Statistical Timing Analysis Driven Post-Silicon-Tunable Clock-Tree Synthesis

Jeng-Liang Tsai, Lizheng Zhang and Charlie Chung-Ping Chen
University of Wisconsin-Madison
Department of Electrical and Computer Engineering
1415 Engineering Drive
Madison, WI 53706
{*jltsai@cae, lizhengz@cae, chen@engr*}*.wisc.edu*

Abstract*— *Process variations cause significant timing uncertainty and yield degradation in deep sub-micron technologies. A solution to counter timing uncertainty is post-silicon clock tuning. Existing design approaches for post-silicon-tunable (PST) clock-tree synthesis usually insert a PST clock buffer for each flip-flop or put PST clock buffers across an entire level of a clock-tree. This can cause significant over-design and long tuning time. In this paper, we propose to insert PST clock buffers at both internal and leaf nodes of a clock-tree and use a bottom-up algorithm to reduce the number of candidate PST clock buffer locations. We then provide two statistical-timing-driven optimization algorithms to reduce the hardware cost of a PST clock-tree. Experimental results on ISCAS89 benchmark circuits show that our algorithms achieve up to a 90% area or a 90% number of tunable clock buffer reductions compared to existing design methods.

I. INTRODUCTION

PST clock-tree has become an important design-for-yield (DFM) technique to counter variations on path delay and clock skew in manufactured chips. In deep sub-micron technologies, yield loss is mainly from the following two sources: (a) *functional* yield loss due to processing defects, and (b) *timing* yield loss due to timing failures caused by processing parameter variations. Timing yield loss is *recoverable* by reducing the sensitivity of the circuit to process variations. However, the increasing intensity of process variations in new technologies, stringent time-to-market requirements, and limits on non-recurring-engineering (NRE) cost have made it difficult to add timing yield in the design objective during circuit optimizations. It is favorable to have generic DFM techniques that can be applied to different designs with the least impact on current design flows.

Rajaram et al. [1] propose to reduce clock skew variations by inserting *cross links* in a given clock-tree. However, this technique cannot take path delay variations into account. Another promising DFM technique is to use PST clock-trees [2]–[4]. By inserting PST clock buffers into the clock-tree, slacks can be redistributed among adjacent timing paths and timing failures may be corrected through *post-silicon clock tuning*. As shown in Fig. 1, the clock distribution network of Intel's recently announced dual-core Itanium® processor uses two levels of PST clock buffers to counter process-induced clock skews and improve the timing yield. The tunable second level clock buffers (SLCBs) at the terminals of L1 route can be dynamically adjusted by on-chip clock phase detection hardware to cancel clock skew variations. They can also be programmed from the test access port (TAP) for timing optimization [2], [3], [5]. There is also a second level of PST clock buffers at every terminal of the L2 route. This level consists $\sim 15K$ clock vernier devices (CVDs) for clock fine-tuning through scan [3].

Post-silicon clock tuning not only improves the timing yield but also reduces clock power by avoid using grid-based clock distribution networks. However, a brute-force design method that inserts a PST clock buffer for each flip-flop or at each clock-tree terminal uses

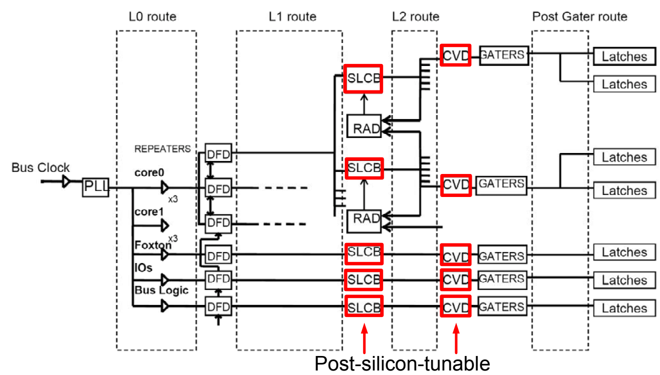


Fig. 1. Clock distribution network of a dual-core Intel® Itanium® processor (figure cited from [3]).

a significant amount of the chip area. Moreover, it also increases the clock power due to the extra capacitance load of PST clock buffers. To the best of the authors' knowledge, there is no systematic ways to construct a PST clock-tree that *provides the maximum tuning capability for timing yield improvements with minimum hardware cost*. We propose to use statistical timing analysis to drive the PST clock-tree synthesis flow. By analyzing the effects of process variations on timing, we can insert PST clock buffers only at the critical locations in a clock-tree and assign appropriate tunable range to PST clock buffers. This can greatly reduce the hardware cost of a PST clock-tree.

The rest of this paper is organized as follows: in Section II we first gives two formulations, PST-A and PST-N, on the PST clock-tree synthesis problem based on two hardware cost metrics. Section III provides a timing yield model for sequential circuits based on SSTA and Monte Carlo integration. The effects of PST clock buffers on timing yield is analyzed and a timing yield model in the presence of PST clock buffers is developed in Section IV. An iterative linesearch algorithm utilizing a fast gradient approximation algorithm is proposed in Section V to solve PST-A. A batch selection algorithm is proposed in Section VI to solve PST-N. Experimental results on the effectiveness of the proposed algorithms are demonstrated in Section VII, and finally Section VIII concludes this work.

II. PROBLEM FORMULATION

The tuning capability of a PST clock-tree is dependent on the number of PST clock buffers and their tunable range. To achieve the maximum timing yield improvement, a brute-force method may insert PST clock buffers at every terminal of the clock-tree and design the PST clock buffers to have a large tunable range. However, this can add a large hardware overhead to the design.

To optimize the hardware cost of a PST clock-tree, it is essential to choose a cost metric that reflects the actual silicon cost. There are several PST clock buffer designs that achieve variable clock delay with very different circuit design techniques. A common PST clock buffer design consists two inverters with a bank of passive loads in between [2], [6]. Each passive load can be connected or disconnected to the inverters by programming the control bit of its pass gate. This type of PST clock buffer relies on RC delay to control the clock delay. To achieve a large tunable range, we need to have a large passive load. Since on-chip capacitors require a large area in a digital VLSI process, the appropriate cost metric for this type of PST clock buffer is the area required to implement the passive loads, which is proportional to the required tunable range of the buffer. Another PST clock buffer design achieves variable delay by changing the driving strength of a buffer. This is either done through controlling the bias voltage of the driver with a digital-analog-converter [4] or introducing contention to the driver [3]. For this type of design, the hardware cost is insensitive to the tunable range and can be treated as a constant. Therefore, the appropriate cost metric with this type of design is the total number of PST clock buffers in the clock-tree. In this paper, we do not make assumptions on the design of a PST clock buffer. Instead, we use both metrics, *total tunable range* and *total number of PST clock buffers*, for the hardware cost and define two PST clock-tree synthesis problems as follows.

Problem PST-A: (To minimize area)
Given a circuit and its buffered clock-tree, determine the required tunable range of each clock buffer such that the total tunable range is minimized and the target timing yield is achieved.

Problem PST-N: (To minimize the number of PST clock buffers)
Given a circuit and its buffered clock-tree, select a minimum subset of clock buffers such that the target timing yield is achieved when the selected clock buffers are converted to PST clock buffers.

The PST-A and PST-N problems require very different optimization approaches but are driven by the same timing yield model. In the following sections, we propose two timing yield models for circuits with and without PST clock buffers based on statistical timing analysis and Monte Carlo integration. Two algorithms for solving the PST-A and PST-N problems are then presented.

III. Timing Yield Model

A sequential circuit is represented by its circuit graph $G = (B, V, E)$, where B is the set of clock buffers and V is the set of flip-flops. E is the set of timing arcs and e_{ij} indicates that there are combinational paths between i and j. The clock skew between i and j is defined as $\alpha_{ij} = T_i - T_j$, where T_i is the clock arrival time at i. The maximum and minimum path delays from i to j are denoted D_{ij} and d_{ij}.

A. Timing Constraints and Slack Vector

A circuit needs to satisfy hold-time and setup-time constraints:

$$\alpha_{ij} + d_{ij} \geq T_h^j, \tag{1}$$
$$\alpha_{ij} + D_{ij} \leq T - T_s^j, \tag{2}$$

where T_h^j and T_s^j are the hold-time and setup-time of flip-flop j and T is the clock period. Define the hold-time slack of (1) as $s_{ij}^d = \alpha_{ij} + d_{ij} - T_h^j$ and the setup-time slack as $s_{ij}^D = T - D_{ij} - \alpha_{ij} - T_s^j$ and collect all the slack variables as an $R^{2|E|\times 1}$ *slack vector* $\mathbf{s}$, a circuit satisfies all the timing constraints if

$$\begin{aligned} \mathbf{s} &\in \mathcal{C}_0, \\ \mathcal{C}_0 &= \{\mathbf{w} \mid w_i \geq 0, i = 1 \dots 2|E|\}. \end{aligned} \tag{3}$$

In other words, a circuit is free from timing failures if its slack vector is in the *feasible region* $\mathcal{C}_0$, which is the non-negative orthant.

Recent statistical timing analysis researches have shown that a delay variable d can be represented in a compact and accurate *canonical delay model* [7]–[9]:

$$d = \mu_d + [\beta_{d,1}\beta_{d,2}\dots\beta_{d,l}] \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_l \end{bmatrix} = \mu_d + \beta_d \mathbf{f}, \tag{4}$$

where μ_d is the mean value of d, $f_1 \dots f_l$ are global and local variation sources. With some derivations [9], the slack vector can be expressed as a multivariate Gaussian distribution

$$\mathbf{s} \sim N(\mu_{\mathbf{s}}, \Sigma_{\mathbf{s}}), \tag{5}$$

where $\Sigma_{\mathbf{s}}$ is the covariance matrix of $\mathbf{s}$.

B. Slack Filtering

The dimension of $\mathbf{s}$ is $2|E|$, which can be very large for large circuits. In practice, many of the timing paths have abundant of slack and do not contribute to the timing yield loss. Therefore, it is desirable to filter out non-critical slack variables to reduce the dimension of the slack vector. We use the following criteria:

$$\frac{\mu_{s_i}}{\sigma_{s_i}} \geq p. \tag{6}$$

For s_i satisfying (6), we delete the i-th rows of $\mathbf{s}$, $\mu_{\mathbf{s}}$, $\Sigma_{\mathbf{s}}$ and the i-th column of $\Sigma_{\mathbf{s}}$. This reduces the dimension of the slack vector to a manageable size n. Alternatively, we can control n by selecting p.

C. Timing Yield Estimation

The *nominal* timing yield of the circuit is

$$\begin{aligned} \mathcal{Y}_0 &= P(\mathbf{s} \in \mathcal{C}_0) \\ &= \int_{\mathcal{C}_0} \cdots \int jpdf(s_1, s_2, \dots, s_n) ds_1 ds_2 \dots ds_n, \end{aligned} \tag{7}$$

where $jpdf(\mathbf{s})$ is the joint probability density function of $\mathbf{s}$. Since the slack variables are correlated, it is difficult to perform multi-dimensional integration analytically. We use Monte Carlo integration, an efficient method to calculate high dimensional integrals, to obtain timing yield estimations. We generate N slack vector samples according to $\mu_{\mathbf{s}}$ and $\Sigma_{\mathbf{s}}$, and calculate the nominal timing yield by

$$\mathcal{Y}_0 \cong \frac{N_0}{N}, \tag{8}$$

where N_0 is the number of samples that falls in $\mathcal{C}_0$.

Checking to see whether a sample $\mathbf{s}$ falls in $\mathcal{C}_0$ is straight-forward, we check if every element in $\mathbf{s}$ is non-negative. It is worth to note that there are other high dimensional integration methods, such as *parallelepiped*, *ellipsoid*, or *binding probability* methods, for timing yield estimation [10]. However, these methods have their restrictions and Monte Carlo integration is a competitive method even without slack filtering.

IV. Timing Yield with PST Clock Buffers

PST clock buffers can be used to redistribute path slacks among adjacent timing paths and possibly fix timing violations. We first study the effect of PST clock buffers on the timing yield, and develop a timing yield model in the presence of PST clock buffers. The optimal timing yield that can be achieved through post-silicon clock-tuning can be estimated efficiently using the derived model.

A. PST Clock Buffer and Slack Vector

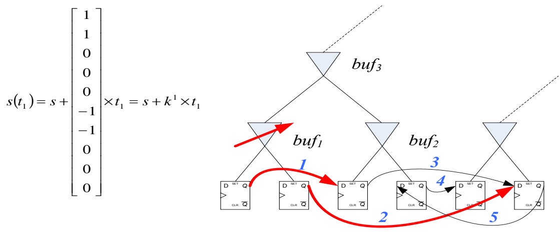


Fig. 2. Effect of changing buf_1 delay on the slack vector.

Fig. 2 shows a circuit with five timing critical paths. When buf_1 is converted to a PST clock buffer, the hold-time and setup-time slacks of paths 1 and 2 can be changed by adjusting the delay of buf_1. Let $s_1 \sim s_5$ be the hold-time slacks and $s_6 \sim s_{10}$ be the setup-time slacks of paths $1 \sim 5$, the slack vector after a change of t_1 on the buf_1 delay, $s(t_1) = s + \mathbf{k^1} t_1$, is shown in Fig. 2. Likewise, the slack vector after a change of t_2 on buf_2 delay is $s(t_2) = s + \mathbf{k^2} t_2$. Similar analysis shows that the effects of tuning the delay of an internal clock buffer can be represented by a linear combination of the effects of tuning leaf level clock buffers. For example, the slack vector after adding t_3 to the buf_3 delay, $s(t_3) = s + \mathbf{k^3} t_3 = s + (\mathbf{k^1} + \mathbf{k^2}) t_3$, is equivalent to adding t_3 to both buf_1 and buf_2 delays.

B. Tuning Vector and Buffer Filtering

Let $\mathbf{t}$ be an $R^{|B| \times 1}$ vector corresponding to the tuning amount of the B clock buffers, the slack vector after applying $\mathbf{t}$ is

$$s(\mathbf{t}) = \mathbf{s} + K\mathbf{t}, \tag{9}$$

where K is the *tuning matrix* and $\mathbf{k^i}$, the i-th column vector of K, is the *tuning vector* of the i-th clock buffer.

After slack filtering, some clock buffers are not connected to timing critical paths and their corresponding tuning vectors are zero vectors. Moreover, tuning the delay of a clock buffer can have the same effect as tuning another clock buffer. Therefore, we can filter out these clock buffers and reduce the number of candidate clock buffers for selection from $|B|$ to m. Fig. 3 shows the algorithm that produces the tuning matrix K and candidate clock buffers U. Fig. 4 illustrates the result of the algorithm on a simple circuit.

C. Parameterized and Optimal Timing Yield

With post-silicon clock-tuning, a circuit is considered functional if there exists a delay configuration $\mathbf{t}$ that can bring its slack vector to the feasible region $\mathcal{C}_0$. Alternatively, we can treat the effect of post-silicon clock-tuning as an enlargement on the feasible region. Let r_i be the tunable range of the i-th candidate clock buffer, the *parameterized* timing yield that can be achieved given the tunable range vector $\mathbf{r}$ is:

$$\begin{aligned} \mathcal{Y}(\mathbf{r}) &= P(\mathbf{s} \in \mathcal{C}(\mathbf{r})), \\ \mathcal{C}(\mathbf{r}) &= \left\{ \begin{array}{lll} \mathbf{w} & | & \mathbf{w} = \mathbf{y} - K\mathbf{t}, \\ & & y_i \geq 0, i = 1 \ldots n \\ & & r_j \geq t_j \geq -r_j, j = 1 \ldots m \end{array} \right\} \\ &= \{\mathbf{w} \mid \mathbf{w} \succeq -K\mathbf{t}, \mathbf{r} \succeq \mathbf{t} \succeq -\mathbf{r}\}, \end{aligned} \tag{10}$$

where $\succeq$ and $\preceq$ are element-wise inequalities. Note that the model (10) is applicable whether $\mathbf{r}$ is a continuous vector or a discrete vector. Therefore, the same parameterized timing yield model can be used both for PST-A and PST-N.

Procedure *SelectCandidate*
Input: Circuit graph $G(V, E, B)$, slack vector $\mathbf{s}$
Output: Tuning matrix K, candidate buffers U

Number clock buffers in *reverse* topological order
$m = 0,\ K \leftarrow \phi,\ U \leftarrow \phi$
for $i = 1 \ldots |B|$ **do**
  if buf_i is a leaf buffer **then**
    $\mathbf{v^i}_j = \begin{cases} 0 & , buf_i \text{ is not connected to } s_j \\ +1 & , buf_i \text{ drives the source(target) of hold-time(setup-time) slack } s_j \\ -1 & , buf_i \text{ drives the target(source) of hold-time(setup-time) slack } s_j \end{cases}$
    if $\mathbf{v^i} \neq 0$ **then**
      $m = m + 1,\ K \leftarrow [K|\mathbf{v^i}],\ U \leftarrow U \cup \{buf_i\}$
    end if
  else
    $\mathbf{v^i} = \sum_{b \in Child(i)} \mathbf{v^b}$
    if $\mathbf{v^i} \neq 0$ and $\mathbf{v^i} \neq \mathbf{v^b},\ \forall b \in Child(i)$ **then**
      $m = m + 1,\ K \leftarrow [K|\mathbf{v^i}],\ U \leftarrow U \cup \{buf_i\}$
    end if
  end if
end for

Fig. 3. Algorithm to identify candidate PST clock buffer locations and generate tuning matrix.

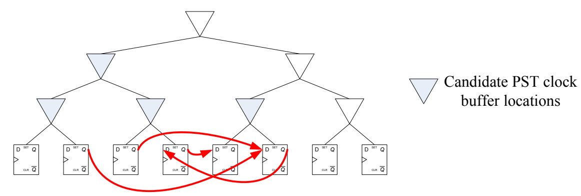


Fig. 4. Candidate PST clock buffer locations obtained by applying the *SelectCandidate* algorithm.

To check if a slack vector sample $\mathbf{s}$ is in the feasible region $\mathcal{C}(\mathbf{r})$, one needs to solve the linear feasibility program

$$\begin{array}{lll} (\mathcal{FP}) & min & 0 \\ & s.t. & -K\mathbf{t} \preceq \mathbf{s}, \\ & & \mathbf{r} \succeq \mathbf{t} \succeq -\mathbf{r}, \end{array} \tag{11}$$

and check to see if (11) is feasible. We use CLP, a high quality Simplex [11] solver of the COIN-OR project [12], to solve the feasibility problem for each slack vector sample and get the parameterized timing yield estimation. Note that although $\mathcal{Y}(\mathbf{r})$ is more costly to obtain than $\mathcal{Y}_0$, the runtime to solve 100,000 instances of the $\mathcal{FP}$ problem with $n \sim 1000$ and $m \sim 2000$ is still less than an hour on a 1.7GHz Pentium-M PC. This is due to the high efficiency of the Simplex algorithm [13].

We can obtain the *optimal* timing yield by assuming all candidate clock buffers have a $(+\infty, -\infty)$ tunable range, or

$$\begin{aligned} \mathcal{Y}_* &= P(\mathbf{s} \in \mathcal{C}_*), \\ \mathcal{C}_* &= \{\mathbf{w} \mid \mathbf{w} \succeq -K\mathbf{t}, \mathbf{t} \in R^m\}. \end{aligned} \tag{12}$$

The optimal timing yield is based on the assumption that all clock buffers are tunable and have infinite tunable ranges. Since there is a diminishing-marginal-return effect between the hardware cost and the timing yield, it is reasonable to set a target timing yield below $\mathcal{Y}_*$. We set the target timing yield as $\mathcal{Y}_t = \mathcal{Y}_0 + 0.9 \times (\mathcal{Y}_* - \mathcal{Y}_0)$ for the following discussions.

V. Total Area Minimization

In this section, we first cast the PST-A problem into a nonlinear optimization problem. We use a *simultaneous perturbation* (SP) [14], [15] algorithm to significantly reduce the time for gradient approximation of the timing yield function using only two Monte Carlo integrations. Finally, an iterative SP linesearch algorithm is proposed to solve the problem efficiently.

A. Nonlinear Optimization Formulation

We formulate the PST-A problem as a nonlinear optimization problem with simple bound constraints as below:

$$\begin{aligned} \max \quad & L_\gamma(\mathbf{r}) = \mathcal{Y}(\mathbf{r}) - \gamma \sum_{i=1\ldots m} r_i \\ s.t. \quad & r_i \geq 0, i = 1\ldots m. \end{aligned} \tag{13}$$

By choosing a positive *penalty parameter* γ, we force the tunable ranges of the candidate buffers that do not contribute to the timing yield improvement to be 'squeezed' toward zero. This formulation is similar to a typical penalty-function-based optimization that minimizes the total tunable range and a penalty term on the timing yield violation. However, it will become clear that this formulation provides benefits on selecting γ and allow us to start the optimization from a feasible solution.

The nonlinear optimization problem can be solved using linesearch algorithms. Linesearch algorithms require gradient information of the objective function. Since we don't have the analytic formula of $\mathcal{Y}(\mathbf{r})$, we need to approximate its gradient using only $\mathcal{Y}(\mathbf{r})$ evaluations. A common gradient approximation method is *finite difference* (FD). A linesearch algorithm using one-sided finite difference approximation follows

$$\mathbf{r}_{k+1} = \mathbf{r}_k + c_k \hat{g}_\gamma(\mathbf{r}_k) \tag{14}$$

$$\hat{g}_\gamma(\mathbf{r}_k) = \begin{bmatrix} \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{e^1}) - \mathcal{Y}(\mathbf{r}_k)}{b_k} - \gamma \\ \vdots \\ \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{e^m}) - \mathcal{Y}(\mathbf{r}_k)}{b_k} - \gamma \end{bmatrix}, \tag{15}$$

where c_k is the step size and b_k is the perturbation size of iteration k. $\hat{g}_\gamma(\mathbf{r}_k)$ is the gradient approximation of $L_\gamma(\mathbf{r})$ at $\mathbf{r}_k$, and $\mathbf{e^i}$ is a unit vector with 1 on the i-th element. Therefore, in each step it takes m parameterized timing yield evaluations to obtain a gradient approximation. This is too computationally expensive.

B. Simultaneous Perturbation

Recent studies have shown that it is possible to use only two function evaluations to approximate the gradient by taking a random perturbation vector $\mathbf{\Delta}_k$ [14], [15]. The gradient approximation with SP is

$$\hat{g}_\gamma(\mathbf{r}_k) = \frac{\mathcal{Y}(\mathbf{r}_k + b_k \mathbf{\Delta}_k) - \mathcal{Y}(\mathbf{r}_k)}{b_k} \begin{bmatrix} \frac{1}{\mathbf{\Delta}_{k,1}} \\ \vdots \\ \frac{1}{\mathbf{\Delta}_{k,m}} \end{bmatrix} - \gamma \mathbf{1}. \tag{16}$$

There are a few conditions that must be met in order to guarantee the convergence of a linesearch algorithm using SP. The most important ones are that b_k and c_k need to go to 0 at appropriate rates and $\mathbf{\Delta}_{k,i}$ are independent and symmetrically distributed with $E[\mathbf{\Delta}_{k,i}] = 0$ and $E[|\mathbf{\Delta}_{k,i}|^{-1}] < \infty$. A common choice of the perturbation vector $\mathbf{\Delta}_k$ is the symmetric Bernoulli ± 1 distribution. It has been shown that under mild conditions, the number of measurements of $\mathcal{Y}(\mathbf{r})$ by SP can approach $\frac{1}{m}$ of that from FD while achieving the same asymptotic mean squared error of the solution. The intuition behind SP is that the gradient approximation in (16) is an *unbiased* approximation and it contains as much information as that from a finite difference approximation. Fig. 5 illustrates the convergence of linesearch algorithms with SP and FD.

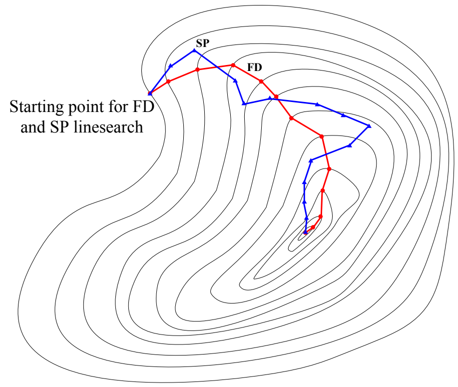


Fig. 5. Illustration of the convergence of SP and FD linesearch [14].

Procedure *Iterative*SP*Linesearch*
Input: $\mathcal{Y}(\mathbf{r})$, $\mathcal{Y}_t$, initial solution $\mathbf{r}_{init}$
Output: Final tunable range $\tilde{\mathbf{r}}$

1: Initialize $b, \eta, c, C, \pi, \epsilon, \chi, \theta$
2: $\tilde{\mathbf{r}} = \mathbf{r}_{prev} = \mathbf{r}_{init}$
3: $\gamma = \frac{\mathcal{Y}(\mathbf{r}_{init}) - \mathcal{Y}_t}{|\mathbf{r}_{init}|}$
4: **repeat**
5: /* maximize $L_\gamma(\mathbf{r})$ using SP linesearch */
6: $k = 1,\ \mathbf{r}_k = \mathbf{r}_{prev}$
7: **repeat**
8: $b_k = \frac{b}{k^\eta},\ c_k = \frac{c}{(C+k)^\pi}$
9: $\mathbf{\Delta}_k \leftarrow \pm 1$ symmetric Bernoulli random vector
10: $\mathbf{\Delta}_k = \max(\mathbf{\Delta}_k, \frac{-\mathbf{r}_k}{b_k})$ {legalization}
11: Approximate $\hat{g}_\gamma(\mathbf{r}_k)$ by (16)
12: $\mathbf{r}_{k+1} = max(0, \mathbf{r}_k + c_k \hat{g}_\gamma(\mathbf{r}_k))$ {legalization}
13: $k = k + 1$
14: **until** $|L_\gamma(\mathbf{r}_k) - L_\gamma(\mathbf{r}_{k-1})| < \epsilon$
15: **if** $\mathcal{Y}(\mathbf{r}_k) > \mathcal{Y}_t$ **then**
16: $\tilde{\mathbf{r}} = \mathbf{r}_k$
17: **end if**
18: $\mathbf{r}_{prev} = \mathbf{r}_k$
19: /* update step size (b, c) and penalty weight γ */
20: $\gamma = \chi\gamma, b = \theta b, c = \theta c$
21: **until** $\mathcal{Y}(\mathbf{r}_k) < \mathcal{Y}_t$

Fig. 6. Algorithm for total tunable range minimization using iterative SP linesearch.

C. Iterative SP Linesearch

We propose an iterative SP linesearch algorithm in Fig. 6 to solve the PST-A problem. The algorithm starts from an initial solution $\mathbf{r}_{init}$, which has a sufficiently large parameterized timing yield ($\mathcal{Y}(\mathbf{r}_{init}) > \mathcal{Y}_t$). For example, we can find a sufficiently large q such that $\mathbf{r}_{init} = q\mathbf{1}$ satisfies this condition. At the beginning of the optimization, we initialize the parameters for SP gradient approximation (b, η, c, C, π) according to the guidelines given in [15]. We let $\chi = 2$, $\theta = 0.9$ and $\epsilon = 0.001$, which controls the convergence rate of the outer iterative loop (line 4-21).

In the first iteration, we choose the penalty parameter γ according to the equation in (line 3) to ensure that the timing yield after the first iteration is still larger than $\mathcal{Y}_t$. In the following iterations, we gradually increase the penalty parameter γ and reduce the step and perturbation sizes (line 20). Within each iteration, we use SP linesearch to find the optimal solution for the given penalty parameter γ (line 5-17). The latest intermediate solution that satisfies the target yield is recorded in $\tilde{\mathbf{r}}$ (line 15-17). There are two legalization steps in the algorithm (line 10, 12). The gradient approximation given by (16) can generate unrealistically large gradients due to a small perturbation step $\Delta_{k,i}$ in the denominator caused by the legalization step (line 10). We use $max(0.5b_k, |\Delta_{k,i}|)$ as the perturbation step size in (16) to resolve this problem. Truncation on the perturbation step can introduce noise to the gradient approximation. The noise has a greater impact to the convergence of the algorithm if it occurs at the beginning of the linesearch iteration when the step size is large. Therefore, choosing $\mathbf{r}_{init}$ instead of the origin as the starting point can reduce the noise effect.

The runtime of the algorithm is dominated by the number of $\mathcal{Y}(\mathbf{r})$ evaluations, which is the same as the number of SP linesearch steps (line 7-14). We can terminate the iterative SP linesearch loop early when a certain number of $\mathcal{Y}(\mathbf{r})$ evaluations is used. For large problems, our algorithm can take less than m steps to find a tunable range vector, less than the time for a traditional FD linesearch to take the very first step.

VI. Reduction on the Number of Tunable Clock Buffers

In this section, we first analyze a greedy algorithm for solving the PST-N problem and point out its issue. We then propose a batch selection algorithm to speedup the process.

A. A Greedy Algorithm

In the PST-N problem, we are only concerned with the number of PST clock buffers used in a PST clock-tree. A digital-to-analog converter controlled PST clock buffer with $\sim 700ps$ tunable range in a $0.18\mu m$ technology has been reported in [4]. This tunable range is sufficient to counter process induced path delay and clock skew variations. Therefore, we assume that a PST clock buffer has an infinite tunable range in the PST-N problem. Under this assumption, a PST clock-tree can be represented by a *selection vector* $\mathbf{r}_{sel}$, where the tunable range $r_{sel,i}$ is ∞ for a selected buffer i, and 0 otherwise. However, to find a selection vector with a minimum number of non-zero elements (PST clock buffers) that satisfies the target timing yield is a combinatorial optimization problem.

A common choice for combinatorial optimizations is a greedy algorithm. The algorithm starts with an empty selection vector and select a buffer in each iteration until the target timing yield is achieved. In each iteration it checks the potential timing yield improvement of every unselected buffer and chooses the one that gives the maximum improvement. The major issue of the greedy algorithm is that it requires $\frac{m+(m-M+1)}{2} \times M$ parameterized timing yield evaluations, where M is the number of non-zero elements in $\mathbf{r}_{sel}$. This is unacceptable for large problems where m and M are both large. We need an algorithm that generates a good selection vector using $\sim m$ parameterized timing yield evaluations.

B. Batch Selection Algorithm

We propose a batch selection algorithm in Fig. 7 to overcome the runtime issue of the greedy algorithm. Instead of selecting one buffer at a time, we scan through all the unselected buffers and select a buffer immediately if it provides a timing yield improvement greater than a threshold value $\mathcal{Y}_{th}$ (line 6-7). The threshold value is decreased exponentially and the selection only takes a few passes to complete (line 13).

The number of $\mathcal{Y}(\mathbf{r})$ evaluations needed for our batch selection algorithm is ωm, where ω is the number of scans to achieve the target timing yield. Since the threshold for buffer selection $\mathcal{Y}_{th}$ decreases exponentially, ω is usually a small constant. Therefore, the overall runtime of our algorithm is $\frac{M}{\omega}$ times faster than the greedy algorithm.

Procedure *BatchSelection*
Input: Timing yield model $\mathcal{Y}(\mathbf{r})$, target yield $\mathcal{Y}_t$
Output: Selection vector $\mathbf{r}_{sel}$

1: $\mathbf{r} = 0,\ \mathcal{Y}_{cur} = \mathcal{Y}_0,\ \mathcal{Y}_{th} = 0.1 \times (\mathcal{Y}_* - \mathcal{Y}_0)$
2: **repeat**
3:   **for** $j = 1 \ldots m$ **do**
4:     **if** $r_j = 0$ **then**
5:       $r_j = \infty$
6:       **if** $\mathcal{Y}(\mathbf{r}) > \mathcal{Y}_{cur} + \mathcal{Y}_{th}$ **then**
7:         $\mathcal{Y}_{cur} = \mathcal{Y}(\mathbf{r})$
8:       **else**
9:         $r_j = 0$
10:       **end if**
11:     **end if**
12:   **end for**
13:   $\mathcal{Y}_{th} = 0.5 \times \mathcal{Y}_{th}$
14: **until** $\mathcal{Y}_{cur} > \mathcal{Y}_t$
15: $\mathbf{r}_{sel} = \mathbf{r}$

Fig. 7. Algorithm for batch PST clock buffer selection.

VII. Experimental Results

We implement our algorithms in C++ and test them on a 1.7GHz Pentium-M computer. We take ISCAS89 benchmark circuits and synthesize and place them using SIS [16] and Dragon [17] to obtain realistic flip-flop placements. In practice, a timing critical path is usually connected to other timing-critical paths and they usually form cycles (otherwise a critical path can be eliminated by introducing useful skew to its source or target flip-flop). We apply an iterative clock scheduling algorithm [6] to identify timing critical cycles. We then take the first $500 \sim 2000$ timing critical paths and build a parameterized timing yield model for each circuit as discussed in Section IV.

For each circuit, we generate an H-tree and assume there is a clock buffer at every branching point and terminal of the H-tree. For S9234.1, we use an eight-level H-tree (256 terminals). For the rest of the circuits, we use ten-level H-trees (1024 terminals). Only PST clock buffers are shown in the subsequent figures.

For PST-A, we compare our iterative SP linesearch algorithm (*IterSP*) with a regular design method (*Regular*) that inserts identical PST clock buffers to all terminals of the clock-tree. For PST-N, we compare our batch selection algorithm (*Batch*) with the greedy algorithm (*Greedy*) and a levelized design method (*Levelized*), which represents a current PST clock-tree design strategy that insert PST clock buffers across an entire level in the clock-tree.

A. Nominal and Optimal Timing Yield

Table I shows the parameters for the timing yield model and the nominal and optimal timing yields of each circuit. The number of timing critical paths included in the timing yield model is in the second column. The third column shows the number of candidate PST clock buffer locations. The number in the parenthesis shows the number of candidate buffers at the leaf level of the H-Tree. The timing yields are obtained by Monte Carlo integration with 100,000 samples for each circuit. The nominal, optimal and target timing yields of each circuit are listed in column four through six. As shown

Circuit	# Paths(n)	# Buf.(m)	$\mathcal{Y}_0(\%)$	$\mathcal{Y}_*(\%)$	$\mathcal{Y}_t(\%)$
S9234.1	500	149(75)	92.15	97.48	96.94
S13207.1	500	417(210)	94.56	99.44	98.95
S15850.1	1000	641(321)	92.69	99.97	99.24
S35932	1000	541(271)	54.93	99.99	95.49
S38584.1	2000	1591(796)	86.69	99.44	98.17

TABLE I

TIMING YIELD MODELS OF THE ISCAS89 BENCHMARK CIRCUITS.

Circuit	*Regular*		*IterSP*			
	Area	CPU	Area	Redu.	Steps	CPU
S9234.1	24.94	1.9m	8.51	65.9%	52	1.1h
S13207.1	68.60	2.8m	11.33	83.5%	63	3.4h
S15850.1	104.85	7.5m	10.09	90.4%	107	6.5h
S35932	101.23	42.2m	1.90	97.4%	73	44.3h
S38584.1	238.26	36.7m	17.19	92.8%	189	30.8h

TABLE II

COMPARISON ON TOTAL AREA BETWEEN A REGULAR DESIGN METHOD AND THE ITERATIVE SP LINESEARCH ALGORITHM.

in the table, post-silicon clock-tuning provides significant timing yield improvements ($5\% \sim 45\%$).

B. Total Area Improvement

We compare the total area (tunable range) achieved by *IterSP* and *Regular* in Table II. For *Regular*, we use the same tunable range for all the leaf level clock buffers and use a binary search to find the tunable range to within a $\pm 10^{-3}$ resolution. The tunable range is then multiplied by the number of leaf candidate clock buffers (the number in the parenthesis in Table I) for area estimation.

As shown in Table II, *IterSP* achieves $> 65\%$ area reduction for PST clock-trees. One of the reasons for the significant improvement is that *IterSP* assigns a tunable range to a clock buffer no larger than what is required and this greatly reduces over design. The other contributor to the large improvement is that *IterSP* distributes the total tunable range among all candidate clock buffers that locate at different levels of the clock-tree.

Fig. 8 shows the tunable range vector $\mathbf{r}_k$ for S9234.1 in the k-th SP linesearch step. As shown in the figure, the gradient approximations provided by SP efficiently guide the linesearch algorithm to supress the tunable ranges of the candidate buffers that do not contribute to significant timing yield improvements in only a small number of steps. In $\mathbf{r}_{52}$, about half of the 149 buffers are reduced almost to zero. This is because there are only 75 linearly independent tuning vectors among the 149 candidate buffers.

We found that averaging four gradient approximations for each SP linesearch step provides a better convergence rate without sacrificing too much runtime. On average, *IterSP* uses $\sim m$ parameterized timing yield estimations.

Although *IterSP* can obtain a good tunable range vector using only $\sim m$ parameterized timing yield estimations, it is still too slow for large problems such as S35932. *IterSP* has an extremely long runtime on S35932 becaues of its low nominal timing yield. As a result, we need to solve $\sim 45K$ instances of a 1000×541 linear feasibility problem for each parameterized timing yield evaluation. For large problems, it is necessary to select a small subset of candidate clock buffer locations and reduce the problem size before applying *IterSP*. Our batch selection algorithm is a good candidate for this goal.

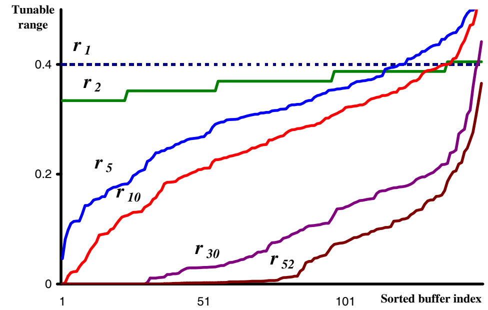


Fig. 8. The tunable range vector of S9234.1 during iterative SP linesearch.

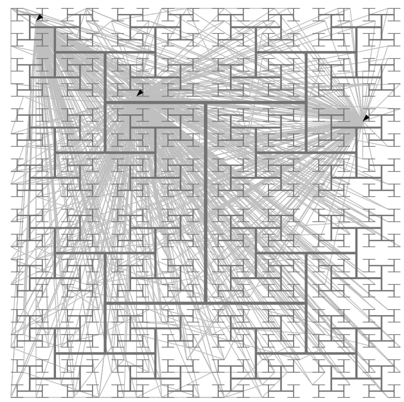


Fig. 9. The PST clock-tree of S35932. The triangles and light gray lines indicate PST clock buffer locations and timing critical paths.

C. PST Clock Buffer Count Reduction

Table III shows the number of PST clock buffers and the runtime necessary to achieve the target timing yield by *Levelized*, *Greedy* and *Batch*. To reduce the number of required PST clock buffers, *Levelized* inserts PST clock buffers at a level as close to the root node of the clock tree as possible provided the target timing yield is satisfied.

The number of PST clock buffers required to achieve the target timing yield does not necessarily depend on the size of the circuit or the number of timing critical paths included in the timing yield model. For example, S35932 only needs three PST clock buffers to achieve the target timing yield. Fig. 9 shows the PST clock-tree and the timing critical path distribution of S35932. It is clear that either the source or the target flip-flops of most of the timing critical paths are driven by one of the three clock buffers.

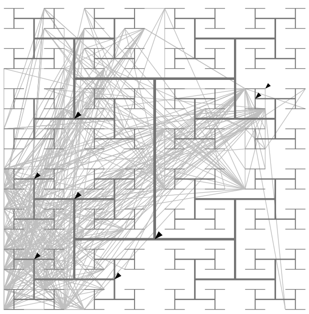
In general, *Levelized* uses PST clock buffers four times more *Greedy* and *Batch*. This is because *Levelized* only inserts one level of PST clock buffers in a clock tree. On the contrary, *Greedy* and *Batch* can utilize *hierarchical tuning* to reduce the number of PST clock buffers. Fig. 10 shows the PST clock-trees of S9234.1 generated by *Greedy* and *Batch*. On the lower left corner of the clock-trees, both algorithms generate PST clock-trees with three levels of PST clock buffers. A PST clock buffer closer to the clock root node can affect many timing paths simultaneously while a PST clock buffer closer to the clock sink nodes can adjust the timing of specific timing paths. *By allowing multiple levels of PST clock buffers, we can explore the correlation between timing critical paths and use fewer PST clock buffers to achieve the target timing yield.*

The comparison of the number of PST clock buffers used by *Greedy* and *Batch* show that *Batch* has comparable solution quality to *Greedy*. Moreover, *Batch* provides ~ 4X speedup on average.

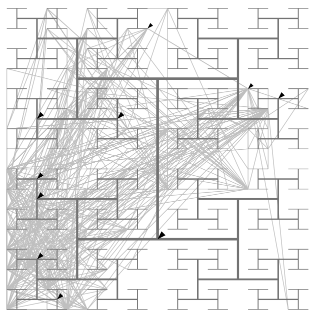
Circuit	# Candidate Buffers	# PST clock buffers					CPU Time		
		Levelized	*Greedy*	Reduction	*Batch*	Reduction	*Levelized*	*Greedy*	*Batch*
S9234.1	149(75)	32	8	75%	8	75%	1.7m	16.9m	4.6m
S13207.1	417(210)	256	16	94%	18	93%	1.9m	35.8m	6.8m
S15850.1	641(321)	128	17	87%	21	84%	13.2m	1.4h	21.6m
S35932	541(271)	16	3	81%	3	81%	1.5h	14.7h	3.1h
S38584.1	1591(796)	512	-	-	162	68%	14.3m	> 2 day	8.9h

TABLE III

COMPARISON ON NUMBER OF PST CLOCK BUFFERS AND RUNTIME AMONG THREE DESIGN METHODS.



(a) Greedy



(b) Batch Selection

Fig. 10. PST clock-trees of S9234.1 generated by *Greedy* and *Batch*.

Therefore, *Batch* is a preferred algorithm for solving large PST-N problems.

D. Runtime

The runtime of *IterSP* and *Batch* ranges from five minutes to two days for the ISCAS89 benchmark circuits. Nevertheless, the majority of the runtime is spent on Monte Carlo simulation for parameterized timing yield estimations. In practice, the runtime can be greatly reduced by parallel computing to minimize the impact on the design turn-around-time (TAT).

VIII. CONCLUSION AND FUTURE WORK

We present two optimization algorithms to solve the PST clock-tree synthesis problems. By allowing hierarchical tuning, our algorithms achieve up to a 90% area or a 90% PST clock buffer count reduction.

The paper suggests that the minimum area PST clock-tree synthesis problem for large circuits remains a difficult problem due to the lack of a closed-form timing yield model. Future researches include developing closed-form timing yield models and post-silicon clock tuning algorithms.

IX. ACKNOWLEDGEMENT

This work was partially funded by National Science Foundation under grants CCR-0093309 & CCR-0204468 and National Science Council of Taiwan, R.O.C. under grant NSC 92-2218-E-002-030.

REFERENCES

[1] Anand Rajaram, Jiang Hu, and Rabi Mahapatra. Reducing clock skew variability via cross links. In *Proceedings of the 41st annual conference on Design automation*, pages 18–23. ACM Press, 2004.

[2] Simon Tam, Stefan Rusu, Utpal Nagarji Desai, Robert Kim, Ji Zhang, and Ian Young. Clock generation and distribution for the first IA-64 microprocessor. *IEEE Journal of Solid-State Circuits*, 35(11):1545–1552, Nov 2000.

[3] Patrick Mahoney, Eric Fetzer, Bruce Doyle, and Sam Naffziger. Clock distribution on a dual-core multi-threaded Itanium-family processor. In *Digest of technical papers of the 2005 international solid-state circuits conference*, pages 292–293, 2005.

[4] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. A post-silicon clock timing adjustment using genetic algorithms. In *Digest of technical papers of the 2003 symposium on VLSI circuits*, pages 13–16, 2003.

[5] Charles E. Dike, Nasser A. Kurd, Priyadarsan Patra, and Javed Barkatullah. A design for digital, dynamic clock deskew. In *Digest of technical papers of the 2003 symposium on VLSI circuits*, pages 21–24, 2003.

[6] Jeng-Liang Tsai, DongHyun Baik, Charlie Chung-Ping Chen, and Kewal K. Saluja. A yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *Proceedings of the 2004 IEEE/ACM international conference on Computer-aided design*, pages 611–618, 2004.

[7] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 621, Washington, DC, USA, 2003. IEEE Computer Society.

[8] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proceedings of the 41st annual conference on Design automation*, pages 331–336, New York, NY, USA, 2004. ACM Press.

[9] Lizheng Zhang, Weijen Chen, Yuhen Hu, and Charlie Chung-Ping Chen. Statistical timing analysis with extended pseudo-canonical timing model. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 952–957, Washington, DC, USA, 2005. IEEE Computer Society.

[10] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. H. J. M. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *Proceedings of the 40th conference on Design automation*, pages 932–937, New York, NY, USA, 2003. ACM Press.

[11] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, N.J., 1963.

[12] Robin Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[13] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305, 2001.

[14] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19(4):482–492, 1998.

[15] James C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronics Systems*, 34(3):817–823, July 1998.

[16] Ellen M. Sentovich, Kanwar Jit Singh, Luciano Lavagno, Cho Moon, Rajeev Murgai, Alexander Saldanha, Hamid Savoj, Paul R. Stephan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. *Electronics Research Laboratory, Memorandum No. UCB/ERL M92/41*, 1992.

[17] Maogang Wang, Xiaojian Yang, and Majid Sarrafzadeh. Dragon2000: standard-cell placement tool for large industry circuits. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 260–263. IEEE Press, 2000.