were run on an Intel Xeon running at 3.20 GHz machine with 3 GB of RAM running Linux Red Hat 3.4.26.fc3. The running time given comprises the entire exploration process including the HLS.

B. Results and Discussion

Table II shows the results of the DSE. From the experiments it can be observed that our clustering method is on average 90% and 92% faster than the annealer method for the CDS-Exp(min) and CDS-Exp(max) method respectively. The drawback is that on average only 36% and 47% of all Pareto optimal designs are found. Table II also shows that our methods on average finds the smallest design that is 7% and 9% larger than the actual smallest case and 28% and 32% longer latency respectively. Although approximately one third and one half, respectively, of the Pareto optimal points are not found the smallest and fastest designs are found in some cases and in most cases almost found. This is important as in most cases these are the designs that are finally used and provide the boundary points of the exploration. In order to expand the search space using our method more clustering stages could be introduced changing the clusters attributes more often. This would find more optimal points at the expense of increasing the design space increasing, therefore, the runtime.

VI. Conclusion

High-level synthesis is becoming a must in state of the art hardware designs. Designers can no longer describe and model entire systems on chips in low-level languages and need to raise the level of abstraction. Tools that bridge the gap between untimed high-level languages and RTL are needed. In this paper, we present a DSE method to speed up the exploration of high-level language design descriptions given in C and SystemC. The presented method, called CDS-ExpA, is based on a clustering method that clusters explorable operations and assigns a fix set of attributes to these based on the global cost function in order to reduce the design space. Two variations of the CDS-Exp are presented. CDS-ExpA(min) creates the smallest possible clusters and CDS-ExpA(max) the largest possible ones. The trade-offs between further reducing the design space by building larger clusters versus smaller is also investigated. Results show that the DSE dramatically reduces the runtime by around 90% at a cost of missing on average 36% and 47% of all Pareto optimal designs. On the other hand the smallest and fastest designs are found in many cases and on average 7% and 9% respectively larger than the actual smallest case and 28% and 32% longer latency designs are found. We believe that this exploration method is a valid solution for initial DSEs as half of the Pareto points are found and closely the smallest and fastest ones which provides valuable design information to the designer at the earliest design stage extremely fast.

References

[1] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high-level synthesis design space exploration," in *Proc. Int. Symp. Very-Large-Scale Integration Design, Automat., Test*, 2009, pp. 106–109.

[2] *Behavioral Description Language* [Online]. Available: http://www.cyberworkbench.com

[3] *Single Assignment C (SA-C)* [Online]. Available: http://www.cs.colostate.edu/cameron

[4] C. Haubelt, T. Schlichter, J. Keinert, and M. Meredith, "SystemCo-Designer: Automatic design space exploration and rapid prototyping from behavioral models," in *Proc. Design Automat. Conf.*, 2008, pp. 580–585.

[5] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian, "Design space exploration of real-time multimedia MPSoCs with heterogeneous scheduling policies," in *Proc. Int. Conf. Hardware/Software Codesign Syst. Synthesis (CODES+ISSS)*, 2006, p. 1621.

[6] S. Mamagkakis, D. Atienza, C. Poucet, F. Catthoor, D. Soudris, and J. M. Mendias, "Automated exploration of pareto-optimal configurations in parameterized dynamic memory allocation for embedded systems," in *Proc. Design, Automat., Test Eur.*, 2006, pp. 874–875.

[7] I. Ahmad, M. Dhodi, and F. Hielscher, "Design-Space exploration for high-level synthesis," in *Proc. IEEE 13th Ann. Int. Phoenix Conf. Comput. Commun.*, 1994, pp. 491–496.

[8] M. Holzer, B. Knerr, and M. Rupp, "Design space exploration with evolutionary multiobjective optimization," in *Proc. Ind. Embedded Syst.*, 2007, pp. 125–133.

[9] C. Haubelt and J. Teich, "Accelerating design space exploration," in *Proc. Int. Conf. Applicat.-Specific Integr. Circuit*, 2003, pp. 79–84.

[10] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A multiobjective cosynthesis framework for multimode embedded systems," in *Proc. IEEE Int. Conf. Applicat.-Specific Syst., Architect., Processors*, 2004, pp. 28–40.

[11] S. Bilavarn, G. Gogniat, J.-L. Philippe, and L. Bossuet, "Design space pruning through early estimation of area/delay trade-offs for FPGA implementations," in *Proc. Int. Conf. Comput. Aided Design*, vol. 25. Oct. 2006, pp. 1950–1968.

[12] I. D. L. Anderson and M. A. S. Khalid, "SC Build: A computer-aided design tool for design space exploration of embedded central processing unit cores for field-programmable gates arrays," *Inst. Eng. Technol. Comput. Digital Tech.*, vol. 3, no. 1, pp. 24–32, Jan. 2009.

[13] B. So, M. W. Hall, and P. C. Diniz, "A compiler approach to fast hardware design space exploration in FPGA-based systems," in *Proc. Conf. Program. Language Design Implement.*, Jun. 2002, pp. 165–176.

[14] B. So, P. C. Diniz, and M. W. Hall, "Using estimates from behavioral synthesis tools in compiler-directed design space exploration," in *Proc. Design Automat. Conf.*, 2003, pp. 514–519.

[15] P. Coussy and A. Moraweic, "All-in-C behavioral synthesis and verification with CyberWorkBench" in *High-Level Synthesis from Algorithm Digital Circuit*. Berlin, Germany: Springer-Verlag, 2008, ch. 7, pp. 113–127.

Effective Corner-Based Techniques for Variation-Aware IC Timing Verification

Luis Guerra e Silva, Joel Phillips, and L. Miguel Silveira

***Abstract*—Traditional integrated circuit timing sign-off consists of verifying a design for a set of carefully chosen combinations of process**

Manuscript received July 11, 2008; revised April 17, 2009 and July 24, 2009. Current version published December 18, 2009. This work was supported by Cadence Design Systems, Inc., San Jose, CA, USA. This paper was recommended by Associate Editor R. Suaya.

L. G. e Silva is with INESC-ID Lisbon and with the Department of Information Systems and Computer Science, Instituto Superior Técnico, Technical University of Lisbon, 1049-001 Lisbon, Portugal (e-mail: lgs@inesc-id.pt).

L. M. Silveira is with INESC-ID Lisbon and with the Department of Electrical and Computer Engineering, Instituto Superior Técnico, Technical University of Lisbon, 1049-001 Lisbon, Portugal and also with Cadence Research Laboratories, Cadence Design Systems, Berkeley, CA 94704 USA (e-mail: lms@inesc-id.pt).

J. Phillips is with Cadence Research Laboratories, Cadence Design Systems, Berkeley, CA 94704 USA (e-mail: jrp@cadence.com).

Digital Object Identifier 10.1109/TCAD.2009.2034343

and operating parameter extremes, referred to as corners. Such corners are usually chosen based on the knowledge of designers and process engineers, and are expected to cover the worst-case fabrication and operating scenarios. With increasingly more detailed attention to variability, the number of potential conditions to examine can be exponentially large, more than is possible to handle with straightforward exhaustive analysis. This paper presents efficient yet exact techniques for computing worst-delay and worst-slack corners of combinational and sequential digital integrated circuits. Results show that the proposed techniques enable efficient and accurate detection of failing conditions while accounting for timing variability due to process variations.

***Index Terms*—Corner, timing, variability, verification.**

I. Introduction

Parametric performance models, where performance metrics, most commonly related to timing and power, are expressed as functions of parameter variations, have been introduced for early prediction and detection of integrated circuit (IC) performance issues due to process variability, inherent to the latest nanometric IC technologies. New analysis techniques that make use of these parametric models have likewise been proposed. The most significant such example is statistical static timing analysis (SSTA), where parameters are treated as distributions rather than fixed numerical values. Several promising SSTA modeling techniques have been proposed [1]–[4], some of which are already implemented in commercially available tools. However, SSTA is mostly used as an aid in design optimization. Therefore, for the most part, the industry golden standard methodology for timing sign-off still resorts to traditional corner analysis techniques.

Even though SSTA techniques have received the most attention in the literature, the parametric delay modeling technologies they advocate have much wider applicability. In particular, they can be used in reducing pessimism and automating well established timing verification methodologies. Conventional IC timing sign-off consists in verifying a design for a set of carefully selected combinations of process and operating parameter extremes, commonly referred to as *corners*, that are expected to cover the worst-case scenarios. However, there is no established systematic methodology for picking such worst-case corners in a realistic manner, and this task usually relies on the experience of design and process engineers. Compounding the problem, for feature sizes in the nanometric scale, the number of parameters to be considered increases significantly. In an effort to overcome this limitation of established timing sign-off methodologies, this paper proposes an efficient automated methodology for computing the worst-timing corners in a digital integrated circuit, when parametric delay models are available. Specifically, we address the computation of worst-delay corners of combinational blocks and of worst-slack corners of sequential circuits. In this approach, parameters only need to be characterized by their respective value ranges. The proposed methodology casts the computation of the worst-timing corners as a search problem, which provides an intellectual paradigm that is more general and useful than most previous approaches.

While it has become commonplace in the literature to argue for a shift away from a corner-based analysis to a statistical methodology, there are important reasons to improve the efficiency of a corner-like methodology. First, such techniques are easily integrated within currently used design and verification paradigms. Second, they impose less stringent requirements on parameter characterization. Finally, efficient worst-case analysis can be a complementary technique to SSTA, by providing insight into unusual circuit operating conditions. This last setting is a primary motivator for our paper.

Recently, Onaissi and Najm [5] have proposed a linear-time approach for timing analysis of combinational circuits that computes a delay upper bound estimate, covering all process corners. Such estimate is just a conservative approximation, and the corresponding worst-delay corner cannot be inferred from such estimate. Further, it is difficult, if not impossible, to trace the corresponding critical path. The goal of our paper is quite different, as we target the determination of the *exact* worst-delay corner and associated paths. Additionally, unlike [5] our paper covers the analysis of sequential circuits.

This paper is organized as follows. Section II introduces a few basic concepts and notation. Section III formulates the worst-delay corner problem and discusses exhaustive procedures for its solution. Section IV describes a worst-delay corner computation technique, proposed in [6]. Section V proposes a novel technique for computing worst-slack corners of sequential circuits. Section VI discusses the experimental results and Section VII presents brief concluding remarks.

II. Background

This section introduces background information. We start by reviewing the concept of timing graph, used to represent the timing information of a circuit. Subsequently, we introduce the parametric affine delay formulation used throughout the paper and sometimes referred to as the canonical representation in the SSTA literature [4].

A. Timing Graph

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. Each edge is annotated with the corresponding *delay*. Further, some vertices are annotated with *timing constraints*, such as required arrival times. 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. A *partial path* is a sequence of edges connecting any two vertices. A complete path will be referred to simply as a *path*.

Cell and interconnect delays are the result of a delay calculation procedure, where slews are forward propagated across the circuit and, using appropriate cell and interconnect models, the delays and output slews for each component are computed. Cell delays are annotated in the edges connecting the vertices corresponding to input/output pins of the cell. Interconnect delays are annotated in the edges connecting the vertices corresponding to port/tap pins of interconnect nets. It is out of the scope of this paper to discuss the delay computation procedure [7] therefore, in the following, we will assume that the timing information of any circuit is already made available in the form of a timing graph.

B. Parametric Delay Formulation

This paper assumes delays to be 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

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

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. This representation is mathematically equivalent to the canonical formulation prescribed in [4], 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 $\Delta\lambda \in [0, 1]^p$.

When delays are given in the form of (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. An important property of affine functions is their *convexity* [8]. Both the min/max and sum operators produce convex functions when operating on convex functions. 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 the simple case of delays that are represented by affine functions this value is fairly easy to compute. Assuming that $\Delta\lambda_i \in [0, 1]$, if in (1) 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, therefore obtaining the maximum value

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

where the maximizing parameter variation assignment is

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

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 much more 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.

III. Worst-Delay Corner

This section formulates the problem of computing the worst-delay corner (WDC) of a combinational circuit, and discusses exhaustive methods for its solution.

A. Problem Formulation

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 (1), then any delay, $d_{i,j}(\Delta\lambda)$, from an input i to an output j can be accurately represented by a piecewise-affine function.

The 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 *late* mode, the worst-delay is the *largest* delay. Assuming that $d_{i,j}^{late}(\Delta\lambda)$ is the piecewise-affine function of the delay in late mode from input i to output j, then the WDC problem is formulated as

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

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, this problem can be cast as a combinatorial optimization problem. 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 whether a given feasible solution is optimal or not. In order to conclude that a feasible solution is optimal, we must somehow compare its cost with the cost of all the 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.

B. Exhaustive Methods

The simplest exhaustive algorithm that can be conceived for computing the WDC consists in evaluating the delay of the circuit for each of the 2^p possible parameter variation corners, and verifying which corner produces the worst circuit delay, which corresponds to the WDC. Clearly, this algorithm has exponential run-time complexity in the number of parameters.

Another possible approach consists instead in computing the WDC over all paths, rather than over all corners. Since the number of paths can grow exponentially with the number of vertices, and this procedure must be applied to every single path, the overall procedure can have, in the worst-case, an exponential run-time complexity.

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 very 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, due to the excessive run-time required for their successful completion.

IV. Dynamic Pruning

In this section, we propose an approach for computing the WDC using branch-and-bound techniques [9], in order to dynamically prune parts of the search space and therefore avoid an explicit enumeration of all the possible solutions.

In the previous sections we did not make explicit the meaning of *worst*, as it can represent the largest or smallest value of a given timing estimate. For the sake of clarity, and without loss of generality, in the following we will assume that the worst value of a given estimate it is its *maximum* value.

Both parameter-based and path-based exhaustive search algorithms described earlier can be improved by employing branch-and-bound techniques. To understand how this can be achieved, we detail a path-based search algorithm that is able to efficiently compute 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

```
1: function WDC-PATH-BNB(G)
2:     w* ← 0                                          ▷ worst delay
3:     Δλ* ← ∅                                         ▷ worst corner
4:     INITIALIZE(G)
5:     for all v ← PO(G) do
6:         ⟨w, Δλ⟩ ← PROCESS-VERTEX(G, v, w*, 0)
7:         if w > w* then
8:             ⟨w*, Δλ*⟩ ← ⟨w, Δλ⟩
9:         end if
10:    end for
11:    return ⟨w*, Δλ*⟩
12: end function
```

```
1: function PROCESS-VERTEX(G, v, w*, d_v^out)
2:     d_v^in ← IN-DELAY-ESTIMATE(v)
3:     d_v^path ← d_v^in + d_v^out
4:     ⟨w, Δλ⟩ ← max_Δλ[d_v^path]
5:     if w ≤ w* then                                  ▷ fanin cone gets pruned
6:         return ⟨w*, 0⟩
7:     else if v ∈ PI(G) then
8:         return ⟨w, Δλ⟩                              ▷ worst delay is updated
9:     else
10:        for all e ← INCOMING-EDGES(v) do
11:            s ← SOURCE-VERTEX(e)                    ▷ get edge source
12:            d_e ← DELAY(e)
13:            d_s^out ← d_v^out + d_e
14:            ⟨w, Δλ⟩ ← PROCESS-VERTEX(G, s, w*, d_s^out)
15:            if w > w* then
16:                ⟨w*, Δλ*⟩ ← ⟨w, Δλ⟩
17:            end if
18:        end for
19:        return ⟨w*, Δλ*⟩
20:    end if
21: end function
```

given step is designated by *current vertex*. The path taken to reach that vertex from the active primary output is designated by *trail*. When reconvergent fanouts exist, the same vertex can be reached from the same primary output, through distinct trails. The worst-delay (e.g., maximum delay), w^*, among the delays of the complete paths already analyzed, is continuously updated, as well as its originating parameter variation assignment, $\Delta\lambda^*$. For each step, where the current vertex is v, the algorithm relies on three parametric delay estimates.

1) d_v^{in} is an upper bound on the delay of all the partial paths that start at a primary input and end in v (e.g., in the fanin cone of v).
2) d_v^{out} is the exact delay of the trail path, that starts in the current vertex, v, and ends in the active primary output.
3) $d_v^{path} = d_v^{in} + d_v^{out}$, which represents an upper bound on the delay of all the complete paths that include the trail.

As v gets closer to the primary inputs, the upper bound given by d_v^{in} gets tighter. If v is a primary input: $d_v^{in} = 0$ and $d_v^{path} = d_v^{out}$ is the exact delay of the trail, rather than an upper bound.

The rationale underlying the proposed algorithm is that if the worst-delay, among all the complete paths going through v and including the trail, $\max_{\Delta\lambda}[d_v^{path}]$, or an upper bound of such delay, is not larger than the worst-delay already computed for some other complete path, w^*, then it is useless to further explore the fanin cone of v, as the worst-delay, w^*, cannot be improved by such action.

The pseudocode of the algorithm is presented in function WDC-PATH-BNB. It receives the timing graph, G, as the single argument and returns a tuple, $\langle w^*, \Delta\lambda^* \rangle$, with the worst-delay value and its originating parameter variation assignment, respectively.

The algorithm starts by invoking INITIALIZE on the timing graph, G, which performs a forward levelized breadth-first traversal of the timing graph and, for each vertex v, computes the parametric formula for d_v^{in}. This formula, and delay upper bounds in general, are computed by performing a max operation over the sum of the delay of each incoming edge with the d^{in} estimate of the corresponding source vertex. The upper bounds can either be constant values, affine functions or piecewise-affine functions, depending on how the max function is implemented.

After completing the initializations, the algorithm processes 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 toward the primary inputs. In each step, a given vertex v is visited (e.g., deemed the *current* vertex), and 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 designated by trail. All the vertices along the trail were visited before v. For a given vertex v, the exact delay of the trail, d_v^{out}, can be computed by adding the delay of all the edges in the trail. That computation is implicitly performed in Process-Vertex. The value of d_v^{path}, computed by adding d_v^{in} and d_v^{out}, is an upper bound on the delay of any path that contains v, starts at any primary input, and reaches the active primary output trough the trail. d_v^{path} is an affine function of the parameter variations. The worst value of d_v^{path}, that we designate by w, and the corresponding corner, that we designate by $\Delta\lambda$, can be computed by applying (2) and (3).

If w is smaller than the largest (worst) known delay, w^*, computed so far, then the worst-delay path cannot contain the trail, and therefore we stop the traversal at this vertex, and backtrace within the trail. If w is larger than w^*, and v is a primary input, then there is a complete path with delay larger than the largest known delay computed so far, and therefore the largest known delay is updated, which corresponds to update the value of w^* with w. If we are not at a primary input, the delay estimate is just an upper bound, and therefore it cannot be used to update w^*. We proceed until all the paths in the circuit are explicitly or implicitly explored. On termination, the largest known delay w^* and the corresponding corner, $\Delta\lambda^*$, are the worst-delay and the WDC of the circuit, respectively.

A similar set of branch-and-bound techniques can be applied when searching in the parameter space (see [6] for details).

V. WORST-SLACK CORNER

Proper operation of a sequential circuit requires that the input data line of any flip-flop must be stable for a specific period of time before the capturing clock edge, designated by *setup time*, t_{setup}. Let us assume that a flip-flop, with clock latency (i.e., delay from clock source) l_i^{in}, connected to the ith primary input of the combinational block, is injecting data, and another flip-flop, with clock latency l_j^{out}, connected to the jth primary output of the combinational block, is capturing the result. Assuming that the clock edge is generated in the clock source at time 0, then it will reach the injecting flip-flop at time l_i^{in}, making the data available at the primary input of the combinational block. If the propagation delay in the combinational block in late mode (i.e., considering that the output of a cell is changed by the last input that changed), from the ith primary input to the jth primary output, is $d_{i,j}^{late}$, then the results will be available in the output at most at time $l_i^{in} + d_{i,j}^{late}$. The next clock edge will reach the capturing flip-flop at time $T + l_j^{out}$. For a correct operation, the results must be available at the jth primary output of the combinational block

TABLE I
RESULTS FOR WORST-DELAY AND WORST-SLACK CORNER COMPUTATION

Design	#C	#N	$\|V\|$	$\|E\|$	Path Exhaustive		Path BnB		Parameter Exhaustive		Parameter BnB	
					#Search	CPU(s)	#Search	CPU(s)	#Search	CPU(s)	#Search	CPU(s)
c432	88	124	415	575	1920392	3.4	561	<0.01	65536	27.45	1701	4.01
c1908	178	211	756	1065	3318560	6.22	1026	0.01	65536	67.88	2125	11.65
c3540	443	494	1882	2756	34153708	64.83	1052	<0.01	65536	299.31	2219	44.95
c5315	554	734	2644	3701	4218632	8.25	740	0.02	65536	417.06	701	23.94
c6288	**1584**	**1653**	**5131**	**6998**	**>1571711079**	**>3000**	**2318098**	**8.62**	**65536**	**803.05**	**1339**	**133.18**
c7552	820	1031	3483	4807	3788036	6.67	922	0.03	65536	550.29	1001	52.64

Design	#C	#S	#N	$\|V\|$	$\|E\|$	Worst Setup Slack				Worst Hold Slack			
						Path Exhaustive		Path Bnb		Path Exhaustive		Path BnB	
						#Search	CPU(s)	#Search	CPU(s)	#Search	CPU(s)	#Search	CPU(s)
s838_1	265	33	334	1002	1409	11526	0.01	176	<0.01	11526	0.03	64	<0.01
s1423	469	75	563	1829	2609	452286	0.85	334	0.03	452286	0.90	214	0.02
s9234_1	726	150	920	2999	4337	159966	0.30	402	0.03	159966	0.30	208	0.03
s5378	799	166	1058	3468	5049	67706	0.12	156	0.04	67706	0.16	70	0.03
s38584	5243	1187	6455	22863	34003	394194	0.81	138	0.43	394194	0.77	30	0.42
s38417	**5517**	**1597**	**7153**	**25750**	**39948**	**15284978**	**29.28**	**298**	**0.61**	**15284978**	**30.51**	**34**	**0.63**
s35932	6825	1763	8916	29305	43056	318414	1.01	222	0.70	318414	1.06	86	0.68

t_{setup} time before the next clock edge reaches the capturing flip-flop. Therefore, the setup time in the capturing flip-flop is observed only if the following condition holds

$$l_i^{in} + d_{i,j}^{late} \leq T + l_j^{out} - t_{setup}. \tag{5}$$

This condition must hold for every $\langle i, j\rangle$ input/output flip-flop pair. For a given output flip-flop j this set of constraints can be compactly written as

$$\max_{i=1,\ldots,n} (l_i^{in} + d_{i,j}^{late}) \leq T + l_j^{out} - t_{setup}. \tag{6}$$

This expression induces a slack s_j^{setup}, defined as

$$s_j^{setup} = T + l_j^{out} - t_{setup} - \max_{i=1,\ldots,n} (l_i^{in} + d_{i,j}^{late}) \tag{7}$$

that is nonnegative when the conditions are met and negative otherwise. The worst-slack corner for s_j^{setup} is the corner where its value is minimized, formally

$$\max_{\Delta\lambda}(-s_j^{setup}) = \max_{\Delta\lambda} \left[\max_{i=1,\ldots,n} (l_i^{in} + d_{i,j}^{late}) - T - l_j^{out} + t_{setup} \right]. \tag{8}$$

The corner, $\Delta\lambda^*$, that maximizes the value of $-s_j^{setup}$ among all outputs $j = 1, \ldots, m$ is given by

$$\max_{\Delta\lambda} \left\{ \max_{j=1,\ldots,m} \left[\max_{i=1,\ldots,n} (l_i^{in} + d_{i,j}^{late}) - T - l_j^{out} + t_{setup} \right] \right\}. \tag{9}$$

Comparing (4) and (9) we can easily detect that they exhibit a similar structure. For building (9), having (4) as a starting point, we only need to add the clock latency, l_i^{in}, inside the max in i, corresponding to the inputs, and subtract the required arrival time, $T + l_j^{out} - t_{setup}$, inside the max in j, corresponding to the outputs. Therefore, it can be concluded that the worst setup slack corner problem can be cast as an instance of the WDC problem, if the original timing graph of the combinational block is modified by adding edges with the input clock latency and required arrival time.

A similar formulation can be employed in the computation of worst hold slacks, as well as slacks induced by other types of parametric arrival times.

VI. EXPERIMENTAL RESULTS

The proposed algorithms were coded in C++. Benchmark circuits, selected from the ISCAS suite were synthesized and mapped to a 90 nm industrial technology. As process parameters, we considered the widths and thicknesses of the eight metal layers needed to route each circuit, resulting in a total of 16 parameters. For each circuit a timing graph was generated having affine cell/interconnect delays annotated as edge properties.

The experimental results for worst-delay corner computation and worst-slack corner computation are presented in Table I. All the experiments where conducted on a machine with an AMD Opteron 850 processor at 2.4 GHz, and 32 GB of RAM. For the benchmark circuits presented, the memory never exceeded 600 MByte. "#C," "#S," and "#N" columns report the number of combinational cells, the number of sequential cells and the number of nets, respectively. "$|V|$" and "$|E|$" report the number of vertices and edges in the corresponding timing graph. For each analysis method, "#Search" and "CPU" report the amount of search and the CPU time in seconds. For path-based methods the amount of search is the number of vertex visits, while for parameter-based methods the amount of search is the number of decisions. The proposed branch-and-bound-based algorithms are exact. The computed worst corner/delay/slack results are the same as their corresponding exhaustive versions. There are no errors or approximations in the analysis. Therefore, due to space constraints we are not showing the corner and delay/slack values.

Table I presents the results for WDC computation in combinational circuits, using path-based and parameter-based approaches. For each approach an exhaustive and a branch-and-bound-based procedure were evaluated. Clearly, the branch-and-bound techniques are quite effective in reducing the amount of search. The computational overhead incurred by the branch-and-bound approaches is largely compensated by the CPU time saved during the search procedure. The path-based approaches seem to be the most effective, even in the exhaustive case. A exception is the design c6288 for which the exhaustive path-based procedure does not terminate after 3000 s, most likely due to the huge number of paths. It is also in this design that the efficiency of the branch-and-bound techniques is most noticeable, as the path-based branch-and-bound procedure completes in less than 9 s.

Table I also presents the results for worst-slack corner computation in sequential circuits, for setup and hold slacks, obtained by applying path-based WDC computation procedures. Once more the branch-and-bound procedure yields a significant reduction in the amount of search, consequently producing a significant reduction in the CPU time. Such reduction is most noticeable for design s38417. Clearly, this problem seems to be much easier to solve than the WDC problem. This is not surprising, since the depth of the combinational blocks in the sequential benchmark circuits is typically much smaller than the depth of the combinational benchmark circuits, and therefore the number of potential paths between two registers it is also much smaller than the number of paths in a combinational circuit.

VII. Conclusion

This paper proposes a set of efficient branch-and-bound-based techniques for automating the computation of the exact worst-timing corners of combinational and sequential circuits, when delays are represented by affine functions of the process parameters. These techniques are particularly adequate for handling variability effects, that are of extreme relevance in the latest nanometric IC technologies. Experimental evidence demonstrates that such techniques are quite effective, outperforming exhaustive approaches by several orders of magnitude.

Acknowledgments

The authors would like to thank A. Kuehlmann, C. Albrecht, V. Kariat, and I. Keller for their helpful discussions.

References

[1] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *Proc. Assoc. Comput. Mach./IEEE Des. Automat. Conf.*, Las Vegas, NV, Jun. 2001, pp. 661–666.

[2] S. Bhardwaj, S. B. K. Vrudhula, and D. Blaauw, "Tau: Timing analysis under uncertainty," in *Proc. Int. Conf. Comput. Aided Des.*, San Jose, CA, Nov. 2003, pp. 615–620.

[3] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single Pert-like traversal," in *Proc. Int. Conf. Comput. Aided Des.*, San Jose, CA, Nov. 2003, pp. 621–625.

[4] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proc. Assoc. Comput. Mach./IEEE Des. Automat. Conf.*, San Diego, CA, Jun. 2004, pp. 331–336.

[5] S. Onaissi and F. N. Najm, "A linear-time approach for static timing analysis covering all process corners," *IEEE Trans. Comput. Aided Des.*, vol. 27, no. 7, pp. 1291–1304, Jul. 2008.

[6] L. Guerra e Silva, J. Phillips, and L. M. Silveira, "Efficient computation of the worst-delay corner," in *Proc. Des. Automat. Test Eur. Exhib. Conf.*, Nice, France, Apr. 2007, pp. 1–6.

[7] L. Guerra e Silva, Z. Zhu, J. Phillips, and L. M. Silveira, "Variation-aware, library compatible delay modeling strategy," in *Proc. Int. Federation Inf. Process. Very Large Scale Integr.-Syst. Chip Conf.*, Nice, France, Oct. 2006, pp. 122–127.

[8] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge, U.K.: Cambridge Univ. Press, 2004.

[9] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Oper. Res.*, vol. 14, no. 4, pp. 699–719, Jul.–Aug. 1966.