INTEGRATION, the VLSI journal 52 (2016) 91–101

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Optimal utilization of adjustable delay clock buffers for timing correction in designs with multiple power modes

Juyeon Kim, Deokjin Joo, Taewhan Kim*

Department of Electrical and Computer Engineering, Seoul National University, Seoul, Korea

A R T I C L E I N F O

Article history:
Received 18 March 2015
Received in revised form
15 June 2015
Accepted 30 August 2015
Available online 11 September 2015

Keywords:
Adjustable delay buffers
Timing
Clock skew
Power modes
Resource allocation
Design optimization

A B S T R A C T

Meeting clock skew constraint is one of the most important tasks in the synthesis of clock trees. Moreover, the problem becomes much hard to tackle as the delay of clock signals varies dynamically during execution. Recently, it is shown that adjustable delay buffer (ADB) whose delay can be adjusted dynamically can solve the clock skew variation problem effectively. However, inserting ADBs requires non-negligible area and control overhead. Thus, all previous works have invariably aimed at minimizing the number of ADBs to be inserted, particularly under the environment of multiple power modes in which the operating voltage applied to some modules varies as the power mode changes. In this work, unlike the previous works which have solved the ADB minimization problem heuristically or locally optimally, we propose an elegant and easily adoptable solution to overcome the limitation of the previous works. Precisely, we propose an $O(n \log n)$ time (bottom-up traversal) algorithm that (1) optimally solves the problem of minimizing the number of ADBs to be allocated with continuous delay of ADBs and (2) enables solving the ADB allocation problem with discrete delay of ADBs to be greatly simple and predictable. In addition, we propose (3) a systematic solution to an important extension to the problem of buffer sizing combined with the ADB allocation to further reduce the ADBs to be used. The experimental results on benchmark circuits show that compared to the results produced by the best known ADB allocation algorithm, our proposed algorithm uses, on average under 30–50 ps clock skew bound, 13.5% and 15.8% fewer numbers of ADBs for continuous and discrete ADB delays, respectively. In addition, when buffer sizing is integrated, our algorithm uses 31.7% and 31.3% fewer numbers of ADBs, even reducing the area of ADBs and buffers by 15.0% and 16.3% for continuous and discrete ADB delays, respectively.

1. Introduction

Clock is one of the most important signals on a chip of synchronous based system, as all the synchronous components on the chip such as flip-flops (FFs) rely on it. Clock tree is a commonly used structure of circuits that distributes the clock signal from the clock source to all the *clock sinks* (e.g., FFs and latches), where the clock signal is required. It is imperative that the maximum of the arrival time difference between the clock sinks, which is known as *global clock skew*, should be maintained under a certain bounded value typically within 10% of the clock period, as a large clock skew may cause timing violation on the circuits. (If no confusion occurs, the *global clock skew* is simply referred to as *clock skew* in this presentation.)

Many research works on the clock tree optimization such as clock routing, clock buffer insertion/sizing, and wire sizing have been performed to control or minimize the clock skew [1–7]. While these approaches were effective, advanced low power design techniques introduced new challenges to the clock skew control problem. Specifically, for multiple power mode designs, where the supply voltage to the circuit components varies dynamically depending on modes, the clock arrival time also varies accordingly.

Even though the previous works can consider the clock skew constraint on every power mode, it would be highly likely that the resulting clock tree uses a substantially long wirelength or there exists no clock tree that satisfies the clock skew constraint on every power mode. On the other hand, post-silicon tuning (e.g., [8–11]) such as inserting adjustable delay buffers (ADBs) is a widely used method to deal with the timing problem caused by process and environment variations. Because the delay of an ADB can be controlled by its delay control inputs [12], the clock skew

* Corresponding author.
E-mail address: tkim@ssl.snu.ac.kr (T. Kim).

http://dx.doi.org/10.1016/j.vlsi.2015.08.005
0167-9260/© 2015 Elsevier B.V. All rights reserved.

variation caused by process variation can be tuned by properly inserting ADBs after the manufacturing stage has been completed. The idea of using ADBs in multiple power modes is to replace some of normal clock buffers with ADBs so that the clock skew constraint on each power mode can be met; when the power mode changes during execution, e.g., from power mode *mode-1* to power mode *mode-2*, the delays of ADBs in clock tree that have been adjusted under *mode-1* are readjusted to meet the clock skew constraint under *mode-2*. Since ADB logic component is much bigger than normal buffer and it requires control line as well as switching logic, the set of related problems to be solved for the ADB-based clock skew optimization in multiple power modes are allocating a minimum number of ADBs, finding the normal buffers (or locations) in the clock tree that are to be replaced by ADBs, and determining the delay value of ADBs to be assigned on each power mode. We call these problems collectively *ADB allocation problem*.

Su et al. [13,14] proposed a linear-time optimal algorithm for the delay assignment problem and exploits the algorithm to solve the rest of two subproblems of the ADB allocation problem heuristically in a greedy manner. Lin et al. [15] proposed an efficient algorithm of two-stage approach which performs a top-down ADB allocation followed by a bottom-up ADB elimination. Even though the approach reduces the run time over that in [13,14], it still does not guarantee an optimality of ADB allocation. Lim and Kim [16] proposed a linear-time algorithm for the ADB allocation problem where they solved the problem optimally for *each* power mode. However, merely collecting the optimal results on individual power modes does not mean globally optimal for all power modes. In this work, we revisit the ADB allocation problem and propose a set of solutions to overcome the limitation of the previous works. More precisely, we propose (1) an $O(n \log n)$ time algorithm that optimally solves the problem of minimizing the number of ADBs to be allocated for all power modes with continuous delay of ADBs and (2) enables solving the ADB allocation problem with discrete delay of ADBs to be greatly simple and predictable. In addition, we propose an effective solution to an important extended problem: (3) the ADB allocation problem combined with buffer sizing. (A preliminary version, which contains concise descriptions and no proofs, of our work can be found in [17].)

It should be mentioned that the work in [16] is completely different from our proposed optimal algorithm by a simple reasoning: For example, [16] requires optimally two ADBs, each in clock nodes 1 and 2, for power mode 1 while requiring optimally two ADBs, each in nodes 3 and 4, for power mode 2. Thus, the combined ADB allocation is four ADBs, each in nodes 1, 2, 3, and 4 to meet timing for all power modes. On the other hand, ours produces an optimal ADB allocation result considering power modes *all together*. The globally optimal allocation may be three ADBs (i.e., not four ADBs), say, each in nodes 1–3. This reasoning clearly foresees that as the number of power modes increases, the gap (i.e., ADB difference) between [16] and ours will increase.

The rest of the paper is organized as follows. Section 2 illustrates the structure of ADB implementation and shows an example of using ADBs for timing correction. Section 3 defines the ADB allocation problem and shows an example to motivate the work. Then, Section 4 proposes an optimal algorithm of ADB allocation with continuous delay values and a modification of the algorithm to support ADBs with discrete delay values. Section 5 proposes a solution to the extended problem of integrating buffer sizing into ADB allocation. Experimental results are provided in Section 6 to show the effectiveness of our proposed ADB allocation algorithms. Finally, a conclusion of the work given in Section 7.

2. ADB structure and example of ADB utilization

Fig. 1 shows the structure of a capacitor bank based implementation of ADB [18]. This implementation of a well-known capacitor bank based ADB consists of two inverters at the input and output ports, and in the middle there is an array of capacitors with switch transistors attached. The switches are controlled by the capacitor bank controller, which controls the number of active capacitors according to the control bits. Activating more capacitors increases the total capacitance between the two inverters, which in turn increases the signal propagation delay between the input and output ports. Inverter based ADB [19] is another implementation structure of ADB, but the adjustable delay values are less fine-grained than that of the capacitor bank based one.

Fig. 2(a) shows an example of clock tree that has four sinks *s*1, *s*2, *s*3, and *s*4, two ADBs replacing two clock buffers, and ADB control logic. Suppose there are two power modes *mode-1* and *mode-2* in this design. Then, the two numbers separated by a slash next to each sink indicate the clock signal arrival times in *mode-1* and *mode-2*. When the clock skew bound is given to 10, the clock tree causes clock skew violations in both modes if ADBs were not used. With the replacement of two clock buffers by ADBs, the two numbers next to each ADB indicate the delay increments (simply called *delay values*) in *mode-1* and *mode-2*: The ADB on the left adds delay of 2 in *mode-1*, thus the clock signal arrival time at *s1* in *mode-1* becomes 6. Likewise, the ADB on the right adds delay of 3 in *mode-2*, increasing the arrival time at *s*3 in *mode-2–6*. To control the ADBs' delay, a mode signal is required. In addition, depending on the implementation of the ADBs, control logic that

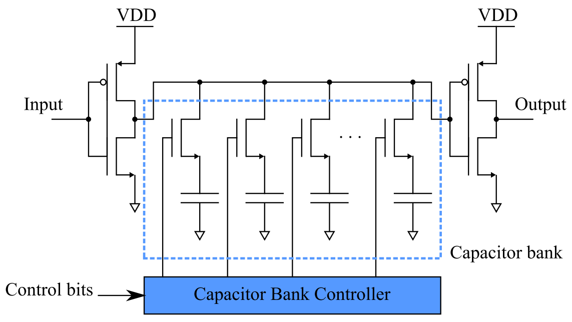


Fig. 1. The structure of a capacitor bank based ADB. The capacitor bank adjusts signal propagation delay from the input to output ports.

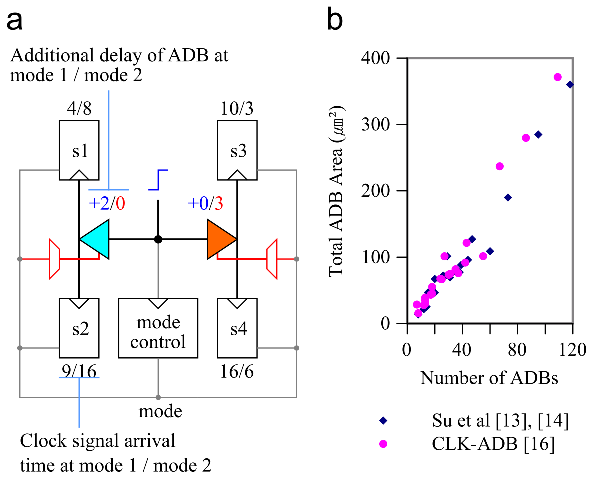


Fig. 2. (a) An example of clock tree with the replacement of two clock buffers with ADBs. (b) The relationship between the number of ADBs and the total ADB area (including logic overhead) used by [13,14,16].

converts the mode signal to ADB's bank controller input is needed. This additional overhead incurred by the insertion of ADBs is also shown in Fig. 2(a).

Fig. 2 (b) shows a scatter plot of the number of ADBs allocated versus the total ADB area including the overhead, obtained by implementing the algorithms in the previous works [13,14,16]. The plot indicates that the number of ADBs has a strong correlation with the total sum of the area of ADBs, justifying that the primary objective of the ADB allocation problem is to minimize the number of ADBs to be inserted.

3. Problem definition and motivation

The problem of ADB allocation in a clock tree can be described as

Problem 1. ADB allocation problem: Given a synthesized clock tree, arrival times of clock sinks in each power mode m, and clock skew bound κ_m in each power mode m, replace the least number of clock buffers with ADBs and assign delays to the ADBs to satisfy κ_m in every power mode m.

Two common features of the previous ADB allocation algorithms [13,14,16] are that they resolve the clock skew violation by synchronizing *the earliest arrival times* of (two) subtrees of interest where they set the delay value of ADB on a root of one of the subtrees to the difference of the earliest arrival times of the subtrees, and the delay value adjustment is performed *mode by mode*. While this method of delay value assignment does minimize the clock skew, their method of applying mode by mode yields suboptimal results that use more ADBs than necessary.

For example, consider the clock tree in Fig. 3(a) with two operating modes *mode-1* and *mode-2*. The initial clock signal arrival times at sinks are shown at the bottom in black numbers. Suppose the clock skew bound $\kappa=10$ for two power modes. Clearly, there are clock skew violations in both *mode-1* and *mode-2*; in *mode-1* the clock skew is 13, which is defined by sinks $s2$ and $s3$, and in *mode-2*, the clock skew is also 11, which is defined by $s3$ and $s6$. The results of ADB allocation produced by the previous algorithm [16] for the clock tree is shown in Fig. 3(a) where buffers B, D, and E are replaced with ADBs and the adjusted arrival times are shown in red and blue numbers next to the initial delays. The delay adjustment procedure is as follows. In *mode-1*, the earliest arrival time $(=5)$ of the subtrees rooted at D is synchronized to the earliest arrival time $(=16)$ of the subtree rooted at C by assigning delay increment of 11 to the ADB in node D. However, the delay adjustment at D increases the arrival time at sink $s4$ from 13 to 24, which causes another skew violation due to the times in $s4$ and $s5$. The violation is then resolved by assigning delay increment of 6 to the ADB in node E. Likewise, in *mode-2* the clock skew violation due to the times at $s3$ and $s6$ is resolved by assigning delay increment of 6 to the ADB in node B. From the ADB allocation and delay assignment, we observe that (1) synchronizing a subtree's earliest arrival time (e.g., time at $s3$ in *mode-1*) introduces the delay increase at another sink (e.g., $s4$), so that additional ADB allocation with delay adjustment shall be needed; (2) even though the skew violation in *mode-2* requires one ADB to be allocated, node B is not the only position at which an ADB could be allocated. An alternative position is D, which coincides with the ADB allocation in *mode-1*. An optimal ADB allocation is shown in Fig. 3 (c) in which only one ADB with delay increment of 3 in *mode-1* and 1 in *mode-2* is inserted to the tree. This example clearly shows that delay adjustment according to the synchronization of the earliest arrival times does not always yield optimal results. Furthermore, in order to find optimal results, ADB allocation should consider all modes simultaneously.

Table 1
Notations.

Symbol	Description
n_i	A node in a clock tree, which is either a buffer or a sink
T_{n_i}	The subtree rooted at node n_i
$arr_{n_i,m}$	Arrival time at sink node n_i at *mode-m*
$lst_{n_i,m}$	The latest arrival time among the sinks on the subtree rooted at node n_i in *mode-m*
κ_m	The given clock skew bound to meet in *mode-m*
$\alpha_{n_i,m}$	Delay value (i.e., increment) of ADB located at node n_i in *mode-m*
H_{n_i}	Set of child nodes of n_i not to be replaced by ADBs

4. Optimal ADB allocation

This section describes our proposed ADB allocation algorithms with continuously and discretely adjustable values of ADBs. The notations commonly used in the presentation is summarized in Table 1.

4.1. The proposed algorithm

First, we demonstrate the procedure of our algorithm for the continuous delay of ADBs, called ADB-Pullup, step-by-step using an example to see how the algorithm works. Then, we describe the flow of the algorithm and the properties of the algorithm.

Let us consider the clock signal arrival times shown in the clock tree in Fig. 4(a). Let $\kappa_m=10$ for all m. Then, ADB-Pullup initially assumes that each sink has a distinct fictitious ADB at the front of it. The blue numbers at the bottom of each sink s_i indicate the delay values, i.e., $\alpha_{s_i,1}$ in *mode-1* and $\alpha_{s_i,2}$ in *mode-2* of the ADB in s_i, $i=1,\dots,10$. We compute the delay value by

$$\alpha_{s_i,m}=\max\{0, lst_{root,m}-\kappa_m-arr_{s_i,m}\} \tag{1}$$

where *root* represents the clock source (root) node of the clock tree. For example, $\alpha_{s1,1}=\max\{0,20-10-7\}=3$ and $\alpha_{s1,2}=\max\{0,20-10-8\}=2$. Note that the value by Eq. (1) for each sink s_i corresponds to the least increase of delay required on the fictitious ADB in s_i to meet the clock skew constraint. Then, ADB-Pullup performs a bottom-up traversal on the clock tree to move up (i.e., pull up) the ADBs towards the root of clock tree.

The decision of allocating an ADB at n_k which is a non-sink and whose α value has been assigned is made according to the evaluation result of the inequality:

$$\alpha_{n_k,m}>lst_{root,m}-lst_{n_i,m} \tag{2}$$

where n_i is the parent node of n_k.

If the inequality is true for at least one power mode, an ADB is allocated.[1] For example, since $\alpha_{b4,2}(=2)>lst_{root,2}-lst_{b2,2}$ $(=20-20=0)$, an ADB is inserted to $b4$. However, since $\alpha_{b5,1}(=0)\le lst_{root,1}-lst_{b2,1}(=20-16=4)$ and $\alpha_{b5,2}(=0)\le lst_{root,2}-lst_{b2,2}$ $(=20-20=0)$, no ADB is inserted to $b5$. (The computation of $\alpha_{b4,2}$ and $\alpha_{b5,1}$ are shown in Fig. 4(a).) Once the decision of allocating ADBs to all children of n_i is made, the α value of n_i is updated by

$$\alpha_{n_i,m}=\max\{\alpha_{n_k,m}:n_k\in H_{n_i}\} \tag{3}$$

where H_{n_i}[2] represents the set of n_i's children that are either sinks or non-sinks, but not the nodes with ADB. For example, since

[1] This implies that the ADB allocation decision is made by considering *all* power modes together.

[2] If $H_{n_i}=\phi$, then $\alpha_{n_i,m}$ is set to 0 for every mode m.

III. Problem Definition and Motivation

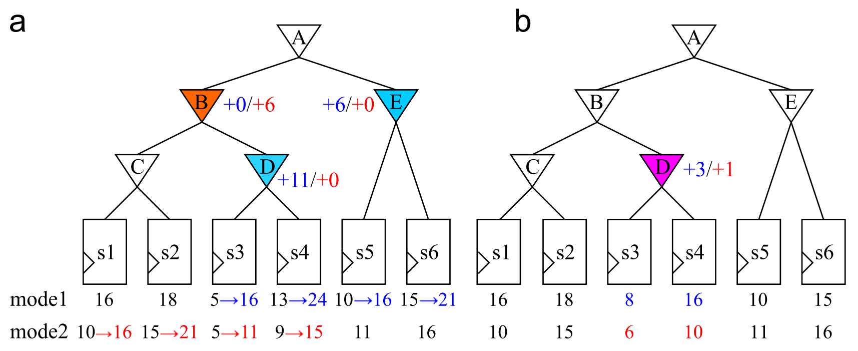


Fig. 3. A motivational example for ADB allocation and delay assignment. (a) A clock tree with clock skew violation when the skew bound is given to 10 units of delay. The initial clock signal arrival times are shown in black numbers. With optimization methods that rely on earliest arrival time synchronization (i.e., [16]), three ADBs are allocated in place of buffers B, D and E. The adjusted arrival times are shown in red or blue numbers. (b) An optimal allocation which uses one ADB. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

$H_{b2}=\{b5,s5,s6\}$, $\alpha_{b2,1}=\max\{\alpha_{b5,1},\alpha_{s5,1},\alpha_{s6,1}\}=\max\{0,0,1\}=1$ and $\alpha_{b2,2}=\max\{\alpha_{b5,2},\alpha_{s1,2},\alpha_{s2,2}\}=\max\{0,0,0\}=0$. (See $b2$ in Fig. 4 (c).)

At this stage, from node n_i where its α values are set, we recursively perform delay-resetting on every descendant, n_k, of n_i by calling function Readjust described in Fig. 5. Readjust subtracts $\alpha_{n_i,m}$ from the sum of delays on each path from a child of n_i to its descendent sink, or set to 0 if $\alpha_{n_i,m}$ is bigger than the previous sum of delays. For example, $\alpha_{b4,1}=3-\min\{1,3\}=2$ and $\alpha_{b4,2}=2-\min\{0,2\}=2$. Fig. 4 (c) shows the results of delay readjustment when the delay value of $b2$ is computed by Eq. (3). Subtree T_{b3} is processed likewise. After all the nodes are processed, ADB-Pullup reports the result of ADB insertion with the updated arrival times as shown in Fig. 4(d).

The flow of ADB-Pullup is depicted in Fig. 5. In the initialization phase, the $\alpha_{n_i,m}$ value of each sink n_i is assigned to the minimum value by which $arr_{n_i,m}+\alpha_{n_i,m}$ is not shorter than $lst_{root,m}-\kappa_m$. This fixes the skew violations by assuming the allocation of a fictitious ADB to each sink. The next phase is "pulling up" these ADBs to non-sink locations of the clock tree, by performing Pullup operation in a topological order. Consider a non-sink node n_i to be processed in the flow. Each child, n_k, of n_i, is checked to see if an ADB is needed according to the evaluation of $\alpha_{n_k,m}>lst_{root,m}-\ lst_{n_i,m}$. If the evaluation is true, an ADB is inserted to n_k, otherwise, the maximum α value (initially 0) to be assigned to n_i is updated if needed. Once the process Pullup at the bottom loop in Fig. 5 is done, the α values at the descendants of n_i are recursively reset according to function Readjust. The time complexity of ADB-Pullupis bounded by $O(KN\log\ N)$ where K is the number of power modes and N is the number of nodes of the input clock tree. Since K is usually very small, the complexity is reduced to $O(N\log N)$. The detailed derivation of the time complexity of ADB-Pullup is the following: $O(N)$ time is taken to sort nodes in topological order and $O(KN)$ time to compute the lst values of all nodes. Likewise, $O(KN)$ time is taken to assign the α values for all leaf nodes and $O(KN)$ to the determination of ADB placement to nodes. Finally, $O(K\log\ N)$ time is taken for each call (i.e., each node) to Readjust function since the number of ancestors of a node is $O(\log\ N)$, resulting in the total time of $O(KN\log\ N)$, which is the most dominant in the steps of ADB-Pullup.

The following summarizes the properties and theorems of ADB-Pullup.

Property 1. *The arrival times at sinks produced by* ADB-Pullup *never exceed $lst_{root,m}$ for every mode m.*

Proof. For simplifying notations, we drop the power mode symbol m in the presentation of the proofs. For a power mode, let $L^{n_l}_{n_k\to s_j}$ be the sum of the α values of the nodes on the path from n_k to a sink s_j which is on the subtree rooted at n_k after Readjust is applied to n_l and $\alpha^{init}_{n_k}$ be the α value of n_k before the application of Readjust to its parent. ($L^{s_j}_{s_j\to s_j}=\alpha^{init}_{s_j}$ since Readjust is not applicable to sinks.)

We claim that the following inequality is hold:

$$L^{n_i}_{n_i\to s_j}+arr_{s_j}\le lst_{root}. \tag{4}$$

We use induction in terms of the depth, D, of the subtree rooted at n_i.

<u>*Basis*</u>: $D=1$ corresponds to the case where n_i is a sink, which means s_j and n_i in Eq. (4) are identical. Thus, $L^{n_i}_{n_i\to s_j}+arr_{s_j}=\alpha^{init}_{s_j}+arr_{s_j}$. By Eq. (1), $\alpha^{init}_{s_j}+arr_{s_j}\le lst_{root}$.

<u>*Induction step*</u>: By the induction hypothesis, for every n_k, it is true that

$$L^{n_k}_{n_k\to s_j}+arr_{s_j}\le lst_{root} \tag{5}$$

where n_i is the parent of n_k and s_j is a sink in the subtree rooted at n_k.

Case 1. When $L^{n_k}_{n_k\to s_j}\ge\alpha^{init}_{n_i}$: after the application of Readjust to n_i, $L^{n_i}_{n_k\to s_j}=L^{n_k}_{n_k\to s_j}-\alpha^{init}_{n_i}$. Thus, $L^{n_i}_{n_i\to s_j}+arr_{s_j}\ =L^{n_i}_{n_k\to s_j}+\alpha^{init}_{n_i}+arr_{s_j}\ =L^{n_k}_{n_k\to s_j}-\alpha^{init}_{n_i}+\alpha^{init}_{n_i}+arr_{s_j}\ =L^{n_k}_{n_k\to s_j}+arr_{s_j}\le lst_{root}$ by Eq. (5).

Case 2. When $L^{n_k}_{n_k\to s_j}<\alpha^{init}_{n_i}$: by Eq. (2), after the application of Readjust to n_i,

$$L^{n_i}_{n_i\to s_j}=\alpha^{init}_{n_i}, \tag{6}$$

and according to Eq. (1) and the definition of lst_{n_i},

$$\alpha^{init}_{n_i}\le lst_{root}-lst_{n_i}\le lst_{root}-arr_{s_j}. \tag{7}$$

Therefore, $L^{n_i}_{n_i\to s_j}+arr_{s_j}=\alpha^{init}_{n_i}+arr_{s_j}\le lst_{root}$.

From Cases 1 and 2, Eq. (5) holds for any n_i. Thus, Property 1 holds for $root$ as well.□

Note that some clock trees have no solution of ADB allocation. For example, consider a simple clock tree shown in Fig. 6 with clock skew bound $\kappa=10$. The clock arrival times 13 at sink $s2$ and 2 at sink $s3$ cause the clock skew violation. However, it can be easily seen that whatever ADB allocations are attempted to A or B, it is not possible to resolve the skew violation. We formally classify the input clock trees into ADB-solvable or ADB-unsolvable as follows:

Definition 1. It is said that a clock tree T with κ is **ADB-unsolvable** if there is a node $n_i\in T$ such that $lst_{n_i}-arr^{min}_{S(n_i)}>\kappa$ in which $S(n_i)$ is the set of sinks which are directly connected to n_i, $arr^{min}_{S(n_i)}$ is the minimum among the arrival times of sinks in $S(n_i)$, and $arr^{min}_{S(n_i)}=\infty$ if $S(n_i)=\phi$. For example, in Fig. 6, $S(A)=\{s_3,s_4\}$ and $lst_A-arr^{min}_{S(A)}=13-2=11>\kappa(=10)$. Thus, the clock tree in Fig. 6 is said to be ADB-

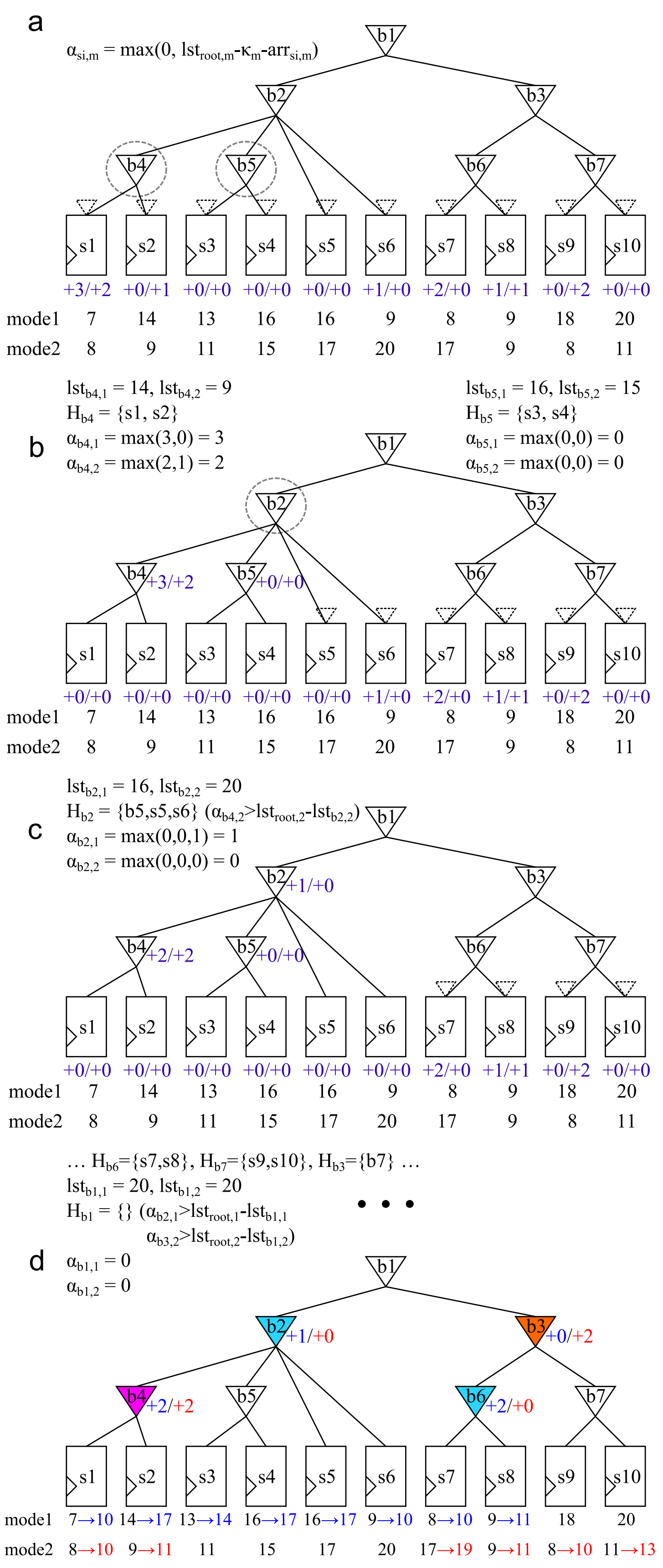
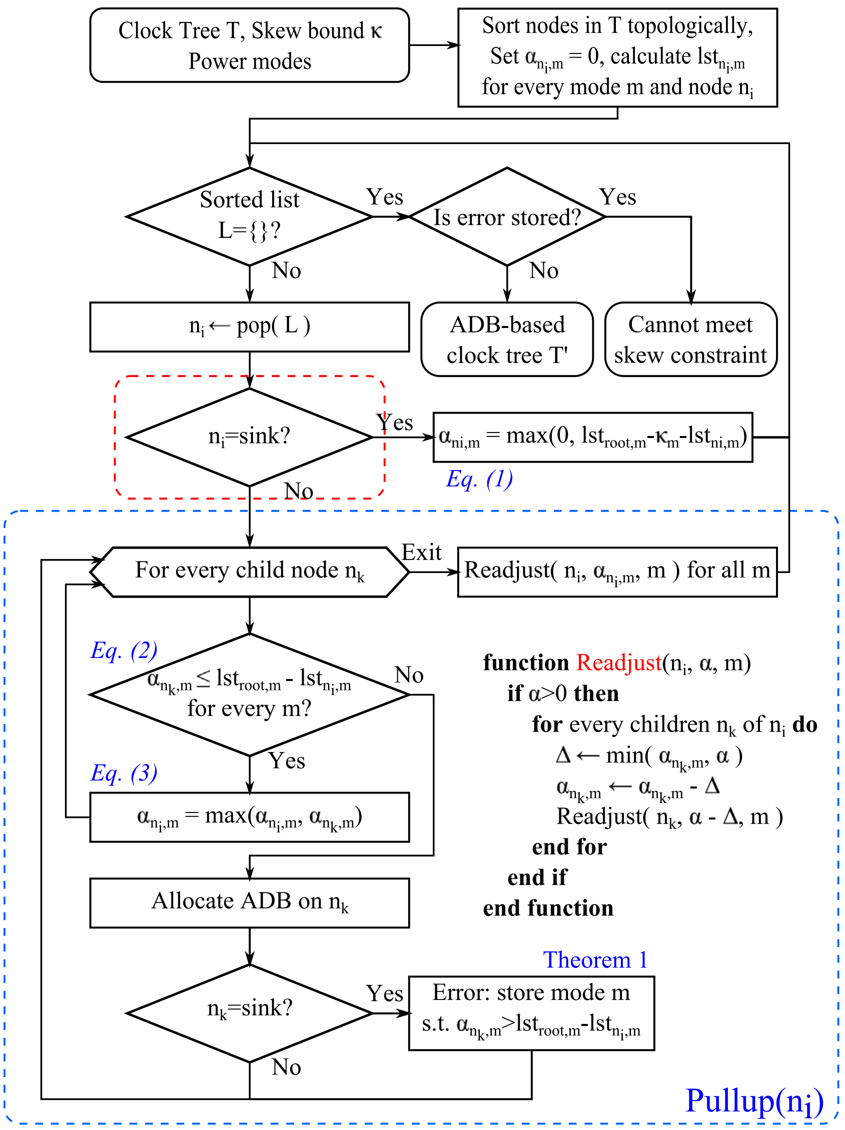

a
αsi,m = max(0, lstroot,m-κm-arrsi,m)
+3/+2 +0/+1 +0/+0 +0/+0 +0/+0 +1/+0 +2/+0 +1/+1 +0/+2 +0/+0
mode1 7 14 13 16 16 9 8 9 18 20
mode2 8 9 11 15 17 20 17 9 8 11
lstb4,1 = 14, lstb4,2 = 9
Hb4 = {s1, s2}
αb4,1 = max(3,0) = 3
αb4,2 = max(2,1) = 2
lstb5,1 = 16, lstb5,2 = 15
Hb5 = {s3, s4}
αb5,1 = max(0,0) = 0
αb5,2 = max(0,0) = 0
b
lstb2,1 = 16, lstb2,2 = 20
Hb2 = {b5,s5,s6} (αb4,2>lstroot,2-lstb2,2)
αb2,1 = max(0,0,1) = 1
αb2,2 = max(0,0,0) = 0
c
… Hb6={s7,s8}, Hb7={s9,s10}, Hb3={b7} …
lstb1,1 = 20, lstb1,2 = 20
Hb1 = {} (αb2,1>lstroot,1-lstb1,1 αb3,2>lstroot,2-lstb1,2)
d
αb1,1 = 0
αb1,2 = 0
mode1 7→10 14→17 13→14 16→17 16→17 9→10 8→10 9→11 18 20
mode2 8→10 9→11 11 15 17 20 17→19 9→11 8→10 11→13

Fig. 4. Example showing step-by-step procedure of ADB-PULLUP: (a) A clock tree T before the ADB insertion by ADB-PULLUP with $\kappa = 10$; allocating $\alpha_{n_i,m}$ for each sink n_i and mode m. (b) After the process of clock subtrees rooted at $b4$ and $b5$. (All children n_k of each subtree rooted at n_i satisfy $\alpha_{n_k,m} \le lst_{root,m} - lst_{n_i,m}$ for all modes. Thus, no ADB is inserted.) (c) After the process of clock subtree rooted at $b2$. ($\alpha_{b4,2} > lst_{root,2} - lst_{b2,2}$, thus, an ADB is inserted at $b4$.) (d) The complete subtree T after the ADB insertion by ADB-PULLUP. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)


Clock Tree T, Skew bound κ
Power modes
Sort nodes in T topologically, Set $\alpha_{n_i,m} = 0$, calculate $lst_{n_i,m}$ for every mode m and node n_i
Sorted list L={}?
Yes
No
Is error stored?
Yes
No
$n_i \leftarrow$ pop(L)
ADB-based clock tree T'
Cannot meet skew constraint
n_i=sink?
Yes
No
$\alpha_{ni,m} = \max(0, lst_{root,m}-\kappa_m-lst_{ni,m})$
Eq. (1)
For every child node n_k
Exit
Readjust(n_i, $\alpha_{n_i,m}$, m) for all m
Eq. (2)
$\alpha_{n_k,m} \le lst_{root,m} - lst_{n_i,m}$ for every m?
No
Yes
Eq. (3)
$\alpha_{n_i,m} = \max(\alpha_{n_i,m}, \alpha_{n_k,m})$
Allocate ADB on n_k
n_k=sink?
Yes
No
Theorem 1
Error: store mode m s.t. $\alpha_{n_k,m} > lst_{root,m}-lst_{n_i,m}$
Pullup(n_i)

```
function Readjust($n_i$, α, m)
  if α>0 then
    for every children $n_k$ of $n_i$ do
      $\Delta \leftarrow \min(\alpha_{n_k,m}, \alpha)$
      $\alpha_{n_k,m} \leftarrow \alpha_{n_k,m} - \Delta$
      Readjust( $n_k$, α - Δ, m )
    end for
  end if
end function
```

Fig. 5. The flow of ADB-Pullup.

unsolvable. For the clock trees in which there is no node $n_i \in T$ such that $lst_{n_i} - arr^{min}_{S(n_i)} > \kappa$, they are said to be **ADB-solvable**.

Theorem 1. *ADB-Pullup allocates ADBs on sinks if and only if the input clock tree is ADB-unsolvable.*

Proof. (⇒) If an ADB is allocated at s_k, which is a child of n_i,

$$\alpha^{init}_{s_k} > lst_{root} - lst_{n_i} \tag{8}$$

for some mode m by Eq. (2).

Since $\alpha^{init}_{s_k} > lst_{root} - lst_{n_i} \ge 0$, Eq. (1) implies

$$\alpha^{init}_{s_k} = lst_{root} - arr_{s_k} - \kappa. \tag{9}$$

Clearly,

$$arr^{min}_{S(n_i)} \le arr_{s_k}. \tag{10}$$

By Eq. (10), $lst_{n_i} - arr^{min}_{S(n_i)} \ge lst_{n_i} - arr_{s_k}$ and by Eq. (9), $lst_{n_i} - arr_{s_k} = lst_{n_i} + \alpha^{init}_{s_k} - lst_{root} + \kappa$, which is greater than κ by Eq. (8). Thus, $lst_{n_i} - arr^{min}_{S(n_i)} > \kappa$.

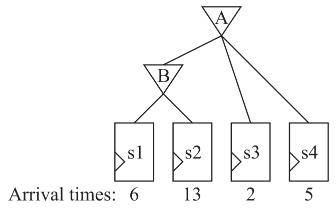


Fig. 6. An example of clock tree that belongs to *ADB-unsolvable* when clock skew bound $\kappa = 10$.

(⇐)Since the clock tree is ADB-unsolvable, there is n_i such that $lst_{n_i} - arr^{min}_{S(n_i)} > \kappa$. Moreover, since $arr^{min}_{S(n_i)} < \infty$, $S(n_i) \ne \phi$. Let $s_k \in S(n_i)$ such that $arr^{min}_{S(n_i)} = arr_{s_k}$. Then, $lst_{n_i} - arr_{s_k} = lst_{n_i} - arr^{min}_{S(n_i)} > \kappa$.

Thus, by Eq. (1), $\alpha^{init}_{s_k} \ge lst_{root} - arr_{s_k} - \kappa > lst_{root} - lst_{n_i}$, which enables the allocation of ADB at s_k according to Eq. (2).□

Note that Property 1, which is a feature that enables to keep the total size of capacitor banks in ADBs within a certain limit, does not hold for the previous ADB allocation algorithms. In addition, Theorem 1 indicates that if there is at least one solution, ADB-Pullup will always find an ADB allocation solution such that the α values of all sinks are 0.

To facilitate the proof of the optimality of our proposed algorithm, we define terms ADB-FREE-PATH and EST-NA$_{i,m}$, and provide one lemma.

Definition 2. If the path from node n (exclusive) to a sink r (exclusive) in a clock tree contains no ADB, the path is called ADB-FREE-PATH and the sink is said to has ADB-FREE-PATH from n.

Definition 3. EST-NA $_{n_i,m}$ represents the earliest one among the arrival times to the sinks which have ADB-FREE-PATH from n_i in power mode m. EST-NA $_{n_i,m} = \infty$ if there exists no such sink.

Lemma 1. *During the process of ADB-PULLUP, if $\alpha^{init}_{n_i,m} > 0$ for a power mode m, a sink exists in subtree T_{n_i} such that the arrival time at the sink is $lst_{root,m} - \kappa_m - \alpha^{init}_{n_i,m}$ and the sink has ADB-FREE-PATH from n_i.*

Proof. Let h denote the height of T_{n_i}. If $h=1$, T_{n_i} has only sinks as the children, Thus, the lemma holds. Let us assume that the lemma is true for $h<L$. We now want to show that the lemma is true for $h=L$: By Eq. (3), if $\alpha^{init}_{n_i,m} > 0$ for any n_i with its height of L, there exists a child node n_{k_j} of H_{n_i} such that $\alpha^{init}_{n_{k_j},m} = \alpha^{init}_{n_i,m}(>0)$. Since $T_{n_{k_j}}$ has a sink whose arrival time is $lst_{root,m} - \kappa_m - \alpha^{init}_{n_{k_j},m}$ on ADB-FREE-PATH from n_i, T_{n_i} also has a sink on ADB-FREE-PATH and arrival time of $lst_{root,m} - \kappa_m - \alpha^{init}_{n_i,m}$.□

Theorem 2. *After the application of ADB-PULLUP to n_i in clock tree T, the resulting subtree T_{n_i} has been allocated with a minimum number of ADBs while meeting the clock skew constraint for T_{n_i}.*

Proof. The proof of the optimality of ADB-PULLUP involves "cut-and-paste" argument. Let $N(T_{n_i})$ denote the number of ADBs in subtree T_{n_i} except the root n_i. Let $X_{n_i} = 1$ if node n_i has an ADB, and $X_{n_i} = 0$, otherwise.

We want to show that (1) $N(T_{n_i})$ is the smallest number among those of all feasible ADB allocations on the subtree rooted at n_i and (2) for each power mode m, T_{n_i} has the largest value of EST-NA $_{n_i,m}$ among those of all feasible ADB allocations with the number of ADBs$=N(T_{n_i})$. For example, if $(N(T_{n_i}), \text{EST}-\text{NA}_{n_i,m}) = (4, 10)$, other feasible solutions could be $(5, 12)$, $(5, 8)$, $(4, 11)$, and $(4, 10)$, but will not be $(3, 12)$ or $(4, 9)$. Let h be the height of T_{n_i}.

- When $h=1$: all children of n_i are sinks. Thus, $N(T_{n_i}) = 0$, which is trivially solvable, and $\text{EST}-\text{NA}_{n_i,m} = \infty$ for every power mode.
- When $h=L$: let us assume this theorem holds for $h<L$. If the theorem is not true for $h=L$, there is a subtree T'_{n_i} produced by an ADB allocation such that (1) $N(T'_{n_i}) < N(T_{n_i})$ or (2) $N(T'_{n_i}) = N(T_{n_i})$ and EST-NA $'_{n_i,m} > \text{EST}-\text{NA}_{n_i,m}$ for some mode m.
  We want to prove that T'_{n_i}, which meets the above condition, dose not exist, following the order illustrated in Fig. 7. In Fig. 7(a), the clock trees T_{n_i} and T'_{n_i} with ADBs are given. We will generate T_{n_i}'' as shown in Fig. 7(b) by replacing one child subtree $T'_{n_{k_j}}$ of T'_{n_i} with $T_{n_{k_j}}$ corresponding to subtree of T_{n_i}. For all cases, we show that (∗) $N(T''_{n_i}) < N(T'_{n_i})$, or $N(T''_{n_i}) = N(T'_{n_i})$ and $\text{EST}-\text{NA}_{n_i,m}'' \geq \text{EST}-\text{NA}'_{n_i,m}$ for every mode m.
  As shown in Fig. 7(d), we can replace every child subtree $T'_{n_{k_j}}$ with $T_{n_{k_j}}$. Let $T^f_{n_i}$ denote the ADB allocation tree produced by the process of replacement. Clearly, $T^f_{n_i}$ satisfies $N(T^f_{n_i}) < N(T'_{n_i})$, or $N(T^f_{n_i}) = N(T'_{n_i})$ and $\text{EST}-\text{NA}^f_{n_i,m} \geq \text{EST}-\text{NA}'_{n_i,m}$ for every mode m. However, $T^f_{n_i}$ has the same values of N and EST-NA as those of T_{n_i}, contradicting the assumption that $N(T_{n_i}') < N(T_{n_i})$, or $N(T'_{n_i}) = N(T_{n_i})$ and $\text{EST}-\text{NA}'_{n_i,m} > \text{EST}-\text{NA}_{n_i,m}$ for some power mode m.
  Now, we prove (∗): We use the fact that the theorem holds for $h<L$, (1) $N(T_{n_{k_j}}) < N(T'_{n_{k_j}})$ or (2) $N(T_{n_{k_j}}) = N(T'_{n_{k_j}})$ and $\text{EST}-\text{NA}_{n_{k_j},m} \geq \text{EST}-\text{NA}'_{n_{k_j},m}$ for every power mode.
  Case1: When $N(T_{n_{k_j}}) < N(T'_{n_{k_j}})$:

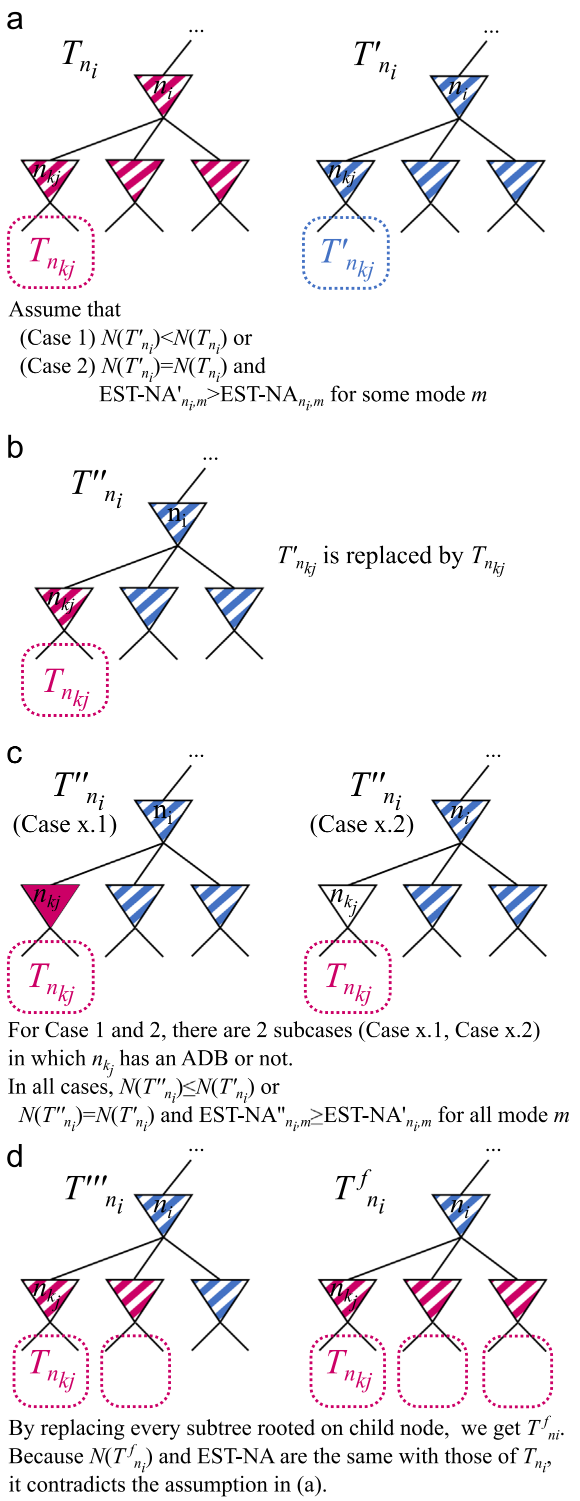

a
T_{n_i}
T'_{n_i}
n_i
n_{kj}
$T_{n_{kj}}$
$T'_{n_{kj}}$
Assume that
(Case 1) $N(T'_{n_i})<N(T_{n_i})$ or
(Case 2) $N(T'_{n_i})=N(T_{n_i})$ and
EST-NA'$_{n_i,m}$>EST-NA$_{n_i,m}$ for some mode m
b
T''_{n_i}
$T'_{n_{kj}}$ is replaced by $T_{n_{kj}}$
c
T''_{n_i}
(Case x.1)
(Case x.2)
For Case 1 and 2, there are 2 subcases (Case x.1, Case x.2)
in which n_{k_j} has an ADB or not.
In all cases, $N(T''_{n_i})≤N(T'_{n_i})$ or
$N(T''_{n_i})=N(T'_{n_i})$ and EST-NA''$_{n_i,m}$≥EST-NA'$_{n_i,m}$ for all mode m
d
T'''_{n_i}
$T^f_{n_i}$
By replacing every subtree rooted on child node, we get T^f_{ni}.
Because $N(T^f_{n_i})$ and EST-NA are the same with those of T_{n_i},
it contradicts the assumption in (a).

Fig. 7. The overall sketch of our proof on Theorem 2.

Case 1.1: When n_{k_j} $(=n_{k_j}'')$ has an ADB: $N(T''_{n_i}) = N(T'_{n_i}) - (N(T'_{n_{k_j}}) + X'_{n_{k_j}}) + (N(T_{n_{k_j}}) + X_{n_{k_j}}) < N(T'_{n_i}) - X_{n_{k_j}} + 1 \leq N(T'_{n_i}) + 1$. Thus, $N(T_{n_i}'') \leq N(T'_{n_i})$. Moreover, since all sinks with ADB-FREE-PATH in T''_{n_i} are also in T'_{n_i}, $\text{EST}-\text{NA}''_{n_i,m} \geq \text{EST}-\text{NA}'_{n_i,m}$ for every m.
Case 1.2: When n_{k_j} has no ADB:
Since $X_{n_{k_j}} = 0$, $N(T_{n_i}'') = N(T'_{n_i}) - (N(T'_{n_{k_j}}) + X'_{n_{k_j}}) + (N(T_{n_{k_j}}) + X_{n_{k_j}}) < N(T'_{n_i})$.

Case 2: When $N(T_{n_{k_j}}) = N(T'_{n_{k_j}})$ and $\text{EST}-\text{NA}_{n_{k_j},m} \ge \text{EST}-\text{NA}'_{n_{k_j},m}$:
Case 2.1: When n_{k_j} has an ADB:
Since $\alpha^{init}_{n_{k_j},m} > 0$ for a power mode m, by Lemma 1, $lst_{n_i,m} - \text{EST}-\text{NA}'_{n_{k_j},m} \ge lst_{n_i,m} - \text{EST}-\text{NA}_{n_{k_j},m} = lst_{n_i,m} - (lst_{root,m} - \kappa_m - \alpha^{init}_{n_{k_j},m}) > \kappa_m$. Thus, n'_{k_j} has an ADB. In addition, since the sinks with ADB-FREE-PATH are the same for T'_{n_i} and T''_{n_i}, $N(T''_{n_i}) = N(T'_{n_i}) - (N(T'_{n_{k_j}}) + X'_{n_{k_j}}) + (N(T_{n_{k_j}}) + X_{n_{k_j}}) = N(T'_{n_i})$ and $\text{EST}-\text{NA}_{n_i,m}{''} = \text{EST}-\text{NA}'_{n_i,m}$ for every m.
Case 2.2: When n_{k_j} has no ADB:
$N(T''_{n_i}) = N(T'_{n_i}) - (N(T'_{n_{k_j}}) + X'_{n_{k_j}}) + (N(T_{n_{k_j}}) + X_{n_{k_j}}) = N(T'_{n_i}) - N(T'_{n_{k_j}}) + N(T_{n_{k_j}}) \le N(T'_{n_i})$. Since $T_{n_i}{''}$ and T'_{n_i} have the same sinks with ADB-FREE-PATH except the sinks in their subtrees rooted at n_{k_j}, $\text{EST}-\text{NA}_{n_{k_j},m}{''} \ge \text{EST}-\text{NA}'_{n_{k_j},m}$ for every m. If n'_{k_j} has an ADB, $N(T_{n_i}{''}) < N(T'_{n_i})$. □

By Theorem 2, it claims that T_{root} produced by the application of ADB-PULLUP uses a minimal number of ADBs while meeting the clock skew constraint.

4.2. Supporting discrete ADB delay

By slightly updating the computation of α values in ADB-PULLUP, it is possible to the ADB allocation with ADBs of discrete delay increments. (We call our updated ADB algorithm ADB-PULLUP-Q.) Precisely, if we want to quantize ADB delay increments with a unit of Q, we simply assign $\alpha_{s_i,m}$ of sink s_i to $\alpha_{s_i,m} = \lceil (lst_{root,m} - \kappa_m - lst_{i,m})/Q \rceil \times Q$ rather than using $lst_{root,m} - \kappa_m - lst_{n_i,m}$ in Eq. (1).

Theorem 3. *If the ADB allocation produced by ADB-PULLUP-Q meets the clock constraint (i.e., if ADB-PULLUP-Q does not return "error"), all the allocated ADBs always use the discrete delay increments.*

A. fter the assignment of the initial α values to every sink s_i, $\alpha_{s_i} = \lceil (lst_{root,m} - \kappa_m - lst_{i,m})/Q \rceil \times Q$, which is a correctly quantized value. In addition, during the PULLUP process at its parent n_i, the value of α_{s_i} may be moved up to n_i when $\alpha_{n_i} \le lst_{root} - lst_{n_i}$. Consequently, α_{n_i} is a correctly quantized value as well. Similarly, during the READJUST process at n_i, the α values of its children are also correctly quantized ones. To generalize, the initial setting and updating of α values in all nodes are correctly quantized values because the outcomes of subtraction, addition, and maximum-selection among the correctly quantized values are also correctly quantized ones.

Note that Theorem 3 does not mean that like to ADB-PULLUP, ADB-PULLUP-Q always finds a valid solution if there exists under the discrete delay of ADBs. We use the following strategy: (1) apply ADB-PULLUP to the input clock tree; (2) if ADB-PULLUP signals "$\alpha > 0$ for some sink", report "the problem is unsolvable" (according to Theorem 1) and stop; (3) apply ADB-PULLUP-Qto the input clock tree; (4) if ADB-PULLUP-Qreturns "no $\alpha > 0$ for any sink", the valid ADB allocation (according to Theorem 3) is found and stop; (5) identify a sink with $\alpha > 0$ and increase its arrival time by Q (based on Property 1) by conducting wire detouring or wire resizing at the sink; (6) if the resulting time at the sink exceeds the longest latency of the initial clock tree, report "fail to find a solution or the problem is unsolvable" and stop; (7) go to (3).

The idea behind this strategy is that since ADB-PULLUP-Q can detect the location where the ADB allocation fails and knows the reason why it fails, by locally tuning the wires at the detected location, the next iteration can be performed with a higher chance of finding a valid solution of ADB allocation.

5. Extension: integration of buffer sizing

We can think of buffer sizing as an ADB allocation imposed by the restriction that the α values in power modes are *pre-defined*. For example, when a buffer b_i in the input clock tree is going to be replaced by a buffer buf_j in the buffer library $\mathcal{L}$ (rather than an ADB), the delay number in each power mode may be increased or decreased, but the number is fixed, which means un-controllable, unlike ADB. Let $\beta^j_{n_i,m}$ be the delay increase or delay decrease in power mode m caused by the replacement of buffer b_i in the input clock tree by $buf_j \in \mathcal{L}$. We can compute all β values from the input clock tree and $\mathcal{L}$. Now, we want to substitute the minimal ADBs determined by ADB-PULLUP(or ADB-PULLUP-Q) with as many buffers in $\mathcal{L}$ as possible to further reduce the number of ADBs to be inserted in the clock tree while still meeting the clock skew constraint for every power mode. Since we have all the β and α values in every node of the clock tree in all power modes, a naive solution is to generate all the combinations of buffer sizing as well as ADB insertion for all nodes, and choose the one that uses the least number of ADBs while meeting the clock skew constraint. However, its computation time grows exponentially as the problem size increases. To be practically feasible, we propose a simple but effective iterative method:

1. For each node n_i in the clock tree, in which ADB-PULLUP (or ADB-PULLUP-Q) has decided that an ADB should be inserted in the node, for each buffer $buf_j \in \mathcal{L}$, we compute
$$\delta^j_{n_i} = \sum_{m=1}^{K} \left(\alpha_{n_i,m} - \beta^j_{n_i,m} \right)^2 \tag{11}$$
where K is the number of modes. For example, if $\alpha_{n_1,1} = +3$, $\alpha_{n_1,2} = +1$, $\beta^1_{n_1,1} = +3$, $\beta^1_{n_1,2} = +2$, $\beta^2_{n_1,1} = +1$, and $\beta^2_{n_1,2} = -1$, then, $\delta^1_{n_1} = (3-3)^2 + (1-2)^2 = 1$ and $\delta^2_{n_1} = (3-1)^2 + (1-(-1))^2 = 8$.
2. Select the pair of node and buffer sizing such that the corresponding δ value is minimal and it satisfies the clock skew and latency constraints. The buffer in the selected node is then resized accordingly. For the previous example, selecting buf_1 is preferred to that of buf_2 for resizing in node n_1 since $\delta^1_{n_1} < \delta^2_{n_1}$. The iteration stops when there is no pair that satisfies the skew and latency constraints or the resizing causes the number of ADBs to increase.
3. Update the arrival times at clock sinks according to the buffer resizing performed in step 2.

Note that the rationale behind the use of δ is that as the smaller the value of δ in a node is, the more the corresponding buffer sizing is likely to close to the ADB that has been inserted to the node, thus, the buffer sizing taking over the role of the ADB with a minimal impact on the overall timing of the clock tree. We call the ADB allocation algorithm combined with buffer sizing ADB-PULLUP-BS for the continuous delay of ADB.

6. Experimental results

The proposed algorithm ADB-PULLUP (continuous delay), ADB-PULLUP-Q (discrete delay), and ADB-PULLUP-BS (combining buffer sizing) have been implemented in Python 3 language on a Linux machine with 8 cores of 3.50 GHz Intel i7 CPU and 16 GB memory. ISCAS'95 and ITC'99 benchmarks were synthesized with *Synopsys IC Compiler* with 45 nm Nangate Open Cell Library. ISPD'09 benchmarks were synthesized using the algorithm in [20]. Each benchmark was partitioned into 6–10 power domains which are

Table 2
Comparison of results produced by CLK-ADB [16], CLK-ADB-RD [16], ADB-Pullup, ADB-Pullup-Q, ADB-Pullup-BS, and ADB-Pullup-QBS. Compared to the results by CLK-ADB [16] and CLK-ADB-RD [16], our proposed algorithm uses, on average under 30–50 ps clock skew bound, 13.5% (=1−0.865) and 15.8% (=(1.070 − 0.901)/1.070) fewer numbers of ADBs for continuous and discrete ADB delays, respectively. In addition, when buffer sizing is integrated, our algorithm uses 31.7% (=1 − 0.683) and 31.3% (=(1.070 − 0.735)/1.070) fewer numbers of ADBs, even reducing the area of ADBs and buffers by 15.0% (=1 − 0.850) and 16.3% (=(1.070 − 0.883)/1.070) for continuous and discrete ADB delays, respectively.

Bench-mark circuit	()FFs/ Bufs	Original skew/lat. (ps)	Skew bound (ps)	Continuous delay						Discrete delay					
				CLK-ADB [16]		ADB-Pullup		ADB-Pullup-BS		CLK-ADB-RD [16]		ADB-Pullup-Q		ADB-Pullup-QBS	
				() ADBs	()Area	()ADBs	Area	()ADBs	Area	()ADBs	Area	()ADBs	Area	()ADBs	Area
			30	27	156.1	25	151.5	20	135.4	42	228.1	29	171.5	25	158.7
s35932	1728/97	264.1/545.1	40	25	147.0	23	140.0	19	126.9	26	151.7	24	146.0	19	129.1
			50	25	144.5	23	137.8	19	124.7	25	145.2	23	139.1	19	125.9
			30	31	212.1	27	196.1	22	180.6	36	235.5	28	202.3	23	186.5
s38417	1564/89	387.1/612.1	40	28	197.9	25	184.6	20	169.0	31	211.8	27	195.4	20	171.3
			50	26	186.5	23	173.1	18	164.4	29	200.8	25	183.9	18	168.1
			30	22	138.3	20	127.3	13	123.2	22	138.2	21	132.8	17	133.8
s38584	1168/66	299.8/552.8	40	18	118.9	16	107.3	11	107.4	21	133.4	20	126.5	16	125.1
			50	18	118.8	16	105.7	11	104.6	18	118.9	16	106.3	11	105.5
			30	29	174.8	25	160.0	19	157.7	35	203.5	29	179.8	24	176.7
B17	1312/89	287.7/654.7	40	26	159.5	22	143.8	15	139.0	30	179.7	24	154.0	16	142.8
			50	26	158.4	22	141.7	15	135.4	26	158.4	22	142.7	15	137.2
			30	150	1010.1	120	896.4	105	849.1	155	1033.8	122	912.9	110	875.3
B18	2752/173	405.1/825.1	40	147	988.9	118	872.3	99	817.0	153	1024.6	119	883.5	101	830.1
			50	144	974.1	118	856.6	94	772.5	149	1003.4	118	861.3	100	810.3
			30	32	202.8	24	171.5	21	191.3	33	207.6	24	172.9	19	160.4
B22	583/42	354.2/690.2	40	32	202.7	24	169.1	21	186.9	32	202.8	24	170.5	21	188.4
			50	31	197.6	24	165.3	21	179.9	32	202.7	24	168.2	21	183.9
			30	13	80.5	13	77.1	11	72.1	13	80.5	13	77.8	12	76.4
F31	273/345	268.8/ 1268.5	40	13	80.5	13	75.8	7	57.2	13	80.5	13	76.5	11	71.4
			50	7	50.9	7	47.4	7	47.4	7	50.9	7	47.5	7	47.5
			30	30	171.8	24	136.9	21	128.5	30	171.8	24	138.3	18	121.9
F34	157/218	211.2/1137.5	40	30	171.5	24	135.1	21	126.3	30	171.8	24	136.5	21	127.7
			50	30	171.5	24	133.3	18	112.9	30	171.5	24	134.8	19	119.4
Average relative values				1	1	0.865	0.894	0.683	0.850	1.070	1.055	0.901	0.928	0.735	0.883

The columns indicated by "Area" represent the sum of the areas of ADBs, ADB control logic and resized buffers in μm^2.

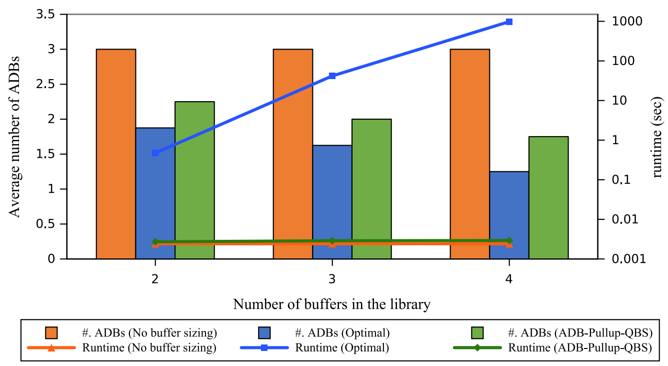


Fig. 8. Comparison of the numbers of ADBs allocated (bar) and run time (line) of ADB-Pullup-Q (orange), exhaustive algorithm (blue), and ADB-Pullup-QBS(green). The run time is in log-scale. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

able to operate in two different supply voltage levels, 0.95 V and 1.1 V.

Table 2 summarizes the results produced by applying CLK-ADB [16] (continuous delay), CLK-ADB-RD (discrete delay)[16], ADB-Pullup, ADB-Pullup-Q, ADB-Pullup-BS and ADB-Pullup-QBS to the benchmark clock trees using four power modes. The columns in the left part of Table 2 represent the number of flip-flop, the number of clock buffers, the worst clock skew, the worst clock latency in the four power modes of the input clock trees, and the clock skew constraint. The columns in the middle part show the results by CLK-ADB [16], ADB-Pullup, ADB-Pullup-BS, ADB-Pullup-QBS. It is observed that ADB-Pullup uses consistently less number of ADBs compared to CLK-ADB. The results shown in the right part indicate that ADB-Pullup-Q uses considerably less ADBs than CLK-ADB-RD. This is because CLK-ADB-RD relies on re-iteration with tighter skew bound when clock skew violation occurs after delay quantization while ADB-Pullup-Q can use quantized delay directly during its bottom-up phase. In addition, ADB-Pullup-BS and

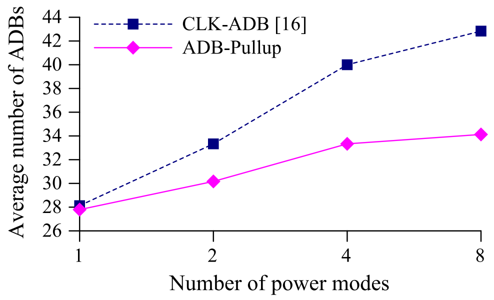


Fig. 9. The changes of the average number of ADBs used by CLK-ADB [16] and ADB-Pullup by varying the number of power modes used.

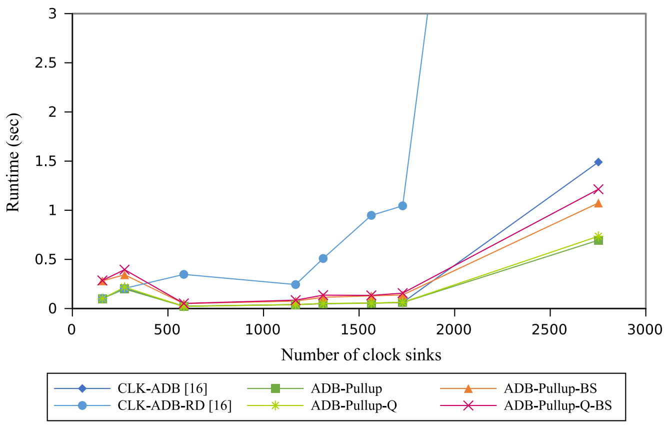


Fig. 10. Run time of CLK-ADB [16], CLK-ADB-RD [16], ADB-Pullup, ADB-Pullup-Q, ADB-Pullup-BS, and ADB-Pullup-QBS. CLK-ADB-RD took about 16 s for a circuit with 2752 sinks.

ADB-Pullup-QBS further reduce the number of ADBs over that of ADB-Pullup and ADB-Pullup-Q.

Additionally, we performed an experiment to check the effectiveness of the proposed algorithm combined with buffer sizing. Fig. 8 shows the results on *ISCAS'89 s382* benchmark circuit, which was synthesized to have 10 clock tree buffers. The experiment was done by varying the clock skew and buffer library, The averaged statistics are shown on the graph in Fig. 8. It is shown that the proposed algorithm clearly uses much fewer number of ADBs over the original result, but uses a little more ADBs than that of the optimal allocation with buffer sizing. However, our run time is very small compared to that of the exhaustive algorithm.

Fig. 9 shows the average numbers of ADBs allocated by CLK-ADB [16] and our ADB-Pullup when the number of modes varies. Clearly, ADB-Pullup always uses less ADBs in all situations. The gap between the results increases as we increase the number of modes used since it is less likely that the ADB allocation in one mode coincides with the allocation in another mode. However, another factor to be considered is that as the number of modes increases, more buffers would be replaced with ADBs, which increases the chance of the coincidence. The actual gap is a complex function of these two factors.

Fig. 10 shows the run time of CLK-ADB [16], CLK-ADB-RD [16], and proposed algorithms. ADB-Pullup and ADB-Pullup-BS take comparable run time with that of CLK-ADB [16], and ADB-Pullup-Q takes a short time compared to that of CLK-ADB-RD [16] because it does not rely on iterations. The run times of ADB-Pullup and ADB-Pullup-Q are theoretically $O(N \log N)$, but they are somewhat arbitrary in practice. This might be because Readjust function does not traverse all the children when an ADB is not allocated and α becomes 0.

7. Conclusions

In this paper, we proposed a polynomial-time optimal algorithm to the problem of ADB allocation on clock trees for the continuous ADB delay. Then, based on the algorithm, we proposed a much simple and predictable solution to the ADB allocation problem for the discrete ADB delay. In addition, we proposed an effective solution to the combined problem of ADB allocation and buffer sizing. From the experimental results on benchmarks, it was shown that compared to the results by the best-known ADB allocation algorithm, our proposed algorithm uses, on average under 30–50 ps clock skew bound, 13.5% and 15.8% fewer numbers of ADBs for continuous and discrete ADB delays, respectively. In addition, when buffer sizing is integrated, our algorithm uses 31.7% and 31.3% fewer numbers of ADBs, even reducing the area of ADBs and buffers by 15.0% and 16.3% for continuous and discrete ADB delays, respectively. Practically, our theoretical outcomes of this work can be applied usefully to the diverse environments (e.g., non-uniform thermal effect) with the dynamically varying clock skew.

Acknowledgments

This research was supported by the ITRC program of ITTP by MSIP (IITP-2015-H8501-15-1005) in Korea, NRF grant funded by the MSIP (2015R1A2A2A01004178), Brain Korea 21 Plus Project in 2015, and the Global Ph.D. Fellowship Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013H1A2A1032650).

References

[1] C.J. Alpert, A. Devgan, S.T. Quay, Buffer insertion with accurate gate and interconnect delay computation, in: Proceedings of IEEE/ACM Design Automation Conference, 1999, pp. 479–484.
[2] J. Cong, C. Koh, K. Leung, Simultaneous buffer and wire sizing for performance and power optimization, In: Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design, 1996, pp. 271–276.
[3] C.C.N. Chu, M.D.F. Wong, An efficient and optimal algorithm for simultaneous buffer and wire sizing, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 18 (9) (1999) 1297–1304.
[4] I.-M. Liu, T.-L. Chou, A. Aziz, M.D.F. Wong, Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion, In: Proceedings of IEEE International Symposium on Physical Design, 2000, pp. 33–38.
[5] T. Okamoto, J. Cong, Buffered Steiner tree construction with wire sizing for interconnect layout optimization, In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 1996, pp. 44–49.
[6] J.-L. Tsai, T.-H. Chen, C.-P. Chen, Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing, IEEE Trans. Comput.-Aided Des. Int. Circuits Syst. 23 (April (4)) (2004) 565–572.
[7] K. Wang, Y. Ran, H. Jiang, M. Marek-Sadowska, General skew constrained clock network sizing based on sequential linear programming, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 24 (May (5)) (2005) 773–782.
[8] S. Hu, J. Hu, Unified adaptivity optimization of clock and logic signals, In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 125–130.
[9] V. Khandelwal, A. Srivastava, Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation, In: Proceedings of IEEE International Symposium on Physical Design, 2007, pp. 11–18.
[10] J.-L. Tsai, L. Zhang, Statistical timing analysis driven post-silicon-tunable clock-tree synthesis, In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2005, pp. 575–581.
[11] E. Takahashi, Y. Kasai, M. Murakawa, T. Higuchi, A post-silicon clock timing adjustment using genetic algorithms, In: Symposium on VLSI Circuits, 2003.
[12] S. Tam, S. Rusu, U. Nagarji Desai, R. Kim, J. Zhang, I. Young, Clock generation and distribution for the first IA-64 microprocessor, IEEE J. Solid-State Circuits 35 (November (11)) (2000) 1545–1552.
[13] Y.-S. Su, W.-K. Hon, C.-C. Yang, S.-C. Chang, Y.-J. Chang, Value assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs, In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2009, pp. 535–538.

[14] Y.-S. Su, W.-K. Hon, C.-C. Yang, S.-C. Chang, Y.-J. Chang, Clock skew minimization in multi-voltage mode designs using adjustable delay buffers, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29(12) (2010) 1921–1930.
[15] K.-Y. Lin, H.-T. Lin, T.-Y. Ho, An efficient algorithm of adjustable delay buffer insertion for clock skew minimization in multiple dynamic supply voltage designs, In: Proceedings of IEEE Asia-South Pacific Design Automation Conference, 2011, pp. 825–830.
[16] K.-H. Lim, T. Kim, An optimal algorithm for allocation, placement, and delay assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs, In: Proceedings of IEEE Asia and South-Pacific Design Automation Conference, 2011, pp. 503–508.
[17] J. Kim, D. Joo, T. Kim, An optimal algorithm of adjustable delay buffer insertion for solving clock skew variation problem, in: Proceedings of IEEE/ACM Design Automation Conference, 2013, pp. 1–6.
[18] N.J.A. Kapoor, S.P. Khatri, A novel clock distribution and dynamic de-skewing methodology, In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2004, pp. 626–631.
[19] K. Park, G. Kim, T. Kim, Mixed allocation of adjustable delay buffers combined with buffer sizing in clock tree synthesis of multiple power mode designs, In: Proceedings of IEEE Design Automation & Test in Europe, 2014, pp. 1–4.
[20] T.-Y. Kim, T. Kim, Clock tree synthesis for TSV-based 3D IC designs, ACM Trans. Des. Autom. Electron. Syst. 16 (October (4)) (2011) 481–4821.