EffiTest2: Efficient Delay Test and Prediction for Post-Silicon Clock Skew Configuration under Process Variations

Grace Li Zhang, Bing Li, Yiyu Shi *Senior Member, IEEE*, Jiang Hu *Fellow, IEEE* and Ulf Schlichtmann *Member, IEEE*

***Abstract*—At nanometer manufacturing technology nodes, process variations affect circuit performance significantly. This trend leads to a large timing margin and thus overdesign in the traditional worst-case circuit design flow. To combat this pessimism, post-silicon clock tuning buffers can be deployed to balance timing slacks of consecutive combinational paths in individual chips by tuning clock skews after manufacturing. A challenge of this method is that path delays of each chip with timing failures should be measured to gather the information for clock skew configuration. However, current methods for delay measurement rely on path-wise frequency stepping, which requires much time from expensive testers. In this paper, we propose an efficient delay test framework (EffiTest2) to solve the post-silicon testing problem by testing only representative paths with delay alignment using the already-existing tunable buffers in the circuit. Experimental results demonstrate that EffiTest2 can reduce the number of frequency stepping iterations by more than 94% with only a slight yield loss.**

***Index Terms*—Process Variations, Post-Silicon Tuning, Clock Skew, Yield, Path Selection, Delay Test, Statistical Prediction**

I. Introduction

Modern IC design faces tremendous challenges to achieve performance goals while maintaining a profitable yield. For example, at advanced technology nodes, increasing process variations together with aging effects require a very large timing margin, thus causing expensive overdesign. To combat such challenges, process variations may be modeled directly in timing analysis, leading to a boom of research on statistical static timing analysis (SSTA) in the last decade [2]–[11]. With the information of distributions of process variations, SSTA methods produce a performance-yield curve with which designers have a chance to make a tradeoff between different design goals. This method can effectively reduce the timing margin compared with traditional worst-case design, but it still does not counter process variations actively. To alleviate the effect of process variations, many researchers have also worked on circuit level to introduce special devices and mechanisms. For instance, the Razor method [12]–[15] boosts circuit performance until timing errors occur, and recently the performance capacity of flip-flops has been exploited to the extreme limit by considering the interdependency between setup and hold time [16]–[21].

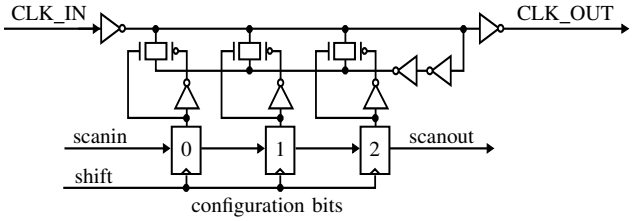


Fig. 1: Post-silicon delay tunable buffer in [25].

Another direction to deal with process variations is to tune chips after manufacturing. To apply this technique, tunable components are inserted into the circuit during the design phase. After manufacturing, chips with timing failures can be rescued by tuning buffers with respect to the effect of process variations, which become deterministic at this phase.

A widely used post-silicon tuning technique is clock tuning using delay buffers with various structures [22]–[25]. An example of industrial applications of this technique is demonstrated in [25]. The structure of the delay buffer (clock vernier device) in this work is illustrated in Fig. 1, where the three registers control the delay between the clock input CLK_IN and output CLK_OUT. During the design phase, tunable buffers of such type are inserted onto the clock paths of selected flip-flops related to potential critical paths. After manufacturing, the delay values of these buffers are adjusted through the test access port (TAP) to create different clock skews to these flip-flops. The objective of this tuning is to allot critical paths more timing slack by shifting clock edges toward stages with smaller combinational delays, so that a manufactured chip can work at the designated frequency.

To apply the post-silicon tuning technique, tunable delay buffers must be inserted into the circuit during the design phase. In recent years, several methods have been proposed to determine buffer locations and evaluate the potential resulting yield improvement. In [26] a clock scheduling method is devel-

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89) as well as NSF (CCF-1525749, CNS-1618824) and SRC (2016-TS-2688).

A preliminary version of this paper was published in the Proceedings of Design Automation Conference (DAC), 2016 [1].

Grace Li Zhang, Bing Li, and Ulf Schlichtmann are with the Chair of Electronic Design Automation, Technical University of Munich (TUM), Munich 80333, Germany (e-mail: grace-li.zhang@tum.de; b.li@tum.de; ulf.schlichtmann@tum.de).

Yiyu Shi is with the Department of Computer Science and Engineering, University of Notre Dame (e-mail: yshi4@nd.edu).

Jiang Hu is with the Department of Electrical and Computer Engineering, Texas A&M University (e-mail: jianghu@tamu.edu).

oped and tunable buffers are selectively inserted to balance the skews resulting from process variations. In [27] the buffer allocation problem is solved with a graph-based algorithm under useful clock skew scheduling. In [28] algorithms are proposed to insert buffers into the clock tree to guarantee a given yield, while minimizing the total area of these tunable buffers or the total number of them. This problem is investigated further in [29], [30] with a sampling-based method to recognize a limited number of locations to insert tunable buffers for yield improvement. In [31] yield loss due to process variations and the total cost of tunable buffers are formulated together for gate sizing. In [32], the placement of tunable buffers is investigated and a considerable improvement is observed when the clock tree is designed using the proposed tuning system. With the locations of tunable buffers known, the improved yield of the circuit can be evaluated efficiently using the method in [33], [34].

After manufacturing, necessary information, e.g., delays of combinational paths, need to be extracted from chips with timing failures for post-silicon clock skew scheduling. In [35] an efficient post-silicon tuning method is proposed to search a configuration tree together with graph pruning and buffer grouping. The methods in [36], [37] measure path delays individually in manufactured chips and tune them accordingly. Furthermore, this post-silicon tuning technique has been applied for on-line adjustment to improve lifetime performance of a circuit in view of process variations and aging [38], [39]. Moreover, the method in [40] applies tunable buffers to compensate dynamic delay uncertainty induced by temperature variations.

In applying post-silicon clock tuning, a major challenge is that delays of combinational paths need to be measured specifically for each chip after manufacturing. However, so far this measurement is still performed by applying frequency stepping to individual paths [35]–[37], which requires much time from expensive testers.

In this paper, we investigate the post-silicon test problem and propose an efficient framework (EffiTest2) to improve test efficiency using statistical prediction and delay alignment. Our contributions are as follows.

1) A path selection approach combining SVD/QRcp decomposition and iterative accuracy evaluation is proposed to choose the paths for post-silicon test. The delays of these paths are used to predict the maximum delays between flip-flops with tunable buffers.

2) Multiple paths are tested in parallel in our framework. The delays of paths in a test batch are aligned statistically during path assignment. Nonrepresentative paths with large random variations are added into test batches to reduce inaccuracy in delay estimation. During delay test, we also adjust the already-existing tunable buffers to align the real delays of paths adaptively so that a frequency step can capture delay information of multiple paths.

3) Since predicted path delays are still in the form of small ranges instead of exact numbers, configuration values of tunable buffers are determined with respect to the upper bounds of these ranges, so that potential timing violations due to the inaccuracy in delay test and prediction can be reduced.

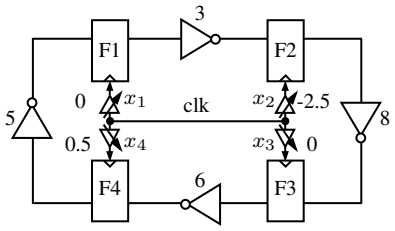


Fig. 2: Post-silicon clock tuning reduces the minimum clock period from 8 to 5.5. Setup time and propagation delay of flip-flops are assumed as 0.

4) Hold time constraints are incorporated by imposing range constraints on configuration values of tunable buffers. These additional constraints limit yield loss due to hold time constraints by a given threshold to avoid further test iterations on short paths.

The rest of this paper is organized as follows. In Section II we give an overview of timing constraints for circuits with post-silicon tunable buffers. We explain the proposed method in detail in Section III. Experimental results are shown in Section IV. Conclusions are drawn in Section V.

II. Background of Post-silicon Clock Tuning

In a circuit with post-silicon tunable buffers, the propagation delays of clock paths to flip-flops with tunable buffers can be adjusted after manufacturing for each chip individually. The concept of this technique can be explained using the example in Fig. 2, where four flip-flops are connected into a loop by combinational paths represented by inverters. The numbers next to the inverters denote delays of the corresponding combinational paths. During the design phase, these combinational delays should be considered as statistical due to process variations. But after manufacturing, the delays in a single chip become fixed values, enabling a concrete clock skew tuning to counter the effect of process variations.

In Fig. 2, if all the delays of the tunable buffers are set to 0, this example is equivalent to the case without post-silicon tuning. The minimum clock period is thus equal to 8 when setup time and propagation delays of the flip-flops are assumed as 0. On the other hand, if clock edges can be moved by adjusting the delays of the tunable buffers, the minimum clock period can be reduced to 5.5. For example, the buffer value x_2 shifts the launching clock edge at F2 2.5 units earlier. Therefore, with a clock period of 5.5, the combinational path between F2 and F3 now has 5.5+2.5=8 time units to finish signal propagation. This shifting of the clock edge reduces the timing budget of the path between F1 and F2 to 5.5-2.5=3 units after post-silicon tuning, which is still sufficient for this path without violating any timing constraint. Note that the buffer delays are defined with respect to a reference clock signal, so that they can have negative values.

This clock tuning concept is similar to assigning useful skews [41] to improve circuit performance. The difference, however, is that the skew scheduling is executed individually for each chip with timing failures after manufacturing. Since critical paths in these chips may differ due to process variations, customized timing schemes generated in response to the

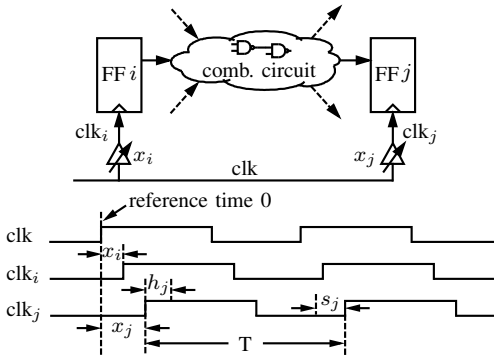


Fig. 3: Timing with tunable buffers.

effect of process variations can rescue these chips by balancing slacks across consecutive register stages.

Timing constraints with clock tunable buffers can be explained using Fig. 3, where two flip-flops with such buffers are connected by a combinational circuit. Assume that the clock signal switches at reference time 0. The clock events at flip-flops i and j happen at time x_i and x_j, respectively. To meet the setup time and hold time constraints, the following constraints must be satisfied

$$x_i + \overline{d}_{ij} \leq x_j + T - s_j \Longleftrightarrow T \geq D_{ij} + x_i - x_j \quad (1)$$
$$x_i + \underline{d}_{ij} \geq x_j + h_j \Longleftrightarrow x_i - x_j \geq d_{ij} \quad (2)$$

where x_i and x_j are delay values of tunable buffers, $\overline{d}_{ij}$ ($\underline{d}_{ij}$) is the maximum (minimum) delay of the combinational circuit between flip-flops i and j, s_j (h_j) is the setup (hold) time of flip-flop j, T is the clock period, $D_{ij} = \overline{d}_{ij} + s_j$, and $d_{ij} = h_j - \underline{d}_{ij}$. The two constraints above indicate that the clock tuning values x_i and x_j should be determined according to D_{ij} and d_{ij}. In delay test, the latter two values are actually measured instead of the path delays $\overline{d}_{ij}$ and $\underline{d}_{ij}$. In the following, we will also refer to D_{ij} and d_{ij} as maximum delay and minimum delay for simplicity.

Owing to area cost, the configurable delay of a clock buffer usually has a limited range. For buffer i, this range is specified as

$$r_i \leq x_i \leq r_i + \tau_i \quad (3)$$

where r_i and τ_i are constants determined by methods such as [28]. In the range (3), x_i may only take discrete values according to the implementation of buffers.

After manufacturing, path delays in chips become deterministic. For a chip with timing failures, the delays of combinational paths related to tunable buffers should be evaluated. Thereafter, the configuration values of tunable buffers are determined by finding a feasible solution meeting the constraints (1)–(3) with respect to the given clock period T. The most challenging task of applying this post-silicon tuning technique is delay evaluation of combinational paths after manufacturing. These delays should be estimated relatively accurately to configure buffers properly. But the cost of this delay test on all failed chips must remain low; otherwise, the benefit of using tunable buffers to improve yield may be offset by the ensuing test cost.

In previous methods [26], [35]–[37], path delays are measured straightforwardly using *frequency stepping*. In this technique, a path is tested with a given clock period. If the sink flip-flop of this path can latch data correctly, the setup time constraint at the sink flip-flop is met, so that an upper bound of the path delay is found. Thereafter, a smaller clock period is applied until data cannot be latched correctly anymore to find a lower bound of the path delay. With a binary search of different frequency steps, the path delay can be approximated by narrowing the range defined by the lower and upper bounds.

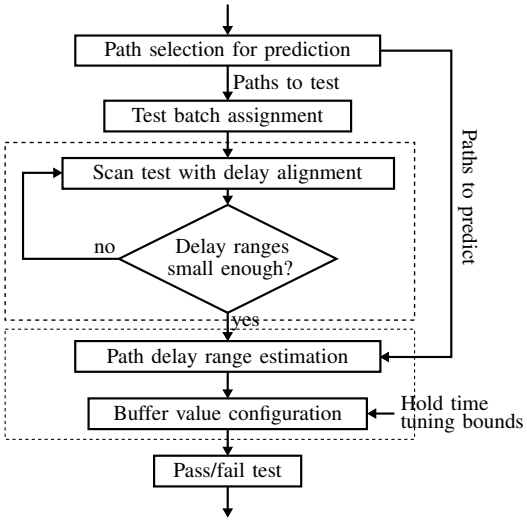


Fig. 4: Delay test and buffer configuration flow in EffiTest2.

In using frequency stepping in this test scenario, the number of iterations (frequency steps) might be large if many paths are tested. Though there are some techniques that can be used to combine tests of several paths to reduce the number of iterations, no method has considered the fact that the tunable buffers in the circuit can be used to align path delays, so that a clock period can sweep the delay ranges of several paths at the same time. For example, if delays of tunable buffers in Fig. 2 could be set to values as shown, the delays of the combinational paths are well balanced and can thus be tested concurrently. To reduce test cost further, the correlation information between path delays provided by statistical timing analysis techniques [11] can also be used. Consequently, only a set of representative paths need to be tested while the maximum delays between flip-flops with tunable buffers are estimated from the test results.

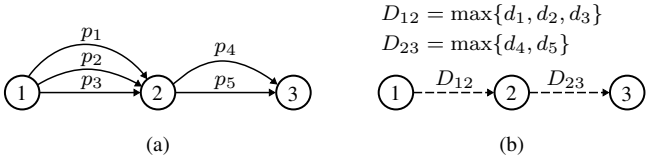
III. Statistical Prediction and Aligned Delay Test for Buffer Configuration

In post-silicon delay test, previous frequency stepping methods test all paths connected to flip-flops with post-silicon tunable buffers individually. According to the test results, the configuration bits of the tunable buffers are adjusted to make the chips work with the required clock period. This exhaustive path-wise test strategy is very expensive because it requires a lot time from testers. Practically, however, not all paths delays need to be evaluated exactly. Instead, they may only need to be estimated with a given accuracy that is sufficient for post-silicon configuration.

In this section, we introduce our method EffiTest2 to reduce the total number of frequency stepping iterations in testing path delays with two techniques: statistical prediction and delay alignment during test. With the tested and estimated delays, tunable buffers in chips with timing failures are then

TABLE I: Notations

$\mathbf{P}$	All critical combinational paths between pairs of flip-flops with at least one tunable buffer
$\mathbf{D}^p$	The statistical delays of combinational paths in $\mathbf{P}$
$\mathbf{D}^m$	The statistical maximum delays between pairs of flip-flops with at least one tunable buffer
$\mathbf{P}_t$	The combinational paths that are tested using frequency stepping
$\mathbf{D}_t^p$	The statistical delays of combinational paths in $\mathbf{P}_t$
$\mathbf{D}_t^m$	The selected maximum delays that can predict $\mathbf{D}^m \backslash \mathbf{D}_t^m$ within a given accuracy
$\mathbf{P}_c$	The chosen combinational paths for maximum delays in $\mathbf{D}_t^m$
$\mathbf{D}_c$	The statistical delays of combinational paths in $\mathbf{P}_c$



(a) (b)

Fig. 5: Test scenario with maximum path delays, where nodes represent flip-flops with tunable buffers. Multiple combinational paths, p_1–p_5 with delays d_1–d_5, respectively, exist between flip-flops with tunable buffers in (a). Post-silicon skew configuration by tunable buffers is determined by the maximum delays of all paths as simplified in (b).

configured to maximize the chance that manufactured chips work with the given clock period T. In the test scenario, we assume that the locations of buffers have been determined, using a method such as [28], [30]. The flow of the proposed method is summarized in Fig. 4. It includes four major steps: path selection in Section III-A, test batch assignment in Section III-B, frequency stepping with delay alignment in Section III-C, buffer configuration in Section III-D, and hold time constraints in Section III-E. The important notations used in the following are listed in Table I.

The statistical delay prediction in Section III-A assumes that path delays follow Gaussian distribution. In cases such as ultra-low voltage designs, this assumption may be invalid. In this scenario, the accuracy evaluation (6)–(7) needs to be adapted accordingly while the overall flow can still be deployed.

A. Path Selection and Statistical Delay Prediction

When tuning manufactured chips to resolve timing failures, the maximum delays between flip-flop pairs need to meet setup time constraints, and the minimum delays hold time constraints, as defined in (1)–(2). Fig. 5 illustrates an example, considering only setup time constraints. In this example, nodes represent flip-flops, solid edges represent combinational paths and dashed edges represent maximum path delays between flip-flops. If a path between a pair of flip-flops is critical, the clock edge to the flip-flop in the middle can be tuned to the other side to give the critical path more slack, provided that the timing constraints between the other pair of flip-flops are not violated.

To determine the configuration of tunable buffers attached to the flip-flops with respect to the setup time constraint (1), the maximum delays between flip-flops need to be evaluated. Since process variations affect combinational paths in manufactured chips differently, many paths between a pair of flip-flops may be critical after manufacturing. Due to test cost, it is impractical to test all combinational paths that can potentially become critical with frequency stepping directly, as assumed in [26], [35]–[37]. Instead, statistical delay prediction can be deployed to estimate the maximum delays between flip-flops using the data of representative combinational paths.

1) Concept of path selection for delay prediction: Statistical delay prediction relies on the correlation between path delays to maintain a high accuracy. Since a high correlation indicates that two delays vary similarly in manufactured chips, the measurement of one delay after manufacturing also discloses information about the other. In high-performance designs, logic gates on a critical path usually are not spread out all over the chip. Therefore, critical paths converging at or leaving from flip-flops with buffers tend to form physical clusters on the chip. This physical proximity results in a high correlation between path delays [11], which can be exploited to reduce the number of paths to be tested. For example, a conditional statistical prediction technique [42] has been used in [43] to predict the timing performance of a circuit from the measurements of on-chip test structures.

Consider a pair of flip-flops to at least one of which a tunable buffer is attached. Under process variations, usually many combinational paths between this pair of flip-flops have a probability to dominate the rest of them. We denote all these paths in the circuit as a set $\mathbf{P}$ and their statistical delays as $\mathbf{D}^p$. The task of delay measurement is to extract sufficient information about delays by testing only a small subset of paths $\mathbf{P}_t \subset \mathbf{P}$. Assume the statistical delays of $\mathbf{P}_t$ are denoted as $\mathbf{D}_t^p$. The statistical prediction from $\mathbf{D}_t^p$ to $\mathbf{D}^p \backslash \mathbf{D}_t^p$ is a well-known problem and has been studied extensively, e.g., in [44].

In post-silicon tuning, the individual delays of paths in $\mathbf{D}^p$, however, are not required. Instead, only the maximum delays between each pair of flip-flops should be determined as illustrated in Fig. 5(b). When process variations are considered, path delays are represented as random variables. The maximum of a set of path delays can be calculated using Monte Carlo simulation or statistical timing analysis. Assume the statistical maximum delays are collected in a set $\mathbf{D}^m$, which contains one statistical maximum delay for every pair of flip-flops to at least one of which a tunable buffer is attached. We then need to establish the relation between the delays $\mathbf{D}_t^p$ of selected combinational paths $\mathbf{P}_t \subset \mathbf{P}$ to $\mathbf{D}^m$, i.e., $\mathbf{D}_t^p \rightarrow \mathbf{D}^m$.

To identify $\mathbf{P}_t$ from $\mathbf{P}$, we need to consider all the combinational paths between each pair of flip-flops with at least one tunable buffer. This leads to a huge amount of paths to be examined. To solve this problem, we introduce a second level of statistical prediction. Instead of predicting the maximum delays $\mathbf{D}^m$, we use the measured delays $\mathbf{D}_t^p$ to predict a subset $\mathbf{D}_t^m \subset \mathbf{D}^m$, provided that the predicted values of $\mathbf{D}_t^m$ can also provide sufficient information for the other maximum delays $\mathbf{D}^m \backslash \mathbf{D}_t^m$, thus establishing a chained relation $\mathbf{D}_t^p \rightarrow \mathbf{D}_t^m \rightarrow \mathbf{D}^m$. Since $\mathbf{D}_t^m$ is a subset of $\mathbf{D}^m$, to identify it from $\mathbf{D}^m$ is the same statistical prediction problem as discussed in [44]. Therefore, we only need to focus on the task to find a set of combinational paths to predict

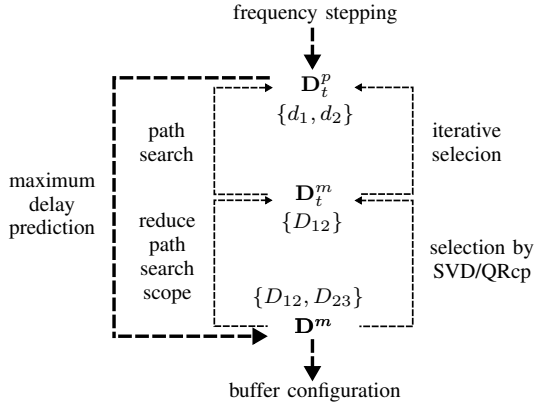


Fig. 6: Relation between delay sets, using the test scenario in Fig. 5 as example. The thin dashed lines represent the relation between the delay sets for identifying representative combinational paths. The thick dashed lines illustrate the real test and prediction procedure.

$\mathbf{D}_t^m$. The information of $\mathbf{D}_t^m$ is derived from the delays of the combinational paths from which $\mathbf{D}_t^m$ is computed. This characteristic allows us to search only the combinational paths between those pairs of flip-flops corresponding to $\mathbf{D}_t^m$, thus reducing the effort of path enumeration significantly.

The relation between the delay sets discussed above is illustrated in Fig. 6. We identify the representative maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$ to reduce the path search scope. Thereafter, the combinational paths between the pairs of flip-flops whose maximum delays are $\mathbf{D}_t^m$ are examined to select the combinational paths $\mathbf{D}_t^p$ for delay test. The details of these steps are described as follows.

2) Determining a set of maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$ using SVD-QRcp: In the delay prediction concept shown in Fig. 6, the first step is to determine a subset of variables $\mathbf{D}_t^m$ from $\mathbf{D}^m$, so that the values of $\mathbf{D}_t^m$ can predict the values of $\mathbf{D}^m$. This problem has been studied previously such as in [44]–[47] using an algorithm based on SVD-QRcp. Assume that a delay from $\mathbf{D}^m$ is written as a linear combination of M random components $\mathbf{S} = [s_1, s_2, \ldots, s_M]^T$, such as in the canonical form in [3]. The delay $\mathbf{D}^m$ can then be expressed as $\mathbf{D}^m = \mathbf{CS}$, where $\mathbf{C}$ is the coefficient matrix. The SVD-QRcp algorithm first performs Singular Value Decomposition (SVD) to decompose $\mathbf{C}$ as

$$\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \tag{4}$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{\Lambda}$ is a diagonal matrix with singular values in a descending order.

The large singular values in $\mathbf{\Lambda}$ reveal the importance of delays that carry orthogonal statistical information. To select the delays $\mathbf{D}_t^m$ to predict $\mathbf{D}^m \backslash \mathbf{D}_t^m$ the correspondence between the singular values and the delays in $\mathbf{D}^m$ needs to be established using the permutation matrix in the QRcp (QR with column pivoting) decomposition. Assume n delays should be selected from $\mathbf{D}^m$. Then the first n columns of $\mathbf{U}$, written as $\mathbf{U}_{[1:n]}$ are decomposed as

$$\mathbf{U}_{[1:n]}^T = \mathbf{QR}\mathbf{\Pi}^T \tag{5}$$

where $\mathbf{\Pi}^T$ is a permutation matrix to identify the n most important random variables from $\mathbf{D}^m$.

To illustrate the decomposition process above, we use an example with three delays in $\mathbf{D}^m$, each of which is expressed as a linear combination of three random components. The SVD and QRcp are performed using the routines from the LAPACK [48] and GSL [49] libraries. The coefficient matrix $\mathbf{C}$ of $\mathbf{D}^m$ and the matrices after decomposition are shown in the following.

$$\underset{\mathbf{C}}{\begin{bmatrix}10 & 6 & 1\\ 13 & 4 & 2\\ 7 & 5 & 1\end{bmatrix}} = \underset{\mathbf{U}}{\begin{bmatrix}-0.59 & 0.43 & -0.68\\ -0.69 & -0.72 & 0.13\\ -0.43 & 0.55 & 0.72\end{bmatrix}} \times \underset{\mathbf{\Lambda}}{\begin{bmatrix}19.83 & 0 & 0\\ 0 & 2.76 & 0\\ 0 & 0 & 0.31\end{bmatrix}}$$
$$\times \underset{\mathbf{V}^T}{\begin{bmatrix}-0.90 & -0.40 & -0.18\\ -0.42 & 0.90 & 0.10\\ -0.12 & -0.16 & 0.98\end{bmatrix}}$$

$$\mathbf{U}_{[1:2]}^T = \underset{\mathbf{Q}}{\begin{bmatrix}-0.69 & -0.72\\ -0.72 & 0.69\end{bmatrix}} \times \underset{\mathbf{R}}{\begin{bmatrix}0.99 & 0.09 & -0.10\\ 0 & 0.72 & 0.69\end{bmatrix}} \times \underset{\mathbf{\Pi}^T}{\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}}$$

In the example above, two delays are selected to predict the third one, so that only the first two columns of $\mathbf{U}$, written as $\mathbf{U}_{[1:2]}$, are used in the QRcp decomposition. In the permutation matrix $\mathbf{\Pi}^T$, the first column shows that we need to select the second delay because the only 1 in this column appears in the second row. Similarly, the second column of $\mathbf{\Pi}^T$ shows that we need to select the first delay.

The decomposition process above requires that we state the number of delays n to be included in $\mathbf{D}_t^m$. This number needs to be decided so that the prediction accuracy is maintained. Assume in a general case that N statistical variables $\mathbf{D}_t$ that are selected to measure their values $\mathbf{d}_t$ in a chip directly, and another variable d_k whose value should be predicted with $\mathbf{d}_t$. Assume also that these delays follow Gaussian distributions, which are widely used in statistical timing analysis [11]. Under this assumption, these delays can be written together as $\mathbf{D} = \begin{bmatrix} d_k \\ \mathbf{D}_t \end{bmatrix} \sim \boldsymbol{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\boldsymbol{\mu}$ is the mean value vector of $\mathbf{D}$, $\mathbf{\Sigma}$ is the covariance matrix of $\mathbf{D}$, $d_k \sim N(\mu_k, \sigma_k)$ and $\mathbf{D}_t \sim \boldsymbol{N}(\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)$. Accordingly, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ can be expressed as $\boldsymbol{\mu} = \begin{bmatrix} \mu_k \\ \boldsymbol{\mu}_t \end{bmatrix}$, and $\mathbf{\Sigma} = \begin{bmatrix} \sigma_k & \mathbf{\Sigma}_{k,t} \\ \mathbf{\Sigma}_{t,k} & \mathbf{\Sigma}_t \end{bmatrix}$, where $\mathbf{\Sigma}_{k,t} = \mathbf{\Sigma}_{t,k}^T$ is the covariance matrix between d_k and $\mathbf{D}_t$.

With the measured values $\mathbf{d}_t$ of $\mathbf{D}_t$, the mean value μ_k' and the variance $\sigma_k'^2$ of d_k under the condition $\mathbf{D}_t = \mathbf{d}_t$ can be expressed as follows [42].

$$\mu_k' = \mu_k + \mathbf{\Sigma}_{k,t}\mathbf{\Sigma}_t^{-1}(\boldsymbol{d}_t - \boldsymbol{\mu}_t) \tag{6}$$
$$\sigma_k'^2 = \sigma_k^2 - \mathbf{\Sigma}_{k,t}\mathbf{\Sigma}_t^{-1}\mathbf{\Sigma}_{t,k}. \tag{7}$$

After delay prediction, d_k is still a random variable because there are purely random process variations that reduce the correlation between delays. However, the variance of the predicted delays becomes smaller due to the second product term in (7), indicating that the real path delay d_k in a chip is confined into a small range with a nonnegligible probability. This range reduction results from the fact that the measurement results of $\mathbf{d}_t$ provide the information of the shared random components between d_k and $\mathbf{D}_t$ to reduce the variability of

Algorithm 1: Select $\mathbf{D}_t^m$ from $\mathbf{D}^m$ to reduce path search scope

Input : Coefficient matrix $\mathbf{C}$ of maximum delays $\mathbf{D}^m$ from SSTA
Output: Representative maximum delays $\mathbf{D}_t^m \subseteq \mathbf{D}^m$

L1 $\mathbf{U} \leftarrow$ Decompose $\mathbf{C}$ using SVD (4);
L2 **for** $i \leftarrow 1$ **to** $|\mathbf{D}^m|$ **do**
L3 $\quad \mathbf{U}_{[1:i]} \leftarrow$ First i columns of $\mathbf{U}$;
L4 $\quad \mathbf{\Pi}^T \leftarrow$ Decompose $\mathbf{U}_{[1:i]}^T$ using QRcp (5);
L5 $\quad$ Select $\mathbf{D}_t^m$ from $\mathbf{D}^m$ using $\mathbf{\Pi}^T$;
L6 $\quad$ **foreach** $d_k \in \mathbf{D}^m \backslash \mathbf{D}_t^m$ **do**
L7 $\quad\quad$ Compute $\sigma_k'^2$ using (7);
L8 $\quad\quad$ **if** $\sigma_k' > \sigma_{th}$ **then**
L9 $\quad\quad\quad$ goto L2;
L10 $\quad\quad$ **end**
L11 $\quad$ **end**
L12 $\quad$ break;
L13 **end**
L15 **return** $\mathbf{D}_t^m$

d_k. Therefore, it may be unnecessary to measure the exact delay of d_k for buffer configuration after delay prediction if the correlation between d_k and $\mathbf{D}_t$ is high. On the other hand, a small correlation allows the delay d_k to vary freely, leading to a relatively large variance even after statistical prediction. Since the standard deviation σ' represents how wide the distribution of the predicted value of d_k spreads, we use it as an indicator of the prediction accuracy. If σ' is lower than a given threshold σ_{th}, the predicted value is considered as having a sufficient accuracy.

In identifying $\mathbf{D}_t^m$ from $\mathbf{D}^m$, if the standard deviation σ' of a delay from $\mathbf{D}^m \backslash \mathbf{D}_t^m$ exceeds σ_{th}, we increase the number of delays from $\mathbf{D}^m$ to be selected and rerun the QRcp decomposition. Since all delays in $\mathbf{D}^m$ contain a purely random component from process variations [3], [11], σ' cannot be reduced to zero. Instead, it must be larger than the standard deviation of the corresponding purely random component. In EffiTest2, we enumerate all the delays in $\mathbf{D}^m$ to identify the maximum σ_{max} of the standard deviations of all the purely random components and use $\sigma_{th} = 2\sigma_{max}$ as the threshold of the prediction accuracy. Therefore, the iterations should always converge because the given threshold σ_{th} is larger than σ_{max}, which is the accuracy when all delays are measured directly. The selection process of $\mathbf{D}_t^m$ from $\mathbf{D}^m$ is summarized in Algorithm 1.

3) Identifying representative paths using iterative selection: The maximum delays $\mathbf{D}_t^m$ returned by Algorithm 1 are actually used to narrow the search scope of combinational paths for delay test. In manufactured chips, only the delays of these combinational paths can be measured with frequency stepping directly [50], [51]. After $\mathbf{D}_t^m$ is identified from $\mathbf{D}^m$, we scan the circuit to find the starting and ending flip-flops corresponding to $\mathbf{D}_t^m$. For example, in Fig. 6 the maximum delay D_{12} is identified from the set $\{D_{12}, D_{23}\}$ using the SVD-QRcp method described above. This maximum delay indicates that the combinational paths between the flip-flops 1 and 2 in Fig. 5 are candidates for delay test. To reduce test cost, only the minimum number of paths from them should be tested using frequency stepping for post-silicon configuration. For example, the delays of paths p_1 and p_2 may already provide sufficient accuracy in predicting the maximum delays

Algorithm 2: Path selection to predict maximum delays $\mathbf{D}^m$

Input : Maximum delays $\mathbf{D}^m$ from SSTA
Delay set $\mathbf{D}_t^m$ from Algorithm 1
Output: Selected paths $\mathbf{P}_t$ for delay test

L1 $\mathbf{P}_c \leftarrow \emptyset$;
L2 **foreach** $d_k \in \mathbf{D}_t^m$ **do**
L3 $\quad \{\text{ff}_{src}, \text{ff}_{dst}\} \leftarrow$ Find flip-flops corresponding to d_k;
L4 $\quad \mathbf{P}_s \leftarrow$ Trace five most critical paths $\text{ff}_{src} \rightarrow \text{ff}_{dst}$;
L5 $\quad \mathbf{P}_c \leftarrow \mathbf{P}_c \cup \mathbf{P}_s$;
L6 **end**
L7 $\mathbf{D}_c \leftarrow$ Delays of $\mathbf{P}_c$;
L8 $\mathbf{D}_t^p \leftarrow \emptyset$;
L9 **for** $i \leftarrow 1$ **to** $|\mathbf{D}_c|$ **do**
L10 $\quad \sigma_{next}^2 \leftarrow \infty$;
L11 $\quad d_{next} \leftarrow null$;
L12 $\quad$ **foreach** $d_c \in \mathbf{D}_c \backslash \mathbf{D}_t^p$ **do**
L13 $\quad\quad \mathbf{D}_t^{p\prime} \leftarrow \{d_c\} \cup \mathbf{D}_t^p$;
L14 $\quad\quad \sigma_{max}^2 \leftarrow 0$;
L15 $\quad\quad$ **foreach** $d_k \in \mathbf{D}^m$ **do**
L16 $\quad\quad\quad$ Compute $\sigma_k'^2$ from $\mathbf{D}_t^{p\prime}$ to $\mathbf{D}^m$ using (7);
L17 $\quad\quad\quad$ **if** $\sigma_k'^2 > \sigma_{max}^2$ **then**
L18 $\quad\quad\quad\quad \sigma_{max}^2 \leftarrow \sigma_k'^2$;
L19 $\quad\quad\quad$ **end**
L20 $\quad\quad$ **end**
L21 $\quad\quad$ **if** $\sigma_{max}^2 < \sigma_{next}^2$ **then**
L22 $\quad\quad\quad \sigma_{next}^2 \leftarrow \sigma_{max}^2$;
L23 $\quad\quad\quad d_{next} \leftarrow d_c$;
L24 $\quad\quad$ **end**
L25 $\quad$ **end**
L26 $\quad \mathbf{D}_t^p \leftarrow \{d_{next}\} \cup \mathbf{D}_t^p$;
L27 $\quad$ **if** $\sigma_{next} \leq \sigma_{th}$ **then**
L28 $\quad\quad \mathbf{P}_t \leftarrow$ Paths corresponding to $\mathbf{D}_t^p$;
L29 $\quad\quad$ break;
L30 $\quad$ **end**
L31 **end**
L33 **return** $\mathbf{P}_t$

$\{D_{12}, D_{23}\}$, while more paths in addition to them may not improve the prediction accuracy further, because the purely random components in the maximum delays then dominate the predicted values.

Because the number of combinational paths between a pair of flip-flops is usually very large, the path candidates related to $\mathbf{D}_t^m$ need to be reduced further. Since the combinational paths related to the same pair of flip-flops are generally located close to each other on the die, their delays exhibit a high correlation due to proximity. Therefore, we need to consider only a small subset of paths between each pair of flip-flops. A static critical path identification is used in our method to extract five combinational paths for each maximum delay in $\mathbf{D}_t^m$ by forward and backward arrival time propagation. The extracted combinational paths are denoted as a set $\mathbf{P}_c$. The delays of these paths are denoted as a set $\mathbf{D}_c$.

The final step for path selection is to choose $\mathbf{P}_t$ from $\mathbf{P}_c$. The objective is that the measured values of the selected paths $\mathbf{P}_t$ should be able to predict the delays of $\mathbf{D}^m$ with a sufficient accuracy. A new challenge in this step is that the set of delays $\mathbf{D}_c$ is not a subset of $\mathbf{D}^m$, so that the SVD-QRcp method cannot be used to identify the paths $\mathbf{P}_t$. To solve this problem, we enumerate all the path delays in $\mathbf{D}_c$ and select the delay that can reduce the maximum of the variances of the predicted values of $\mathbf{D}^m$ the most. The selection step stops when the maximum of the standard deviations σ_k' is smaller than the threshold σ_{th} as used in Algorithm 1.

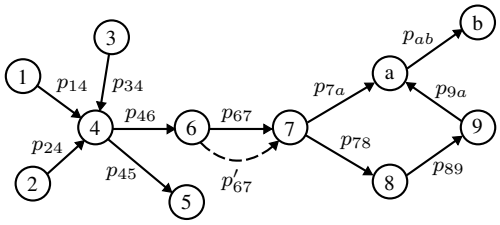


Fig. 7: Test scenario with multiple combinational paths. The nodes represent flip-flops. The solid edges represent combinational paths whose delays need to be tested using frequency stepping. The dashed edges represent an additional path that can also be tested without increasing the number of test batches.

The procedure of selecting combinational paths for delay test is summarized in Algorithm 2. In L1–L7 the combinational paths related to $\mathbf{D}_t^m$ are saved in $\mathbf{P}_c$ and their statistical delays in $\mathbf{D}_c$. To select representative paths from $\mathbf{P}_c$, the loop L9–L31 adds one delay d_{next} from $\mathbf{D}_c$ into $\mathbf{D}_t^p$ in each iteration. The newly selected delay d_{next} is the one from $\mathbf{D}_c$ that, together with the already selected delays in $\mathbf{D}_t^p$, predicts the maximum delays $\mathbf{D}^m$ with the best accuracy. To identify this delay, each delay in $\mathbf{D}_c \backslash \mathbf{D}_t^p$ is evaluated in L12–L25 as d_c, where d_c and the current $\mathbf{D}_t^p$ are combined as $\mathbf{D}_t^{p\prime}$ to evaluate the accuracy of predicting the maximum delays $\mathbf{D}^m$ in the loop L15–L20 using (7). The prediction accuracy is indicated as the maximum variance σ_{max}^2 of predicted values of $\mathbf{D}^m$, and the delay d_c that can produce the smallest σ_{max}^2 is selected and added into $\mathbf{D}_t^p$. Meanwhile, the accuracy indicator σ_{max}^2 is assigned to σ_{next}^2. When σ_{next} becomes lower than the threshold σ_{th}, the selection procedure finishes and the current $\mathbf{P}_t$ is returned as the paths to be tested using frequency stepping.

B. Path Test Multiplexing

To predict the maximum delays $\mathbf{D}^m$ between flip-flops, the delays $\mathbf{D}_t^p$ of representative combinational paths $\mathbf{P}_t$ need to be tested. In frequency stepping, the delay of a path is compared with the period of the test clock signal by checking whether the sink flip-flop of the path latches data correctly. A violation of the setup time constraint indicates the maximum delay exceeds the test clock period. Since the data latching state of a flip-flop can only indicate whether there is a timing violation, only the delay of one path converging to it can be tested in one clock cycle. In addition, paths leaving from a flip-flop cannot be tested in parallel, because the values of the flip-flops need to be set to trigger specific paths. In practice, the constraints may be relaxed because some paths can share parts of test patterns. In the following discussion, we will only assume the strictest case without allowing this sharing of test vectors to simplify the description of the proposed test multiplexing, which can be adapted easily to deal with the relaxed test scenarios.

Consider the test scenario shown in Fig. 7, where the nodes represent flip-flops and the edges represent combinational paths. During delay test, the paths p_{14}, p_{24}, and p_{34} cannot be processed in parallel, because they converge at the same flip-flop. Similarly paths p_{45} and p_{46} cannot be tested at the same time due to the shared source flip-flop. On the contrary, paths that can be tested in parallel can be arranged into the same group. For example, paths p_{14}, p_{46}, p_{67}, p_{7a}, and p_{ab} can be tested with the same clock period together. These paths are called a *batch* in the following discussion. In real test scenarios, there might be cases that some paths in a test batch cannot be activated by ATPG vectors at the same time. These paths can be set as mutually exclusive and arranged into different test batches. The proposed method does not consider the logic inconsistencies that might arise while activating/propagating faults. However, we can use existing methods, e.g., [52] and [53], to obtain testing compatibility, i.e., subsets of paths which can be tested simultaneously for a given set of paths that need to be tested. Accordingly, testing compatibility of the representative combinational paths $\mathbf{P}_t$ after path selection in Algorithm 2 can be derived with these methods. The compatibility of $\mathbf{P}_t$ can be incorporated into path test multiplexing to generate test batches in which paths can be tested simultaneously.

Since the delays of paths in a test batch can be measured in parallel, naturally we should arrange paths to be tested into as few batches as possible to reduce the overall number of frequency stepping iterations. To identify the minimum number of test batches, we formulate this path arrangement task into an Integer Linear Programming (ILP) problem.

Assume there are N_t ($|P_t| = N_t$) paths $p_1, p_2, \ldots, p_{N_t}$ to be tested. For the path p_j, we assign a 0-1 variable $b_{i,j}, i = 1, 2, \ldots, N_t$ to indicate whether p_j is assigned into the ith batch. For all the selected paths, the variables can be written into an $N_t \times N_t$ submatrix, as shown in the first N_t columns of the following matrix,

$$\left(\begin{array}{cccc|ccc} b_{1,1} & b_{1,2} & \ldots & b_{1,N_t} & b_{1,N_t+1} & \ldots & b_{1,N_t+N_a} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,N_t} & b_{2,N_t+1} & \ldots & b_{2,N_t+N_a} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{N_t,1} & b_{N_t,2} & \ldots & b_{N_t,N_t} & b_{N_t,N_t+1} & \ldots & b_{N_t,N_t+N_a} \end{array}\right) \quad (8)$$

where the columns correspond to the paths to be tested, and the rows correspond to test batches.

Because a path delay needs to be measured only once, the sum of the variables in a column in (8) should be equal to one, written as

$$\sum_{i=1}^{N_t} b_{i,j} = 1, \quad 1 \le j \le N_t. \quad (9)$$

To prevent paths from converging at or leaving from the same flip-flop to be arranged in the same batch, we add the following constraints for each flip-flop,

$$\sum_{p_j \in I_F} b_{i,j} \le 1, \quad \sum_{p_j \in O_F} b_{i,j} \le 1, \quad 1 \le i \le N_t \quad (10)$$

where I_F is the set of paths converging at the flip-flop and O_F the set of paths leaving the flip-flop.

To reduce the number of batches, the number of rows containing at least one 1 value in (8) should be minimized. For the ith row corresponding to the ith test batch, we assign a 0-1 variable B_i to indicate whether this test batch is occupied, so that

$$b_{i,j} \le B_i, \quad 1 \le j \le N_t, \ 1 \le i \le N_t. \quad (11)$$

By minimizing $\sum_{i=1}^{N_t} B_i$, we can minimize the number of test

batches that are really occupied by test paths.

In test iterations, the delays of the paths in the same test batch are always swept by the same test clock. If the delays of these paths differ significantly, the test clock can only capture the delay information of a part of them, while the other paths are swept by changing the period of the clock signal in other test iterations. Consequently, the number of iterations may have to be increased. To improve test efficiency, we arrange the paths with comparable delays into the same test batch according to their statistical delay information.

Since comparable delays in a test batch mean that large delays tend to be assigned in the same batch and small ones in other ones, we simplify the delay balancing problem in path arrangement by pushing paths with large delays into into the same test batch as much as possible. For each test batch, we assign a variable $W_i, i = 1, 2, \ldots, N_t$ to represent the sum of the delays of the paths in the ith batch. Therefore, W_i can be defined as

$$W_i = \sum_{j=1}^{N_t} b_{i,j}\mu_j, \quad 1 \le i \le N_t \tag{12}$$

where μ_j is the mean value of the jth path. If the jth path is assigned into the ith batch, its delay contributes to W_i. Afterwards, we maximize the weighted sum of $\sum_{i=1}^{N_t} \varepsilon_i W_i$, where ε_i are constants and $\varepsilon_i > \varepsilon_{i+1}$. With the weights ε_i in the descending order, the paths with large delays tend to be assigned to the first test batches to improve the efficiency of frequency stepping.

After test batches are formed, there might still be some unoccupied slots in a test batch because paths might not be distributed evenly at flip-flops with buffers. For example, the test scenario in Fig. 7 requires at least three test batches because there are three edges converging at node 4. Therefore, these test batches can cover not only the edge p_{67} but also p'_{67}. Because the batches of paths should be tested anyway, we add additional paths to these empty test slots to gather more delay information.

Additional paths are added according to the prediction accuracy of their corresponding maximum delays. As discussed in Section III-A2, the predicted standard deviation is used as an indicator of the prediction accuracy. Since a large standard deviation σ'_k calculated by (7) represents that the corresponding maximum delay cannot be estimated with enough accuracy, we first identify those maximum delays whose predicted standard deviations are larger than a given percentage of their original standard deviations, 10% in our framework. Thereafter, for each of these delays, we find a combinational path from the corresponding source flip-flop to the sink flip-flop to reduce the predicted variance of the delay. These newly identified combinational paths are written as a set $\mathbf{P}_a$ with delays $\mathbf{D}_a$ and $|\mathbf{D}_a| = N_a$. To incorporate these paths into the test batches, we assign 0-1 variables $b_{i,j}, i = 1, 2, \ldots, N_t, j = N_t + 1, N_t + 2, \ldots, N_t + N_a$ as shown in the extended submatrix (8). Since it is preferred, but not mandatorily required, to add these new paths into the test batches, their appearance in the test batches can be constrained as

$$\sum_{i=1}^{N_t} b_{i,j} \le 1, \quad N_t \le j \le N_t + N_a. \tag{13}$$

Consequently, a new path is included into one of the test batches when the sum above is equal to 1. To incorporate the new paths into test batches as many as possible, we maximize the objective $\sum_{1\le i\le N_t, N_t+1\le j\le N_t+N_a} b_{i,j}$. With the new columns in (8), (11) and (12) should be revised to incorporate the extended indexes as

$$b_{i,j} \le B_i, \quad 1 \le i \le N_t,\ 1 \le j \le N_t + N_a \tag{14}$$

$$W_i = \sum_{j=1}^{N_t+N_a} b_{i,j}\mu_j, \quad 1 \le i \le N_t. \tag{15}$$

Considering the three objectives discussed above, we formulate the path assignment task into an ILP problem as

$$\text{Minimize} \quad \alpha \sum_{i=1}^{N_t} B_i - \beta \sum_{i=1}^{N_t} \varepsilon_i W_i - \gamma \sum_{\substack{1\le i\le N_t \\ N_t+1\le j\le N_t+N_a}} b_{i,j} \tag{16}$$

$$\text{Subject to} \quad (9)\text{–}(10) \text{ and } (13)\text{–}(15) \tag{17}$$

where α, β and γ are constants with $\alpha \gg \beta \gg \gamma$ to guarantee the minimum number of test batches are generated. After solving this problem, only the rows with at least a one in (8) are kept as test batches, denoted as $\mathbf{B}$.

C. Test with Delay Alignment by Tuning Buffers

After path batches are identified, they should be tested using frequency stepping to determine the path delays. In this section, we discuss how the delays of paths in a single batch are measured. Note this is the only step in the proposed framework that is executed by expensive testers able to generate various clock signals with a high accuracy.

In frequency stepping, a clock period is applied to the chip under test and the paths in a test batch are sensitized by test vectors. If the setup time constraint (1) at a flip-flop is violated, the data at this flip-flop cannot be latched correctly. This error shows that $D_{ij} + x_i - x_j$ is larger than T so that T is its lower bound. On the other hand, if the clock period is large enough so that there is no timing violation, the constraint (1) is met and T is an upper bound of $D_{ij} + x_i - x_j$. By applying different clock periods in a binary search style, the value of D_{ij} can be approximated with a given accuracy.

Consider the case shown in Fig. 8(a), where a delay has given upper and lower bounds. These bounds are initialized with $\mu \pm 3\sigma$, where μ and σ are the mean value and the standard deviation of the delay calculated by statistical timing analysis. When the delay is tested with a given clock period T in an iteration, either a new upper bound or a new lower bound of it is generated. Consequently, the corresponding delay range is partitioned into two parts by T and the real delay value falls into one of them. To partition the delay range efficiently, it is preferable that T is aligned to the center of the range. Otherwise, T might not partition the delay range evenly, but instead slices it in small steps, leading to many test iterations to estimate the delay, as illustrated in Fig. 8(b).

When several path delays in one test batch are considered as in Fig. 8(c), it is not always possible to partition all the delay ranges evenly with one clock period. However, we can still find a clock period T that partitions several delay ranges

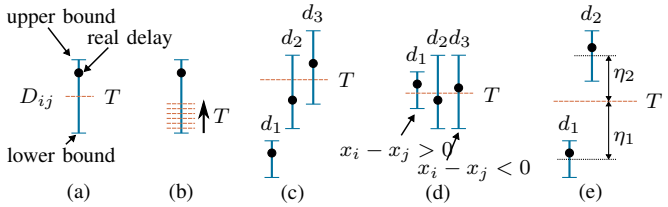


Fig. 8: Frequency stepping and delay range alignment.

at the same time, so that the ranges of these delays can be reduced in one test iteration.

To use a clock period T to partition multiple delay ranges, there must be some overlap between the delay ranges, such as d_2 and d_3 in Fig. 8(c). According to (1), the actual constraint that is tested using T is $D_{ij}+x_i-x_j$. Since the tunable buffers are already deployed in the circuit and their values x_i and x_j can be adjusted through the scan chain, we change the value of x_i-x_j to align the delay ranges, as illustrated in Fig. 8(d). Consequently, a clock period can partition more delay ranges so that the delays can be measured more efficiently compared with the case in Fig. 8(c). In EffiTest2, at-speed scan test is deployed for delay tests. At-speed scan test has been applied in [36], [37] and investigated thoroughly in [54]. In this method, scan chains are loaded with test vectors and two clock pulses are applied at the functional frequency. Because the configuration bits of buffers can be scanned into the chip under test together with the test vectors, the proposed technique requires no change to the existing test platform.

In real circuits, the buffer values x_i and x_j can only be adjusted in a limited range as specified by (3). In addition, these buffer values may affect more than one path delay. For example, in Fig. 7 the buffer value of node 4 affects all the paths converging at or leaving from it. To test the path delays efficiently, we need to find a proper set of buffer values to align the ranges of path delays as much as possible.

Assume that the upper and lower bounds of D_{ij} between nodes i and j are u_{ij} and l_{ij}, respectively. When the buffers at the source and sink nodes of the path are considered, the lower bounds and the upper bounds are shifted by $x_i - x_j$ as defined in (1). Therefore, the distance η_{ij} between a given T and the center of the shifted range of the path delay D_{ij} can be expressed as

$$\eta_{ij} = |T - ((u_{ij} + l_{ij})/2 + x_i - x_j)|. \tag{18}$$

If we minimize the sum of η_{ij} from all delay ranges, the resulting T will approximate the centers of delay ranges as much as possible, while the buffer values x_i and x_j are also determined.

Minimizing the sum of η_{ij} directly, however, cannot handle the special case in Fig. 8(e) where the two delay ranges still do not overlap even after the buffer values have been adjusted to the limit. In this case, the sum of distances $\eta_1 + \eta_2$ is independent of where T is placed between the centers of the two ranges. To solve this problem, we sort the centers of delay ranges determined in the previous test iteration. Thereafter, we assign the weight k_0 to the range whose center is in the middle of the sorted list, and reduce the weights of other ranges by k_d successively. In the proposed method, we set $k_0 \gg k_d$, so that the ranges at the middle of the sorted list have slightly higher priorities. With this weight assignment, the weights of the two ranges in Fig. 8(e) are different so that the next test clock period T should align at the center of the range with the larger weight.

Algorithm 3: Test procedure with frequency stepping

Input: $\mathbf{B}$: the queue of test batches

```
L1  foreach B_k ∈ B do
L2      while B_k contains an edge do
L3          T ← solve (19)–(26);
L4          test_with_frequency_stepping(B_k, T);
L5          foreach p_ij in B_k do
L6              if passed(p_ij) then
L7                  u_ij=T − x_i + x_j;
L8              else
L9                  l_ij=T − x_i + x_j;
L10             end
L11             if u_ij − l_ij < ϵ then
L12                 remove_edge(p_ij, B_k);
L13             end
L14         end
L15     end
L16 end
```

The optimization problem to determine the clock period T and the corresponding set of buffer values x_i and x_j to align delay ranges can thus be expressed as

$$\text{Minimize} \quad \sum_{i,j} k_{ij}\eta_{ij} \tag{19}$$

$$\text{Subject to} \quad \forall \text{ path } p_{ij} \text{ in the test batch}$$

$$T - ((u_{ij} + l_{ij})/2 + x_i - x_j) \le \mathcal{M} z_{ij}^p \tag{20}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \le \mathcal{M}(1 - z_{ij}^p) \tag{21}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) + \eta_{ij} \le \mathcal{M}(1 - z_{ij}^p) \tag{22}$$

$$-(T-((u_{ij} + l_{ij})/2 + x_i - x_j)) \le \mathcal{M} z_{ij}^n \tag{23}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \le \mathcal{M}(1 - z_{ij}^n) \tag{24}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j) + \eta_{ij} \le \mathcal{M}(1 - z_{ij}^n) \tag{25}$$

$$r_i \le x_i \le r_i + \tau_i,\ r_j \le x_j \le r_j + \tau_j \tag{26}$$

where (20)–(25) are linear constraints transformed from (18) and $\mathcal{M}$ is a very large positive constant [55]; z_{ij}^p and z_{ij}^n are two 0-1 variables corresponding to the two cases that $T - ((u_{ij} + l_{ij})/2 + x_i - x_j)$ are no less than zero and no greater than zero, respectively. (26) defines the ranges of buffer values as in (3).

After the clock frequency and the corresponding buffer values are determined by solving the ILP problem (19)–(26), the paths in the current batch are tested. According to the test result, either the upper bounds or the lower bounds of their delays are updated. If the distance between the range bounds u_{ij} and l_{ij} of a path is smaller than a threshold ϵ, which is set to a constant times of the maximum of the mean values of the path delays, 0.005 in our framework, the path is removed from the current batch. The test iterations finish when all paths in the batch have been removed. The pseudocode of the test process is shown in Algorithm 3. The testing process of one test batch only requires the calculation of buffer configuration and one clock frequency. The test patterns are determined once and no adaptive test generation based on the measurements from the tester is needed.

D. Buffer Configuration with Delay Estimation

To rescue chips with timing failures after manufacturing, buffers can be configured according to results of the delay test and prediction. Unlike delay alignment using existing tuning buffers to reduce the number of test iterations above, this step really configures tuning buffers so that the corresponding chips can operate at the designated clock frequency. After a path in P_t has been tested by frequency stepping, its delay is confined to a range with a lower bound and an upper bound. For another delay d_k that is not measured directly but is to be estimated, (6) and (7) are used to calculate the mean value μ'_k and the standard deviation σ'_k. According to (6) and (7), σ'_k is determined exclusively by the covariance matrix, but μ'_k is affected by $\boldsymbol{d}_t$, which are the delays measured by frequency stepping. When calculating μ'_k, we use the upper bounds of $\boldsymbol{d}_t$ so that the estimated delays are conservative. Since the variances of estimated delays are often non-zero, which indicate that purely random variations still affect path delays, we assign a lower bound and an upper bound $\mu'_k - 3\sigma'_k$ and $\mu'_k + 3\sigma'_k$ for an estimated delay, so that all path delays are constrained similarly for the following buffer configuration.

A real delay may take any value in the range defined by the lower and upper bounds, but the exact location of this delay in the range is unknown due to test resolution and delay estimation. To tackle uncertainty, a conservative method to configure the buffers is to assume the upper bounds of the ranges to be path delays, so that the chip always works with the resulting buffer configuration. This method, however, may incorrectly report some chips as nonfunctional due to pessimistic delay overestimation. To solve this problem, we try to find a buffer configuration for a chip while assuming the delays are as close to their corresponding upper bounds as possible. By minimizing the distance of the assumed delays from their corresponding upper bounds when determining the buffer configuration, the chance that the chip works after configuration becomes large, so that the final pass/fail test will accept most post-silicon configured chips as functional. Because the variances of predicted maximum delays differ from each other, the distance to the upper bounds are also scaled by the standard deviations of the predicted delays.

The optimization problem to find a buffer configuration while minimizing the distance ξ of the assumed delays from the corresponding upper bounds is described as follows.

$$\text{Minimize} \quad \xi \tag{27}$$

$$\text{Subject to} \quad \forall \text{ path } p_{ij}$$

$$T_d \geq D'_{ij} + x_i - x_j \tag{28}$$

$$l_{ij} \leq D'_{ij} \leq u_{ij},\ \xi \geq (u_{ij} - D'_{ij})/\sigma'_{ij} \tag{29}$$

$$r_i \leq x_i \leq r_i + \tau_i,\ r_j \leq x_j \leq r_j + \tau_j \tag{30}$$

where D'_{ij} is the assumed delay value of a path during buffer configuration; σ'_{ij} is the standard deviation of the corresponding predicted delay; T_d is the designated clock period for the design; (28) and (30) are derived from (1) and (3), respectively. By solving the optimization problem (27)–(30), a set of buffer configuration values x_i and x_j can be found.

E. Tuning Bounds due to Hold Time Constraints

In the discussion above, we do not consider hold time constraints. However, tuning buffers may affect hold time constraints significantly if they are configured improperly. For example, in Fig. 3, if x_j is much larger than x_i, the constraint (2) may be violated.

As shown in (2), hold time constraints are affected by $x_i - x_j$ instead of individual values of x_i and x_j. In our method, we do not test against hold time violations after configuring buffers. Instead, we set a lower bound λ_{ij} for $x_i - x_j$ by sampling the statistical distribution of d_{ij} in (2) so that a given yield can be maintained.

Consider the case that d_{ij} in (2) is sampled M times for all short paths and its value in the kth sample is $d_{ij,k}$. For the kth sample, we use a 0-1 variable y_k to represent that the lower bound λ_{ij} meet

$$\lambda_{ij} - d_{ij,k} \geq \mathcal{M}(y_k - 1), \quad \text{for all short paths } p_{ij} \tag{31}$$

where $\mathcal{M}$ is a very large constant. The yield of the circuit with respect to hold time can thus be constrained as

$$\sum y_i/M \geq Y, \quad i = 1, 2, \ldots M \tag{32}$$

where Y is a given yield for hold time constraints, set to 0.99 in our method. To allow buffers to have the largest freedom in value configuration, we minimize the sum of all the lower bounds $\sum_{i,j} \lambda_{ij}$. After λ_{ij} are determined, the buffer configuration values can be constrained to avoid hold time violation, as shown below

$$x_i - x_j \geq \lambda_{ij}. \tag{33}$$

This constraint is added into the optimization problems in Section III-C and Section III-D to incorporate hold time constraints to determine buffer values x_i and x_j.

IV. Experimental Results

The proposed framework was implemented in C++ and tested using a 3.20 GHz CPU. We demonstrate the results with four circuits, s9234 to s38584, from the ISCAS89 benchmark set and four circuits, mem_ctrl to pci_bridge32, from the TAU13 variation-aware timing analysis contest. Details of these circuits are shown in Table II, where $\boldsymbol{n_s}$ denotes the number of flip-flops and $\boldsymbol{n_g}$ the number of logic gates. The numbers of inserted tunable buffers are shown in the column $\boldsymbol{n_b}$, which were less than 1% of the numbers of flip-flops in the benchmark circuits. The locations of these buffers were determined using [30]. We set the maximum allowed buffer ranges to 1/8 of the original clock period and all tuning delays with 20 discrete steps [22]. The logic gates in the circuits were sized and mapped using a 45 nm library. The standard deviations of transistor length, oxide thickness and threshold voltage were set to 15.7%, 5.3% and 4.4% of the nominal values [56]. The correlation of variations between logic gates is defined using the curve in [57]. The ILP solver for the optimization problems was Gurobi [58].

In Table II the column $|\mathbf{D^m}|$ shows the numbers of maximum delays between flip-flops whose delays need to be evaluated for post-silicon buffer configuration. If a flip-flop is attached a tunable buffer, the maximum delays from all its

TABLE II: Test Results With Delay Alignment and Statistical Prediction

Circuit				EffiTest2						Frequency Stepping				Runtime		
	n_s	n_g	n_b	$\lvert\mathbf{D}^m\rvert$	$\lvert\mathbf{D}_t^m\rvert$	$\lvert\mathbf{P}_t\rvert$	$\lvert\mathbf{B}\rvert$	n_a	n_v	n'_a	n'_v	$r_a(\%)$	$r_v(\%)$	$T_p(s)$	$T_t(s)$	$T_s(s)$
s9234	211	5597	2	87	6	7	5	46.33	6.62	871.31	10.02	94.68	33.91	5.93	0.06	0.00
s13207	638	7951	6	456	12	12	2	51.25	4.27	4550.88	9.98	98.87	57.20	16.34	0.10	0.00
s15850	534	9772	5	546	10	11	4	47.05	4.28	5452.90	9.99	99.14	57.17	50.46	0.12	0.01
s38584	1426	19253	14	437	14	14	3	61.42	4.39	4366.07	9.99	98.59	56.09	89.94	0.15	0.03
mem_ctrl	1065	10327	10	2210	15	36	4	105.48	2.93	22038.12	9.97	99.52	70.62	299.63	0.69	0.06
usb_funct	1746	14381	17	1402	13	28	2	87.98	3.14	13975.14	9.97	99.37	68.48	143.13	0.36	0.04
ac97_ctrl	2199	9208	21	1714	13	20	2	58.78	2.94	16879.47	9.85	99.65	70.16	146.53	0.24	0.02
pci _bridge32	3321	12494	33	4501	22	58	6	181.60	3.13	44784.95	9.95	99.59	68.53	1712.57	0.92	0.79

fanin flip-flops to it and from it to all its fanout flip-flops are added into $\mathbf{D}^m$ as discussed in Section III-A. Consequently, these numbers may still be large, specially in the test cases mem_ctrl and pci_bridge32, despite only a small number of buffers (n_b) are inserted into the circuits. The column $\boldsymbol{|\mathbf{D}_t^m|}$ shows the numbers of maximum delays $\mathbf{D}_t^m$ after applying SVD-QRcp decomposition in Algorithm 1. These maximum delays are identified from $\mathbf{D}^m$ and used to narrow the search scope of real combinational paths. Due to statistical prediction, the size of $\mathbf{D}_t^m$ is much smaller than that of $\mathbf{D}^m$, so that the number of paths that are really tested can be reduced effectively. The numbers of real combinational paths to be tested are shown in the column $\boldsymbol{|\mathbf{P}_t|}$. These paths are identified by iterative selection in Algorithm 2. The tested delays of these paths are used to predicted $\mathbf{D}^m$. The efficiency of this delay prediction can be demonstrated by the comparison between $|\mathbf{P}_t|$ and $|\mathbf{D}^m|$ clearly, where the numbers of test paths are only about 2%–3% of the numbers of the maximum delays in most cases. The paths in $\mathbf{P}_t$ are grouped in test batches as described in Section III-B, and the numbers of these batches are shown in the column $|\mathbf{B}|$. Since multiple combinational paths are multiplexed during delay test, these numbers can be much smaller than those of $\mathbf{P}_t$.

In the experiments, we tested 10 000 simulated chips by sampling statistical delays from statistical timing analysis. The column $\boldsymbol{n_a}$ shows the average number of frequency stepping iterations for each chip using EffiTest2, and the column $\boldsymbol{n_v}$ shows the average number of iterations per path, where $n_v = n_a/|\mathbf{P}_t|$. For comparison, we implemented the method applying frequency stepping to each path individually, as assumed in [26], [35]–[37]. The column $\boldsymbol{n'_a}$ in Table II shows the average number of test iterations for each chip. These large numbers confirm that the straightforward frequency stepping method is impractical for large circuits. Furthermore, the column $\boldsymbol{n'_v}$ shows the average number of frequency stepping iterations per path where $n'_v = n'_a/|\mathbf{D}^m|$. The columns $\boldsymbol{r_a(\%)}$ and $\boldsymbol{r_v(\%)}$ show the reduction ratios of the test iterations per chip and the test iterations per path achieved using EffiTest2, where $r_a = (n'_a - n_a)/n'_a * 100$ and $r_v = (n'_v - n_v)/n'_v * 100$. Combining statistical prediction and aligned delay test, EfiiTest2 can reduce the overall test effort by more than 94% (94.68%∼99.65%). In addition, the ratios of test iterations per path $r_v(\%)$ demonstrate that test reduction can reach from 33.91% to 70.62%. This reduction comes only from test multiplexing and aligned delay test, while the statistical prediction technique does not affect this ratio. Both comparisons confirm that the proposed EffiTest2 framework reduces test effort significantly.

The runtimes of the proposed method are shown in the last three columns in Table II, where $\boldsymbol{T_p}$ is the runtime for path identification, batch assignment and hold time bound computation before delay test starts. Because these steps are performed offline, the runtime is already acceptable. The column $\boldsymbol{T_t(s)}$ shows the average runtime when computing the clock period T and the buffer configuration values for all test batches of a chip. Since this computation can be performed in parallel while path batches are tested, the runtime is also acceptable compared with the execution time of scan test. The last column $\boldsymbol{T_s(s)}$ shows the runtime to determine the final buffer values using the method in Section III-D. This step is not performed on high-end testers so that the efficiency is good enough.

In the proposed framework, the results of aligned delay test produce lower and upper bounds for delays. This inaccuracy cannot be avoided due to the nature of delay test and it affects the yields of the circuits after buffer configuration. In addition, the technique of statistical prediction also introduces configuration inaccuracy in the estimated delays. Consequently, it is expected that the yield values of the circuits should drop from the ideal yield values with delays assumed being measured exactly. We tested several cases with two clock periods T_1 and T_2 and the results are shown in Table III. The yield in this table was calculated by checking the setup and hold time constraints between pairs of flip-flops for the 10000 simulated chips after buffer configuration. If the timing constraints were satisfied for a simulated chip that failed initially without buffer configuration, we assumed the chip after buffer configuration was rescued, so that the yield was improved by 0.01%. For T_1 and T_2 the original yield values without buffers were 50% and 84.13%, corresponding to the cases of setting the target clock period to mean and mean plus standard deviation of the clock period calculated by SSTA, respectively. The column y_i shows the yield values with a perfect delay measurement; the column y_t shows the yield values with delays measured by the proposed method; and the column y_r shows the yield drops due to the inaccuracy in the tested delays, where $y_r = y_i - y_t$. In these results, we can see that the yield drops are around 1-2%, where the improved yield values are still far better than those without buffers.

Since the results of the statistical prediction technique in Section III-A depend on the correlations between path delays, we manually increased the standard deviations of all delays by 10%. These increased variations are added to the purely random part of the delays so that the correlations between delays are decreased accordingly. Figure 9 shows the yield results of three cases with respect to the clock period T_2 in

TABLE III: Yield Comparison

Circuit	T_1			T_2		
	$y_i(\%)$	$y_t(\%)$	$y_r(\%)$	$y_i(\%)$	$y_t(\%)$	$y_r(\%)$
s9234	52.77	52.51	0.26	85.01	84.99	0.02
s13207	63.58	61.98	1.60	89.88	89.81	0.07
s15850	68.19	67.08	1.11	93.51	92.63	0.88
s38584	64.67	63.01	1.66	92.33	91.56	0.77
mem_ctrl	59.08	58.35	0.73	89.30	88.95	0.35
usb_funct	54.98	53.69	1.29	86.72	85.85	0.87
ac97_ctrl	58.37	57.84	0.53	88.01	87.81	0.20
pci _bridge32	60.56	58.87	1.69	89.10	88.08	1.02
Yield w/o tuning	50.00			84.13		

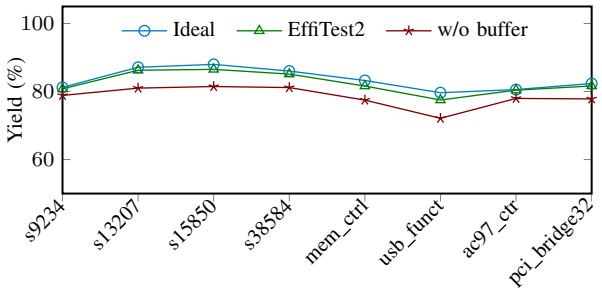


Fig. 9: Yield with enlarged random variation.

Table III: 1) ideal yield with buffers configured with presumed accurate delays; 2) with buffers configured using tested and predicted delays in EffiTest2; 3) no buffers in the circuits. Compared with the results in Table III, the yield values in Fig. 9 are lower due to the increased random variation. The first two cases, however, demonstrate clearly that the yield results were still improved impressively using tunable buffers when compared with the cases without them. When testing and configuring the buffer values with EffiTest2, the yield values dropped slightly from the ideal cases in Fig. 9, because of the expected inaccuracy in delay test and prediction. These yield values, however, still followed the ideal cases closely, confirming the strength of EffiTest2.

To verify the effectiveness of aligned delay ranges described in Section III-B and Section III-C, we applied them directly to reduce test iterations without statistical prediction. Figure 10 shows the comparison of the numbers of test iterations per path in three cases: 1) path-wise frequency stepping, where around ten iterations were needed for each path; 2) test multiplexing without delay alignment using buffers; 3) multiplexing with delay alignment using buffers in EffiTest2. The second case used the method in Section III-B and Section III-C, but all the buffers values were set to zero during test. Comparing the results of the first case and the second case, we can see that test multiplexing is a powerful technique to reduce test iterations. When the technique of delay alignment is applied, test iterations can be reduced further, as demonstrated by the third case. These results confirm that even without taking advantage of the correlations between path delays, the proposed method can still reduce test cost significantly.

In the statistic prediction technique described in Algorithm 1 and 2, the iterations stop when the predicted maximum variances reach a threshold σ_{th}. In delay prediction, the variance of a predicted variable cannot be lower than that of its purely random component. To experiment with different thresholds σ_{th} used in Algorithm 1 and 2, we first find the maximum

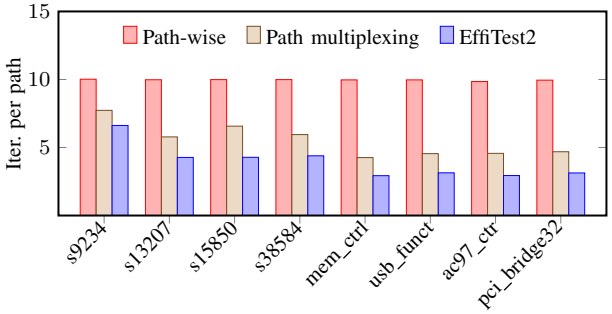


Fig. 10: Test comparison without statistical prediction.

of these purely random components of the predicted delays and set the threshold as constant times of them. Figure 11(a) shows the effect of these threshold values. As the threshold value reaches $3.5 \times \sigma_{max}$, the yield values of the circuits start to drop, because of the large range of the predicted delays. In EffiTest2, this constant was set to 2.0. Similarly, in the test procedure, the binary search of frequency stepping quits when an accuracy is reached. We have also tested different threshold values of ϵ in Algorithm 3 and the results are shown in Figure 11(b), where the x axis shows the constant times of the maximum of the mean values of the delays. As this number reaches 0.01, slight accuracy loss starts to appear. This number was set to 0.005 in EffiTest2 to maintain the test quality.

In Algorithm 1 and 2 we also increased the number of variables in $\mathbf{D}_t^m$ gradually in the loops, instead of using a binary search to reduce execution time. Figure 12 shows the trend of accuracy improvement with the two cases s13207 and pci_bridge32. For these two circuits, the accuracy does not improve notably after the number of variables becomes relatively large. Using the threshold setting discussed in Section III-A, this number was set to 12 for s13207 and 22 for pci_bridge32 in the experiments. Furthermore, it can be observed that the curves for these two circuits do not decrease monotonously, so that a binary search may not return the best result. For example, 15 instead of 12 variables for s13207, and 24 instead of 22 variables for pci_bridge32, should be selected if a binary search would be used for these two cases.

To demonstrate the prediction accuracy using a given number of combinational paths for each maximum delay in $\mathbf{D}_t^m$ as discussed in Section III-A, we compared the accuracy of all delays in $\mathbf{D}_m$ when the number of selected combinational paths is varied to from 1 to 10 in Algorithm 2. For a circuit, the threshold of the prediction accuracy σ_{th} for path selection in Algorithm 2 is set with respect to the standard deviations of purely random components of path delays described in Section III-A. Accordingly, the predicted maximum standard deviations σ_{max} in $\mathbf{D}^m$ are different in the tested circuits. With more paths selected for testing, σ_{max} decreases due to the correlation information, however, with an increase of test cost. Fig. 13 shows the trend of maximum standard deviations σ_{max} of predicted values of $\mathbf{D}_m$ with respect to the number of selected paths. With the increase of the selected number, σ_{max} decreases, meaning the prediction accuracy is improved. When the number of selected paths is larger than 5, the accuracy does not change noticeably. Therefore, we set this number to 5 in EffiTest2 to maintain the accuracy. The other circuits that are

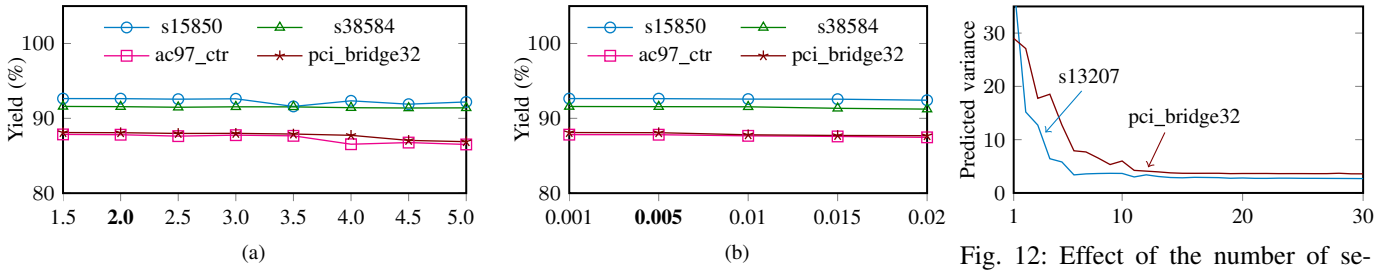


Fig. 11: Prediction threshold and its effect on yield in Algorithm 1 and Algorithm 2 (a), and test threshold over yield in Algorithm 3 (b).

Fig. 12: Effect of the number of selected variables over prediction accuracy.

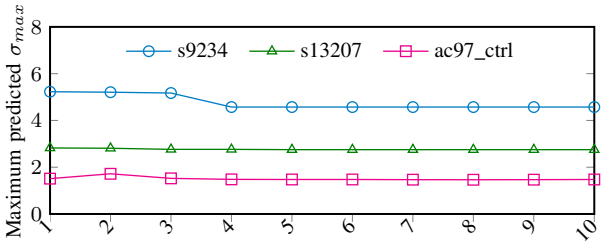


Fig. 13: Comparisons of predicted standard deviations using different numbers of combinational paths. Lower values indicate better prediction accuracy.

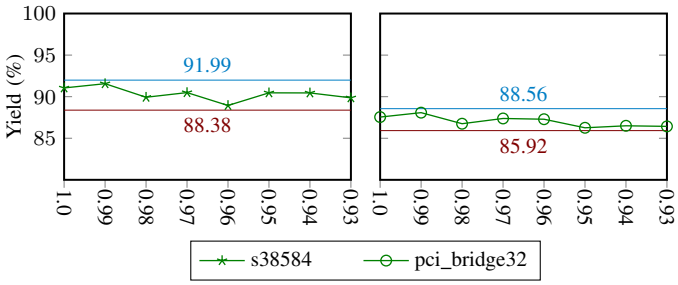


Fig. 14: Hold time threshold effect over yield.

not included in Fig. 13 show less trend changes in the predicted standard deviations in terms of the number of combinational paths.

In EffiTest2, we do not test short paths using frequency stepping so that test iterations can be reduced. Instead, we set adjustment ranges as described in Section III-E to reduce hold time violations. These constraints are controlled by the threshold Y in (32), whose effect over the yield values of s38584 and pci_bridge32 are shown in Fig. 14. In each of these two figures, the two straight lines and the corresponding numbers show upper bound and lower bound of the yield values, where the former is computed by ignoring all hold time violations and the latter is computed by not adding the constraints in (32)–(33). When the threshold Y decreases to 0.98, the yield values of the circuits start to drop. Therefore, we set Y to 0.99 in EffiTest2.

V. Conclusions

In this paper we have proposed an efficient framework to reduce test cost in configuring tunable buffers in high-performance designs. By providing customized clock schemes to manufactured chips, timing failures may be alleviated by intentional clock skews with respect to the effect of process variations. The proposed framework combines statistical prediction and aligned delay test with path multiplexing to reduce test cost during post-silicon configuration. Consequently, the number of test iterations can be reduced by more than 94%, while the improved yield of the circuit is well maintained. The effectiveness of these techniques has been confirmed by experimental results using ISCAS89 and TAU13 benchmark circuits. Future work will consider techniques combining post-silicon tuning and flexible timing such as in [59].

References

[1] G. L. Zhang, B. Li, and U. Schlichtmann, "EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers," in *Proc. Design Autom. Conf.*, 2016, pp. 60:1–60:6.

[2] 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.*, 2003, pp. 621–625.

[3] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proc. Design Autom. Conf.*, 2004, pp. 331–336.

[4] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C. Chen, "Correlation-preserved non-gaussian statistical timing analysis with quadratic timing model," in *Proc. Design Autom. Conf.*, 2005, pp. 83–88.

[5] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis," in *Proc. Design Autom. Conf.*, 2006, pp. 155–160.

[6] A. Goel, S. B. K. Vrudhula, F. Taraporevala, and P. Ghanta, "A methodology for characterization of large macro cells and IP blocks considering process variations," in *Proc. Int. Symp. Quality Electron. Des.*, 2008, pp. 200–206.

[7] A. Goel, S. Vrudhula, F. Taraporevala, and P. Ghanta, "Statistical timing models for large macro cells and IP blocks considering process variations," *IEEE Trans. Semicond. Manuf.*, vol. 22, pp. 3–11, 2009.

[8] B. Li, N. Chen, M. Schmidt, W. Schneider, and U. Schlichtmann, "On hierarchical statistical static timing analysis," in *Proc. Design, Autom., and Test Europe Conf.*, 2009, pp. 1320–1325.

[9] B. Li, N. Chen, Y. Xu, and U. Schlichtmann, "On timing model extraction and hierarchical statistical timing analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 3, pp. 367–380, 2013.

[10] R. Kumar, B. Li, Y. Shen, U. Schlichtmann, and J. Hu, "Timing verification for adaptive integrated circuits," in *Proc. Design, Autom., and Test Europe Conf.*, 2015, pp. 1587–1590.

[11] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: from basic principles to state of the art," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 4, pp. 589–607, Apr. 2008.

[12] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. Int. Symp. Microarch.*, 2003, pp. 7–18.

[13] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. M. Bull, "Razor II: In situ error detection and correction for pvt and ser tolerance," in *Proc. Int. Solid-State Circuits Conf.*, pp. 400–401.

[14] M. Fojtik, D. Fick, Y. Kim, N. R. Pinckney, D. M. Harris, D. Blaauw, and D. Sylvester, "Bubble razor: Eliminating timing margins in an ARM cortex-m3 processor in 45 nm CMOS using architecturally independent error detection and correction," *J. Solid-State Circuits*, vol. 48, no. 1, pp. 66–81, 2013.

[15] I. Kwon, S. Kim, D. Fick, M. Kim, Y. Chen, and D. Sylvester, "Razor-lite: A light-weight register for error detection by observing virtual supply rails," *IEEE J. Solid-State Circuits*, vol. 49, no. 9, pp. 2054–2066, 2014.

[16] N. Chen, B. Li, and U. Schlichtmann, "Iterative timing analysis based on nonlinear and interdependent flipflop modelling," *IET Circuits, Devices & Systems*, vol. 6, no. 5, pp. 330–337, 2012.

[17] E. Salman, A. Dasdan, F. Taraporevala, K. Küçükçakar, and E. G. Friedman, "Exploiting setup-hold-time interdependence in static timing analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 6, pp. 1114–1125, 2007.

[18] S. Srivastava and J. S. Roychowdhury, "Interdependent latch setup/hold time characterization via Euler-Newton curve tracing on state-transition equations," in *Proc. Design Autom. Conf.*, 2007, pp. 136–141.

[19] A. B. Kahng and H. Lee, "Timing margin recovery with flexible flip-flop timing model," in *Proc. Int. Symp. Quality Electron. Des.*, 2014, pp. 496–503.

[20] Y. Yang, K. H. Tam, and I. H. Jiang, "Criticality-dependency-aware timing characterization and analysis," in *Proc. Design Autom. Conf.*, 2015, pp. 167:1–167:6.

[21] G. L. Zhang, B. Li, and U. Schlichtmann, "Piecetimer: a holistic timing analysis framework considering setup/hold time interdependency using a piecewise model," in *Proc. Int. Conf. Comput.-Aided Des.*, 2016, p. 100.

[22] S. Tam, S. Rusu, U. Nagarji Desai, R. Kim, J. Zhang, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.

[23] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi, "Post-fabrication clock-timing adjustment using genetic algorithms," *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 643–650, Apr. 2004.

[24] P. Mahoney, E. Fetzer, B. Doyle, and S. Naffziger, "Clock distribution on a dual-core, multi-threaded Itanium®-family processor," in *Proc. Int. Solid-State Circuits Conf.*, 2005, pp. 292–293.

[25] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, and M. Horowitz, "The implementation of a 2-core, multi-threaded Itanium family processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[26] J. Tsai, D. Baik, C. C.-P. Chen, and K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 611–618.

[27] J. Kim and T. Kim, "Adjustable delay buffer allocation under useful clock skew scheduling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 4, pp. 641–654, Apr. 2017.

[28] J. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 575–581.

[29] G. L. Zhang, B. Li, and U. Schlichtmann, "Sampling-based buffer insertion for post-silicon yield improvement under process variability," in *Proc. Design, Autom., and Test Europe Conf.*, 2016, pp. 1457–1460.

[30] G. L. Zhang, B. Li, J. Liu, Y. Shi, and U. Schlichtmann, "Design-phase buffer allocation for post-silicon clock binning by iterative learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 392–405, 2018.

[31] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," in *Proc. Int. Symp. Phys. Des.*, 2007, pp. 11–18.

[32] K. Nagaraj and S. Kundu, "A study on placement of post silicon clock tuning buffers for mitigating impact of process variation," in *Proc. Design, Autom., and Test Europe Conf.*, 2009, pp. 292–295.

[33] B. Li, N. Chen, and U. Schlichtmann, "Fast statistical timing analysis for circuits with post-silicon tunable clock buffers," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 111–117.

[34] B. Li and U. Schlichtmann, "Statistical timing analysis and criticality computation for circuits with post-silicon clock tuning elements," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1784–1797, 2015.

[35] Z. Lak and N. Nicolici, "A novel algorithmic approach to aid post-silicon delay measurement and clock tuning," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1074–1084, May 2014.

[36] K. Nagaraj and S. Kundu, "An automatic post silicon clock tuning system for improving system performance based on tester measurements," in *Proc. Int. Test Conf.*, 2008, pp. 1–8.

[37] D. Tadesse, J. Grodstein, and R. I. Bahar, "AutoRex: An automated post-silicon clock tuning tool," in *Proc. Int. Test Conf.*, 2009, pp. 1–10.

[38] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 442–447.

[39] Z. Lak and N. Nicolici, "On using on-chip clock tuning elements to address delay degradation due to circuit aging," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 12, pp. 1845–1856, Dec. 2012.

[40] A. Chakraborty, K. Duraisami, A. V. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino, "Dynamic thermal clock skew compensation using tunable delay buffers," *IEEE Trans. VLSI Syst.*, vol. 16, no. 6, pp. 639–649, Jun. 2008.

[41] J. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[42] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*. Upper Saddle River: Pearson Prentice Hall, 2007.

[43] Q. Liu and S. S. Sapatnekar, "A framework for scalable postsilicon statistical delay prediction under process variations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 8, pp. 1201–1212, Aug. 2009.

[44] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori, "Representative critical-path selection for aging-induced delay monitoring," in *Proc. Int. Test Conf.*, 2013, pp. 1–10.

[45] J. Yen and L. Wang, "Simplifying fuzzy rule-based models using orthogonal transformation methods," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 29, no. 1, pp. 13–24, 1999.

[46] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Proc. Design Autom. Conf.*, 2010, pp. 386–391.

[47] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori, "Aging- and variation-aware delay monitoring using representative critical path selection," *ACM Trans. Design Autom. Electr. Syst*, vol. 20, no. 3, pp. 1–39, Jun. 2015.

[48] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Society for Industrial and Applied Mathematics, 1999.

[49] M. Galassi *et al.*, *GNU Scientific Library Reference Manual*, 3rd ed.

[50] S. Pateras, "Achieving at-speed structural test," *IEEE Des. Test. Comput.*, vol. 20, no. 5, pp. 26–33, 2003.

[51] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli, "High-frequency, at-speed scan testing," *IEEE Des. Test. Comput.*, vol. 20, no. 5, pp. 17–25, 2003.

[52] M. Michael and S. Tragoudas, "Function-based compact test pattern generation for path delay faults," *IEEE Trans. VLSI Syst.*, vol. 13, no. 8, pp. 996–1001, Aug. 2005.

[53] I. Pomeranz, S. Reddy, and P. Uppaluri, "NEST: a nonenumerative test generation method for path delay faults in combinational circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 12, pp. 1505–1515, 1995.

[54] P. Pant, J. Zelman, G. Colon-Bonet, J. Flint, and S. Yurash, "Lessons from at-speed scan deployment on an intel itanium microprocessor," in *Proc. Int. Test Conf.*, 2010, pp. 1–8.

[55] D. Chen, R. Batson, and Y. Dang, *Applied Integer Programming: Modeling and Solution*. Wiley, 2011.

[56] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. Custom Integr. Circuits Conf.*, 2001, pp. 223–228.

[57] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 4, pp. 619–631, 2007.

[58] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013. [Online]. Available: http://www.gurobi.com

[59] G. L. Zhang, B. Li, M. Hashimoto, and U. Schlichtmann, "VirtualSync: Timing optimization by sychronizing logic waves with sequential and combinational components as delay units," in *Proc. Design Autom. Conf.*, 2018.

Grace Li Zhang received the master's degree from the school of microelectronics, Xidian University, Xi'an, China, in 2014. She is currently pursuing the Ph.D. degree with the Chair of Electronic Design Automation, Technical University of Munich (TUM). Her research interests include high-performance and lower-power design, as well as emerging systems.

Bing Li received the bachelor's and master's degrees in communication and information engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2000 and 2003, respectively, and the Dr.-Ing. degree in electrical engineering from Technical University of Munich (TUM), Munich, Germany, in 2010. He is currently a researcher with the Chair of Electronic Design Automation, TUM. His research interests include high-performance and lower-power design, as well as emerging systems.

Yiyu Shi (SM'06) is currently an associate professor in the Departments of Computer Science and Engineering and Electrical Engineering at the University of Notre Dame. He received his B.S. degree (with honors) in Electronic Engineering from Tsinghua University, Beijing, China in 2005, the M.S and Ph.D. degree in Electrical Engineering from the University of California, Los Angeles in 2007 and 2009 respectively. His current research interests include three-dimensional integrated circuits, and machine learning on chips. In recognition of his research, he has received many best paper nominations in top conferences. He was also the recipient of IBM Invention Achievement Award in 2009, Japan Society for the Promotion of Science (JSPS) Faculty Invitation Fellowship, Humboldt Research Fellowship for Experienced Researchers, IEEE St. Louis Section Outstanding Educator Award, Academy of Science (St. Louis) Innovation Award, Missouri S&T Faculty Excellence Award, National Science Foundation CAREER Award, IEEE Region 5 Outstanding Individual Achievement Award, and the Air Force Summer Faculty Fellowship.

Jiang Hu (F'16) received the B.S. degree in optical engineering from Zhejiang University (China) in 1990, the M.S. degree in physics in 1997 and the Ph.D. degree in electrical engineering from the University of Minnesota in 2001. He worked with IBM Microelectronics from January 2001 to June 2002. In 2002, he joined the electrical engineering faculty at Texas A&M University. His research interest is in optimization for large scale computing systems, especially VLSI circuit optimization, adaptive design, hardware security, resource allocation and power management in computing systems.

Dr. Hu received the Best Paper Award at the ACM/IEEE Design Automation Conference in 2001, an IBM Invention Achievement Award in 2003, and the Best Paper Award from the IEEE/ACM International Conference on Computer-Aided Design in 2011. He has served as a technical program committee member for DAC, ICCAD, ISPD, ISQED, ICCD, DATE, ISCAS, ASP-DAC and ISLPED. He is general chair for the 2012 ACM International Symposium on Physical Design, and an associate editor of IEEE transactions on CAD 2011-2016. Currently he is an associate editor for the ACM transactions on Design Automation of Electronic Systems. He received a Humboldt research fellowship in 2012.

Ulf Schlichtmann (S'88–M'90) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering and information technology from Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively. He was with Siemens AG, Munich, and Infineon Technologies AG, Munich, from 1994 to 2003, where he held various technical and management positions in design automation, design libraries, IP reuse, and product development. He has been a Professor and the Head of the Chair of Electronic Design Automation with TUM, since 2003. He served as the Dean of the Department of Electrical and Computer Engineering, TUM, from 2008 to 2011. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems.