An Optimal Algorithm of Adjustable Delay Buffer Insertion for Solving Clock Skew Variation Problem

Juyeon Kim[1]
juyeon@ssl.snu.ac.kr

Deokjin Joo[1]
jdj@ssl.snu.ac.kr

Taewhan Kim[1,2]
tkim@ssl.snu.ac.kr

[1]School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea
[2]Nano Systems Institute (NSI), Seoul National University, Seoul, Korea

ABSTRACT

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 inserted with continuous delay of ADBs* and (2) *enables solving the ADB insertion 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 insertion* to further reduce the ADBs to be used.

1. INTRODUCTION

Clock is one of the most important signals on a chip, 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), 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 *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.

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 dynamically.

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 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, for example 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 the these problems collectively *ADB insertion 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 insertion problem heuristically in a greedy manner. Lin, Lin, and Ho [15] proposed an efficient algorithm of two-stage approach which performs a top-down ADB insertion 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. Lim and Kim [16] proposed a linear-time algorithm for the ADB insertion problem where they solved the problem optimally for *each* power mode. However, merely collecting the optimal results on in-

DAC'13, May 29 - June 07 2013, Austin, TX, USA.

dividual power modes does not mean globally optimal for all power modes. In this work, we revisit the ADB insertion 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 inserted for all power modes with continuous delay of ADBs* and (2) *enables solving the ADB insertion 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 insertion problem combined with buffer sizing.*

2. ADB STRUCTURE AND INSERTION OF ADB

Fig. 1 shows the structure of a capacitor bank based implementation of ADB [17]. 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.

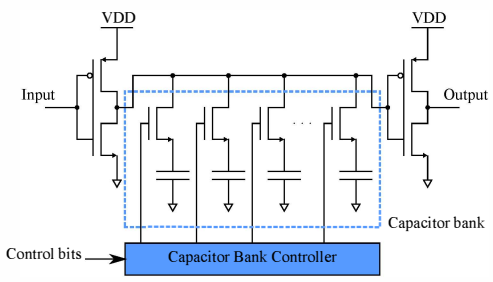

Figure 1: The structure of a capacitor bank based ADB.

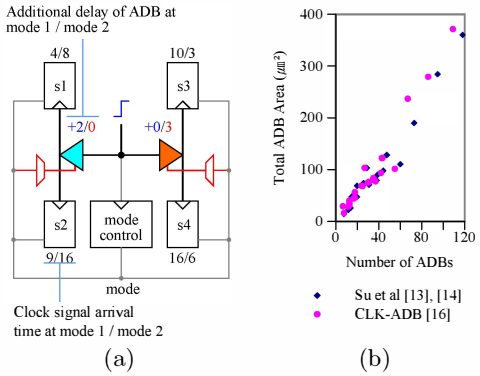

Figure 2: (a) An example of clock tree with the replacement of two clock buffers with ADBs. (b) The relationship between the number of ADBs and the total ADB area (including logic overhead) used by [13, 14, 16].

Fig. 2(a) shows an example of clock tree that has four sinks $s1$, $s2$, $s3$, and $s4$, 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 $s3$ in *mode-2* to 6. To control the ADBs' delay, a mode signal is required. In addition, depending on the implementation of the ADBs, control logic that 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 inserted versus the total ADB area including the overhead, obtained by implementing the algorithms in the previous work [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 insertion problem is to minimize the number of ADBs to be inserted.

3. PROBLEM DEFINITION AND MOTIVATION

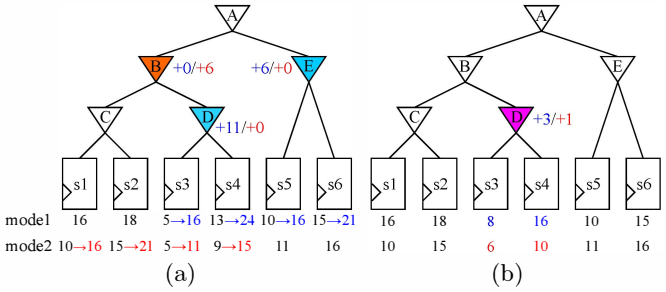

Figure 3: A motivational example for ADB allocation and delay assignment. (a) A clock tree with clock skew violation when skew bound is given to 10 units of delay. The initial clock signal arrival times are shown in black numbers. With optimization methods that rely on earliest arrival time synchronization (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 that uses only one ADB.

PROBLEM 1. ***ADB insertion problem:*** *Given a synthesized clock tree, arrival times of clock sinks in each power mode, and clock skew bound* κ*, replace the least number of clock buffers with ADBs and assign delays to the ADBs to satisfy* κ *in all power modes.*

Two common features of the previous ADB insertion algorithm [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 sub-optimal results that use more ADBs than necessary. For example, Fig. 3 shows a comparison of two results of ADB allocation where Fig. 3(a) corresponds to the result by the sub-optimal ADB allocation algorithm in [16] while Fig. 3(b) corresponds to an optimal result for the same input clock tree as that in Fig. 3(a).

4. ADB INSERTION ALGORITHM

Table 1: Notations

Symbol	Description
$n_{(\cdot)}$	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 of the subtree rooted at node n_i in *mode-m*;
κ	The given clock skew bound to meet;
$\alpha_{n_i,m}$	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.

4.1 The Proposed Optimal 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. (The definitions of the notations used in the presentation are in Table 1.)

Let us consider the clock signal arrival times shown in the clock tree in Fig. 4(a). Let $\kappa = 10$. 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, $i = 1, \cdots, 10$. We compute the delay value by

$$\alpha_{s_i,m} = max\{0,\ lst_{root,m} - \kappa - arr_{s_i,m}\} \qquad (1)$$

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

$$\alpha_{n_k,m} > lst_{root,m} - lst_{n_i,m} \qquad (2)$$

where n_i is the parent node of n_k.

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

$$\alpha_{n_i,m} = max\{\alpha_{n_k,m} : n_k \in H_{n_i}\} \qquad (3)$$

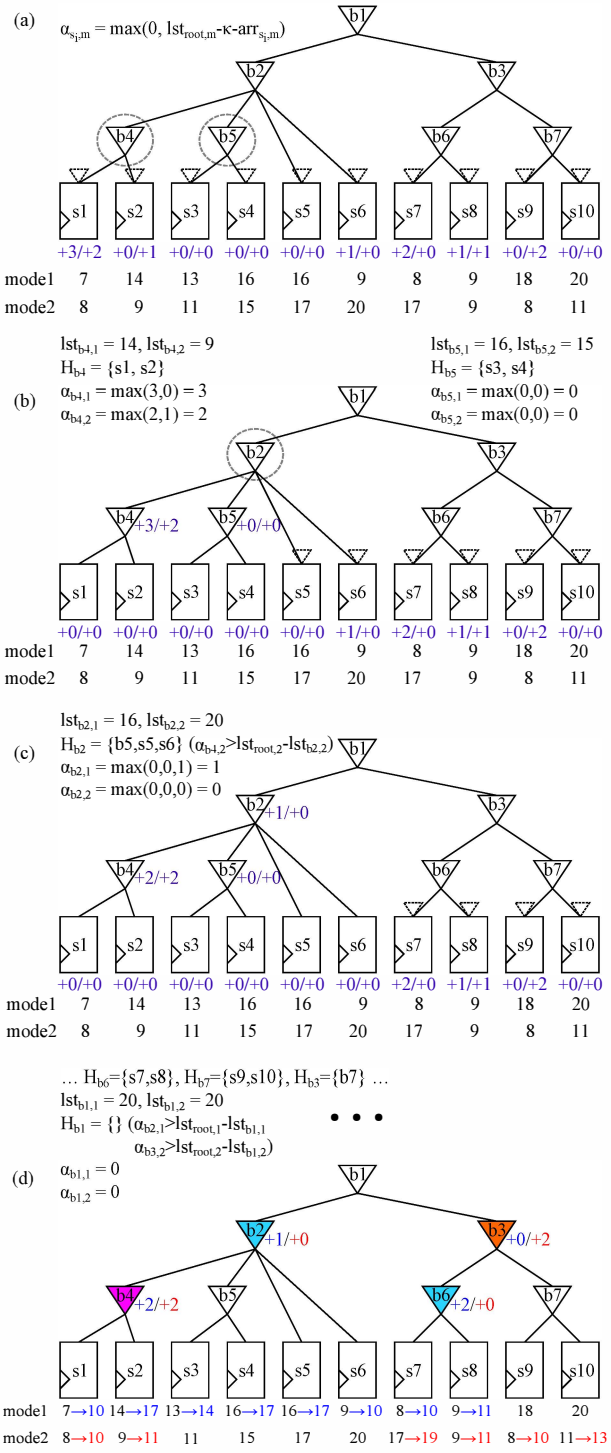

Figure 4: Example showing step-by-step procedure of ADB-PULLUP: (a) A clock tree T before the ADB insertion by ADB-PULLUP with $\kappa = 10$; allocating $\alpha_{n_i,m}$ for each sink n_i and mode m. (b) After the process of clock subtrees rooted at $b4$ and $b5$. (All children n_k of each subtree rooted at n_i satisfy $\alpha_{n_k,m} \leq lst_{root,m} - lst_{n_i,m}$ 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.

where H_{n_i}[1] represents the set of n_i's children that are either sinks or non-sinks, but not the nodes with ADB. For example, since $H_{b2} = \{b5, s5, s6\}$, $\alpha_{b2,1} = \max\{\alpha_{b5,1}, \alpha_{s5,1}, \alpha_{s6,1}\} = max\{0, 0, 1\} = 1$ 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 perform delay-resetting on every child, 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 $lst_{root,m} - \kappa$. 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 re-set according to function READJUST. The time complexity of ADB-PULLUP is 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 following summarizes the properties and theorems of ADB-PULLUP. (All the proofs are left out due to the space limitation.)

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

THEOREM 1. *The result produced by* ADB-PULLUP *has a positive value of* α *in a sink if and only if it is impossible for the given clock tree to meet the clock skew bound with ADB allocation.*

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 insertion algorithms. In addition, Theorem 1 indicates that if there is at least one solution, ADB-PULLUP will always find an ADB insertion solution such that the α values of all sinks are 0.

THEOREM 2. *After the execution of* ADB-PULLUP *on* n_i, *subtree* T_{n_i} *of clock tree* T *rooted at* n_i *has been inserted with a minimum number of ADBs while meeting the clock skew constraint for* T_{n_i}.

By theorem 2, for T_{root} ADB-PULLUP minimally inserts ADBs while meeting the clock skew constraint.

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

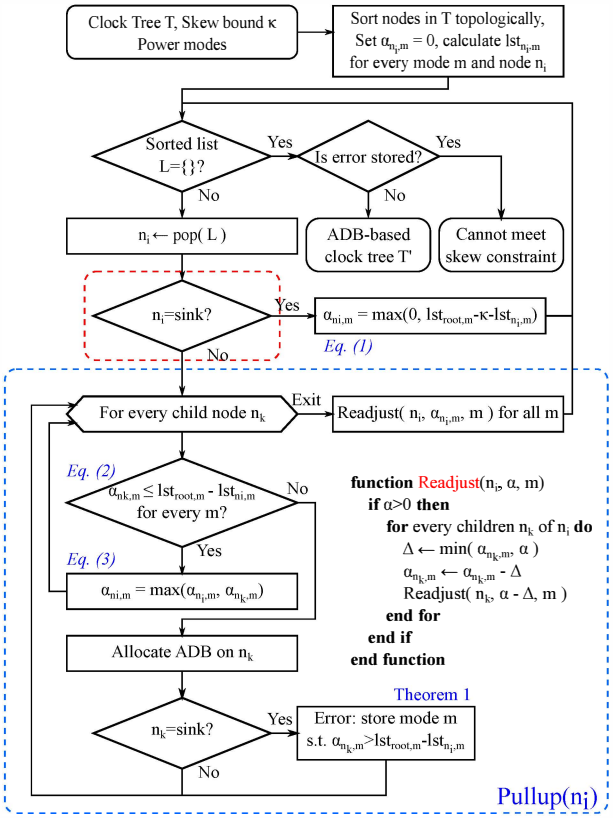

Figure 5: The flow of ADB-PULLUP.

4.2 Supporting Discrete ADB Delay

ADB-PULLUP can be easily fitted to support the discrete delay of ADBs, which we call ADB-PULLUP-Q. If we want to quantize ADB delays with a unit of Q, we simply assign $\alpha_{s_i,m}$ of sink s_i to $\alpha_{s_i,m} = \lceil (lst_{root,m} - \kappa - lst_{i,m})/Q \rceil \times Q$ rather than using $lst_{root,m} - \kappa - lst_{n_i,m}$ in *Eq.(1)*.

THEOREM 3. *If the result produced by* ADB-PULLUP-Q *meets the clock constraint, all the ADBs inserted always use the discrete delays.*

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-Q to the input clock tree; (4) if ADB-PULLUP-Q returns "*no* $\alpha > 0$ *for any sink*", the valid ADB insertion (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 insertion 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 insertion.

5. EXTENSION: INTEGRATION OF BUFFER SIZING

We can think of buffer sizing as an ADB insertion imposed by the restriction that the α values in power modes are predefined. 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

$$\delta^j_{n_i} = \sum_{m=1}^{K} (\alpha_{n_i,m} - \beta^j_{n_i,m})^2 \quad (4)$$

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 insertion 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 16 cores of 2.67Ghz Intel Xeon CPU and 51GB memory. ISCAS'95 and ITC'99 benchmarks were synthesized with *Synopsys IC Compiler* with 45nm Nangate Open Cell Library. ISPD'09 benchmarks were synthesized using the algorithm in [18]. Each benchmark was partitioned into 6 to 10 power domains which are able to operate in two different supply voltage levels, 0.95V and 1.1V.

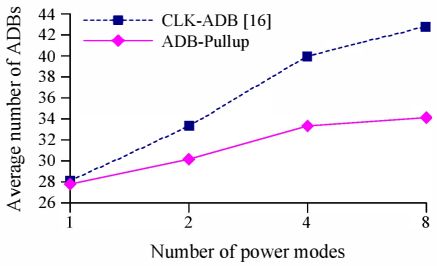

Figure 6: The changes of the average number of ADBs used by CLK-ADB [16] and ADB-Pullup by varying the number of power modes used.

Table 2 summarizes the results produced by applying CLK-ADB [16] (continuous delay), CLK-ADB-RD (discrete delay) [16], ADB-Pullup, ADB-Pullup-Q and ADB-Pullup-BS 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. It is observed that ADB-Pullup uses consistently less number of ADBs compared to CLK-ADB. In addition, ADB-Pullup-BS further reduces the number of ADBs with a slight area saving over that by ADB-Pullup. The results shown in the right part indicates 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.

Fig. 6 shows the average numbers of ADBs inserted 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 consider 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.

7. CONCLUSIONS

In this paper, we proposed a polynomial-time optimal algorithm to the problem of ADB insertion on clock trees for the continuous ADB delay. Then, based on the algorithm, we proposed a much simple and predictable solution to the ADB insertion problem for the discrete ADB delay. In addition, we proposed an effective solution to the combined problem of ADB insertion and buffer sizing. From the experimental results on benchmarks, it was shown that compared to the results by the best known ADB insertion algorithm, our proposed algorithms reduced the number of ADBs by 13.5%, 15.4%, and 31.6% (15.0% total area reduced) on average when continuous ADB delay, discrete ADB delay, and the integration of buffer sizing were used, respectively.

Table 2: Comparison of results produced by CLK-ADB [16], CLK-ADB-RD [16], ADB-PULLUP, ADB-PULLUP-Q and ADB-PULLUP-BS.

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	
				#ADBs	Area	#ADBs	Area	#ADBs	Area	#ADBs	Area	#ADBs	Area
s35932	1728/97	264.1/545.1	30	27	156.1	25	151.5	20	135.4	42	228.1	25	151.7
			40	25	147.0	23	140.0	19	126.9	26	151.7	23	140.2
			50	25	144.5	23	137.8	19	124.7	25	145.2	23	137.9
s38417	1564/89	387.1/612.1	30	31	212.1	27	196.1	22	180.6	36	235.5	28	200.9
			40	28	197.9	25	184.6	20	169.0	31	211.8	26	189.4
			50	26	186.5	23	173.1	18	164.4	29	200.8	23	173.3
s38584	1168/66	299.8/552.8	30	22	138.3	20	127.3	14	123.2	22	138.2	20	127.4
			40	18	118.9	16	107.3	11	107.4	21	133.4	17	112.0
			50	18	118.8	16	105.7	11	104.6	18	118.9	16	105.8
B17	1312/89	287.7/654.7	30	29	174.8	25	160.0	19	157.7	35	203.5	26	164.7
			40	26	159.5	22	143.8	15	139.0	30	179.7	22	143.9
			50	26	158.4	22	141.7	15	135.4	26	158.4	22	141.8
B18	2752/173	405.1/825.1	30	150	1010.1	120	896.4	104	849.1	155	1033.8	120	897.2
			40	147	988.9	118	872.3	99	817.0	153	1024.6	118	873.1
			50	144	974.1	118	856.6	90	772.5	149	1003.4	118	857.4
B22	583/42	354.2/690.2	30	32	202.8	24	171.5	21	191.3	33	207.6	24	171.7
			40	32	202.7	24	169.1	21	186.9	32	202.8	24	169.3
			50	31	197.6	24	165.3	21	179.9	32	202.7	24	165.5
F31	273/345	268.8/1268.5	30	13	80.5	13	77.1	11	72.1	13	80.5	13	77.2
			40	13	80.5	13	75.8	7	57.2	13	80.5	13	75.9
			50	7	50.9	7	47.4	7	47.4	7	50.9	7	47.4
F34	157/218	211.2/1137.5	30	30	171.8	24	136.9	21	128.5	30	171.8	24	137.0
			40	30	171.5	24	135.1	21	126.3	30	171.8	24	135.3
			50	30	171.5	24	133.3	18	112.9	30	171.5	24	133.5
Average Relative Values				1	1	0.86	0.89	0.68	0.85	1.07	1.05	0.87	0.90

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

8. ACKNOWLEDGMENTS

This work was supported by Basic Science Research Program through National Research Foundation (NRF) grant (No.2011-0029805), the Center for Integrated Smart Sensors funded by the Ministry of Education, Science and Technology as Global Frontier Project (CISS 2011-0031863) and supported by the MKE (Ministry of Knowledge Economy), Korea, under ITRC (Information Technology Research Center) support program supervised by NIPA (National IT Industry Promotion Agency) (NIPA-2012-H0301-12-1011).

9. REFERENCES

[1] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *DAC*, 1999.

[2] J. Cong, C. Koh, and K. Leung, "Simultaneous buffer and wire sizing for performance and power optimization," in *ISLPED*, 1996.

[3] C. C. N. Chu and M. D. F. Wong, "An efficient and optimal algorithm for simultaneous buffer and wire sizing," *IEEE TCAD*, 1999.

[4] I.-M. Liu, T.-L. Chou, A. Aziz, and M. D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion," in *ISPD*, 2000.

[5] T. Okamoto and J. Cong, "Buffered steiner tree construction with wire sizing for interconnect layout optimization," in *ICCAD*, 1996.

[6] J.-L. Tsai, T.-H. Chen, and C.-P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *IEEE TCAD*, 2004.

[7] K. Wang, Y. Ran, H. Jiang, and M. Marek-Sadowska, "General skew constrained clock network sizing based on sequential linear programming," *IEEE TCAD*, 2005.

[8] S. Hu and J. Hu, "Unified adaptivity optimization of clock and logic signals," in *ICCAD*, 2007.

[9] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," in *ISPD*, 2007.

[10] J.-L. Tsai and L. Zhang, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *ICCAD*, 2005.

[11] E. Takahashi, Y. Kasai, M. Murakawa, and 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, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE JSSC*, 2000.

[13] Y.-S. Su, W.-K. Hon, C.-C. Yang, S.-C. Chang, and Y.-J. Chang, "Value assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs," in *ICCAD*, 2009.

[14] ——, "Clock skew minimization in multi-voltage mode designs using adjustable delay buffers," *IEEE TCAD*, 2010.

[15] K.-Y. Lin, H.-T. Lin, and T.-Y. Ho, "An efficient algorithm of adjustable delay buffer insertion for clock skew minimization in multiple dynamic supply voltage designs," in *ASPDAC*, 2011.

[16] K.-H. Lim and 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 *ASPDAC*, 2011.

[17] N. J. A. Kapoor and S. P. Khatri, "A novel clock distribution and dynamic de-skewing methodology," in *ICCAD*, 2004.

[18] T.-Y. Kim and T. Kim, "Clock tree synthesis for TSV-based 3D IC designs," *ACM ToDAES*, 2011.