA, June 2008.

[8] S. Onaissi, K. Heloue, and F. Najm. PSTA-Based Branch and Bound Approach to the Silicon Speedpath Isolation Problem. In *Proceedings of ICCAD*, pages 217–224, San Jose, CA, November 2009.

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

[10] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proceedings of DAC*, pages 331–336, San Diego, CA, June 2004.

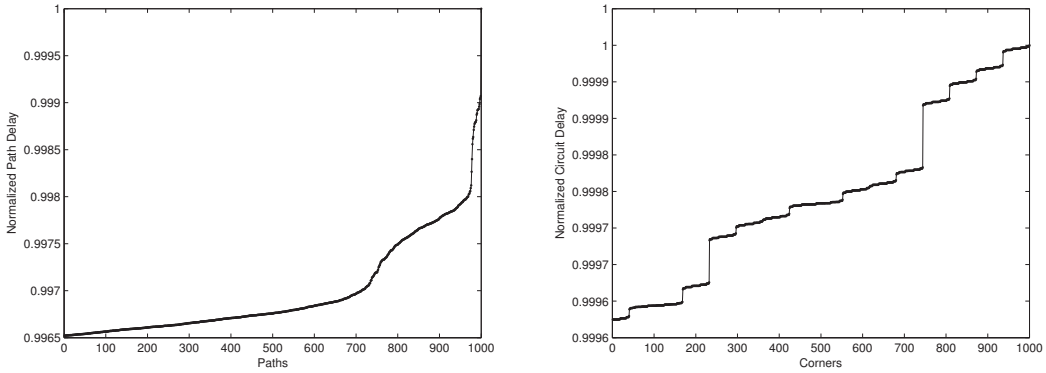

Figure 3: Worst delay of the top 1000 critical paths and of the top 1000 critical corners, in c6288 (note that the plots are in different scales).

Design	#PI	#PO	#Logic	#Net	#Vertex	#Edge	#Path
c17	5	2	4	9	21	22	22
c432	36	7	88	124	415	575	550302
c499	41	32	133	174	633	886	253056
c880	60	26	147	207	674	908	13862
c1355	41	32	133	174	633	886	253056
c1908	33	25	178	211	756	1065	1011690
c2670	157	64	282	515	1321	1700	27426
c3540	55	22	443	494	1882	2756	9035812
c5315	178	123	554	734	2644	3701	1247156
c6288	32	32	1584	1653	5131	6998	90242948060
c7552	206	107	820	1031	3483	4807	979140

Table 1: Benchmark characterization.

Design	1 Path		100 Paths		1000 Paths		1 Corner		100 Corners		1000 Corners	
	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU
c17	20	< 0.01	72	< 0.01	72	< 0.01	3463	0.33	7761	0.71	12033	1.15
c432	510	< 0.01	6646	0.02	41664	0.31	853	2.46	1707	4.51	6435	17.39
c499	459	0.01	4702	0.03	19821	0.13	405	2.32	1271	6.66	6473	32.69
c880	523	< 0.01	4695	0.01	16900	0.15	315	1.60	1337	6.43	6371	30.58
c1355	354	0.01	4523	0.01	19962	0.18	647	3.39	1573	7.98	6483	33.75
c1908	755	< 0.01	10124	0.05	54852	0.47	301	2.05	799	4.95	5679	35.35
c2670	595	0.02	6411	0.03	27629	0.22	239	3.04	1077	13.30	6941	85.81
c3540	868	0.03	24605	0.11	131781	0.86	559	12.76	1735	39.08	6759	146.45
c5315	671	0.03	10789	0.08	55575	0.40	205	7.80	821	30.33	5799	213.71
c6288	18159	0.12	1056387	4.54	7878747	53.00	241	25.40	803	83.46	5677	580.72
c7552	746	0.04	11103	0.11	91632	0.68	411	23.65	1439	83.10	6453	365.96

Table 2: Results for the exact computation of the top critical paths/corners.

Design	$\varepsilon \in [-1\%\ d_0\ ,\ +1\%\ d_0]$					$\varepsilon \in [-3\%\ d_0\ ,\ +3\%\ d_0]$				
	#P	#D	#I	#R	CPU	#P	#D	#I	#R	CPU
c17	17	0	0	5	<0.01	16	1	0	5	< 0.01
c432	542175	7823	119	185	0.94	534859	11070	2027	2346	192.94
c499	249758	2642	78	578	4.02	247074	608	1408	3966	265.30
c880	13607	106	25	124	0.22	13358	234	86	184	0.64
c1355	249216	2928	212	700	7.09	245356	2012	1120	4568	350.68
c1908	1006661	4092	345	592	7.44	997977	3767	5144	4802	944.29
c2670	27276	66	17	67	0.08	27225	56	38	107	0.21
c3540	9026219	9232	155	206	1.24	9013233	17063	3584	1932	241.40
c5315	1239613	6747	267	529	5.65	1229638	10584	4434	2500	420.90
c6288	-	-	-	-	> 1000	-	-	-	-	> 1000
c7552	976853	1888	52	347	1.58	974982	2614	590	954	20.07

Table 3: Results for silicon speedpath selection.