Efficient Computation of the Worst-Delay Corner

Luis Guerra e Silva, L. Miguel Silveira
Cadence Labs / INESC-ID
IST / TU Lisbon
Lisbon, Portugal
{lgs,lms}@inesc-id.pt

Joel R. Phillips
Cadence Berkeley Labs
Cadence Design Systems
San Jose, CA 95134, U.S.A.
jrp@cadence.com

Abstract

Timing analysis and verification is a critical stage in digital integrated circuit design. As feature sizes decrease to nanometer scale, the impact of process parameter variations in circuit performance becomes extremely relevant. Even though several statistical timing analysis techniques have recently been proposed, as a form of incorporating variability effects in traditional static timing analysis, corner analysis still is the current timing signoff methodology for any industrial design. Since it is impossible to analyze a design for all the process corners, due to the exponential size of the corner space, the design is usually analyzed for a set of carefully chosen corners, that are expected to cover all the worst-case scenarios. However, there is no established systematic methodology for picking the right worst-case corners, and this task usually relies on the experience of design and process engineers, many times leading to over design. This paper proposes an efficient automated methodology for computing the worst-delay process corners of a digital integrated circuit, given a linear parametric characterization of the gate and interconnect delays.

1 Introduction

As integrated circuit feature sizes decrease, the impact of process and operational parameter variations in circuit performance becomes very significant [7]. Under these circumstances, proper timing of the circuit is now considerably harder to predict and ensure. The ability to accurately identify the parameter settings that correspond to critical timing conditions is therefore increasingly important.

Statistical timing analysis (STA) has been introduced as a form of incorporating variability effects in traditional static timing analysis. Even though several promising STA modeling techniques have been proposed [3, 1, 9], their practical applicability is still quite limited, as their usage could ultimately entail an overhaul of the timing verification flow [5]. Industrial tools and design flows are not yet prepared to handle statistical information, and EDA companies are still evaluating the best ways to incorporate it in their products and design flows, as it does not represent a natural extension to traditional static timing flows. Further, STA requires complex parameter characterization, like multidimensional statistical distributions, and most foundries still do not provide that information on their fabrication technologies in a consistent manner, either due to confidentiality issues, or simply because they are still trying to figure out themselves the best way to represent and convey such information. Additionally, most often process control can be best described by ranges, rather than statistics.

Even though STA is not yet a mature methodology, mainly due to poor parameter characterization and lack of tool support, the *parametric* delay and slew formulations that it prescribes, can be used to determine the parameter settings that correspond to the critical timing conditions of a circuit. Since it is impossible to analyze a design for all possible parameter settings, the design is usually analyzed for a small set of carefully selected settings, that are expected to cover the worst-case fabrication and operation scenarios. These settings are usually designated as *corners*, since they correspond to extreme conditions. Unfortunately, picking the right corners in a realistic manner is not trivial and most often than not either such corners are missed or gross overdesign may happen.

This paper proposes an efficient automated methodology for computing the exact worst-delay corner of a digital integrated circuit, given a parametric characterization of the cell and interconnect delays. In our approach, parameters only need to be characterized by their respective value ranges, as opposite to STA where they need to be characterized by statistical distributions. Additionally, our approach produces meaningful and insightful information for the designer, like corners and specific circuit paths where they induce critical timing conditions, which makes it much more useful in effectively guiding manual or automated circuit optimization than other approaches.

Recently, [6] proposed a linear-time approach for timing analysis that computes a delay upper bound estimate, covering all process corners. No matter how tight, this estimate is just an approximation, and the worst-case corner for the delay upper bound may not be the true worst-delay corner of the circuit. The goal of our work is quite different, as we target the determination of the *exact* worst-delay corner and associated paths.

The paper is organized as follows. Section 2 introduces a few basic concepts. Section 3 formulates the worst-delay corner problem and discusses possible exhaustive solutions. Section 4 proposes the use of branch-and-bound techniques. Section 5 discusses a few practical issues. Finally, Section 6 presents the experimental results and Section 7 presents some concluding remarks.

978-3-9810801-2-4/DATE07 © 2007 EDAA

2 Background

2.1 Timing Analysis

The timing information of a circuit is usually modeled by a *timing graph*, where vertices model pins in the circuit, and edges to pin-to-pin delays in cells or interconnect. Each edge is annotated with the corresponding *delay*. Some vertices are annotated with *timing constraints*, such as required arrival times. The timing graph is the result of a delay calculation procedure. Since its discussion is out of the scope of this paper, we assume that the timing information of a circuit is made available in the form of a timing graph.

Two main approaches have been proposed for timing analysis: block-based and path-based. In the block-based approach, characterized by linear runtime, arrival times are pushed through the timing graph in a levelized fashion, performing sum operations with delays over the edges and min/max operations over the vertices with multiple incoming edges. The alternative path-based approach consists in individually computing the delay of each path in the circuit by adding the delay of each of its edges. Even though more accurate, this approach is computationally much more expensive than the former, since the number of paths can grow exponentially with the number of vertices (pins).

In the following, we shall consider a *timing graph* as a directed acyclic graph, $G = (V, E)$, composed of *vertices*, $v \in V$, and directed *edges*, $e \in E$, connecting them. The *primary inputs* are vertices with no incoming edges. All vertices with no outgoing edges are *primary outputs*, but there may also be primary outputs with outgoing edges. The sets of primary inputs and outputs of G are respectively $PI(G)$ and $PO(G)$. A *complete path* is a sequence of edges, connecting a primary input to a primary output, and will be referred to simply as a *path*. A *partial path* is a sequence of edges connecting any two vertices.

2.2 Parametric Formulation

In this work, instead of assuming delays to be constant real-numbered values, we assume them to be described by affine functions [8] of process/operational parameter variations, corresponding to a first-order linearization of every delay, d, around a nominal point, λ_0, in the parameter space,

$$d(\lambda - \lambda_0) = d(\lambda_0) + \left.\frac{\partial d}{\partial \lambda}\right|_{\lambda_0} (\lambda - \lambda_0) = d(\lambda_0) + \left.\frac{\partial d}{\partial \lambda}\right|_{\lambda_0} \Delta\lambda \tag{1}$$

where $\Delta\lambda = \lambda - \lambda_0$, represents the incremental parameter variation vector. Considering the parameter space to have size p, Eqn. (1) can be rewritten more compactly as

$$d(\Delta\lambda) = d_0 + \sum_{i=1}^{p} d_i \Delta\lambda_i = d_0 + d^T \Delta\lambda \tag{2}$$

where d_0 is the nominal value of d, computed at the nominal values of the parameters, λ_i, $i = 1, 2, \ldots, p$, and d_i is the sensitivity of d to parameter λ_i, computed at the nominal point λ_0. Information on parametric delay computation can found in [4]. The application of a linear parametric formulation in the context of statistical timing analysis was first proposed in [9]. This representation is mathematically equivalent to the canonical formulation prescribed in [9], but the interpretation and subsequent treatment is, as we shall see, quite different. Throughout this paper, and without loss of generality, we will assume that all the parametric formulas have been normalized such that $\Delta\lambda \in [0,1]^p$.

2.3 Affine Operations

The max of affine functions is a piecewise-affine function, and the max of piecewise-affine functions is also a piecewise-affine function. Similarly, the sum of affine functions is an affine function, and the sum of a piecewise-affine function and an affine function is a piecewise-affine function. Therefore, any arrival time can be exactly represented by a piecewise-affine function, since it is the result of a sequence of max and sum operations between piecewise-affine functions and affine function. If no simplification is performed, the piecewise-affine representation of arrival times should grow linearly with the number of paths, and therefore can be exponential in the number of vertices.

Affine functions are *convex* [2]. An important property of the max operator over affine functions, or convex piecewise-affine functions, is that it always produces *convex* functions. The same applies to the sum operator. The convexity implies that *the largest value for a given affine or piecewise-affine function is obtained by setting each variable to one of its extreme values.* In the context of timing analysis this corresponds to state that the largest delay or arrival time is obtained by setting each parameter to one of its extreme values, in this case either 0 or 1. For the simple case of delays, that are represented by affine functions, this value is fairly easy to compute. If, in Eqn. (2), we set to 1 all the parameter variations with positive sensitivities, and to 0 the remaining ones, we are maximizing the value of the affine delay function over the parameter space, and therefore we obtain the maximum value,

$$\max_{\Delta\lambda}[d(\Delta\lambda)] = d(\Delta\lambda^*) = d_0 + \sum_{i=1}^{p} d_i \Delta\lambda_i^* \tag{3}$$

where the maximizing parameter variation assignment is

$$\Delta\lambda_i^* = \begin{cases} 0 & if\ d_i \leq 0 \\ 1 & if\ d_i > 0 \end{cases}, \; i = 1, 2, \ldots, p \tag{4}$$

The minimum value can be computed by replacing $\Delta\lambda_i^*$ by $(1 - \Delta\lambda_i^*)$ in Eqn. (3). For affine functions this computation takes linear time in the number of parameters, however, for piecewise-affine functions (that we use for arrival times) this computation is much more expensive, since it requires an implicit or explicit enumeration of all the 2^p possible corners, making it exponential in the number of parameters.

3 The Worst-Delay Corner Problem

3.1 Formulation

The *worst-delay corner* (WDC) problem, consists in computing the assignment (corner) to the parameter variation vector, $\Delta\lambda$, that maximizes the largest arrival time among all the primary outputs of a given circuit. As we have seen, since arrival times are represented by piecewise-affine functions which are convex, their largest value is obtained by setting each parameter variation to one of its extreme values. Therefore, in essence this problem can be cast as a combinatorial optimization problem where, by searching in a finite but typically large set of elements, we want to optimize a given cost function. In this case the set of elements can be the set of all the 2^p possible corners, and the cost function is the arrival time at a given primary output. The major difficulty with this type of discrete problems, as opposed to continuous linear problems, is that we do not have any optimality conditions to check if a given (feasible) solution is optimal or not. Therefore, in order to conclude that a feasible solution is optimal, we must somehow compare its cost with the cost of all other feasible solutions. This amounts to always explore the entire solution space, either *explicitly* or *implicitly*, by a complete or partial *enumeration* of all the feasible solutions and their associated costs.

3.2 Exhaustive Methods

The simplest conceivable algorithm for computing the WDC is to just evaluate the delay of the circuit for all the 2^p possible corners, and verify which corner produces the largest arrival time at a primary output. This corner clearly corresponds to the WDC. By using a block-based timing analysis procedure, the arrival times can be computed in linear time of the number of vertices. However, since we must run such a procedure for each of the 2^p corners, the overall algorithm will be exponential in the number of parameters.

Instead of performing an exhaustive search in the parameter space as outlined in the previous paragraph, we can perform such a search in the path space. Essentially, this corresponds to performing an exhaustive path-based timing analysis and, for each path, computing the corresponding affine delay function, by adding the delay functions of the edges along that path. Given the affine delay function of a path, we can easily compute the WDC for that path by applying Eqns. (3) and (4). For each path, the procedure of computing the delay function and obtaining the WDC of the path is linear in the number of parameters. However, since the number of paths can grow exponentially with the number of vertices, and we must perform this procedure for every single path, the overall procedure has a worst-case exponential complexity in the number of vertices.

As can easily be concluded, both exhaustive methods exhibit exponential run-time complexity, either in the number of parameters or in the number of vertices. For small circuits, or when a small number of parameters is of interest, they may constitute viable options. However, even average size circuits will render both approaches unpractical.

4 Dynamic Pruning

In this section we propose an approach for computing the WDC that, by using branch-and-bound techniques, is able to dynamically prune parts of the search space and therefore avoid an explicit enumeration of all possible solutions. We start by briefly explaining the basic foundations of branch-and-bound techniques and subsequently present path-space and parameter-space search algorithms based on them.

4.1 Branch-and-Bound

Most combinatorial problems, including the one at hand, can only be solved by explicitly or implicitly evaluating a specific, nonlinear, cost function over the entire solution space, in order to compute the solution that yields the optimal cost. Branch-and-bound techniques focus on pruning useless regions of the solution space, thus avoiding the explicit evaluation of all the possible solutions that they may contain. During the execution of the algorithm, the best known value for the cost function is maintained, corresponding to the cost of the best solution already found. If by some simple and quick procedure we are able to determine that the cost of all the solutions contained in a certain subspace is worse than the best known cost, then it is useless to explore that subspace, since no improvement on the cost function will be obtained. Therefore, that portion of the solution space can be pruned, and an explicit enumeration of all the solutions it may contain is avoided. Even though in the worst case this approach can be as bad as the exhaustive enumeration, on average for a wide range of applications, it has proven to perform significantly better.

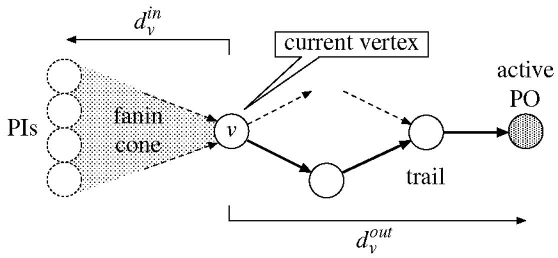


Figure 1. Illustration of delay estimates.

4.2 Path Space Exploration

In the following we will detail a branch-and-bound based algorithm that computes the worst-delay corner by finding one path where it occurs. Considering one primary output at a time, the algorithm performs an implicit search over all the complete paths that end at that 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 explored in a certain 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 largest delay, w^*, among the complete paths already analyzed is continuosly updated, as well as the corresponding corner, $\Delta\lambda^*$. For each current vertex v, the algorithm relies on three parametric delay estimates, illustrated in Figure 1:

- 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 path going through v, that contains the trail.

The rationale underlying this algorithm is that, if for a given current vertex v, the following condition is verified,

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

then there is no path, going through v and containing the trail, with delay larger than w^*, and therefore it is useless to further explore the fanin cone of v.

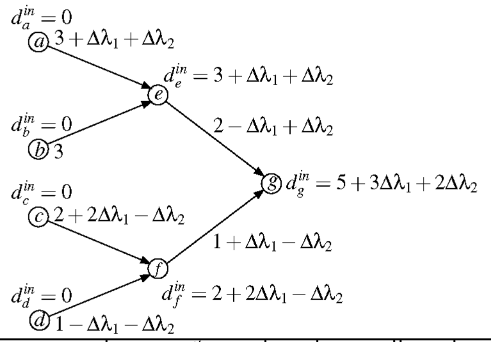
The pseudocode for the algorithm is presented in function WDC-PATH-BNB. It receives the timing graph G as the only argument and it returns a tuple with the worst delay value, w^*, and its associated corner, $\Delta\lambda^*$.

```
function WDC-PATH-BNB($G$)
    $w^* \leftarrow 0$                                  ▷ worst delay
    $\Delta\lambda^* \leftarrow <>$                     ▷ worst corner
    INITIALIZE($G$)
    for all $v \leftarrow PO(G)$ do
        $\langle w, \Delta\lambda \rangle \leftarrow$ PROCESS-VERTEX($G, v, w^*, 0$)
        if $w > w^*$ then
            $\langle w^*, \Delta\lambda^* \rangle \leftarrow \langle w, \Delta\lambda \rangle$
        end if
    end for
    return $\langle w^*, \Delta\lambda^* \rangle$
end function
```

```
function PROCESS-VERTEX($G, v, w^*, d_v^{out}$)
    $d_v^{in} \leftarrow$ IN-DELAY-ESTIMATE($v$)
    $d_v^{path} \leftarrow d_v^{in} + d_v^{out}$
    $\langle w, \Delta\lambda \rangle \leftarrow \max_{\Delta\lambda}[d_v^{path}]$
    if $w \leq w^*$ then                                ▷ fanin cone gets pruned
        return $\langle w^*, 0 \rangle$
    else if $v \in PI(G)$ then
        return $\langle w, \Delta\lambda \rangle$       ▷ worst delay is updated
    else
        for all $e \leftarrow$ INCOMING-EDGES($v$) do
            $s \leftarrow$ SOURCE-VERTEX($e$)            ▷ get source vertex
            $d_e \leftarrow$ DELAY($e$)
            $d_s^{out} \leftarrow d_v^{out} + d_e$
            $\langle w, \Delta\lambda \rangle \leftarrow$ PROCESS-VERTEX($G, s, w^*, d_s^{out}$)
            if $w > w^*$ then
                $\langle w^*, \Delta\lambda^* \rangle \leftarrow \langle w, \Delta\lambda \rangle$
            end if
        end for
        return $\langle w^*, \Delta\lambda^* \rangle$
    end if
end function
```

The algorithm starts by invoking INITIALIZE on the timing graph, G. This function, whose pseudocode is not presented due to space constraints, performs a forward levelized traversal of the timing graph, starting at the primary inputs and ending at the primary outputs. For each vertex v, it computes, the parametric formula for the delay estimate d_v^{in}, that is an upper bound on the delay from any primary input to v. This formula is computed by performing a block-based timing analysis, where the max operation computes conservative upper bounds. The upper bounds can either be constant values, affine functions or piecewise-affine functions, depending on how the max operation is implemented. See Section 5.2 for further details.

After completing the initializations, the algorithm processes all the primary outputs, one at a time. For every primary output it invokes the recursive function PROCESS-VERTEX, that performs a backwards depth-first traversal of the timing graph towards the primary inputs. At each step, a given current vertex v is visited, and each one of its fanins is scheduled to be visited in the next step. Therefore, the current vertex v is always connected to the active primary output by the incomplete path used to reach v, that we already defined as trail. All the vertices in the trail were visited before v. For a given vertex v, we can exactly compute the delay of the trail, d_v^{out}, by adding the delay of all the edges in the trail. That computation is implicitly performed in PROCESS-VERTEX. Adding d_v^{in} and d_v^{out} we obtain d_v^{path}, that is an upper bound on the delay of any path that contains v, starting at any primary input, and reaching the primary output trough the trail. d_v^{path} is an affine function of the parameter variations, therefore its worst value and the corresponding worst corner can be computed using Eqns. (3) and (4). If the worst value of d_v^{path}, w, is smaller than the largest known delay, w^*, computed so far, that means that the worst-delay path does not contain the trail, and therefore we stop the traversal at this vertex. If w is larger than w^*, and v is a primary input, it means that there is a complete path with delay larger than the largest known delay computed so far, and therefore the largest known delay is updated. If v is not at a primary input, the delay estimate is just an upper bound, and therefore it cannot be used to update the largest known delay. The algorithm proceeds until all the paths in the circuit are explicitly or implicitly explored. At the end, the largest known delay w^* and the corresponding corner, $\Delta\lambda^*$, are the worst delay and the worst-delay corner of the circuit, respectively.


$d_a^{in} = 0$
a
$3 + \Delta\lambda_1 + \Delta\lambda_2$
$d_e^{in} = 3 + \Delta\lambda_1 + \Delta\lambda_2$
e
$d_b^{in} = 0$
$2 - \Delta\lambda_1 + \Delta\lambda_2$
b
3
g
$d_g^{in} = 5 + 3\Delta\lambda_1 + 2\Delta\lambda_2$
$d_c^{in} = 0$
c
$2 + 2\Delta\lambda_1 - \Delta\lambda_2$
$1 + \Delta\lambda_1 - \Delta\lambda_2$
f
$d_d^{in} = 0$
$d_f^{in} = 2 + 2\Delta\lambda_1 - \Delta\lambda_2$
d
$1 - \Delta\lambda_1 - \Delta\lambda_2$

v	trail	d_v^{path}	w	$\Delta\lambda$	w^*	$\Delta\lambda^*$
g	$\emptyset$	$5+3\Delta\lambda_1+2\Delta\lambda_2$	10	$\langle 1,1 \rangle$	0	$\emptyset$
e	$\langle e,g \rangle$	$5+2\Delta\lambda_2$	7	$\langle X,1 \rangle$	0	$\emptyset$
a	$\langle a,e \rangle, \langle e,g \rangle$	$5+2\Delta\lambda_2$	7	$\langle X,1 \rangle$	7	$\langle X,1 \rangle$
b	$\langle b,e \rangle, \langle e,g \rangle$	$5-\Delta\lambda_1+\Delta\lambda_2$	6	$\langle 0,1 \rangle$	7	$\langle X,1 \rangle$
f	$\langle f,g \rangle$	$3+3\Delta\lambda_1-2\Delta\lambda_2$	6	$\langle 1,0 \rangle$	7	$\langle X,1 \rangle$

Figure 2. Execution of WDC-PATH-BNB.

Figure 2 illustrates the execution of the algorithm for a small timing graph. It should be noticed that w^* is only updated when vertex a is analyzed because only then the trail is a complete path, and therefore d_v^{path} is the exact delay of that path, and not an upper bound. Further, the fanin cone of f is not analyzed because $w \leq w^*$. This corresponds to pruning a portion of the path space, namely paths $\{\langle c,f \rangle, \langle f,g \rangle\}$ and $\{\langle d,f \rangle, \langle f,g \rangle\}$.

4.3 Parameter Space Exploration

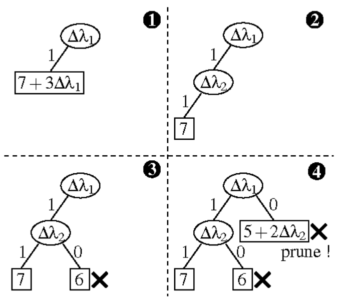
In the previous section we detailed a branch-and-bound algorithm for computing the worst-delay corner by exploring the path space and finding one path where it occurs. In this section we try a different approach, and propose another branch-and-bound algorithm exploring the parameter space. By analyzing the worst delay obtained for specific corners, the algorithm is able to effectively prune regions of the parameter space. In this context the timing graph will only be used to compute worst delay estimates for a partial or complete assignment (corner) of the parameter variation vector. For partial assignments we can only compute an upper bound on the worst delay. For complete assignments we obtain the exact value of the worst delay.

The pseudocode for the proposed algorithm is presented in function WDC-PARAMETER-BNB, that receives and returns the same information as the previously studied WDC-PATH-BNB. As we have mentioned the algorithm

```
1:  function WDC-PARAMETER-BNB($G$)
2:      $w^* \leftarrow 0$                                   ▷ worst delay
3:      $\Delta\lambda^* \leftarrow <>$                      ▷ worst corner
4:      $T \leftarrow$ DT-INIT()
5:      while $\Delta\lambda \leftarrow$ DECIDE($T$) do
6:          DT-REGISTER-DECISION($T$,$\Delta\lambda$)
7:          $w \leftarrow$ WORST-DELAY($G$,$\Delta\lambda$)
8:          if $w \leq w^*$ then
9:              DT-REGISTER-PRUNE($T$,$\Delta\lambda$)
10:         else if IS-COMPLETE($\Delta\lambda$) then
11:             $\langle w^*, \Delta\lambda^* \rangle \leftarrow \langle w, \Delta\lambda \rangle$
12:         end if
13:     end while
14:     return $\langle w^*, \Delta\lambda^* \rangle$
15: end function
```

will try to prune regions of the parameter variation space by analyzing the worst delay produced by certain partial and complete assignments of the parameter variation vector. In order to help us keep track of all the partial and complete assignments already analyzed we will use a binary tree, commonly designated by *decision tree*. Each node in the decision tree represents one element of the parameter variation vector and can have at most a left and right child. Each child is a subtree. The left child represents a partial or a complete assignment of the parameter variation vector where the corresponding element assumes value 1. For the right child this value is 0. The leaves of the tree are the delay estimates computed considering the parameter variation vector assignments in the upper levels. Therefore, if a leaf is at level $p+1$ (assuming the root at level 1), it means that it corresponds to a complete assignment, and therefore it contains an exact worst delay. On the other hand, if a leaf is at level $k < p$, it means that it corresponds to a partial assignment, and therefore it contains an upper bound on the worst delay.

The algorithm starts by calling DT-INIT, that initializes the data structures for the decision tree. Afterwards it enters a cycle, where for each iteration the function DECIDE, based on the current state of the decision tree, and the regions of the parameter variation space that need to be explored, will produce a partial or complete assignment, $\Delta\lambda$, for the parameter variation vector. This assignment is then annotated to the decision tree by DT-REGISTER-DECISION. Subsequently, the worst delay estimate for this assignment is computed by WORST-DELAY, and stored in w. If the worst delay estimate is smaller or equal to the largest known delay estimate achieved so far, w^*, it means that any assignment contained in the partial assignment $\Delta\lambda$ will not provide an improvement over w^* and therefore can simply be ignored. In order to prevent DECIDE from further exploring this region of the parameter variation space, we call DT-REGISTER-PRUNE that will insert a marker in the decision tree. No further expansions will be performed beyond this node, effectively pruning the subtree from consideration. If the worst delay estimate is larger than the largest known delay estimate computed so far and $\Delta\lambda$ is a complete assignment, it means that $\Delta\lambda$ improves the largest known delay estimate and therefore w^* and $\Delta\lambda^*$ are updated. If the worst delay estimate is larger than the largest known delay estimate computed so far, but $\Delta\lambda$ is only a partial assignment, no conclusion can be drawn, since the worst delay estimate obtained for a partial assignment is just an upper bound, whose value will eventually get smaller as new el-



$\Delta\lambda_1$	$\Delta\lambda_2$	delay
0	X	$5+2\Delta\lambda_2$
1	X	$6+2\Delta\lambda_2$
X	0	$5+3\Delta\lambda_1$
X	1	$7+3\Delta\lambda_1$
0	0	5
0	1	7
1	0	6
1	1	7

Figure 3. Execution of WDC-PARAMETER-BNB.

ements of the parameter variation vector are assigned. The algorithm proceeds until all the regions of the parameter space (e.g. all possible parameter variation vector assignments) are either explicitly explored or pruned.

Figure 3 illustrates the decision tree produced by the execution of the algorithm for the timing graph in Figure 2. The table on the right side presents the delay estimates, which are upper bounds for partial parameter variation assignments, and are exact delay estimates for complete assignments. In steps (1) and (2) we generate a complete parameter variation assignment, $\Delta\lambda = \langle 1,1 \rangle$, in order to obtain the first estimate for w^*, which is 7. In step (3) we analyze the complete assignment $\Delta\lambda = \langle 1,0 \rangle$ and conclude that it produces a delay of 6, which is smaller than the current $w^* = 7$. In step (4) we analyze the partial assignment $\Delta\lambda = \langle 0,X \rangle$ and conclude it produces a delay of $5+2\Delta\lambda_2$, that in the worst case assumes value 7. Since this delay is equal to the largest known delay found so far, we can discard (e.g. prune) the remainig subtree. This effectively prunes part of the parameter variation space, namely assignments $\langle 0,1 \rangle$ and $\langle 0,0 \rangle$. After this all the parameter variation space has been explored, and the final solution is $w^* = 7$ and $\Delta\lambda^* = \langle 1,1 \rangle$.

5 Practical Issues

This section addresses two issues concerning the practical application of the proposed corner computation algorithms: the computation of corners induced by timing constraints, and the computation of loose and tight upper bounds on the max function.

5.1 Corners Induced by Timing Constraints

Slacks are induced by required arrival time (RAT) constraints in specific circuit pins. If RATs are constant values, finding the setting of parameter variation values that produces the minimum slack in a pin is equivalent to finding the setting of parameter variation values that produces the largest delay in one of the pins where these constraints exist. Therefore we can use our algorithm to compute the parameter settings that produce the worst delay in every pin with RAT constraints. Among these pins, we choose the parameter settings for the pin where the slack is the smallest. Therefore, computing the worst-slack corner amounts to a sequence of worst-delay corner computations.

When the constraints are also parametric, as in the case of setup and hold constraints, where clock latencies also depend on process parameters, it is also possible to cast the problem as a worst-delay corner problem. Due to lack of space we will address this issue in a future publication.

5.2 Bounding the max

The algorithms presented in the previous sections explicitly or implicitly rely on computing upper bounds on the arrival times at specific vertices in the timing graph, namely the primary outputs. Their correctness is independent of the upper bounds computed. Their performance, however, is dependent on the *tightness* of those bounds. These bounds are actually computed on the max between two or more parametric formulas. Tighter bounds will potentially allow for larger regions of the search space to be pruned, and therefore may have a significant impact in performance. On the other hand, tighter bounds are usually much more expensive to compute. In order to evaluate the impact of tighter bounds on the performance of the proposed corner finding approach, we describe the method for computing loose and tight max bounds, for which experimental results will be presented in the next section.

In the following, given two parametric formulas a and b, we want to compute c, such that c is an upper bound on the max between a and b, e.g. $c \geq \max[a,b]$. The simplest and cheapest upper bound on the max can be computed by just picking for each coefficient of c to be the max of the coefficients of a and b. Formally,

$$c_i = \max[a_i, b_i] \ , \ i = 0, 1, \ldots, p \qquad (6)$$

This bound is very cheap to compute, but it is also loose.

The tightest upper bound on the max, with only one bounding function (one plane) c, can be computed by solving the following LP,

$$\begin{array}{ll} \min & \varepsilon \\ \text{s.t.} & \varepsilon \geq c(\Delta\lambda^{(q)}) - \max[a(\Delta\lambda^{(q)}), b(\Delta\lambda^{(q)})], \ q = 1, \ldots, 2^p \\ & c(\Delta\lambda^{(q)}) \geq \max[a(\Delta\lambda^{(q)}), b(\Delta\lambda^{(q)})] \\ & 0 \leq \Delta\lambda_l \leq 1, \ l = 1, \ldots, p \end{array} \qquad (7)$$

where $\Delta\lambda^{(q)}$ is the q-th process corner. This bound is the tightest (for one plane), but it is expensive to compute.

6 Experimental Results

A realistic circuit block was synthesized and mapped to an industrial 90nm technology. As process parameters, we considered the widths and thicknesses of the six metal layers needed to route the block. During parasitic extraction of the design, we computed the nominal values and sensitivities of each parasitic element (resistors and grounded capacitors), relative to each one of the 12 parameters, and from that we computed parametric interconnect delays. From the circuit block we extracted 3 combinational circuits that we used as benchmarks. Table 1 presents information about the timing graph of each circuit, including the number of process parameters considered.

Table 2 presents the CPU time and the search size for the execution of the exhaustive (Exh) and branch-and-bound versions of the WDC computation by searching in the path and parameter spaces. The path space branch-and-bound versions were executed using both loose (BnB-L) and tight (BnB-T) max bounds, as detailed in Section 5.2.

By analyzing the experimental results it is easy to conclude that the proposed branch-and-bound technique is very effective, since it reduces the CPU times and search sizes by several orders of magnitude, both in parameter space and path space. The parameter space search seems to be significantly more expensive than path space search. Additionally, as expected, when tighter bounds are used the amount of search is slightly reduced, since more pruning should occur, which indicates potential for some moderate improvement, if tighter, but still cheap, bounds can be computed.

Name	#Vertex	#Edge	#PI	#PO	#Par
mult	2507	3324	20	19	12
add	679	890	41	22	12
share	375	493	26	13	12

Table 1. Benchmark information.

	Name	Parameter		Path		
		Exh	B-n-B	Exh	BnB-L	BnB-T
CPU Time (s)	mult	356.52	10.94	2.68	0.02	141.20
	add	27.54	0.2	0.01	<0.01	22.21
	shared	9.52	0.05	0.01	<0.01	11.69
Search Size	mult	4096	125	3249498	1623	1170
	add	4096	27	9144	595	466
	shared	4096	19	3846	52	52

Table 2. Worst-delay corner computation.

7 Conclusions

This paper proposes an efficient, branch-and-bound based, automated methodology for computing the exact worst-delay process corners of a digital integrated circuit, given a linear parametric characterization of the gate and interconnect delays. Experimental evidence shows that the proposed approach is particularly effective, leading to reductions in CPU time up to several orders of magnitude, when computing circuit timing while accounting for parameter variability.

References

[1] S. Bhardwaj, S. B. K. Vrudhula, and D. Blaauw. Tau: Timing analysis under uncertainty. In *Proceedings of The International Conference on Computer Aided-Design*, pages 615–620, San Jose, CA, November 2003.

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

[3] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic. Fast Statistical Timing Analysis by Probabilistic Event Propagation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 661–666, Las Vegas, NV, June 2001.

[4] Luis Guerra e Silva and Zhenhai Zhu and Joel Phillips and L. Miguel Silveira. Variation-Aware, Library Compatible Delay Modeling Strategy. In *Proceedings of the IFIP VLSI-SoC Conference*, Nice, France, October 2006.

[5] F. N. Najm. On the Need for Statistical Timing Analysis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 764–765, Anaheim, CA, June 2005.

[6] S. Onaissi and F. N. Najm. A Linear-Time Approach for Static Timing Analysis Covering All Process Corners. In *Proceedings of The International Conference on Computer Aided-Design*, San Jose, CA, November 2006.

[7] L. Scheffer. Explicit Computation of Performance as a Function of Process Variation. In *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Monterey, CA, December 2002.

[8] J. Stolfi and L. H. de Figueiredo. Self-Validated Numerical Methods and Applications. In *Operations Research*, July 1997.

[9] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 331–336, San Diego, CA, June 2004.