Clock Tree Synthesis Considering Slew Effect on Supply Voltage Variation

CHUN-KAI WANG, YEH-CHI CHANG, HUNG-MING CHEN, and CHING-YU CHIN,
National Chiao Tung University

This work tackles a problem of clock power minimization within a skew constraint under supply voltage variation. This problem is defined in the ISPD 2010 benchmark. Unlike mesh and cross link that reduce clock skew uncertainty by multiple driving paths, our focus is on controlling skew uncertainty in the structure of the tree. We observe that slow slew amplifies supply voltage variation, which induces larger path delay variation and skew uncertainty. To obtain the optimality, we formulate a symmetric clock tree synthesis as a mathematical programming problem in which the slew effect is considered by an NLDM-like cell delay variation model. A symmetry-to-asymmetry tree transformation is proposed to further reduce wire loading. Experimental results show that the proposed four methods save up to 20% of clock tree capacitance loading. Beyond controlling slew to suppress supply-voltage-variation-induced skew, we also discuss the strategies of clock tree synthesis under variant variation scenarios and the limitations of the ISPD 2010 benchmark.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids

General Terms: Algorithms, Design

Additional Key Words and Phrases: Clock tree optimization, slew, voltage variation, robust design

ACM Reference Format:
Chun-Kai Wang, Yeh-Chi Chang, Hung-Ming Chen, and Ching-Yu Chin. 2014. Clock tree synthesis considering slew effect on supply voltage variation. ACM Trans. Des. Autom. Electron. Syst. 20, 1, Article 3 (November 2014), 23 pages.
DOI: http://dx.doi.org/10.1145/2651401

1. INTRODUCTION

Clock network costs the most power for a synchronous design and directly affects circuit speed. With shrinking technology, variations result from manufacturing, the operating environment, and even analysis [Blaauw et al. 2008]. These variations require more guard band in the design process. In addition, to achieve lower power dissipation, a voltage scaling technique is broadly adopted in modern design, which introduces severe voltage variation.

One way to reduce clock skew uncertainty is improving the delay correlation between clock paths by using a multiple-driving-paths clock network such as mesh [Restle et al. 2001; Xiao et al. 2010], cross link [Rajaram et al. 2006; Mittal and Koh 2011], and multilevel tree [Lee and Markov 2011]. However, this work adopts the other way, that is, to reduce skew uncertainty by reducing path delay variability. Therefore we focus on the network structure of the tree.

To reduce voltage-variation-induced path delay variability, previous works [Shih et al. 2010; Bujimalla and Koh 2011] minimize the number of buffer stages by full-filling

---

A preliminary version of this article was presented at the ISPD 2012 Conference [Chang et al. 2012].
Authors' addresses: C.-K. Wang (corresponding author), Y.-C. Chang, H.-M. Chen, and C.-Y. Chin, Department of Electronics Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan; email: oldkai.ee90@gmail.com.

DOI: http://dx.doi.org/10.1145/2651401

buffers. However, the mapping from voltage variation to delay variation depends on the buffer's input signal transition. Slow transition amplifies supply-voltage-variation-induced path delay variation and skew uncertainty. In this work, the slew effect is considered so that a clock buffer insertion and wire sizing method can efficiently reduce clock latency variation and skew uncertainty.

The contributions of this work are as follows.

—We formulate buffer insertion and wire sizing on a symmetric clock tree as a mathematical programming problem, which ensures the optimality.
—The slew effect on voltage variation to delay variation is considered by a NonLinear Delay Model (NLDM)-like cell delay variation model.
—A technique projecting a symmetric tree to an asymmetric tree is proposed to further reduce clock wire loading.
—We discuss the strategies of Clock Tree Synthesis (CTS) for different variation scenarios and the limitations of the ISPD 2010 benchmark using Sze [2010].

The organization of this article is as follows. Section 2 introduces the problem formulation, our observation/motivation on the slew effect, and the overall flow in this work. Section 3 introduces a global optimization for the problem involving buffer insertion and wire sizing on a symmetric tree. Section 4 introduces a local optimization that projects a symmetric tree solution to an asymmetric tree. Section 5 presents experimental results, the discussion of CTS strategies under different variation scenarios, and the limitations of the ISPD 2010 benchmark.

2. PRELIMINARIES

In this section, we first review the problems of the ISPD 2010 benchmark and then deliver our observation about the slew effect and the overall flow optimizing a clock tree.

2.1. Review of ISPD 2010 Problem

To achieve a low-power and robust clock network, ISPD held a High-Performance Clock Network Contest in 2009 and 2010 [Sze et al. 2009; Sze 2010]. To reflect the variation effect realistically, instead of the clock latency range used in 2009, which is the maximum difference of clock arrival time of arbitrary sink pairs by two different supply voltages, ISPD 2010 measured the skew using a Monte Carlo method with the variation source of wire dimension and supply voltage. The details of the ISPD 2010 problem are as follows.

A synchronous circuit layout with edge-triggered flip-flops as sinks of the clock network is the starting input.

Given.

(1) a set of sinks $S = \{s_1, s_2, \ldots, s_n\}$ with physical position and capacitance loading,
(2) a clock source s_0 and its position,
(3) a buffer library,
(4) a wire library,
(5) two variation sources: the width of a wire segment varies uniformly in $\pm 5\%$, and the voltage source of a buffer varies uniformly in $\pm 7.5\%$,
(6) a set of placement blockages $B = \{b_1, b_2, \ldots, b_m\}$,
(7) a $W \times H$ layout region, and
(8) a Local Clock Skew (LCS) distance: a skew is negligible if the distance of its sink pair is larger than the LCS distance.

Objective.
Minimize the capacitance loading of a clock network.

Constraints.

(1) no 95% LCS violation: only max 5% LCS (in 500 times Monte Carlo simulations) can exceed a skew limit,
(2) no slew-rate violation, and
(3) no buffer overlaps a blockage.

The blockages are placement blockage, and clock routing can go through them. However, the clock buffers cannot be placed on the blockages. LCS is the worst local clock skew for a sample of NGSPICE simulation under wire dimension variation and supply voltage variation. To evaluate the robustness of a clock network, 95% LCS is measured. A clock network with lower value of 95% LCS is more robust, and 95% LCS is a value that cuts off 95% samples' LCS with the rest 5%. In the ISPD 2010 benchmark, the evaluator performs 500-sample Monte Carlo to derive 95% LCS of a clock network. In general, max loading and transition speed are defined in the cell library for signal integrity, and the ISPD 2010 benchmark defines a slew constraint that the slew transition in any position of a clock network must be faster than 100ps.

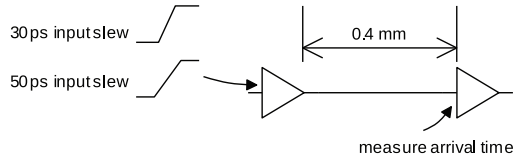
2.2. Our Observation on Slew Effect and Overall Flow

To reduce voltage-variation-induced skew, previous works [Bujimalla and Koh 2011; Shih and Chang 2010; Shih et al. 2010] minimized the number of buffer stages by full-filling buffers, or minimizing nominal clock latency by assuming that latency variation is proportional to nominal latency [Lee et al. 2010]. In Bujimalla and Koh [2011], the authors assumed that, for a buffer stage, voltage variation to delay variation is constant. However, the influence of voltage variation to delay variation depends on the slew rate; slow input slew would amplify voltage variation to delay variation of a buffer stage. Figure 1 shows the slew effect for the case of a single-buffer stage by two input slews, 30ps and 50ps. Regardless of a rising or falling transition, the 30ps input slew has a more compact histogram of arrival time. The reason is that a faster input slew reduces the fuzziness of a gate's switching time, as shown in Figure 2.

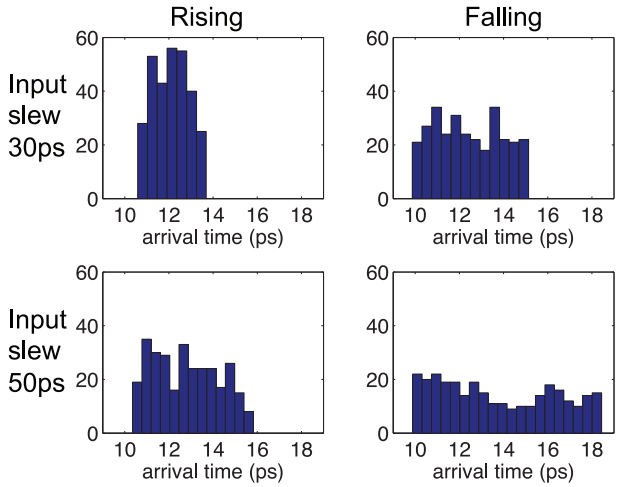
The slew effect is also experimented as shown in Figure 3 in cases of multiple-buffer stages. A slew of 0.4mm buffer distance is about 30ps and a slew of 0.9mm buffer distance is about 50ps. Results show that minimizing buffer levels does not sufficiently minimize voltage-variation-induced skew, and a smaller clock latency does not guarantee less clock latency variation.

Although the example demonstrates that controlling slew is effective to reduce voltage-variation-induced skew, inserting buffers with a slew consideration is still a problem. Conventional Ginneken's buffer insertion [van Ginneken 1990] minimizes nominal delay by dynamic programming (DP) from sinks toward the source; however, it is not suitable to minimize delay variation because, when minimizing delay variation, slew has to be considered. Nevertheless, slew is propagated from predecessor buffer stages that are unknown during DP bottom up. Hu et al. [2007] proposed a slew-constrained buffer insertion based on Ginneken's method. But applying this method induced another problem: which value should be used as the constraint? In addition, it gives up more solution space than only applying a single value as the slew constraint.

We propose a two-stage overall flow shown in Figure 4. In the first stage, we solve buffer insertion and wire sizing for a symmetric tree. There are two buffer insertion methods in first stage: (1) length-based buffer insertion and (2) mathematical-programming-based buffer insertion and wire sizing. Either one of the two buffer insertion methods will be selected. The length-based buffer insertion runs fast, but



(a) experimental setting of single stage



(b) experimental result of single stage

Fig. 1. Slew effect in single-buffer stage. (a) Two ramp input signals with different slew rate 30ps and 50ps are experimented to drive a buffer stage; (b) Monte Carlo simulations are performed by the voltage variation setting of the ISPD 2010 benchmark. The histogram shows that the slower input slew amplifies the supply-voltage-variation-induced delay variation.

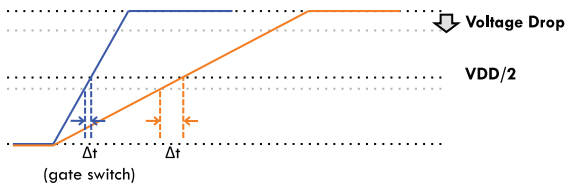
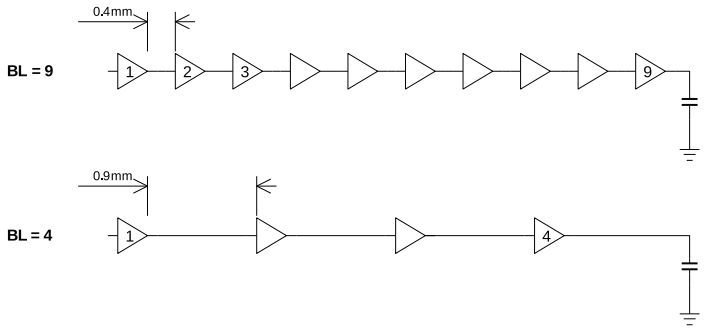


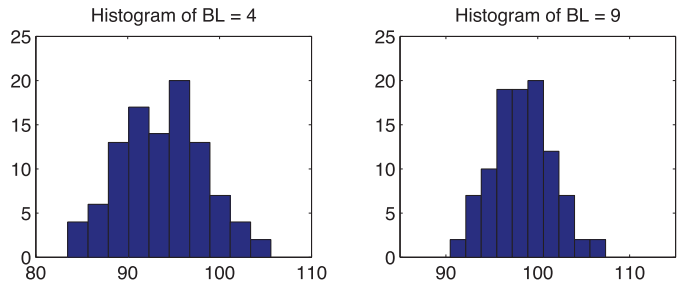
Fig. 2. Consider a gate switch when the input signal is VDD/2. A gate of slow input slew has more uncertainty than that of sharp input slew.

the mathematical-programming-based buffer insertion and wire sizing explores better solutions. The results of the first stage could be viewed as a final solution, or could be the input of the second stage. The second stage further saves wire loading by projecting a symmetric tree to an asymmetric tree. The two-stage overall flow is a heuristic that the first stage sacrifices solution space to facilitate global optimization, and the second stage searches unexplored solution space of the asymmetric structure to improve quality.

The two-stage method can avoid some difficulties when a process directly attempts to design buffer stages for an asymmetric tree. First, the fine-tuning for an asymmetry tree is a complex process. Assuming a skew estimation reports that variation-induced skew is too large for an asymmetry tree, a process may adjust the buffer position, buffer size, and wire size to reduce clock latency variation. However, at the same time,



(a) experimental setting of multiple stages



(b) experimental result of multiple stages

Fig. 3. Slew effect in multiple-buffer stages. Two buffer insertions drive the same loading. Their costs of buffer loading are the same. (a) One is 9 buffer stages with 12x inv-1, and the other is 4 buffer stages with 27x inv-1; (b) variation of arrival time of 9 stages is less than that of 4 stages. The experiment shows: (1) minimizing the number of buffer stages results in slow slew and may enlarge timing uncertainty; (2) a smaller latency does not guarantee less latency variation.

these adjustments on an asymmetry clock tree always generate new nominal skew. The process of rebalancing nominal skew and that of reducing clock latency variation are coupled to each other, which is complex and time consuming. Second, it is difficult to analyze variation-induced skew for an asymmetric tree because the paths from the clock source to sinks are all different. In contrast, the identical path of a symmetric tree can facilitate the analysis of path delay variation. Third, it is difficult to design a slew rate, to design buffer stages, and to ensure a valid variation-induced sKew at the same time for an asymmetry tree. Assuming that the slew of a node has been planned, to achieve this plan, it needs to know the driving buffer of this node and the input slew of the driving buffer; therefore, it must know every stage of input slew and buffer size. Nevertheless, it will not be known whether the plan of buffer insertion and wire sizing is able to completely drive an asymmetric tree until a Deferred Merging Embedding (DME) bottom-up phase is finished. So the process becomes that of first guessing a buffer insertion and wire sizing solution and then performing DME to verify whether the solution is feasible or not, and a method guessing a new solution is needed when the current solution is unfeasible.

3. GLOBAL OPTIMIZATION ON A SYMMETRIC TREE

To facilitate global optimization of CTS with a slew effect consideration, in the first stage we sacrifice solution space, only considering the symmetric structure. We explain the first stage in Section 3.1 by introducing a symmetric tree generation. Section 3.2 then introduces a skew estimation that will be utilized in buffer insertion and wire

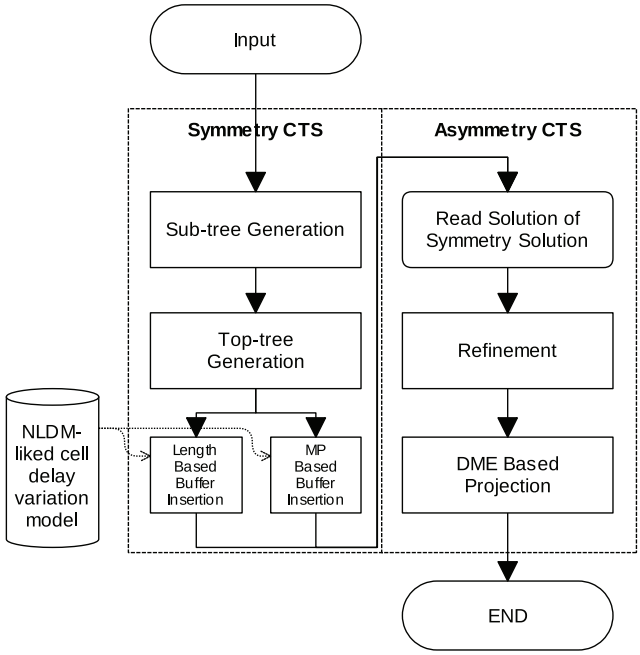


Fig. 4. Overview of optimizing a clock tree. The first stage is a global optimization by the symmetric structure, and the second stage is local optimization that transforms a symmetric tree to an asymmetric tree. Two buffer insertion methods are proposed, and either one of two methods would be selected in the first stage. The slew effect is considered by an NLDM-like cell delay variation model.

sizing later. Two types of buffer insertion and wire sizing are introduced. Section 3.3 introduces a length-based buffer insertion and wire sizing that generates quality solutions in fast runtime. And Section 3.4 introduces a mathematical-programming-based buffer insertion and wire sizing that applies a mathematical programming solver to boost solution performance.

3.1. The Symmetric Tree

We adopt a symmetric tree generation method proposed by Shih et al. [2010]. A weakness in nature of the symmetric structure is that it costs longer wire length to maintain symmetry. If we adopt a traditional H-tree, the longer wire length would need more buffer stages to drive the tree. Consequently, it costs more wire and buffer capacitance loading, and variation-induced skew increases. Second, when the gap between a symmetric and an asymmetric tree is large, the projection from symmetry to asymmetry would be less accurate. Shih et al. [2010] proposed a method to minimize the wire length of a symmetric tree, which is adopted in this work as the starting point. The difference between Shih et al. [2010] and this work is that Shih et al. [2010] adopt a symmetric tree to drive a mesh, but this work adopts a symmetric tree to drive a set of subtrees. These subtrees are all in asymmetric structure. Before applying the symmetric tree generation by Shih et al. [2010], we have to generate a set of subtrees.

The set of subtrees must have an identical driving buffer so that the top-level symmetric tree can see an identical loading. The other parameter to be decided is the number of subtrees; since this work applies a symmetric tree in binary structure, the number of subtrees must be a power of two.

ALGORITHM 1: *subtreeGeneration*(*allsinks*, *B*)

Input: *allsinks* and buffer library *B*
Output: driving buffer *b* and the number of sub-trees *n*

```
1  sort buffer in B in increase order of size;
2  min_cost ← inf;
3  for b′ ∈ B do
4  |  V ← generating a set of sub-trees by b′, using DME;
5  |  n′ ← 2^⌈log‖V‖⌉;
6  |  cost ← power dissipation of b′ and n′;
7  |  if min_cost > cost then
8  |  |  min_cost ← cost;
9  |  |  b ← b′;
10 |  |  n ← n′;
11 |  end
12 end
13 return b and n
```

The subtree generation steps are listed in Algorithm 1. To decide the number of subtrees and the type of their driving buffer, we sweep buffer types in the library to generate sets of subtrees by DME. If the number generated by the procedure is not a power of two, the one closest to and larger than the generated one will be selected. And each set has a cost for its power dissipation, the lowest cost one of which would be selected for the number of subtrees and the driving buffer.

After Algorithm 1 returns the target number of subtrees and the driving buffer, these two values are used as the arguments for the DME function. Then the DME function uses the driving buffer to bottom-up subtrees and terminates when the number of subtrees is equal to the target.

3.2. Skew Estimation

Skew estimation can prevent keeping too much guard band. An asymptotic approximation for the mean and variance of clock skew is proposed by Kugelmass and Steighlitz [1990] as

$$E(skew) = \sigma \left[\frac{4 \ln N - \ln \ln N - \ln 4\pi + 2C}{(2 \ln N)^{1/2}} + O\left(\frac{1}{\log N}\right) \right], \tag{1}$$

$$Var(skew) = \frac{\sigma^2}{\ln N} \frac{\pi^2}{6} + O\left(\frac{1}{\log^2 N}\right), \tag{2}$$

where σ is the standard deviation of clock latency, N the number of sinks, and $C(= 0.05772\ldots)$ is Euler's constant. Bujimalla and Koh [2011] applied (1) and (2) with an assumption that skew is a normal distribution to estimate $95\%skew$:

$$95\%skew = \left[E(skew) + 2 \times Var(skew) \right]. \tag{3}$$

To utilize (1), (2), and (3) in this work, computing σ is needed:

$$\sigma^2 = \sum_{i=0}^{n} {\sigma_i}^2 + \sum_{i=0}^{n-1} (\rho_R e_{Ri} + \rho_F e_{Fi}) \sigma_i \sigma_{i+1}, \tag{4}$$

$$(e_{Ri}, e_{Fi}) = \begin{cases} (0,1), & \text{if falling input} \\ (1,0), & \text{if rising input} \end{cases}, \tag{5}$$

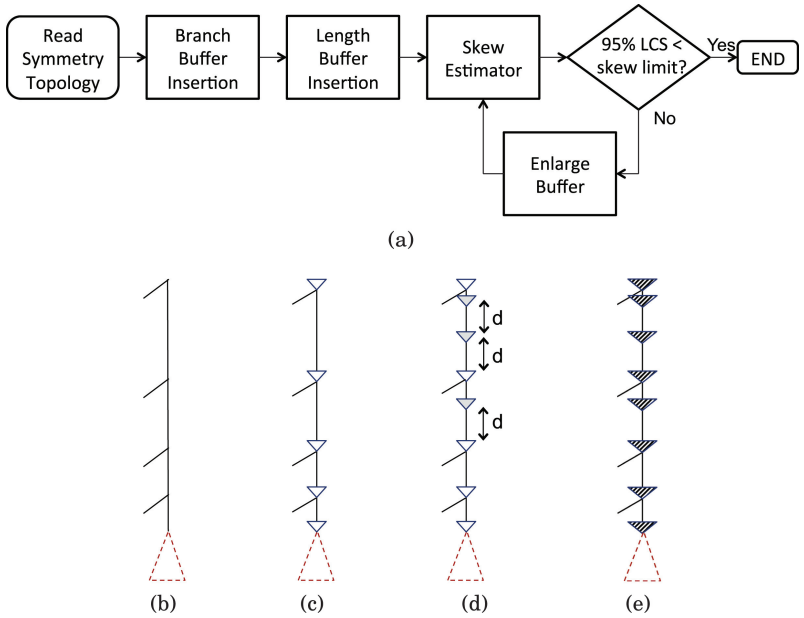


Fig. 5. Length-based buffer insertion. (a) Flow of length-based buffer insertion; (b) a symmetric topology is adopted; (c) inserting buffers on all branches; (d) inserting buffers by a distance d; (e) enlarging the buffer size.

where σ_i is the standard deviation of a buffer-stage delay, ρ_R/ρ_F the correlation coefficient for a rising/falling transition before a falling/rising transition, and e_R/e_F denotes that the buffer-stage input transition is rising/falling. σ_i is looked up by an NLDM-like cell delay variation model. In this work, the NLDM-like cell delay variation model is extracted by SPICE simulations and Response Surface Model (RSM) fitting [NIST 2012]. Parameters used to look up σ_i are about the input signal transition and output RC network. The details of the parameters to look up σ_i will be given in Section 3.4.3.

3.3. Length-Based Buffer Insertion

Our first buffer insertion method is a length-based one [Alpert and Devgan 1997]. The length-based method inserts buffers by a constant distance d. To minimize supply-voltage-variation-induced skew, d is decided by experiments of different distances on a long wire. According to Monte Carlo simulations, the distance of minimum delay variation is defined as d.

Figure 5 illustrates the overall flow of the length-based buffer insertion in this work. We first insert buffers on branches and then insert buffers by d from sinks to root. This ensures that the loading of one buffer stage is not particularly larger than any of the other buffer stages. If skew estimation reports a skew violation, we enlarge the buffer. If all types of buffer cannot satisfy the skew constraint, the buffer of smallest variation-induced skew would be used.

3.4. Mathematical Programming Buffer Insertion and Wire Sizing

The other buffer insertion and wire sizing is based on mathematical programming, in which the objective is to minimize power dissipation and the two constraints are skew and slew. Sections 3.4.1 to 3.4.3 introduce the programming variables, the objective formulation, and the constraint formulations. Section 3.4.4 enhances performance by

Table I. Notations Used in Mathematical-Programming-Based Buffer Insertion Wire Sizing

Notation	Description
v_i	number of buffer stages in i_{th} level
b_{ijk}	binary variable, buffer size of j_{th} buffer stage in i_{th} level is k
w_{ijk}	binary variable, wire size of j_{th} buffer stage in i_{th} level is k
η_i	lower bound of buffer stages in i_{th} level
ξ_i	upper bound of buffer stages in i_{th} level
d_{max}	maximum distance between two buffer stages
d_{min}	minimum distance between two buffer stages
L_i	wire length of i_{th} level
l_{ij}	wire length of j_{th} buffer stage in i_{th} level
P_i	power cost of i_{th} level
B_{ij}	power cost of buffer of j_{th} buffer stage in i_{th} level
W_{ij}	power cost of wire of j_{th} buffer stage in i_{th} level
β_i	power cost of buffer size i
ω_i	power cost of wire size i
σ	standard deviation of clock latency
σ_{ij}	standard deviation of j_{th} buffer stage delay in i_{th} level
$slew'_{ij}$	terminal slew of j_{th} buffer stage in i_{th} level
$slew_{ij}$	input slew of j_{th} buffer stage in i_{th} level
e_{ij}	binary variable, input signal is rising/falling for j_{th} buffer stage in i_{th} level
h_{ij}	binary variable, j_{th} buffer stage in i_{th} level is on branch
b_{ij}	buffer size of j_{th} buffer stage in i_{th} level
b'_{ij}	next stage buffer size of j_{th} buffer stage in i_{th} level
w_{ij}	wire size of j_{th} buffer stage in i_{th} level
y_{ij}	binary variable, j_{th} buffer stage in i_{th} level is realized
z_{ij}	binary variable, j_{th} buffer stage in i_{th} level is the last buffer stage

reducing the complexity of slew propagation. All notations used in the mathematical formulations are listed in Table I.

3.4.1. Programming Variables. Figure 6 illustrates the way that programming variables indicate a solution. In the beginning, we insert buffers at every branch, which levels the mathematical programming model and simplifies the RC topology for each buffer stage. For level i, three programming variables describe a solution: v_i denotes the number of buffer stages, b_{ijk} that the buffer size of the j_{th} stage is equal to type k, and w_{ijk} that the wire size of the j_{th} stage is equal to type k.

$$v_i \in N \tag{6}$$

$$b_{ijk} \in \{0, 1\} \tag{7}$$

$$w_{ijk} \in \{0, 1\} \tag{8}$$

Constraints on b_{ijk} and w_{ijk} ensure the solution's uniqueness, and v_i is bounded to reduce the solution space.

$$\eta_i \leq v_i \leq \xi_i \tag{9}$$

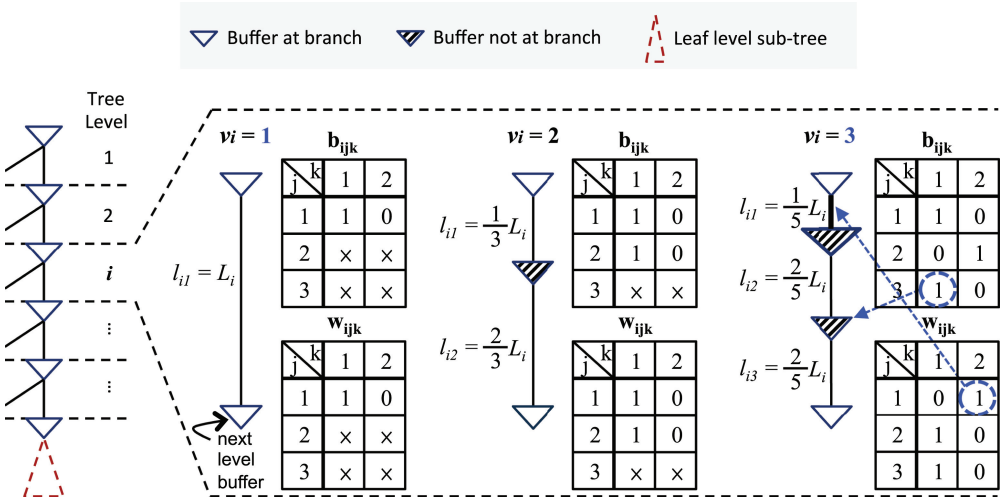


Fig. 6. In mathematical-programming-based buffer insertion and wire sizing, buffers are inserted on branches to level the problem. Three programming variables include: v_i, denoting the number of buffer stages in the i_{th} level; b_{ijk}, denoting that the size of the j_{th} buffer is equal to k, and w_{ijk}, denoting that the size of the j_{th} wire is equal to k. l_{ij} is the wire length of the j_{th} stage in the i_{th} level and is defined by level wire length L_i and v_i. In this example, v_i is bounded by [1,3]. And in the example of $v_i = 3$, 3 buffer stages in the i_{th} level, $b_{i31} = 1$ represents the 3rd-stage buffer size is equal to type-1 (a smaller buffer), and $w_{i12} = 1$ represents the 1st-stage wire size is equal to type-2 (a wider wire width).

$$\sum_{k=1}^{|b|} b_{ijk} = 1 \tag{10}$$

$$\sum_{k=1}^{|w|} w_{ijk} = 1 \tag{11}$$

To balance loading for buffer stages, buffers in a level are inserted uniformly. l_{ij} denotes the wire length of the j_{th} buffer stage. It is defined by level wire length L_i and v_i:

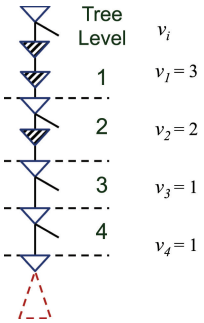
$$l_{ij} = \begin{cases} \frac{L_i}{(2v_i - 1)}, & \text{if j} = 1 \\ \frac{L_i}{(2v_i - 1)} \times 2, & \text{otherwise.} \end{cases} \tag{12}$$

Since the stage of $j = 1$ is on a branch, compared to other stages, it has double capacitance loading per unit stage wire length. We make its wire length half of other stages to balance loading.

3.4.2. Power Formulation. Total power is the summation of all levels' buffer loading and wire loading:

$$power = \sum_{i=1}^{n} 2^{i-1} P_i, \tag{13}$$

$$P_i = B_{i,1} + 2\sum_{j=2}^{v_i} B_{ij} + 2\sum_{j=1}^{v_i} W_{ij}, \tag{14}$$



y_{ij}

i \ j	1	2	3
1	1	1	1
2	1	1	0
3	1	0	0
4	1	0	0

z_{ij}

i \ j	1	2	3
1	0	0	1
2	0	1	0
3	1	0	0
4	1	0	0

Fig. 7. Boolean variables y_{ij} and z_{ij}. In the example of four levels, each level has maximum of 3 buffer stages and minimum of 1 buffer stage. A true y_{ij} denotes that the corresponding buffer stage is realized, and a true z_{ij} denotes that the corresponding buffer stage is the last buffer stage in the corresponding level. They are utilized to formulate skew and slew constraints.

$$B_{ij} = \sum_{k=1}^{|b|} \beta_k \times b_{ijk}, \tag{15}$$

$$W_{ij} = \sum_{k=1}^{|w|} l_{ij} \times \omega_k \times w_{ijk}, \tag{16}$$

where B_{ij} denotes the buffer capacitance on the j_{th} stage in the i_{th} level, β_k is the buffer capacitance of buffer size k, and where W_{ij} denotes the wire capacitance, and ω_k is the wire capacitance of wire size k.

3.4.3. Skew and Slew Constraint. We rewrite (4) as

$$\sigma^2 = \sum_{ij} y_{ij} {\sigma_{ij}}^2 + \sum_{ij} (\rho_R e_{Rij} + \rho_F e_{Fij}) \sigma_{ij} \sigma'_{ij}. \tag{17}$$

Substituting σ in (1), (2), and (3) formulates the skew constraint. In (17), a boolean variable y_{ij} is introduced:

$$y_{ij} = \begin{cases} 1, & \text{if } v_i \geq j \\ 0, & \text{otherwise} \end{cases}, \tag{18}$$

y_{ij} denotes that the j_{th} buffer stage in the i_{th} level is realized. For example, in Figure 7, the upper bound of the number of buffer stages in the second level of the tree is three, but v_2 is set to 2. Therefore, the third-stage buffer is not realized ($y_{23} = 0$).

To look up σ_{ij} by the NLDM-like cell delay variation model requires

$$\sigma_{ij} = g(e_{ij}, slew_{ij}, h_{ij}, w_{ij}, l_{ij}, b'_{ij}, b_{ij}), \tag{19}$$

$$slew'_{ij} = f(e_{ij}, slew_{ij}, h_{ij}, w_{ij}, l_{ij}, b'_{ij}, b_{ij}). \tag{20}$$

The parameters are:

—rise/fall transition e_{ij};
—input slew $slew_{ij}$;
—output branch/unbranch h_{ij};
—output wire size w_{ij};
—output wire length l_{ij};
—next stage buffer input capacitance b'_{ij}; and
—current stage buffer b_{ij}.

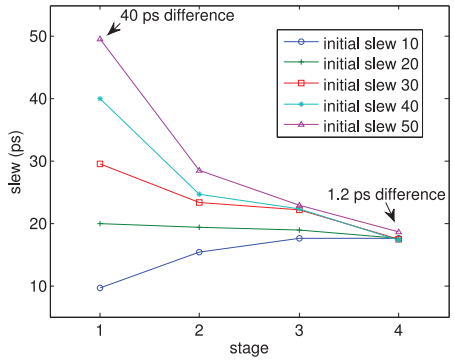


Fig. 8. Slew of a node is dominated by its near predecessors. After propagating through three buffer stages, the difference in slew decreases from 40ps to 1.2ps.

$slew'_{ij}$ is the terminal slew value of the j_{th} stage in the i_{th} level and is looked up in the same manner.

3.4.4. Enhancement on Efficiency. It could be seen in (19) and (20) that to look up σ_{ij}, input slew $slew_{ij}$ is required, and a propagation from source to sinks for slew must be performed first. To propagate slew across levels, a boolean variable z_{ij} is utilized which denotes that the j_{th} buffer stage is the last buffer stage in the i_{th} level:

$$z_{ij} = \begin{cases} 1, & \text{if } v_i = j \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

Then the input slew of the first buffer stage for all levels can be derived by

$$slew_{i+1,1} = \sum_{k=\eta_i}^{\xi_i} z_{ik} slew'_{ik}. \tag{22}$$

However, the slew propagation across levels results in a problem, that is, $slew_{i+1,1}$ grows exponentially by $(\xi_i - \eta_i + 1)$ times per level.

By a property of slew that it is dominated by near predecessors, neglecting far predecessors only affects accuracy little. An example in Figure 8 shows that, after three-stage slew propagation, the difference decreases from 40ps to 1.2ps only. This slew deviation corresponds to 3% deviation of a stage delay variation. This work adopts three-stage slew propagation as shown in Figure 9, and the formulation is

$$slew'_{ij} = slew'^{(3)}_{ij}, \tag{23}$$

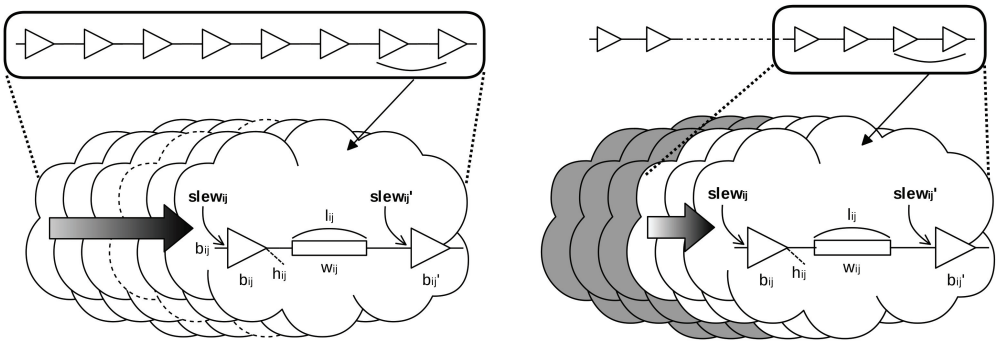
$$slew'^{(k)}_{ij} = f\left(slew_{ij}{}^{(k-1)}, \ldots\right), \tag{24}$$

$$slew'^{(0)}_{ij} = slew'^{(0)}, \tag{25}$$

where $slew'^{(k)}_{ij}$ means that the leaf terminal slew of the j_{th} buffer stage in the i_{th} level is calculated by k-stage slew propagation, $slew'^{(0)}$ is a user-defined parameter, and $f(.)$ in (24) is equivalent to (20).

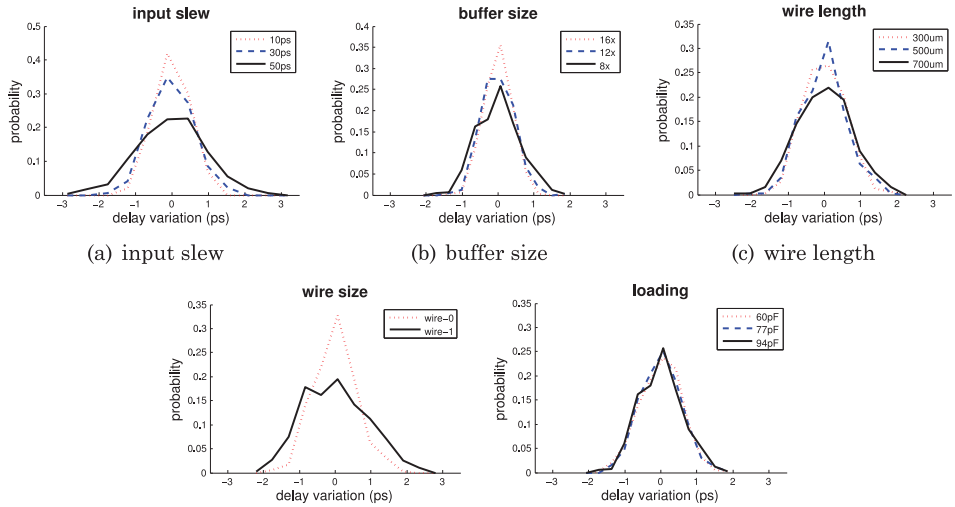
4. ASYMMETRIC CLOCK TREE

After global optimization, the second stage relaxes the constraint of symmetric structure. To reduce wire loading, a symmetric tree from the first stage is projected onto an asymmetric tree. In the projection, the skew performance needs to be preserved.



(a) slew propagation from clock source to sinks (b) slew propagation in local stages

Fig. 9. For mathematical programming, slew propagation from the clock source results in exponential growth with the number of tree levels as shown in (a). Since slew is dominated by near predecessors, its efficiency can be enhanced by only local propagation as shown in (b).



(a) input slew (b) buffer size (c) wire length

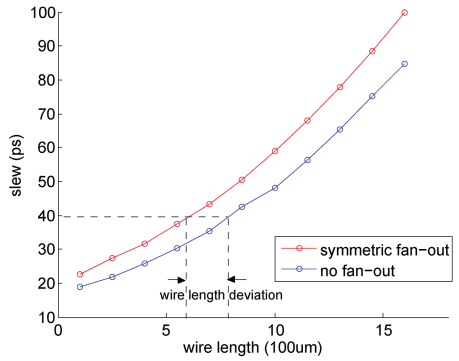
(d) wire size (e) loading

Fig. 10. Parameters affect variation of a buffer-stage delay.

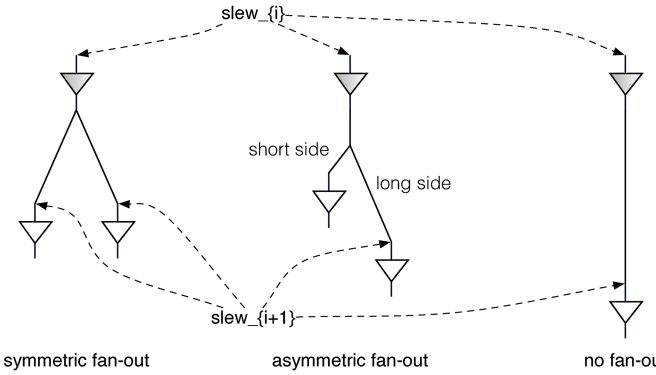
The idea to preserve skew performance is to control the delay variation of each buffer stage. As each buffer stage in an asymmetric tree is well controlled, the total clock latency variation and the variation-induced skew would be equal to the symmetric tree. Such a buffer-stage delay variation control is due to three parameters, namely slew, buffer type, and wire size, which are introduced in Section 4.1. A solution refinement between the symmetric and the asymmetric tree synthesis is introduced in Section 4.2, which addresses the weakness of the symmetric tree solution to reduce the number of inserted buffers in the asymmetric tree. Based on DME, the asymmetric tree synthesis controlling the variation for each buffer stage is introduced in Section 4.3.

4.1. Performance Preservation of Skew

By observing the NLDM-like cell delay variation model, we know the manner in which a factor affects delay variation. Figure 10 is the part of the raw data of the NLDM-like cell delay variation model that delivers our observation on the ISPD 2010 benchmark.



(a) correlation between slew and wire length for a buffer stage



(b) when transferring a symmetric tree to an asymmetric tree, slew of a buffer stage is preserved.

Fig. 11. When transferring a symmetric to an asymmetric tree, the extreme condition for a buffer stage changing is from symmetric fanout to no fanout; however, the wire length deviation is bounded because of the correlation between slew and wire length.

Input slew, buffer size, wire size, and wire length strongly influence the delay variation of a buffer stage; however, the asymmetric tree synthesis of this work neglects wire length. The reason is that wire length and slew are highly correlated, for example, in Figure 11(a), when the slews of two buffer stages in series are constrained, the wire length deviation is limited. The effect of wire length neglect will be discussed in the next paragraph. Consequently, in the asymmetric tree synthesis of this work, delay variation of a buffer stage is controlled by slew, buffer size, and wire size. An example projecting a buffer stage from a symmetric tree to an asymmetric tree is illustrated in Figure 11(b), in which the input slew $slew_i$ and $slew_{i+1}$ in the asymmetric tree are the same as they are in the symmetric tree, and so are the buffer size and the wire size. (In Section 4.3, the buffer size and the wire size of a buffer stage are possible to change, however, their new values are restricted so that delay variation of the buffer stage will not be increased. Note that the asymmetric tree synthesis does not consider the location of the buffer and wire in the symmetric tree, and its detailed process will be introduced in Section 4.3.)

The neglect of wire length results in the deviation of performance preservation, but the amount of this deviation is tolerable. In the extreme condition, the deviation increases 10% delay variation for a buffer stage. The performance preservation considers

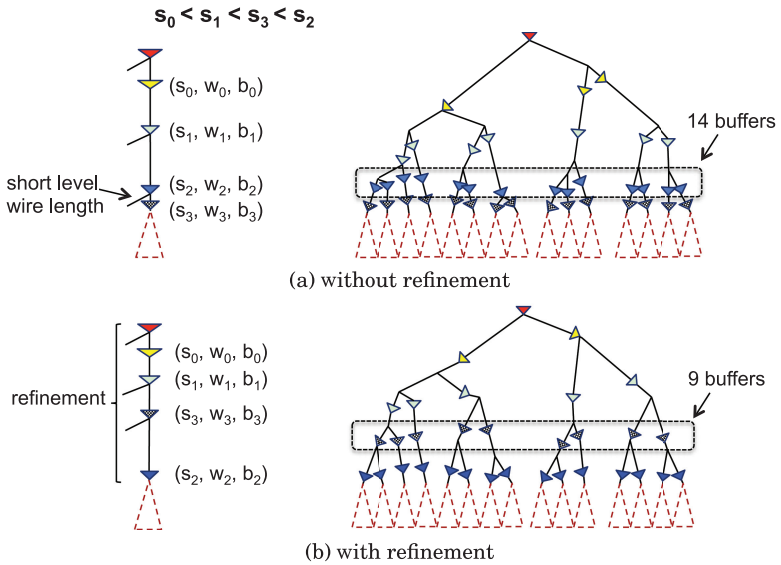


Fig. 12. Solution refinement is a process between the symmetric and the asymmetric clock tree synthesis. The refinement shifts stages of small input slew toward the root. It reduces capacitance loading near leaf level. (a) An asymmetric tree without solution refinement; (b) an asymmetric tree with solution refinement.

slew, buffer size, and wire size for a buffer stage; in other words, it neglects the buffer stage's RC tree topology. The RC tree topology change from a symmetric to an asymmetric tree is such that the tapping point of an asymmetric tree is skewed because of a delay difference between the two merged substrees. The long side will dominate delay variation, and the extreme condition is that the short side degenerates to zero length (no fanout) as shown in Figure 11(b); however, the wire length deviation is bounded because of the correlation between slew and wire length. According to our experiments, for a buffer stage, the extreme condition that a symmetric fanout stage is transferred to a no-fanout stage results in 10% larger delay variation; for a whole tree, the skew performance results in average 9.6% larger skew in an asymmetric tree than in the original symmetric tree.

4.2. Solution Refinement between a Symmetric and an Asymmetric Clock Tree

A solution refinement, which moves the buffer stage of sharp input slew toward the tree root, is adopted to reduce the number of buffers inserted in asymmetric tree synthesis. A tight stage slew constraint limits a stage's wire length; as a result, before being merged with other subtrees, a subtree may need a preceding driving buffer stage to maintain sharp slew. When such a condition occurs near leaf level, a large number of preceding buffers would be inserted, which costs buffer loading. It is sometimes inevitable that a short-level wire length is generated near the leaf level of a symmetric tree as shown in Figure 12(a); however, when synthesizing an asymmetric tree, we can move these sharp slew buffer stages shown as $(s3, w3, b3)$ in Figure 12(b) toward the root so that leaf-level subtrees could be merged as deeply as possible, thus reducing buffer loading from 14 to 9. Therefore, after receiving the buffer insertion and wire sizing solution of a symmetric tree, we shifted the sharp slew buffer stage toward the tree root in order to apply this buffer insertion plan in the asymmetric tree synthesis. Note that the refinement only modifies the plan of buffer insertion for the asymmetric tree synthesis; in other words, no modification was actually done on the original symmetric tree.

4.3. DME-Based Projection

The asymmetric CTS is based on DME and, during the DME bottom-up phase, the parameters, namely input slew of a stage s, leaf slew of a stage s', buffer size b, and wire size w, are maintained for each stage.

Asymmetric CTS steps are listed in Algorithm 2. For each stage, (s, s', b, w) from a refined solution of a symmetric tree are read. According to these parameters, *genStageSubtree* generates a set of stage subtrees N and a table T, which records pairs of a stage subtree and the corresponding stage buffer. Then stage buffers are inserted as roots of stage subtrees and become merging candidates for the next stage. The while-loop (lines 2–6) continues until only one tree is in the merging candidate pool, that is, our final asymmetric clock tree.

We can elaborate more on *genStageSubtree*. In Section 4.2, the solution refinement shifts the small slew stage toward the tree root to reduce the number of buffers. *genStageSubtree* further saves the number of buffers by using stronger driving-strength buffers. A stage buffer may be swapped by a stronger driving-strength one if the stronger buffer drives more subtrees, which in turn saves power. Note that variability of the stronger buffer must be less than that of the original stage buffer. Therefore *genStageSubtree* records the original stage buffer for all subtrees (lines 11–13) and invokes *genStageSubtreeBySingleBuf* several times (lines 14–16) to test whether a stronger buffer saves power.

genStageSubtreeBySingleBuf generates a set of stage subtrees L and records stage buffers in T. In *genStageSubtreeBySingleBuf*, a subtree of smallest delay n_1 has highest priority to be merged. n_2 is the merging partner of n_1 and they must satisfy the stage slew constraint by a driving buffer b'. After n_1 and n_2 are selected (lines 23–37), they are merged into a new subtree n_{new}. Merging candidates are updated and b' is recorded in T as the stage buffer of n_{new} (lines 38–42). If n_1 finds no merging partner, n_1 is removed from merging candidate container N and saved in stage subtree container L (lines 43–46). The merging process continues until no more merging is possible and all stage subtrees are stored in L.

4.3.1. Bottleneck and Complexity. The runtime bottleneck of Algorithm 2 happens at line 5, that is, to insert the buffer and adjust the wire length to match the slew target. It costs the most runtime, because it embeds NGSPICE simulation to derive an accurate slew rate and delay.

The complexity of Algorithm 2 is analyzed as follows.

(1) The complexity of *genStageSubtreeBySingleBuf* is equal to DME.
   —If the driving buffer is very strong so that one buffer can drive the whole tree, the first *genStageSubtreeBySingleBuf* in line 15 will return $|N| = 1$, and the later iterations of *genStageSubtreeBySingleBuf* will cost nothing because no subtrees are merged. As a result, an asymmetric tree is completed by one *genStageSubtree*, and the complexity is equal to DME. The slew check in line 28 of *genStageSubtreeBySingleBuf* is an additional constant cost for each merging of DME, which does not increase complexity.
   —When a driving buffer cannot drive a whole tree, *genStageSubtree* returns a set of subtrees in N, which is L in *genStageSubtreeBySingleBuf*. The cost of generating a subtree in L is equal to a successful merging, because generating a subtree in L and a successful merging both decrease one subtree in merging candidate N in *genStageSubtreeBySingleBuf*. The only difference is that n_1 find no merging partner with a valid slew in lines 26–37.
   —The complexity of DME is $O(n^2)$, where n is the number of sinks.

ALGORITHM 2: *AsymmetryTree*(*allsinks*, B)

Input: all sinks, buffer library B, wire library W, a refined solution of a symmetric clock tree
Output: an asymmetric tree

$N \leftarrow$ all sinks;
while $|N| > 1$ **do**
  $(s, s', w, b) \leftarrow$ read solution of symmetric tree for current stage, or there is no feasible solution and exit;
  $(N, T) \leftarrow genStageSubtree(s, s', w, b, N)$;
  for all sub-trees $\in N$, insert buffers as their roots by a table of stage buffer T and adjust wire length to match s' //*concurrent*;
end
There is only one sub-tree in N, connect its root to clock source and top down node embedding;

genStageSubtree(s, s', w, b, N)
for *all sub-trees* $n \in N$ **do**
  $T(n) \leftarrow b$;
end
for *all* $b' \in B$ *&& driving strength satisfies that strength*(b) $\leq$ *strength*(b') $\leq$ $\alpha \times$ *strength*(b), *by order of strength* **do**
  $(N, T) \leftarrow genStageSubtreeBySingleBuf(b', s, s', w, N)$;
end
return (N, T);

genStageSubtreeBySingleBuf(b', s, s', w, N)
$L \leftarrow \phi$;
while $|N| > 1$ **do**
  $n_1 \leftarrow$ smallest delay sub-tree $\in N$;
  $isMergeble \leftarrow false$;
  $minCost \leftarrow$ inf;
  for $n_{test} \leftarrow$ *all other sub-trees* $\in N$ **do**
    $n_{new} \leftarrow merge(n_1, n_{test})$ by wire size w;
    $s_{test} \leftarrow calcSlew(b', n_{new})$ //buffer b' drives sub-tree n_{new};
    if $s_{test} \leq s'$ **then**
      $cost \leftarrow calcMergeCost(n_{new})$;
      if $cost < minCost$ **then**
        $minCost \leftarrow cost$;
        $n_2 \leftarrow n_{test}$;
      end
      $isMergeble \leftarrow true$;
    end
  end
  if $isMergeble$ **then**
    $n_{new} \leftarrow merge(n_1, n_2)$;
    $N \leftarrow N \setminus \{n_1, n_2\}$;
    $N \leftarrow N \cup \{n_{new}\}$;
    $T(n_{new}) \leftarrow b'$;
  end
  else
    $N \leftarrow N \setminus n_1$;
    $L \leftarrow L \cup n_1$;
  end
end
$L \leftarrow L \cup n_0$, n_0 is the last sub-tree $\in N$;
return (L, T)

Table II. ISPD 2010 Benchmark Information

	#Sinks	LCS (ps)	LCS Dist.(μm)	W(μm)	H(μm)	#Blocks
cns01	1107	7.5	600	8000	8000	4
cns02	2249	7.5	600	13000	7000	1
cns03	1200	4.9	370	3072	493	2
cns04	1845	7.5	600	2130	2690	2
cns05	1016	7.5	600	2319	2545	1
cns06	981	7.5	600	1950	891	0
cns07	1915	7.5	600	2537	1448	0
cns08	1134	7.5	600	1837	1628	0

Table III. Physical Properties of Buffers

	Inverted	Input Cap (fF)	Output Cap (fF)	Output Res (Ω)
inv-0	True	35	80	61.2
inv-1	True	4.2	6.1	440

Table IV. Physical Properties of Wires

	Unit Res (Ω/nm)	Unit Cap (fF/nm)
wire-0	0.0001	0.0002
wire-1	0.0003	0.00016

(2) The complexity of Algorithm 2 is as follows. Assuming each *genStageSubtreeBySingleBuf* scales down the number of subtrees by α, we have two cases.
—*Worst case* ($\alpha = 1$).
Assuming a *genStageSubtree* sweeps β buffer sizes and that *AsymmetryTree* calls *genStageSubtree* at most γ times, there are totally $\beta \times \gamma$ times of *genStageSubtreeBySingleBuf*. The complexity is $O(\beta \times \gamma \times n^2)$.
—*General case* ($0 < \alpha < 1$).
m is an integer satisfying two rules: $n \times \alpha^m \leq 1$ and $1 < n \times \alpha^{m-1}$. This means that a tree is completed after m number of *genStageSubtreeBySingleBuf* are performed. The complexity is $O(\sum_{i=0}^{m-1} (n \times \alpha^i)^2) = O(\sum_{i=0}^{m-1} \alpha^{2i} n^2)$.

5. EXPERIMENTAL RESULTS AND LIMITATIONS OF ISPD 2010 BENCHMARK

This section demonstrates experimental results and compares our methods in Section 5.1. Section 5.2 discusses limitations of the ISPD 2010 benchmark and strategies of CTS for different variation scenarios.

5.1. Experimental Results

The proposed approach is implemented in C++, and the mathematical programming solver is IBM ILOG CPLEX v12.2 [CPLEX 2010]. Experimental results are evaluated by the ISPD 2010 benchmark (Table II), which is based on IBM and Intel real-case microprocessor design. The buffer library (Table III) and wire library (Table IV) are based on PTM [2011] 45nm technology. The variation setting is the same one as that of the contest, that is $\pm$7.5% vdd variation and $\pm$5% wire width variation. Monte Carlo simulations by NGSPICE are performed to evaluate performance.

The experiments are carried out on a 2.4 GHz Intel Xeon CPU Linux workstation with 16GB memory. The runtime limit of mathematical programming is set to 600 seconds and 14 threads are utilized for concurrent SPICE simulations of asymmetric CTS. The correlation coefficients of stage delay variation are ρ_R of 0.4, and ρ_F of 0. These values are extracted by a least-square error fitting on an inverter chain, as

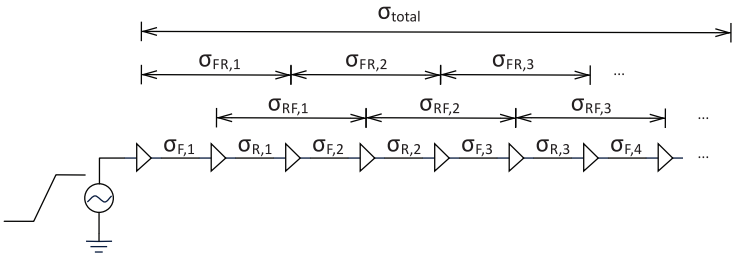


Fig. 13. Extraction of ρ_R and ρ_F by an inverter chain. Variance of two-stage delay is collected by Monte Carlo simulations, and a least-square-fitting (26)–(29)-derive ρ_R and ρ_F.

shown in Figure 13.

$$\min \quad \sum_i (\sigma_{RF_i} - \sigma_{\hat{RF}_i})^2 + (\sigma_{FR_i} - \sigma_{\hat{FR}_i})^2 \tag{26}$$

$$\text{such that} \quad {\sigma_{total}}^2 = \sum_i \left({\sigma_{R_i}}^2 + {\sigma_{F_i}}^2\right) + 2\sum_i (\sigma_{R_i}\sigma_{F_i}\rho_R + \sigma_{F_i}\sigma_{R_i}\rho_F) + O(n^3) \tag{27}$$

$${\sigma_{\hat{FR}_i}}^2 = {\sigma_{F_i}}^2 + {\sigma_{R_i}}^2 + 2\sigma_{F_i}\sigma_{R_i}\rho_F \tag{28}$$

$${\sigma_{\hat{RF}_i}}^2 = {\sigma_{R_i}}^2 + {\sigma_{F_{i+1}}}^2 + 2\sigma_{R_i}\sigma_{F_{i+1}}\rho_R \tag{29}$$

All σ_{F_i} and σ_{R_i} are looked up by the NLDM-like cell delay variation model and σ_{total}, σ_{RF_i}, and σ_{FR_i} are derived by Monte Carlo simulations. The higher-order terms $O(n^3)$ are neglected.

Table V shows a comparison of the statistics on LCS, capacitance loading, CPU time, and wall-clock time. Acronyms are used to denote the following methods.

—*SMeshMB* is a symmetric tree driving a bottom mesh, which is done in Shih et al. [2010].
—*Contango 2.0* is an asymmetric tree with clock latency minimization, which is done in Lee et al. [2010].
—*AMB* is an asymmetric tree with buffer-stage minimization, which is done in Bujimalla and Koh [2011].
—*AMB_CL* inserts a cross link based on *AMB*, which is done in Mittal and Koh [2011].

And our four methods are as follows.

—*length-symm* is a symmetric clock tree of length-based buffer insertion.
—*mp-symm* is a symmetric clock tree of mathematical-programming-based buffer insertion.
—*length-asym* is an asymmetric clock tree transformed from *length-symm*.
—*mp-asym* is an asymmetric clock tree transformed from *mp-symm*.

The clock network produced by *mp-asym* has the smallest capacitance loading. It is smaller than *SMeshMB* [Shih et al. 2010], *Contango 2.0* [Lee et al. 2010], *AMB* [Bujimalla and Koh 2011], and *AMB_CL* [Mittal and Koh 2011] up to 1.2×. For our four methods' comparison, mathematical-programming-based buffer insertion wire sizing improved 4% capacitance, and transformation from symmetric to asymmetric versions improved up to 5% of capacitance. The runtime overhead results from: (1) the mathematical programming solver and (2) SPICE simulations of asymmetric CTS, especially in large cases, are cns01 and cns02. The present study shows that concurrent SPICE simulations of asymmetric CTS can effectively reduce CPU time to wall clock. It can

Table V. Experimental Results of ISPD 2010 Benchmark

BM		Skew, Capacitance, and RunTime							
		SMeshMB	Contango 2.0	AMB	AMB_CL	length-symm[1]	mp-symm	length-asym	mp-asym
cns01	95%LCS(ps)	7.16	7.01	5.79	7.32	7.32	7.35	~~7.77~~	6.41
	cap(pF)	445.3	198.3	177.5	142.6	146.0	**124.4**	124.5	143.3
	cpu time (sec)	0.4	12015	2790	1092	114	795	1477	2426
	wall clock (sec)					97	688	335	860
cns02	95%LCS(ps)	7.33	7.34	6.69	7.42	7.38	7.49	~~8.93~~	6.73
	cap(pF)	933.6	375.9	329.9	265.2	268.3	**255.3**	250.4	275.2
	cpu time (sec)	2.42	25006	7787	4314	295	935	3319	6223
	wall clock (sec)					120	763	800	1659
cns03	95%LCS(ps)	4.88	4.18	3.46	4.49	4.76	4.64	~~6.41~~	4.83
	cap(pF)	183.7	55.86	50.81	36.61	34.17	34.33	34.24	**33.21**
	cpu time (sec)	1.57	3840	2094	383	71	274	313	441
	wall clock (sec)					37	241	120	228
cns04	95%LCS(ps)	4.01	4.46	3.79	6.70	7.14	6.70	~~7.64~~	6.96
	cap(pF)	196.3	71.84	57.44	51.07	42.77	41.78	40.00	**38.03**
	cpu time (sec)	0.27	6075	2763	934	73	244	335	970
	wall clock (sec)					27	199	127	720
cns05	95%LCS(ps)	3.81	4.41	3.68	4.78	5.88	6.22	5.72	5.80
	cap(pF)	89.09	37.69	28.93	25.13	22.13	20.98	19.50	**18.33**
	cpu time (sec)	0.10	2406	1110	278	36	207	150	716
	wall clock (sec)					13	185	65	609
cns06	95%LCS(ps)	7.40	6.05	4.01	6.41	5.61	5.82	5.75	7.04
	cap(pF)	160.4	47.81	36.12	32.68	28.55	28.01	26.03	**23.78**
	cpu time (sec)	0.28	2660	1142	285	70	75	184	232
	wall clock (sec)					17	23	57	70
cns07	95%LCS(ps)	6.24	4.58	5.65	5.86	6.62	6.80	7.08	6.75
	cap(pF)	228.2	72.66	57.93	48.32	43.91	43.39	39.79	**39.30**
	cpu time (sec)	0.30	2351	2968	818	75	122	283	511
	wall clock (sec)					29	76	112	220
cns08	95%LCS(ps)	7.64	5.15	4.24	5.07	6.50	6.89	6.58	6.95
	cap(pF)	228.2	52.49	40.43	32.70	28.41	28.08	27.25	**25.69**
	cpu time (sec)	0.28	1987	1497	327	76	82	206	241
	wall clock (sec)					29	36	73	90
geo mean of cap		5.17	1.76	1.45	1.20	1.09	1.05	1.04	1.00

further reduce runtime by replacing SPICE simulation with a static timing analysis tool, for example, of a composite current source model.

Figure 14 describes experiments on benchmark cns01 with different skew constraints. The Figure shows that *mp*-based ones are more flexible than *length*-based ones for different skew specifications. Table VI shows the comparison between Monte Carlo results and the asymptotic skew approximation.

5.2. Limitations of ISPD 2010 Benchmark

To evaluate a clock network by means of the ISPD 2010 benchmark, one should beware of the variation setting. We list the notables of variation setting as follows.

[1]*length-symm* is slightly different from Chang et al. [2012]. Because the symmetric tree in Chang et al. [2012] is not obstacle avoiding, and we speed up runtime by parallelizing the fine-tuning SPICE simulations.

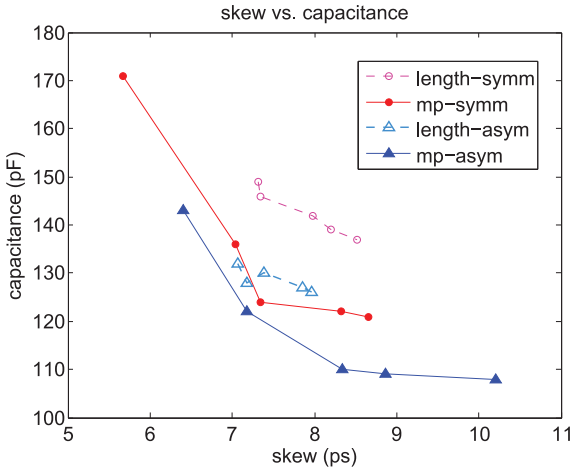


Fig. 14. Trade-off between skew and capacitance on cns01. *mp-asym* has the smallest capacitance. Mathematical-programming-based buffer insertion wire sizing is more flexible for different skew constraints.

Table VI. Skew Estimation vs. Monte Carlo Result

BM	Monte Carlo 95%LCS	Estimated 95%LCS
cns01	6.41	8.58
cns02	6.73	10.21
cns03	4.83	5.00
cns04	6.96	7.49
cns05	5.80	7.29
cns06	7.04	7.18
cns07	6.75	7.42
cns08	6.95	7.10

—The problem addressed by Bujimalla and Koh [2011] and Lee and Markov [2011] is that the setting of the ISPD 2010 benchmark allows to reduce a buffer-stage variation by stacking buffers such that each buffer has its own voltage source. Increasing the number of stacking buffers can smooth the effects of voltage variation. The works done in Bujimalla and Koh [2011] and Lee and Markov [2011] adopt a setting of single-location single-voltage to eliminate smoothing effects by the stacking buffer, as shown in Figure 15. It is worth mention that the comparison in Table V is fair because the number of stacking buffers used in our proposed method is not greater than others. The max number of stacking buffers used in *length-asym*, *mp-asym*, *Contango 2.0* [Lee et al. 2010], *AMB* [Bujimalla and Koh 2011], and *AMB_CL* [Mittal and Koh 2011] are all 30x inv-1, while those used in *length-symm*, *mp-symm*, and *SMeshMB* [Shih et al. 2010] are all 20x inv-1.

—The slew effect addressed in this work affects performance evaluation of a multiple-driving-paths network. Networks such as mesh and cross link reduce the variation-induced skew by improving the delay correlation between paths. However, when the slew effect is not controlled well, the baseline of path delay variation is different, possibly misleading the real performance of a network.

—The ISPD 2010 benchmark considers variation of supply voltage and wire width only. When more variation sources are considered such as threshold voltage and gate

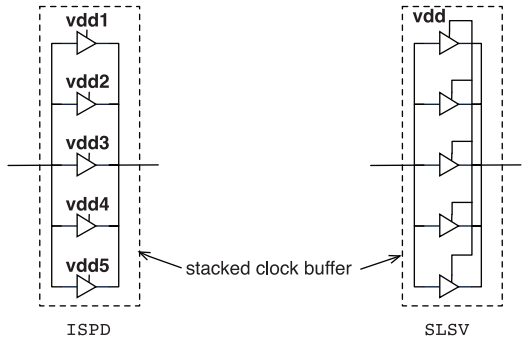


Fig. 15. By the setup of the ISPD 2010 benchmark, buffers stacked at a same location have differing voltage source. This smooths voltage variation for a buffer stage. By the setup of Single-Location Single-Voltage (SLSV), all buffers have only one voltage source.

length, the primitive delay variation of a buffer stage increases. In this scenario, the proposed CTS method is concerned, and a strategy of CTS to minimize skew uncertainty should seek to minimize the number of buffer stages. To minimize delay variation of a buffered long wire, the slew effect considered in this work delivers an idea that there is an optimal number of buffer stages. When the number of buffer stages is less than the optimal value, inserting an additional buffer stage sharpens slew and reduces delay variation. When the number is more than the optimal value, the slew effect remains but is weaker than the primitive delay variation of an additional buffer stage. Once the primitive delay variation of a buffer stage become severe, the optimal number of buffer stages will be less. In the worst case, to minimize delay variation is just to minimize the number of buffer stages. We call this scenario *primitive variation dominance*.

6. CONCLUSIONS

This work proposed a method to tackle supply voltage variation and to synthesize a lower-power and robust clock tree. The proposed method includes two stages. The first stage facilitates global optimization by adopting a symmetric structure. Buffer insertion and wire sizing are formulated in mathematical programming, and the slew effect is considered by an NLDM-like cell delay variation model. The second stage performs local optimization in which a transformation from a symmetric to an asymmetric tree further saves wire and buffer loading. Experimental results demonstrate that the proposed method saves capacitance loading up to 20%.

Beyond the proposed method, limitations of the ISPD 2010 benchmark are addressed. To evaluate the performance of a clock network synthesizer by the ISPD 2010 benchmark, one should be aware of the variation setting. For example, single-location single-voltage can prevent the voltage variation smoothing by stacking buffers; when evaluating performance of a multiple-driving-paths clock network, ignoring the slew effect may result in a different baseline of path delay variation and mislead the performance; when variation becomes more severe, *primitive variation dominance* may occur, and the strategy of CTS in this scenario should seek to minimize the number of buffer stages.

ACKNOWLEDGMENTS

The authors would like to thank for anonymous reviewers for their comments that guided us to improve the quality of the article.

REFERENCES

C. J. Alpert and A. Devgan. 1997. Wire segmenting for improved buffer insertion. In *Proceedings of the Design Automation Conference (DAC'97)*. 588–593.

D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. 2008. Statistical timing analysis: From basic principles to state of the art. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 27, 4, 589–607.

S. Bujimalla and C.-K. Koh. 2011. Synthesis of low power clock trees for handling power-supply variations. In *Proceedings of the International Symposium on Physical Design (ISPD'11)*. 37–44.

Y.-C. Chang, C. K. Wang, and H.-M. Chen. 2012. On constructing low power and robust clock tree via slew budgeting. In *Proceedings of the International Symposium on Physical Design (ISPD'12)*. 129–136.

CPLEX. 2010. IBM ilog cplex optimizer v12.2. http://www01.ibm.com/software/integration/optimization/cplex-optimizer/.

S. Hu, C. J. Alpert, J. Hu, S. K. Karandikar, Z. Li, W. Shi, and C. N. Sze. 2007. Fast algorithms for slew-constrained minimum cost buffering. *IEEE Trans. Comput.-Aided Des.* 26, 11, 2009–2022.

S. D. Kugelmass and K. Steighlitz. 1990. An upper bound on expected clock skew in synchronous systems. *IEEE Trans. Comput.* 39, 12, 1475–1477.

D.-J. Lee, M.-C. Kim, and I. L. Markov. 2010. Low-power clock trees for cpus. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'10)*. 444–451.

D.-J. Lee and I. L. Markov. 2011. Multilevel tree fusion for robust clock networks. In *Proceedings of the International Conference on Computer Aided Design (ICCAD'11)*. 632–639.

T. Mittal and C.-K. Koh. 2011. Cross link insertion for improving tolerance to variations in clock network synthesis. In *Proceedings of the International Symposium on Physical Design (ISPD'11)*. 29–36.

NIST. 2012. NIST/SEMATECH e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/.

PTM. 2011. Predictive technology model. http://ptm.asu.edu/.

A. Rajaram, J. Hu, and R. Mahapatra. 2006. Reducing clock skew variability via crosslinks. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 25, 6, 1176–1182.

P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie. 2001. A clock distribution network for microprocessors. *IEEE J. Solid-State Circ.* 36, 5, 792–799.

X-.W. Shih and Y.-W. Chang. 2010. Fast timing-model independent buffered clock-tree synthesis. In *Proceedings of the Design Automation Conference (DAC'10)*. 80–85.

X.-W. Shih, H.-C. Lee, K.-H. Ho, and Y.-W. Chang. 2010. High variation-tolerant obstacle-avoiding clock mesh synthesis with symmetrical driving trees. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'10)*. 452–457.

C. N. Sze. 2010. ISPD 2010 high performance clock network synthesis contest: Benchmark suite and results. In *Proceedings of the International Symposium on Physical Design (ISPD'10)*. 143–143.

C. N. Sze, P. Restle, G.-J. Nam, and C. Alpert. 2009. ISPD 2009 clock network synthesis contest. In *Proceedings of the International Symposium on Physical Design (ISPD'09)*. 149–150.

L. P. P. P. van Ginneken. 1990. Buffer placement in distributed rc-tree networks for minimal elmore delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'90)*. 865–868.

L. Xiao, Z. Xiao, Z. Qian, Y. Jiang, T. Huang, H. Tian, and E. F. Y. Young. 2010. Local clock skew minimization using blockage-aware mixed tree-mesh clock network. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'10)*. 458–462.

Received November 2013; revised June 2014; accepted July 2014