

# Statistical Path Selection for At-Speed Test

Vladimir Zolotov, Jinjun Xiong, Hanif Fatemi, Chandu Visweswariah

**Abstract**—Process variations make at-speed testing significantly more difficult. They cause subtle delay changes that are distributed in contrast to the localized nature of a traditional fault model. Due to parametric variations, different paths can be critical in different parts of the process space, and the union of such paths must be tested to obtain good process space coverage. This paper proposes an integrated at-speed structural testing methodology, and develops a novel branch-and-bound algorithm that elegantly and efficiently solves the hitherto open problem of statistical path tracing. The resulting paths are used for at-speed structural testing. A new Test Quality Metric (TQM) is proposed and paths which maximize this metric are selected. After chip timing has been performed, the path selection procedure is extremely efficient. Path selection for a multi-million gate chip design can be completed in a matter of seconds.

## I. INTRODUCTION

Achieving the required performance specification for advanced-nanometer contract-manufactured ASICs poses considerable challenges. The effects of process and environmental variations are increasing with each new technology generation [1]. The number of significant sources of variation is also increasing, leading to a variational space of high dimensionality and a wide distribution in ASIC performance [1]. Delay measurements of simple test structures, such as ring oscillators, can no longer guarantee adequate functional performance. Certain process and environmental parameters may in fact affect the ASIC logic differently than its test structures. Moreover, expert knowledge of the design’s functionality is often not available, which dictates that a performance verification methodology must be applicable to a wide variety of designs. Furthermore, the cost of test equipment used must adhere to customer-established budgets.

At-speed structural test (ASST) has become an important tool for filtering out performance-deficient chips. For example, in [2], methods for at-speed built-in self test (BIST) and deterministic test are reported. More recently,

in [3], the authors describe the at-speed test methodology for Freescale’s e600 core. An ASST methodology for contract-manufactured ASICs is also presented in [4].

Existing ASST mainly targets transition delay faults that assume a large and localized change of delay [5]. However, at-speed test focused on detecting these catastrophic defects does not address the problem of delay faults caused by process variation [6]. Delay defects due to process variations manifest themselves as subtle and distributed effects. These small delay changes accumulate along signal propagation paths to cause delay faults. The delay variations exhibit statistical correlation, due to dependence on common process parameters. Moreover, delays owing to unpredictable process changes have assumed a greater share of customer fails than catastrophic defects [7]. As a result, traditional ASST tests are inefficient for detecting process variation delay faults [6]. For example, automatic test pattern generation (ATPG) tailored to the transition delay fault model is likely to select short signal propagation paths on account of their being easier to sensitize. Therefore, at-speed transition fault test is not guaranteed to exercise critical paths required to measure performance [8].

Path delay test has been proposed to verify performance [8], [9]. However, straightforward use of a path delay fault model is impractical because of the large number of paths in a chip. Several publications select paths for delay testing using deterministic (or single-corner) static timing. For example, in [10], a method based on graph traversal to automatically identify the longest path through a gate is reported; in [8], true critical path selection using deterministic static timing by ranking endpoints in order of increasing slack is studied; and in [9], the authors compare the delays of the longest sequential and combinational testable paths through each gate. But paths that are critical at one process corner may not be critical elsewhere in the process space, and *vice versa*. Thus, this approach is not able to identify paths that statistically are the most probable to fail, because deterministic timing does not have information about probability distributions and correlations of path delays.

The delays of different paths on a chip have been observed to correlate if the paths pass through the same gates and interconnects [6]. Hence, methods for critical path selection based on statistical modeling of delays

Dr. Zolotov, Dr. Xiong, and Dr. Visweswariah are with IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. Dr. Fatemi’s work was done when he was a summer intern with IBM Thomas J. Watson Research Center, and he is now with Synposys Inc.

have been reported recently. In [11], a statistical fault coverage metric based on identifying the longest path per gate is presented. However, this would lead to an explosion in the number of paths tested in current ASICs that can have as many as 25 million gates. Moreover, robust path test patterns that have a large number of specified bits (to keep switching activity low) cannot be compacted as efficiently as transition test patterns. Testing a large number of paths would therefore lead to large test data volume and high test cost. In [12], false-path-aware statistical timing analysis to select critical paths for delay test is described. The method uses worst case slacks to identify critical nodes and does not consider process or environmental parameters. The technique presented in [12] assumes variations of gate delays to be mutually independent, which is incorrect because the same process parameters affect all delays. Computation of the longest path going through each gate is proposed in [13]. However, this approach is infeasible for large industrial circuits because of the large number of gates in practical designs and the statistical nature of gate delays. Finally, in [6], critical path selection modeling delay defect size as a random variable is presented. Performance violation due to process variation in defect-free circuits is not investigated. The authors use Monte-Carlo simulation to compute the criticality probabilities of the circuit paths, which is inefficient for large circuit instances. Moreover, no mention is made of actual process or environmental parameters. Furthermore, none of the proposed methods have been combined with an at-speed test methodology.

In this paper, we present an integrated approach to at-speed structural test (ASST) for performance verification, explicitly considering a multidimensional variational process space. We develop a novel technique for optimally selecting paths for detecting delay faults due to process variations. The paths are selected using the results of statistical timing analysis. Our approach is based on a new Test Quality Metric (TQM). Tight bounds on the TQM are developed which are leveraged in an efficient branch-and-bound path enumeration procedure. Our experiments on industrial chips having multi-million gates show that the proposed algorithm is able to select a thousand paths within a few seconds of CPU time.

The rest of this paper is organized as follows. Section II provides the necessary background while Section III discusses the fault model and our integrated methodology for ASST. Section IV describes our novel algorithm for statistical path selection based on the concept of a test quality metric. Section V describes implementation of the proposed technique and experimental results. Section VI draws conclusions and discusses

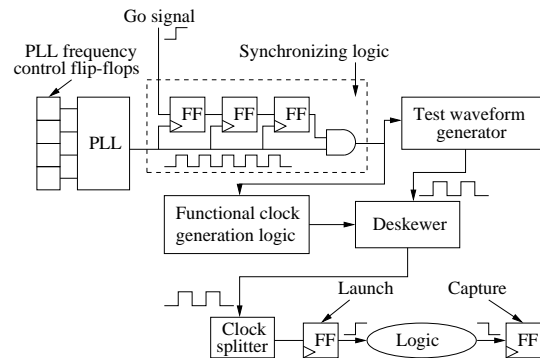


Fig. 1. Illustration of ASST methodology.

future work.

## II. BACKGROUND AND MOTIVATION

### A. Chip performance testing

Various flavors of ASST are described in several text books and papers. Here we assume the ASST architecture described in [4]. This approach applies to ASIC chips with level-sensitive scan design (LSSD).

ASST, illustrated in Figure 1, is used to exercise each clock domain of the ASIC at its specified performance while using a low-cost tester. For ASST, we first scan bits into the phase-locked loop (PLL) frequency control latches and lock the PLLs for correct operation. Once the PLLs are operational, an asynchronous GO signal is asserted by the tester to begin an at-speed (or faster) functional clock sequence. Details of the GO synchronizing latches are presented in [4]. The test waveform generator (TWG) consists of two  $n$ -bit shift registers that can be programmed to create any arbitrary functional clock waveform for  $n$  clock cycles. This is especially useful in the event that a complex functional clocking sequence of a certain length is needed to sensitize a critical functional path. The TWG outputs are combined to create the at-speed functional clock pulses for test using the deskewer. Details of the TWG, deskewer and ASST signal can be found in [4].

A detailed view of the testing logic is shown schematically in Fig. 2. In scan mode, the tester writes test patterns into the chip flip-flops and reads test responses from the flops. The on-chip functional clock launches transitions corresponding to test patterns stored in the flops. The transitions propagate through combinational logic to destination flops where the response is captured by the clock. The propagation of the transition through combinational logic occurs during one clock cycle. If the delay of the combinational circuit is longer than the clock cycle, the capture flop gets a wrong response. The correctness of the response is verified by the tester after

scanning out the state of the flops. The launching of the test transition and capturing of the response is performed at the actual chip frequency, which can be much higher than the tester frequency. The separate use of scan clock and testing clock enables us to employ a low-cost tester with low frequency, requires no functional design changes by the customer, and imposes little hardware overhead.

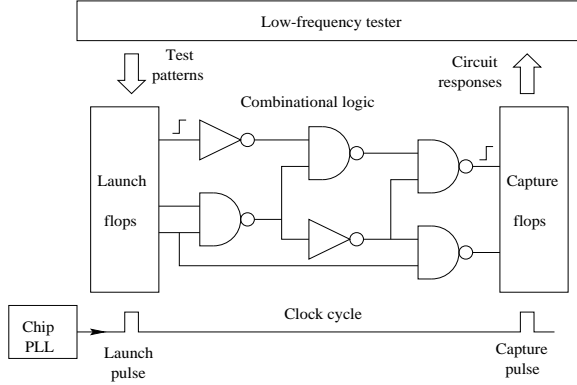


Fig. 2. At-speed structural testing.

The test consists of pairs of boolean vectors. Each pair tests the combinational circuit by activating its paths, i.e., sensitizing the propagation of a test transition through them. By sweeping the clock frequency it is possible to measure the time of signal propagation. The on-chip Test Waveform Generator (TWG) enables the lengthening and shortening of clock pulses. However, since this procedure occupies too much tester time, it is useful for laboratory work but not for manufacturing test. Manufacturing test typically occurs at a fixed frequency with a pass or fail result for each chip.

In order to ensure quality, the test should be designed to screen out as many bad chips as possible. On the other hand, the test should not be too long because tester time is expensive.

Test patterns for detecting delay defects are generated in two steps. First, the delay defect to be tested is specified. Second, the set of test patterns to detect this defect is generated. Test patterns can optionally be compacted to achieve faster turnaround time on the tester. Automatic test pattern generation (ATPG) for detecting specified delay defects is well-known in the literature. Therefore, here we only consider the selection of paths based on a path delay fault model. Interestingly, for detecting process variation defects, it is not necessary to test all paths. Paths delays are correlated with each other. Therefore it is enough to test a subset of paths that cover a significant range of process parameter variations. In this paper, we consider the problem of selecting

paths that are most suitable for testing process variation defects.

Our method of path selection is based on statistical timing analysis, so we provide below a brief outline of static timing and its mathematical models.

### B. Deterministic static timing analysis

Static timing analysis models a digital circuit by a timing graph. Nodes represent pins of gates and wires, while edges represent all pin-to-pin transitions. Both the data and clock networks are modeled and timed. The timing graph is leveled and early and late arrival times (ATs) propagated forward. At flops, setup and hold tests define required arrival times (RATs) at the clock and data pins. Other timing tests similarly define RATs at the corresponding nodes of the timing graph. The graph is then reverse-leveled and early and late RATs are propagated backwards.

Every path has a *start point* and *end point*. For flop-to-flop paths, which are of most interest for testing purposes, the start point is the output data pin of a launching flop and the end point is the input data pin of a capturing flop. Fig. 3 shows the timing graph corresponding to the combinational part of the circuit of Fig. 2. The nodes corresponding to gate output pins are shown in black. In general, start points include primary inputs, and end points include primary outputs, although paths involving chip I/Os are much more difficult to test at-speed.

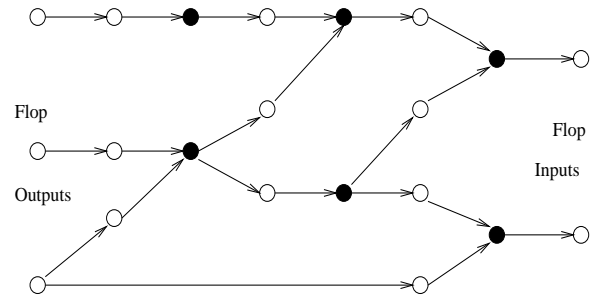


Fig. 3. Example of a timing graph.

The definitions and theorems below assume late-mode timing, but they can be readily extended to early-mode timing.

*Definition 1:* The timing slack of a node  $i$  is defined as  $RAT(i) - AT(i)$ .

The slack shows how much earlier than required a signal arrives. Negative slack implies a timing violation at some end-point fed by this node.

*Definition 2:* Timing slack of a path  $\pi$  that goes from node  $a$  to  $b$  with delay  $d_\pi$  is defined as  $RAT(b) - d_\pi - AT(a)$ .

Path slack shows how much faster the signal propagates through a path than required. Negative path slack implies that the path is too slow and causes a timing violation.

The following theorems are commonly used in static timing.

*Theorem 1:* Let node  $n$  have timing slack  $s$ . Then among all paths  $\pi$  going through this node there is at least one path  $\pi_{crit}$  whose slack  $s_{crit} = s$ . Moreover, among these paths there is no path  $\pi_i$  with slack  $s_i < s$ .

*Definition 3:* The slack of a set of paths  $\Pi = \{\pi_1, \pi_2, \dots\}$  is defined as the minimum of the timing slacks of these individual paths.

*Theorem 2:* For any ensemble of paths  $\Pi$  with slack  $s$ , there is at least one path  $\pi_{crit}$  whose slack  $s_{crit} = s$ . Moreover, among these paths there is no path  $\pi_i$  with slack  $s_i < s$ .

*Definition 4:* Circuit slack is the timing slack of all paths in the circuit, i.e., the minimum of path slacks across all paths.

Circuit slack is clearly also the minimum of the timing slack at all end-points. If timing slack of a circuit is negative, then the circuit has timing violations.

### C. Statistical static timing analysis

Block-based statistical static timing analysis (SSTA) is a natural extension of deterministic STA. SSTA models timing quantities (delays, ATs, RATs, slacks, slews or rise/fall times, setup and hold times) not as numbers but as parameterized probability distributions [14], [15]. Then theorems 1 and 2 can be reformulated as follows.

*Theorem 3:* Let node  $n$  (or ensemble of paths  $\Pi$ ) have statistical slack  $s$ . Then for any path  $\pi_i$  going through this node (or constituent path  $\pi_i$ ) and any real number  $t$ ,  $P(s_i \leq t) \leq P(s \leq t)$ . In other words, the CDF of  $s$  is the left-most of all CDFs  $s_i$ .

The most common and widely used statistical timing model is a linear canonical form [14], [15] in which any timing quantity  $T$  is represented by a linear affine form

$$T = T_0 + \sum_{i=1}^n a_i \Delta X_i + a_R \Delta R, \quad (1)$$

where  $\Delta X_i$  and  $\Delta R$  are normalized Gaussian random variables with zero means. Variables  $\Delta X_i$  model globally correlated variations of process parameters. Variable  $\Delta R$  models uncorrelated variations.  $T_0$  is the mean or nominal value of the timing quantity. Coefficients  $a_i$ ,  $a_R$  are sensitivities to the corresponding variations. All the operations on timing quantities (addition, subtraction, minimum and maximum) required for SSTA are extended to this representation.

Recently, many versions of SSTA have been proposed, extending the canonical model and concomitant operations to nonlinear delay dependencies and non-Gaussian distributions. Although we present our path selection technique in the context of linear Gaussian statistical timing, it can easily be extended to any form of block-based statistical timing. In the rest of this paper, we will assume that statistical timing adjusts due to CPPR (Common Path Pessimism Removal, see [16]) and PLL feedback have already been applied before the final RAT calculation in a backward sweep.

## III. INTEGRATED ASST METHODOLOGY

### A. Delay faults due to process variations

Delay faults due to process variations are quite different from traditional transition delay faults. The difference between a good and faulty chip in the case of process variation faults is quite subtle. The location of the fault in a faulty chip cannot be specified; all we can say is that there exists at least one path whose delay exceeds the cycle time. The excess delay is distributed over all the gates and wires of the path. In fact, some gates and wires may speed up and others slow down, but the net effect is a timing violation.

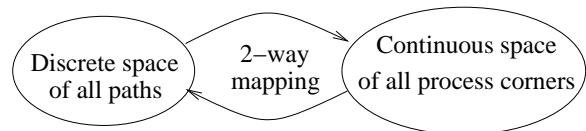


Fig. 4. Mapping between paths and process space.

In traditional delay testing, typically one fault is targeted at a time. Several practical reasons predicate this decision. First, the probability of multiple simultaneous faults is assumed to be low. Second, it is too difficult to target multiple faults simultaneously due to the enormous search space. Third, good single-fault testing is assumed to serendipitously lead to reasonable multiple-fault detection. Most of these assumptions are not valid for path delay testing. In fact, the existing literature does not fully consider the difference between testing individual delay faults and process variation delay faults. This difference significantly affects construction of an optimal test for process variation delay faults.

Process variations cause correlated effects and almost all paths are simultaneously impacted (but to different degrees) due to process variations. Fig. 4 shows the mapping between the discrete path space and the continuous process space. For each path, there is a point/subspace in the process space where this path has the worst slack. Likewise, for each process corner (i.e., unique

combination of process parameter values), there exists one or more paths which is/are the most critical at that process corner. During manufacturing test, we can detect path delay changes only if the faulty delay exceeds cycle time. So it is clear that the more critical the path tested, the higher the likelihood of an effective test. Loosely speaking, the goal is to find the superset of all paths that can be critical anywhere in the process space, or all combinations of process parameters that can cause timing fails and representative corresponding paths that fail. The good news is that since path delays exhibit myriad correlations, a small number of paths is able to cover a large multi-dimensional process space.

### B. At-speed test generation

Fig. 5 shows the flow chart for our integrated ASST methodology. The process begins by reading in the design netlist, timing models for library cells, and related timing constraints. We then run statistical timing in a special ASST mode to compute the signal ATs, RATs, and slacks as functions of random variables in the form (1). From the statistical distributions of the random variables, we can predict the probability of signals being correct at the flip-flops for a given clock frequency. This enables us to compute the criticality of every node in the design, which is defined as the probability of each node being on the critical path among all manufactured chips. There exist efficient algorithms to compute node criticalities [17], [18]. The circuit nodes having the highest criticalities are selected.

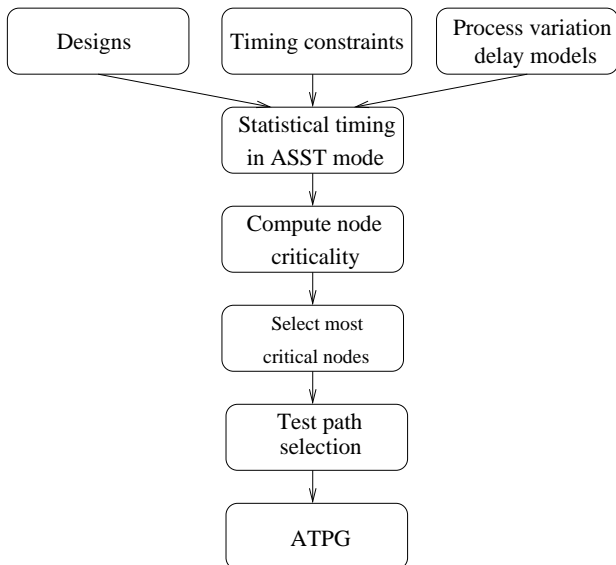


Fig. 5. Test generation for process variation faults.

Next, optimal and efficient path selection is carried out. This step is the heart of the contribution of this

paper, and we will discuss it in detail in the next section.

Finally, the selected paths are supplied to automatic test pattern generation (ATPG) software for generating test patterns to sensitize the selected paths.

## IV. PATH SELECTION FOR ASST

Among the billions of paths in a chip, our goal is to select a minimal set which is most likely to detect delay defects due to process variations. In order to mathematically formulate this problem we need a formal metric of test quality.

### A. Test Quality Metric (TQM)

The goal of ASST is to screen out as many faulty chips as possible. Thus, we define the metric  $Q(\Pi)$  as the probability that a tested chip has no timing violations conditional on paths  $\Pi$  passing at-speed testing.

$$Q(\Pi) = P(\text{Chip\_is\_good} | \text{Test\_passed}). \quad (2)$$

Let  $S_C$  be the statistical chip slack in worst-case field conditions (i.e., worst-case voltage, temperature and aging). Now consider the subset of the chip that comprises the paths being tested during ASST. Let  $S_T$  be the “test slack” of this subset of the chip, timed at the conditions experienced in the test chamber (including mode bit settings for ASST mode rather than functional mode and inclusion of only ASST-testable timing tests, both of which make important timing changes). Now we can express TQM in terms of chip and test slacks  $S_C$  and  $S_T$ .

$$Q(\Pi) = P(S_C \geq 0 | S_{\Pi} \geq 0). \quad (3)$$

Taking into account that the slack of the set of tested paths is the statistical minimum of their individual slacks  $s_i$ ,

$$Q(\Pi) = P(S_C \geq 0 | \min(s_i) \geq 0). \quad (4)$$

If block-based statistical timing computes chip, path and test slacks in a parameterized statistical form, for example, the linear canonical form (1), then TQM can be computed as the conditional probability (4).

Our goal is to select paths with the highest TQM. At first glance, the problem may seem similar to a longest path algorithm. However, longest path computation exploits the fact that any sub-path of the longest path is also the longest path between its start and end points. These properties permit a dynamic programming approach to path tracing employed by static timing analysis programs to generate path reports. Unfortunately, these properties do not hold for graphs with edges weighted with expressions like (1) and the metric (4). Therefore, we use general depth-first path traversal combined with branch-and-bound to solve this more difficult problem.

### B. TQM bounds

Branch and bound depends critically on the quality of bounds for its efficiency. Ideally, bounds should be tight and easy to compute. We begin by rewriting (3) as

$$Q(\Pi) = P(S_C \geq 0 | S_{\Pi} \geq 0) = \frac{P(S_C \geq 0, S_{\Pi} \geq 0)}{P(S_{\Pi} \geq 0)}. \quad (5)$$

$S_C$  represents *all* paths of the chip and therefore is always less than the slack of any set of paths  $\Pi$ . Thus if chip slack  $S_C \geq 0$ , it implies  $S_{\Pi} \geq 0$ . So we can simplify (5) as

$$Q(\Pi) = \frac{P(S_C \geq 0)}{P(S_{\Pi} \geq 0)}. \quad (6)$$

$S_C$  does not depend on the paths selected for testing. Therefore, we can use a simpler surrogate metric

$$q(\Pi) = P(S_{\Pi} \geq 0) \quad (7)$$

which we minimize in order to maximize TQM.

Our bounds are based on the well-known formula for timing slack of a set of paths going through a common segment. Fig. 6 shows this configuration. The timing slack of the set  $\Pi$  of *all the paths* going through a path segment  $Z_{1,n} = (a_1, a_2, \dots, a_n)$  is given by

$$S_{\Pi} = RAT(a_n) - AT(a_1) - D(Z_{1,n}) \quad (8)$$

where  $D(Z_{1,n})$  is the delay of path segment  $Z_{1,n}$ . A similar formula is used in [17] for criticality computation.

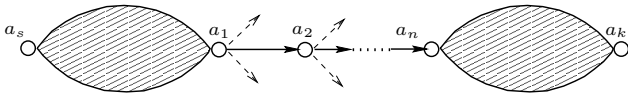


Fig. 6. Set of paths going through a common segment.

The slack  $s_i$  of any path  $\pi_i$  from the set of paths  $\Pi = \{\pi_1, \pi_2, \dots\}$  is not worse (i.e., not smaller) than the slack  $S_{\Pi}$  of this set of paths. Thus, we have an inequality for our metric

$$q(\pi_i) = P(s_i \geq 0) \geq P(S_{\Pi} \geq 0). \quad (9)$$

The lower bound on  $q(\pi_i)$  is thus  $P(S_{\Pi} \geq 0)$ . This lower bound together with (8) gives us a method to quickly check if the best candidate path going through a path segment can possibly be better than a previously found path or set of paths.

### C. Basic path selection algorithm

Our path selection method is a branch-and-bound (**BnB**) algorithm. We describe the main principles of our approach on a basic version of the algorithm that selects only a single path with the minimum value of the metric

(7). Then we extend the method to selection of the set of paths that are the most promising for detecting process variation delay faults.

The algorithm constructs paths starting from end points and tracing backwards towards start points. The operation of path selection can be considered from two points of view: traversal of the timing graph and traversal of a decision tree. The decision tree for the timing graph of Fig. 3 is shown in Fig. 7. Each node of the decision tree represents a unique path from an end-point to the node. The leaves of the tree represent paths between end points and start points. The decision tree is never constructed explicitly. It is just a representation of a depth first search (DFS) tree of graph traversal. At each node of the decision tree the algorithm makes a decision either to continue expansion of the path or not. Therefore, the algorithm traverses only a small part of the whole tree.

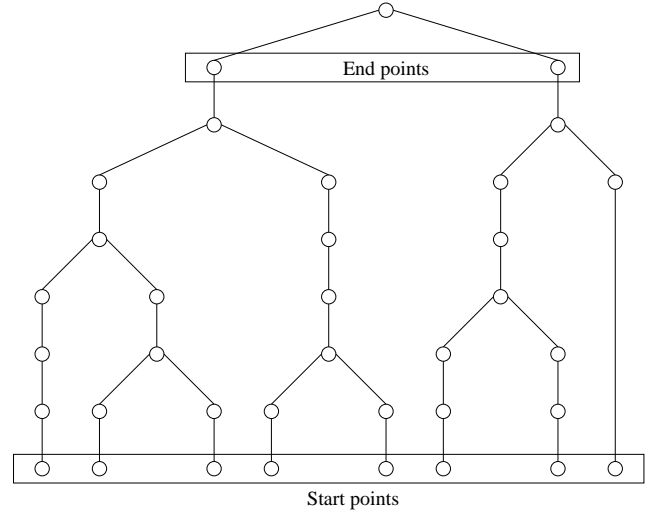


Fig. 7. Decision tree for path selection.

The pseudocode of our **BnB** algorithm is as shown in Fig. 8.

---

Set the best path found  $\pi = \phi$ .  
 Set TQM of the best path  $q(\pi) = 1$  (the worst value).  
 For each end point  $a_j$ , call  
**Recursive Traversal Procedure** with parameters:  
 (1) the best path found  $\pi$  and its metric  $q(\pi)$ ;  
 (2) the current node  $a_{cur} = a_j$  from which to continue traversal;  
 (3) the sub-path being traversed  $\pi_{cur} = (a_j)$  (At this moment the sub-path has only one node);

---

Fig. 8. Pseudoe code for our branch-and-bound **BnB** algorithm.

The algorithm of the **Recursive Traversal Procedure**

is further detailed as shown in Fig. 9.

---

If the current node is a start point

- (a) Compute the metric  $q(\pi_{cur})$  of the current candidate path  $\pi_{cur}$  and compare it with the metric of the best path found  $q(\pi)$ .
- (b) If  $q(\pi_{cur}) < q(\pi)$   
Update the best path  $\pi = \pi_{cur}$  and its metric  $q(\pi) = q(\pi_{cur})$ .
- (c) Return back.

Compute bound  $q_b$  of the metric for the paths going through the currently traversed sub-path  $\pi_{cur}$  using formulas (8), (9).

If  $q_b \geq q(\pi)$

Return back because this direction of query is guaranteed to be fruitless.

Otherwise for each predecessor  $a_{cur+1,j}$  of the current node  $a_{cur}$ , call

- Recursive Traversal Procedure with parameters:
- (1) the best path found  $\pi$  and its metric  $q(\pi)$ ;
  - (2) the current node  $a_{cur} = a_{cur+1,j}$  from which to continue traversal;
  - (3) the sub-path being traversed  $\pi_{cur} = (a_{cur+1,j}, \pi_{cur})$  (Extend the current sub-path with the currently processed node).
- 

Fig. 9. Pseudo code for the recursive traversal procedure.

From this description we see that our BnB algorithm iterates through all the end points of the timing graph and performs backward traversal. At each step the algorithm deals with a subgraph shown in Fig. 10. The path  $a_k, a_{k-1}, \dots, a_1$  is already traversed by the algorithm. The algorithm uses (8), (9) to estimate the lower bound of the test quality metric for any path going through node  $a_k$ . This estimate is compared with the test quality metric  $q_\pi$  of the best path  $\pi$  found so far. From this comparison the algorithm decides either to continue the current traversal further or to return back to continue previously postponed depth-first traversals. If the algorithm decides to continue the current traversal, it iterates through nodes  $a_{k+1,1}, a_{k+1,2}, \dots, a_{k+1,n}$  and processes them by the same Recursive Traversal Procedure. When the algorithm arrives at a start point it has the full path  $\pi_{cur}$  from the start point to the end point. The algorithm compares the test quality metric of this path and the best path  $\pi$  found so far. If the newly found path  $\pi_{cur}$  is better, the algorithm replaces the best path found. After that the algorithm returns to continue previously postponed traversals.

Note that, for simplicity of presentation, the above branch-and-bound algorithm is presented through a backward traversal of the timing graph. In practice, this can

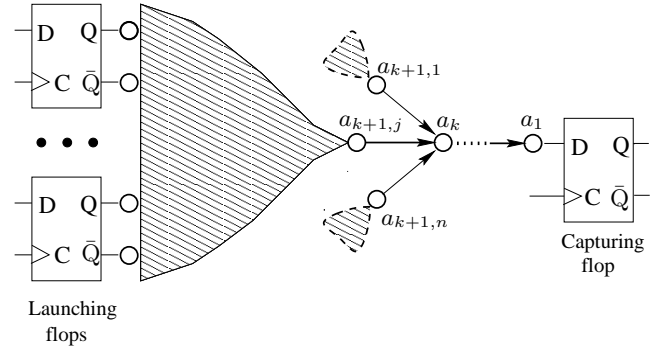


Fig. 10. Subgraph for path traversal.

be done through either forward traversal or even two-way traversal.

#### D. Extensions of the path selection algorithm

The basic path selection algorithm selects only a single path with the best metric. For testing purposes, we need to select many paths to ensure good detection of process variation delay faults. The most obvious way of selecting several paths is by multiple applications of the basic selection algorithm. Obviously this is not very efficient because the same timing graph is traversed multiple times. However, the basic algorithm can easily be modified to select many paths by a single traversal.

One problem statement is to select  $N$  top paths with the best individual single path metric (or SPM in short). Our basic BnB algorithm can be modified to solve this problem, and we denote this algorithm as **BnB-SPM**. The difference between BnB-SPM and the basic BnB algorithm is *The Recursive Traversal Procedure*, which requires the following modifications:

- Instead of updating a single best path we accumulate  $N$  best paths at any given time.
- Each time we compute the bound  $q_b$  of the metric of paths going through a current sub-path  $\pi_{cur}$ , we compare it with the metric of the worst path  $\pi_{worst}$  from the set of paths found. If the bound is better than the metric of the worst path found we continue the traversal otherwise we return back to continue the postponed traversals.
- Each time we get to the start point of the candidate path  $\pi_{cur}$  we compare its metric with the metric of the worst path  $\pi_{worst}$  from the set of paths found. If  $\pi_{cur}$  is better than  $\pi_{worst}$  we substitute  $\pi_{cur}$  for  $\pi_{worst}$  in the set of the best paths found.

Another problem is selection of  $N$  paths  $\pi_1, \pi_2, \dots, \pi_N$  such that together they have the highest joint path metric (or JPM in short). An efficient (though potentially sub-optimal) solution can be obtained by

combining the basic BnB algorithm with a greedy selection method, which is denoted as **BnB-JPM**. The modification of BnB-JPM to the basic BnB algorithm is an altered *Recursive Traversal Procedure* as shown below:

- Instead of updating a single best path we accumulate  $N$  best paths at any given time.
- Each time we process a sub-path  $\pi_{cur}$  we check whether among the best paths found there is a worst path  $\pi_{worst}$  such that if we substitute a path with the slack of sub-paths  $\pi_{cur}$  for this path, then the total quality metric is improved. If so, we continue the traversal for extending the sub-path  $\pi_{cur}$ . Otherwise we return to continue the depth-first traversal.
- Each time we get to the start point of the candidate path  $\pi_{cand}$ , we check whether among the best paths found there is a worst path  $\pi_{worst}$  such that if we substitute  $\pi_{cand}$  for  $\pi_{worst}$ , the total quality metric of the set of paths is improved. If so, we substitute  $\pi_{cand}$  for  $\pi_{worst}$  in the set of  $N$  best paths.

Similarly we can construct an algorithm for selecting a minimal set of paths with a user-specified value of the total quality metric. Even though the greedy selection in BnB-JPM may result in suboptimal solutions, our experiments show that it in fact produces solutions with good quality.

### E. Complexity Analysis

The complexity of our branch-and-bound based algorithm depends on three factors: (F1) the effectiveness of our bound metric for pruning; (F2) finding the worst path  $\pi_{worst}$  to check if replacement is needed; and (F3) replacement of the worst path with the updated metric. This section shows how (F2) and (F3) can be rendered efficient by appropriate choice of data structures. The effectiveness of our bound metric (i.e., (F1)) is empirically demonstrated in the results section.

For the basic BnB algorithm, it is trivial to show that (F2) and (F3) can be performed in constant time as only one worst path is kept and its TQM can be trivially updated. As for the BnB-SPM algorithm, we employ a priority queue data structure to maintain the  $N$  paths and their TQMs. In this case, (F2) can be performed in constant time, while (F3) can be achieved in  $O(\log N)$  or  $O(\log \log N)$  time, depending on how the priority queue is implemented.

The BnB-JPM algorithm needs to maintain the joint path metric (or JPM) during graph traversal. A straightforward implementation would require at least  $O(N^2)$  time for (F2), as we need to loop through all  $N$  paths to see which one should be replaced by the current path

(segment), and each step in the loop involves computation of JPM with  $N - 1$  existing paths and the current path, an  $O(N)$  operation by itself. To make it more efficient, we employ a similar *binary partition tree* data structure as suggested in [17] with some modifications.

For simplicity, we assume that the required path number  $N = 2^t$  for some integer  $t$  with the understanding that the concept is applicable to arbitrary  $N$ . We build a balanced binary partition tree as shown in Fig. 11, where each leaf node corresponds to one of the  $N$  paths. The goal is to be able to compute the slack of a new set of paths from the old set in which just one path has been replaced. The initial setup is as follows: each leaf node contains the corresponding path's slack  $s_i$ ; through one bottom-up traversal of the tree, we compute the statistical minimum slack of every node's two direct child nodes, and save it at the node (at the root, we have the minimum slack of all  $N$  paths); through another top-down traversal of the tree, we compute the complement slack of every node by taking the statistical minimum of its parent node's complement slack and its sibling node's slack (with the root's complement slack initialized to infinity). At the leaf node, the complement slack would be the minimum path slack of the other  $N - 1$  paths, which is combined with the slack of the new path.

This is illustrated by an example in Fig. 11 in which  $a$  through  $h$  represent 8 paths. Each node of the tree has two values shown one below the other, i.e., the *min* slack of the paths in that sub-tree, and the *min* slack of the paths *not* in the sub-tree. The upward and downward traversals necessary to compute the complement slack of  $a$  are shown by dotted arrows, so now we are in a position to evaluate the impact of replacing one path by a new candidate path.

Both bottom-up and top-down traversal<sup>1</sup> of the tree take  $O(N)$  time. Once initialization is done, (F2) only needs  $O(N)$  time by looping through the leaf nodes only once; and each step of the loop takes  $O(1)$  time to compute the new JPM, as the minimum slack of all other  $N - 1$  paths is known. After finding which path to replace (if replacement is needed), (F3) can be done again in  $O(N)$  time by one bottom-up leaf-to-root traversal and updating those nodes' minimum slack, followed by another top-down traversal and updating all nodes' complement slacks. Afterwards, the data structure is up-to-date with the replaced path. So the complexity of (F2) and (F3) is  $O(N)$ .

<sup>1</sup>Note that during the bottom-up or top-down traversal (updates) of the binary partition tree, we can further speed-up the traversal with early termination of the traversal if we detect that the effects of updating will not propagate further.

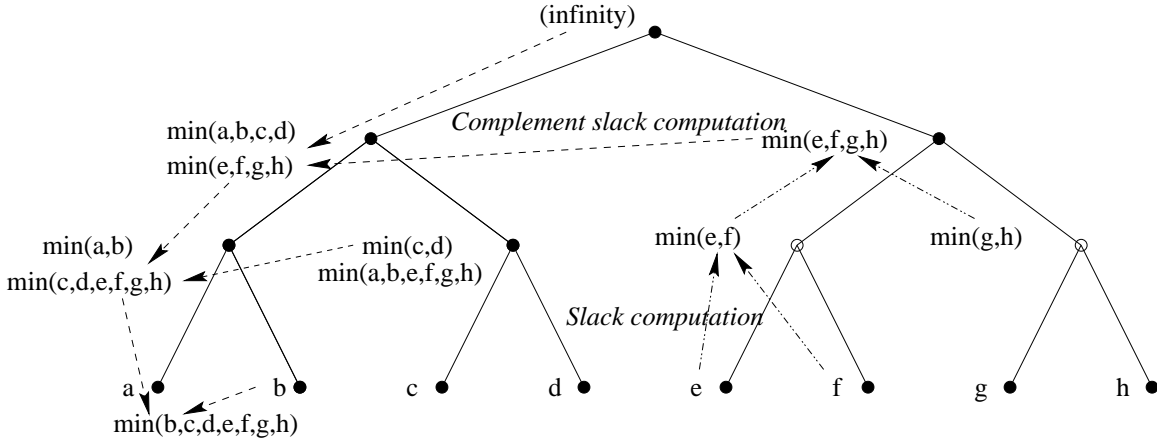


Fig. 11. Binary partition tree.

TABLE I

STATISTICS OF BRANCH-AND-BOUND ALGORITHMS WITH TEST SIZE = 60 FOR BOTH BnB-SPM AND BnB-JPM.

Design	# cells	BnB-SPM					BnB-JPM				
		Paths inspected	Paths rejected	Path depth (min/max/avg)	Branches pruned	Prune depth (min/max/avg)	Paths inspected	Paths rejected	Path depth (min/max/avg)	Branches pruned	Prune depth (min/max/avg)
D1	4.5K	243	183	17/25/21.4	311	1/28/9.7	67	7	15/27/22.5	63	1/16/5.0
D2	382.3K	275	215	49/79/71.1	2158	1/72/45.5	73	13	37/75/67.9	354	1/68/22.0
D3	1.2M	221	161	1/23/12.4	173	1/16/4.1	62	2	1/31/18.1	96	1/4/1.1

## V. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented the proposed BnB-SPM and BnB-JPM algorithms in C++ with an industry quality sign-off SSTA engine, EinsStat [14]. For comparison purpose, we have also implemented the deterministic (DTR) path selection algorithm. The only difference between DTR and our branch and bound based algorithms is the way paths are selected, i.e., DTR selects the top  $m$  least slack paths based on conventional multi-corner timing. For fair comparison, each path from DTR is also evaluated using the metrics proposed in this paper based on either individual single path TQM (SPM) or joint path TQM (JPM).

A set of industrial 90 nm and 65 nm ASIC designs are used as benchmarks to test the algorithms. The process variation data are set according to foundry rules, and they include both front-end device parameters and back-end metal parameters. The sizes of the design in terms of placeable objects are shown in column 2 of Table I.

### A. Efficiency of BnB Algorithms

We first present evidence on the quality of our bounds, a key contributor to the efficiency of our algorithms. Table I shows the number of complete paths considered during the branch and bound procedure, a small fraction of the total number of paths in the chip; paths that were rejected (i.e., removed from the list of top  $N$  paths at some point in the algorithm); path depth of considered

paths; branches pruned and prune depth. A low prune depth indicates early pruning in the timing graph whereas a high prune depth means that the depth-first search had to work hard before being able to prune branches. From the table, it is clear that the pruning is extremely effective (from the small number of paths rejected and low prune depth relative to the graph depth) and hence the path tracing procedure is extremely efficient. These conclusions are equally valid for BnB-JPM as shown in the right half of Table I.

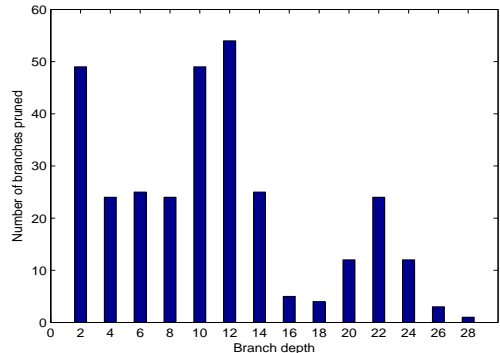


Fig. 12. Histogram of BnB-SPM pruning branches for D1.

To further show the effectiveness of our bound metric used for pruning, we report the distribution of pruning depth for design D1 in Fig. 12. As can be seen, even though the logic depth of paths can be quite long ( $> 28$ ),

our BnB algorithm is able to prune a lot of branches in the early stages of the search, thus greatly reducing our search space.

We also show the run-time scalability of our BnB algorithms as a function of test size in Fig. 13 for design *D3*. From the figure, we clearly see that both BnB-SPM and BnB-JPM scale almost linearly with test size. For a million gate design, we are able to find thousands of paths within a few seconds. The efficiency is made possible because of both effective pruning and fast solution updating due to good data structures.

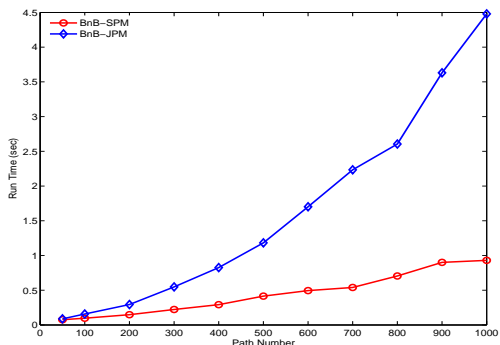


Fig. 13. Run time scalability for *D3*.

### B. Test Quality of BnB Algorithms

For a given test size  $N$ , the BnB-SPM algorithm intelligently searches the entire solution space and finds paths which optimize the targeted TQM. Fig. 14 shows the TQM of every single path generated by the BnB-SPM algorithm for design *D1*, with the required test size set as 1, 5, and 10. Each bar of the plot shows the TQM of a selected path. We see that every larger test is an exact superset of smaller tests, and that TQM of selected paths decreases monotonically. This optimality property is especially desirable for testing, as it gives test engineers the guarantee that the best set of paths is tested irrespective of the path budget.

In contrast, the optimality property does not hold for the DTR algorithm. To show this, we repeat the same experiment for DTR as above, and report the results in Fig. 15. Because DTR selects paths based on the least slack according to multi-corner timing, it does not consider the quality from a process space coverage point of view, hence the least slack path does not necessarily correspond to the best TQM. For example, by comparing the single path test with the test having 5 and 10 paths, we see that the TQM of the only path of the first test is lower than the TQM of many paths of the latter two tests. This translates to the fact that by increasing the

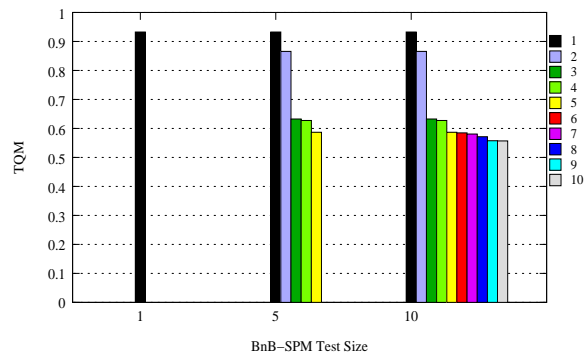


Fig. 14. BnB-SPM's test quality metric TQM for different test sizes.

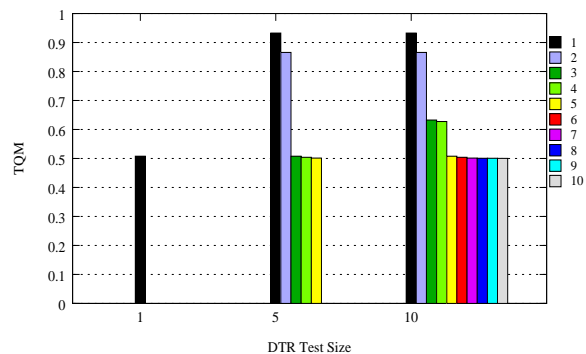


Fig. 15. DTR's test quality metric TQM for different test sizes.

number of paths, it is possible to find a better path than an already-chosen path. With this uncertainty and sub-optimality, test engineers can never be sure that the best path set is being tested.

We also compare the quality of path selection between BnB and DTR. Irrespective of the algorithm applied, the selected path set is evaluated by both the SPM and JPM metrics, and the results are reported in Table II. As can be seen, BnB-SPM always finds the best test set in terms of SPM metrics compared to DTR and BnB-JPM, as evidenced by larger SPMs. This is expected as BnB-SPM targets the best SPM metric for each individual path. Not surprisingly, BnB-JPM, which targets the best JPM, may in fact find paths with less favorable SPM compared to BnB-SPM and DTR. But the joint metric of BnB-JPM is the best among the three. Interestingly, we also note that even though BnB-SPM is not targeting JPM directly, the produced JPM metrics are reasonably good. This shows the merit of the SPM metric in guiding us finding a good test set.

Fig. 16 shows the TQM comparison between DTR and BnB-SPM based on SPM metrics for *D1* with test size as 60. We sort all paths' TQM from both algorithms in decreasing order and plot TQM against the sorted path ID. We observe that the BnB-SPM algorithm always produces higher TQM than DTR, which further

TABLE II  
TQMs COMPARISON WITH TEST SIZE = 60.

Design	DTR	BnB-SPM	BnB-JPM
SPM (min/max/avg)			
D1	0.500/0.901/0.628	0.587/0.932/0.694	0.500/0.932/0.560
D2	0.500/0.766/0.619	0.691/0.766/0.706	0.557/0.766/0.628
D3	0.500/0.784/0.536	0.542/0.784/0.576	0.500/0.784/0.535
JPM			
D1	0.922	0.953	0.958
D2	0.775	0.771	0.777
D3	0.977	0.980	0.997

illustrates the superiority of our BnB-SPM algorithm.

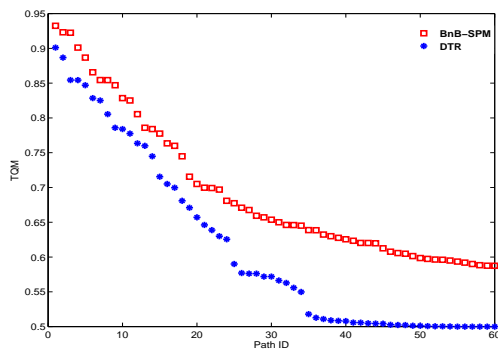


Fig. 16. TQM comparison between BnB-SPM and DTR.

## VI. FUTURE WORK AND CONCLUSIONS

This paper presented a novel algorithm for optimal path selection for at-speed testing. A test quality metric (TQM) was defined and maximized during branch-and-bound traversal of the timing graph, solving the hitherto open problem of statistical path tracing. Numerical results demonstrated the consistency and efficiency of the algorithm. Chips with millions of gates were processed in a matter of seconds to generate paths for testing. The proposed algorithm is useful not only for testing, but for reporting the statistically most critical paths, and for variation-aware optimization of the design.

This research suggests some avenues of future work. First, the path selection procedure in this paper does not take into account that some paths may be unsensitizable or difficult to sensitize during the ATPG step. This can be either taken into account by selecting redundant paths, or invoking ATPG in the inner loop of the path selection and rejecting unsensitizable paths. Second, a different problem formulation suggests itself. Although at each step of the algorithm the test quality metric is maximized, the cumulative coverage of the process space by a given set of paths may not necessarily be optimal since multiple paths can occupy the same or similar regions of the process space. The greedy approach proposed works

well in practice, but is not optimal. One possible way to improve the quality of these solutions is by accumulating more paths than necessary and then selecting the best ones from them.

## REFERENCES

- [1] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold. Process and environmental variation impacts on ASIC timing. *IEEE International Conference on Computer-Aided Design*, pages 336–342, November 2004. San Jose, CA.
- [2] B. Bailey, A. Metayer, B. Svrcek, N. Tendolkar, E. Wolf, E. Fiene, M. Alexander, R. Woltenberg, and R. Raina. Test methodology for Motorola’s high-performance e500 core based on PowerPC instruction set architecture. *International Test Conference*, pages 574–583, 2002.
- [3] N. Tendolkar, D. Belete, A. Razdan, H. Reyes, B. Schwarz, and M. Sullivan. Test methodology for Freescale’s high performance e600 core based on PowerPC instruction set architecture. *International Test Conference*, pages 1–9, 2005.
- [4] V. Iyengar, T. Yokota, K. Yamada, T. Anemikos, R. Bassett, M. Degregorio, R. Farmer, G. Grise, M. Johnson, D. Milton, M. Taylor, and F. Woytowich. At-speed structural test for high-performance ASICs. *Proc. International Test Conference*, pages 2.4:1–10, October 2006. Santa Clara, CA.
- [5] M. L. Bushnell and V. D. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Kluwer Academic Publishers, 2000.
- [6] L-C. Wang, J-J. Liou, and K-T. Cheng. Critical path selection for delay fault testing based upon a statistical timing model. *IEEE Transactions on Computer-Aided Design of ICs and Systems*, 23(11):1550–1565, November 2004.
- [7] K-T. Cheng, S. Dey, M. Rodgers, and K. Roy. Test challenges for deep sub-micron technologies. *Proc. 2000 Design Automation Conference*, pages 142–149, June 2000. Los Angeles, CA.
- [8] A. Crouch, J. C. Potter, and J. Doege. AC scan path selection for physical debugging. *IEEE Design & Test*, 20:34–40, Sept.-Oct. 2003.
- [9] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi, and H. Balachandran. K longest paths per gate (KLPG) test generation for scan-based sequential circuits. *International Test Conference*, pages 223–231, 2004.
- [10] M. Sharma and J. H. Patel. Finding a small set of longest testable paths that cover every gate. *International Test Conference*, pages 974–982, October 2002. Baltimore, MD.
- [11] W. Qiu, X. Lu, J. Wang, Z. Li, D. M. H. Walker, and W. Shi. A statistical fault coverage metric for realistic path delay faults. *VLSI Test Symposium*, pages 37–42, 2004.
- [12] J-J. Liou, A. Krstic, L-C. Wang, and K-T. Cheng. False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation. *Proc. 2002 Design Automation Conference*, pages 566–569, June 2002. New Orleans, LA.
- [13] W. Qiu and D. M. H. Walker. An efficient algorithm for finding k longest testable paths through each gate in a combinational circuit. *International Test Conference*, pages 592–601, September 2003. Charlotte, NC.
- [14] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. *Proc. 2004 Design Automation Conference*, pages 331–336, June 2004. San Diego, CA.
- [15] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. *IEEE International Conference on Computer-Aided Design*, pages 621–625, November 2003. San Jose, CA.

- [16] D. J. Hathaway, J. P. Alvarez, and K. P. Belkhale. Network timing analysis method which eliminates timing variations between signals traversing a common circuit path. *U. S. Patent 5,636,372*, June 1997.
- [17] J. Xiong, V. Zolotov, C. Visweswariah, and N. Venkateswaran. Criticality computation in parameterized statistical timing. *Proc. 2006 Design Automation Conference*, pages 63–68, July 2006. San Francisco, CA.
- [18] J. Xiong, V. Zolotov, and C. Visweswariah. Incremental criticality and yield gradients. *Design and Test in Europe*, March 2008. Messe Munich, Germany.