Buffer Insertion to Remove Hold Violations at Multiple Process Corners

Inhak Han, Daijoon Hyun, and Youngsoo Shin
School of Electrical Engineering, KAIST
Daejeon 34141, Korea

***Abstract*— Buffer insertion to remove hold violations at multiple process corners is addressed for the first time. The problem is formulated as integer linear programming (ILP); it is combined with circuit partitioning heuristic so that larger circuits can also be handled. A heuristic buffer insertion algorithm is then proposed and compared to ILP, which demonstrates only a slight increase of the number of buffers (2.4% on average). Two additional intuitive methods are implemented to demonstrate why new heuristic algorithm is needed: conventional buffer insertion at each process corner one by one and conventional buffer insertion at all process corners simultaneously followed by combining insertion results.**

I. Introduction

Timing analysis of synchronous circuits consists of setup (or long path) analysis and hold (or short path) analysis. Setup violations imply that target clock frequency cannot be achieved; so clock frequency is often lowered to resolve the violations. On the other hand, hold violations cannot be fixed once chips are manufactured, so it is very important to fix all hold violations beforehand [1].

Buffer insertion [2]–[6] is a popular technique to fix hold violations. Some other techniques include gate downsizing, using multiple threshold voltage, using dummy metal to increase load capacitance [2], and clock skew scheduling to adjust clock arrival time [7].

Process variations have become a major factor to affect timing analysis, so static timing analysis is now performed at multiple process corners. In particular, hold constraints have to be checked at the corners incurring large arrival times (like SS corner) as well as those causing small arrival times (such as FF corner) [8]. Consider one data path whose delay is slightly larger than the sum of clock skew and hold time at FF corner; hold constraint is satisfied. The path can however violate hold constraint at SS corner, if the path and clock path consist of large-sized gates and small-sized ones respectively which increases the delay of the path small and clock skew largely when the corner changes from FF to SS. In addition, new process corners are continually added (such as SF and FS corners), and hold constraints should also be checked at them for the same reason. We thus have to solve the problem of buffer insertion at multiple process corners.

Two conventional methods have been used to solve the problem. One way is to insert buffers at each corner one by one [9], and the other is to perform buffer insertion at each corner in parallel and combine the insertion results [10]. These methods usually insert more number of buffers compared with the buffer insertion method which takes into account timing information of all corners at once. Consider Fig. 1 where paths with hold violations at FF corner and those at SS corner are respectively presented by dashed- and thick-line. If hold violations are fixed using conventional buffer insertion, two buffers in front of primary inputs are added at FF corner and two buffers in back of primary outputs are inserted at SS corner; while only three buffers are added when hold violations at all corners are fixed at the same time.

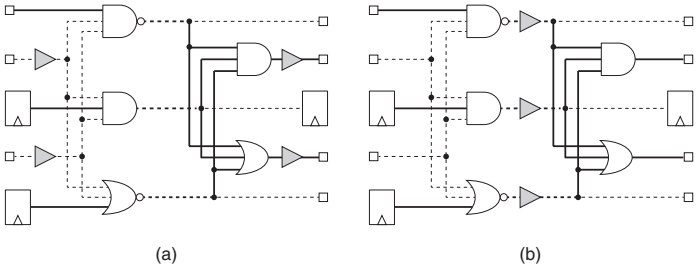

Fig. 1. An example of buffer insertion at two process corners: (a) four buffers are inserted if hold violations are fixed at two corners one by one, and (b) three buffers are inserted if violations are fixed at two corners together.

A. Contributions

Our main contributions can be summarized as follows.

- ILP formulation of buffer insertion to remove hold violations at multiple process corners, and circuit partitioning heuristic to apply ILP to larger circuits (Section II).
- Implementation of two intuitive buffer insertion algorithms, called *Sequential* and *Parallel* (Section III).
- A new buffer insertion algorithm to overcome the limitations of intuitive methods (Section III).

II. ILP Formulation

The input to our buffer insertion problem is a netlist of circuit with timing constraints in a set of process corners $\mathbb{C}$, specifically arrival time (AT) at primary inputs and required arrival time (RAT) at primary outputs. A set of buffer types is denoted by $\mathbb{B}$.

A netlist is modeled by a timing graph $G = (V, E_g \cup E_w, \{d_i\}, \{\delta_{i,j}\})$, where each vertex in V corresponds to a node, E_g is a set of edges representing the timing path from gate input to gate output, and the edge in E_w represents the timing path corresponding to interconnect. An example is shown in Fig. 2, where edges in E_g and edges in E_w

978-1-4673-9569-4/16/$31.00 ©2016 IEEE

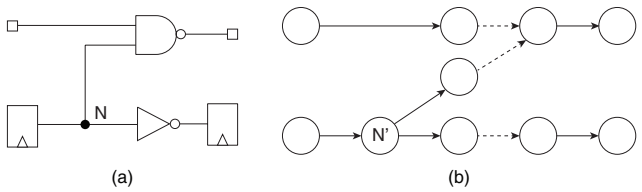

Fig. 2. (a) An example circuit and (b) corresponding timing graph.

are represented by dashed- and solid-arrows, respectively. An edge $e \in E_g$ is associated with a set of delays $\{d_i(e)\}$ with i indicating a corner, i.e. $i \in \mathbb{C}$. An edge $e \in E_w$ is associated with wire delays $\{d_i(e)\}$; it also has a set of candidate buffer delays $\{\delta_{i,j}(e)\}$, where j is the index of buffer type. Multiple fanout node is represented as a vertex, e.g. N and N' in Fig. 2.

For each vertex v corresponding to a primary output, hold RAT and setup RAT at corner i are denoted by $r_i(v)$ and $R_i(v)$, respectively. The earliest AT at v, $a_i(v)$, has to be no smaller than $r_i(v)$ due to hold time constraint, and the latest AT, $A_i(v)$, is not allowed to exceed $R_i(v)$ due to setup time constraint:

$$a_i(v) \geq r_i(v), \tag{1}$$
$$A_i(v) \leq R_i(v). \tag{2}$$

The earliest AT at v is computed iteratively from earliest ATs at all its fanins:

$$a_i(v) = \min_{u \in \text{fanins of } v} [a_i(u) + d_i((u,v))], \tag{3}$$

where (u, v) indicates the edge from u to v. If v is a primary input, its earliest AT is given by $\lambda_i(v)$. Equation (3) is not a proper form for ILP formulation, so it is transformed into

$$a_i(v) \leq a_i(u) + d_i((u,v)). \quad \forall u \in \text{fanins of } v \tag{4}$$

Similarly, the latest AT at v can be described by

$$A_i(v) \geq A_i(u) + d_i((u,v)). \quad \forall u \in \text{fanins of } v \tag{5}$$

The latest AT at a primary input v is given by $\Lambda_i(v)$.

Hold violations may be fixed by adding extra buffers to some edges $e \in E_w$ as long as setup time constraints are not violated. Let the number of j-th type of buffer be denoted by $n_j(e)$; total buffer delay on e at corner i, $\delta_i(e)$, is given by

$$\delta_i(e) = \sum_{j \in \mathbb{B}} \delta_{i,j}(e)\, n_j(e), \tag{6}$$

where $\delta_{i,j}(e)$ denotes the delay of j-th type of buffer. We assume that $\mathbb{B}$ contains buffers in ascending order of delay and so $\delta_{i,1}(e)$ is a minimum value.

The buffer insertion problem can now be defined as the problem to assign $n_j(e)$ for each $e \in E_w$ and each $j \in \mathbb{B}$. The objective is to minimize the total sum of $n_j(e)$ while all hold and setup time constraints, described by (1) and (2), are satisfied.

TABLE I
THE NUMBER OF BUFFERS AND RUNTIME OF ILP AND PARTITIONING+ILP

Circuit	# Cells	ILP		Partitioning+ILP	
		# BUFs	Time (s)	# BUFs	Time (s)
s35932	6910	729	48	729	17
b15	7757	307	97	326	56
ac97_ctrl	12766	1440	302	1440	60
usb_funct	14130	1456	266	1468	73
b17	23593	684	2779	694	164
RC4_dec	44609	4445	25840	4445	217
des3_perf	88841	-	Timeout	5572	574
vga_lcd	113950	-	Timeout	15691	2167

$$\text{Minimize} \quad \sum_{e \in E_w} \sum_{j \in \mathbb{B}} n_j(e)$$

Subject to

$\forall i \in \mathbb{C}$,

$$a_i(v) = \lambda_i(v), \qquad \forall \text{input } v \in V$$
$$A_i(v) = \Lambda_i(v), \qquad \forall \text{input } v \in V$$
$$a_i(v) \leq a_i(u) + d_i((u,v)), \qquad \forall (u,v) \in E_g$$
$$A_i(v) \geq A_i(u) + d_i((u,v)), \qquad \forall (u,v) \in E_g$$
$$a_i(v) \leq a_i(u) + d_i((u,v)) + \delta_i((u,v)), \qquad \forall (u,v) \in E_w$$
$$A_i(v) \geq A_i(u) + d_i((u,v)) + \delta_i((u,v)), \qquad \forall (u,v) \in E_w$$
$$\delta_i(e) = \sum_{j \in \mathbb{B}} \delta_{i,j}(e) \cdot n_j(e), \qquad \forall e \in E_w$$
$$a_i(v) \geq r_i(v), \qquad \forall \text{output } v \in V$$
$$A_i(v) \leq R_i(v), \qquad \forall \text{output } v \in V$$

A. Application to Large Circuits

The application of ILP to some example circuits is demonstrated in Table I. It is inherently limited to small circuits due to large runtime; the circuits of more than 80K gates take more than 12 hours (denoted as Timeout). To extend the application of ILP to some larger circuits, we partition the netlist and apply ILP to each sub-circuit one by one (named *Partitioning+ILP*).

Each primary output with hold violation is visited; its fanin cone is considered as an initial sub-circuit. We then iteratively merge two sub-circuits that share the most number of edges with hold violations (i.e. with negative hold slack); merging is actually executed if the total number of gates in merged sub-circuit will not exceed some given value, so that each merged sub-circuit can be solved using ILP in reasonable runtime. The described merging method is inspired by the fact that as the number of shared edges with negative hold slack (between two candidate sub-circuits) increases, *Partitioning+ILP* becomes worse, i.e. it inserts more buffers than optimal ILP does.

ILP is then applied to each sub-circuit one by one. An example of the process is illustrated in Fig. 3: there are two sub-circuits, fanin cone of A and fanin cone of B; paths with hold violations are represented by thick lines. Sub-circuits are visited in descending order of total negative hold slack; let us

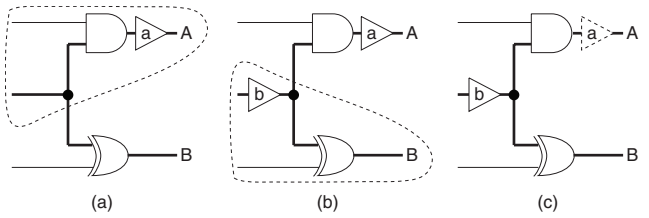

Fig. 3. Partitioning+ILP: (a) buffer insertion in fanin cone of A and (b) buffer insertion in fanin cone of B, and (c) removal of unnecessary buffer a.

assume that the fanin cone of A is visited first. Buffers a and b are added as a result of buffer insertions as shown in Fig. 3(a) and (b). As a post-process, buffer a is removed because all hold violations are fixed with only one buffer b. This post-process is performed by iteratively discarding a buffer with hold slack larger than its own delay.

The result of applying *Partitioning+ILP* to example circuits are shown in the last two columns of Table I. It inserts 6% more buffers in circuit b15 but less than 1.5% in the remainders, even though its runtime is much smaller. Its performance is determined by the number of shared edges with negative hold slack divided by the number of sub-circuits, which is 2.6 in b15 and less than 1.0 in the other circuits.

III. Heuristic Buffer Insertion Algorithms

For industrial-scale large circuits, a fast and scalable buffer insertion algorithm is required. We study two intuitive algorithms for this purpose. In *Sequential* method, buffer insertion is performed at each process corner one by one; some buffers inserted at one corner may be discarded at the others if they cause setup violations. In *Parallel* method, buffer insertion is performed at all corners in parallel, and the insertion results are appropriately combined. We finally propose a new heuristic algorithm to overcome the limitations of two intuitive methods.

A. Sequential Method

The algorithm *Sequential* is presented in Fig. 4, which starts from heuristic buffer insertion at the first corner (L3). The buffer insertion at a single corner, *Buffer_Insertion*, iteratively performs the followings until all hold violations are fixed (L10) [4], [5]. Out of all unmarked edges, the one e_{max} that is shared by most number of paths with hold violations ($SLK_{hd} < 0$) is picked; unmarked edges are those that have setup slack large enough to accommodate extra buffers. If more than two edges are on the same number of paths, we select the edge which has the largest candidate buffer delay (L11). Note that buffer delay varies with edges due to different load capacitance and data transition time, and any type of buffer can be used to find the edge with largest buffer delay. The idea is to fix largest amount of hold violations by inserting one buffer on that edge. We then repeatedly insert one buffer into e_{max} until hold slack of that edge becomes positive while keeping positive setup slack (L12). During the iteration, j-th buffer, where j is minimum that satisfies the buffer delay $\delta_{i,j}(e_{max})$ is larger than the amount of hold violation, is

Algorithm *Sequential* $(G, \mathbb{C})$
L1 **for** each corner $i \in \mathbb{C}$ **do**
L2 *Buffer_Removal* (G, i)
L3 *Buffer_Insertion* (G, i)
L4 *Timing_Update* on G at i, $\forall i \in \mathbb{C}$
L5 **if** $\exists(e, i)$; $SLK_{su} < 0$ or $SLK_{hd} < 0$ at i **then**
L6 **go to** L1

Function *Buffer_Removal* (G, i)
L7 **while** buffers $\{b\}$ with $SLK_{su} < 0$ at i exist **do**
L8 Remove a buffer b, on the least number of paths with $SLK_{hd} <$ delay of b at i
L9 *Timing_Update* on G at i

Function *Buffer_Insertion* (G, i)
L10 **while** unmarked edges $\{e\}$ with $SLK_{hd} < 0$ at i exist **do**
L11 $e_{max} \leftarrow e$ which is shared by most number of paths with $SLK_{hd} < 0$ at i and has maximum $\delta_{i,j}(e)$
L12 **while** $SLK_{hd} < 0$ and $\delta_{i,1}(e_{max}) < SLK_{su}$ at i **do**
L13 Add j-th buffer to e_{max}; j is minimum that satisfies $\delta_{i,j}(e_{max}) > -SLK_{hd}$ while $\delta_{i,j}(e_{max}) < SLK_{su}$
L14 Update SLK_{hd} and SLK_{su}
L15 *Timing_Update* on G at i
L16 *Edge_Marking* (G, i)

Function *Edge_Marking* (G, i)
L17 **for** each edge e; $SLK_{su} < \delta_{i,1}(e)$ at i **do**
L18 Mark e

Fig. 4. *Sequential* algorithm.

inserted; if hold violation cannot be fixed by only one buffer, the buffer with largest delay is added (L13). Finally timing information is updated (L15) and each edge whose setup slack is smaller than minimum buffer delay is marked (L17-L18), which we call *Edge_Marking*, to prevent inserting buffers into the edges at other corners as well as at current corner (L16).

Now we visit the second corner and eliminate setup violations caused by the buffers inserted at the first corner. We iteratively drop one buffer with negative setup slack so that least number of paths with hold violations increase after removing that buffer (L8). The process repeats until all setup violations are fixed (L7), which we call *Buffer_Removal* (L2). *Buffer_Insertion* is then performed (L3); two buffers on the same edge are replaced by another type of buffer if the delay of the alternative buffer is larger than their aggregate delay and does not cause setup violation.

We iteratively go through the same steps performed at the second corner while visiting the remaining corners one by one (L1). After all corners have been visited, we update timing information at all corners (L4), and we start all over again if setup or hold violation exists (L5-L6).

The sequence of visiting corners affects the number of buffers. As we visit the corner with larger setup slack later, the number of dropped buffers tends to be smaller and so the total number of inserted buffers also does. We thus visit corners in ascending order of worst setup slack.

B. Parallel Method

The algorithm of *Parallel* is shown in Fig. 5, where each step executed at all corners is done in parallel. It first marks

Algorithm *Parallel* $(G, \mathbb{C})$
L1 *Edge_Marking* $(G, i),\ \forall i \in \mathbb{C}$
L2 Copy G to $G_i,\ \forall i \in \mathbb{C}$
L3 *Buffer_Insertion* $(G_i, i),\ \forall i \in \mathbb{C}$
L4 **for** each edge e **do**
L5 $e_i \leftarrow$ edge in G_i corresponding to $e,\ \forall i \in \mathbb{C}$
L6 $e_{max} \leftarrow e_i$ with maximum total delay of added buffers
L7 Insert buffers into e s.t. buffers on e = buffers on e_{max}
L8 *Timing_Update* on G at $i,\ \forall i \in \mathbb{C}$
L9 Copy G to $G_i,\ \forall i \in \mathbb{C}$
L10 *Buffer_Removal* $(G_i, i),\ \forall i \in \mathbb{C}$
L11 **for** each edge e **do**
L12 $e_i \leftarrow$ edge in G_i corresponding to $e,\ \forall i \in \mathbb{C}$
L13 $e_{min} \leftarrow e_i$ with minimum total delay of added buffers
L14 Remove buffers on e s.t. buffers on e = buffers on e_{min}
L15 *Timing_Update* on G at $i,\ \forall i \in \mathbb{C}$
L16 **if** $\exists(e, i);\ SLK_{hd} < 0$ at i **then**
L17 **go to** L1

Fig. 5. *Parallel* algorithm.

each edge whose setup slack is too small to accommodate extra buffer at all corners (L1), which helps utilize timing information of other corners when performing *Buffer_Insertion*. The resulting graph is then copied to new graph G_i and *Buffer_Insertion* is performed on G_i at each corner $i \in \mathbb{C}$ (L2-L3).

We now merge the insertion results into G so that G satisfies all hold constraints at all corners, which is done by performing the following steps for each edge e in G (L4). Among all the edges e_i in G_i corresponding to e, the one e_{max} on which the buffers have the largest aggregate delay is picked (L5-L6). Note that each aggregate delay is obtained and compared at one corner and the corner can be arbitrarily chosen. Buffers are then inserted into e so that e has the same buffers as e_{max} does (L7).

Since each edge contains the buffers with largest aggregate delay, the possibility that setup violations occur is high. The violations are removed by copying the resulting graph to G_i and performing *Buffer_Removal* on G_i at each corner i (L9-L10). We then combine the removal results into G so that G satisfies all setup constraints at all corners, which is done by performing the following steps for each edge e in the graph G: selecting the edge e_{min} where the buffers have the smallest aggregate delay among all the edges e_i; and dropping buffers on e so that e has the same buffers as e_{min} does (L11-L14). The algorithm starts all over again, if any hold violation exists; it is finished, otherwise (L15-L17).

C. Proposed Method

The *Sequential* and *Parallel* methods check hold and setup constraints at only a single corner when inserting buffers, so they cannot find the edges to insert buffers for minimizing the number of buffers in the view of whole corners. This is inherent limitation of these methods.

An additional limitation comes from discarded buffers. If some buffers are removed, remaining buffers could exist on undesirable edges because their locations are determined under the assumption that the dropped buffers exist. An example is illustrated in Fig. 6, where all paths violate hold constraint at the first corner, except a dotted path which cannot accommodate extra buffer due to setup constraint at the second corner. In the example circuit, only four buffers are needed to satisfy all timing constraints as shown in Fig. 6(a), while five buffers are inserted via *Sequential* like Fig. 6(b): three buffers are added in front of primary inputs at the first corner and the one buffer surrounded with dotted line is dropped at the second corner during first iteration; and the three buffers with thick lines are inserted at the first corner during second iteration.

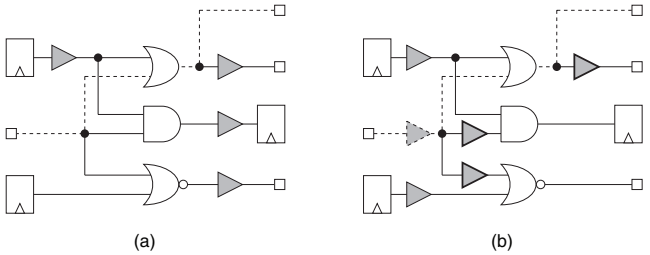

Fig. 6. (a) Buffer insertion with minimum number of buffers and (b) buffer insertion using *Sequential*, where a dotted line represents a path with setup slack smaller than minimum buffer delay at the second corner and all the other paths violate hold constraint at the first corner.

Algorithm *Proposed* $(G, \mathbb{C})$
L1 *Edge_Marking* $(G, i),\ \forall i \in \mathbb{C}$
L2 **for** each unmarked edge e **do**
L3 $p(e) \leftarrow \sum_{i \in \mathbb{C}}$(#paths including e with $SLK_{hd} < 0$ at i)
L4 Insert buffers into the edge e with maximum $p(e)$
L5 *Timing_Update* on G at $i,\ \forall i \in \mathbb{C}$
L6 **if** $\exists(e, i);\ SLK_{hd} < 0$ at i **then**
L7 **go to** L1

Fig. 7. Proposed algorithm.

To overcome these limitations, we propose a new method whose algorithm is presented in Fig. 7. We start from performing *Edge_Marking* at all corners (L1). Each unmarked edge e is then assigned the number $p(e)$ which is the sum of the numbers of paths with hold violations including e over all corners (L2-L3), and we repeatedly insert one buffer into the edge e with the maximum value of $p(e)$ (L4). The insertion repeats until negative hold slack at any coner changes into positive or setup slack at any corner becomes too small to insert extra buffers. If more than two edges have the same $p(e)$, we add buffers on the edge which has the largest buffer delay. The algorithm starts over again, if hold violation exists at any corner; it is finished, otherwise (L6-L7).

IV. Experimental Results

Experiments are carried out using a set of sequential circuits from ISCAS and ITC benchmarks, and also from open cores [12]; they are listed in Table II. Each circuit is synthesized [13] using 28nm commercial library. Placement and clock tree synthesis are performed [14], which are then followed by our buffer insertion implemented in C language. Two process corners (FF and SS) are assumed, which are then combined with interconnect corners to assess buffer insertion

TABLE II
COMPARISON OF THE NUMBER OF BUFFERS AMONG DIFFERENT BUFFER INSERTION METHODS

Circuit	#Cells	ILP	*Partitioning +ILP*	Proposed	*Sequential*	*Parallel*
		#BUFs	#BUFs	ΔBUFs (%)	ΔBUFs (%)	ΔBUFs (%)
s35932	6910	729	-	0.4	2.3	3.4
b14	7393	289	-	4.5	15.2	20.4
b15	7757	307	-	5.9	13.4	17.9
mem_ctrl	7805	671	-	1.3	3.7	9.1
s38584	8711	872	-	1.9	9.6	14.9
b18	9487	632	-	3.8	12.5	15.7
s38417	9579	1148	-	0.4	4.1	5.8
ac97_ctrl	12766	1440	-	0.8	6.8	10.9
usb_funct	14130	1456	-	1.1	8.2	13.8
b19	18557	1044	-	3.3	13.2	15.4
b17	23593	684	-	6.4	10.7	16.2
RC4_dec	44609	4445	-	3.8	4.8	8.5
des3_perf	88841	-	5572	0.9	1.2	2.8
vga_lcd	113950	-	15691	0.8	8.1	15.7
fft_64	114405	-	6427	0.7	5.8	17.0
Average				2.4	8.0	12.5

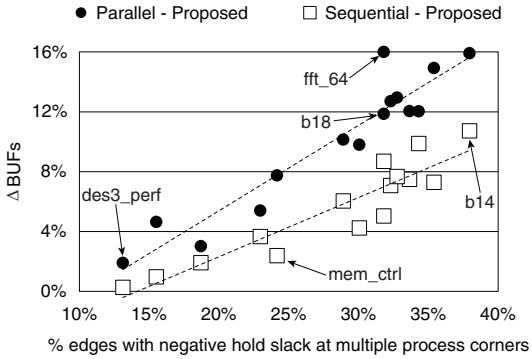

Fig. 8. Correlation between the percentage of edges with negative hold slack at multiple process corners and the difference of ΔBUFs between each intuitive method and 'Proposed'.

in more than two corners in Section IV-B. Clock period of each circuit is set to 1.1 times the maximum data-path delay (out of all corners) to accommodate some timing margin.

A. Analysis of Heuristic Buffer Insertion Methods

The number of buffers through ILP or *Partitioning + ILP* (for some larger circuits) is shown in columns 3-4 of Table II; it serves as a reference to assess the proposed insertion algorithm (column 5) as well as two intuitive algorithms (last two columns). The number of buffers via three heuristic algorithms is reported as a percentage increase from ILP result.

The number of buffers increases as timing information of less number of corners is utilized when inserting buffers. In 'Proposed' method, timing information at all corners is utilized every time buffers are inserted; the insertion results at the previous corners are reflected in *Sequential*; but *Parallel* does not see any timing information of other corners at all. 'Proposed' increases the number of buffers by only 2.4% compared to ILP, while *Sequential* and *Parallel* show 8.0% and 12.5% increase respectively.

The difference of the number of buffers between each intuitive method and 'Proposed' has wide variation according to circuits, which is affected by the proportion of edges with negative hold slack at multiple corners to edges with negative hold slack at any corner. As the number of edges with negative hold slack at multiple corners becomes larger, it is more likely that the edges to insert buffers for minimizing the number of buffers are different between at single corner and at all corners; i.e. the edge on most number of paths with hold violations is different between at single corner and at all corners. This figure is shown in Fig. 8 for all test circuits of Table II. Strong positive correlation proves that our conjecture is true. The proportion of the edges with negative hold slack at multiple corners varies from 13% to 38%, where the proportions of des3_perf, mem_ctrl, and b14 are 13%, 24%, and 38% respectively, and the difference of the number of buffers between *Parallel* and 'Proposed' in those circuits are 1.9%, 7.7%, and 15.9% respectively.

The difference of the number of buffers between each intuitive method and 'Proposed' is also affected by the number of buffers discarded due to setup violation in intuitive method; it becomes larger as the proportion of the number of discarded buffers to the number of inserted buffers increases. See circuits fft_64 and b18 in Fig. 8. Although they have the same proportion of the edges with negative hold slack at multiple corners, the difference of the number of buffers between *Parallel* and 'Proposed' is 16% in fft_64 and 12% in b18; 24% of buffers are dropped in fft_64 while only 17% of buffers are eliminated in b18 during *Parallel*.

B. Number of Process Corners

We compared the results of three heuristic methods with different number of corners. Three numbers of corners (2, 4, and 6) are used for the experiment, where process corners contain device corners (SS and FF) and interconnect corners (Cmin, RCmin, and Cmax). As the number of corners

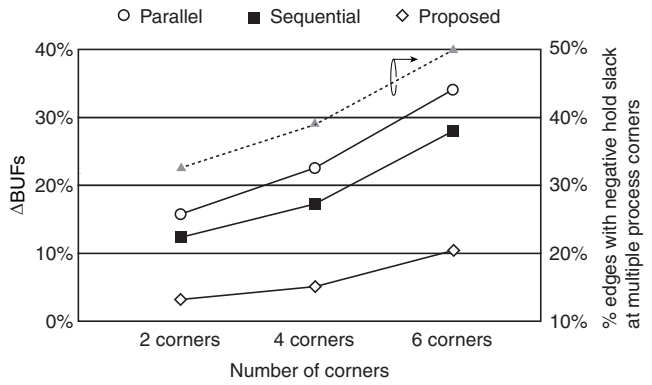

Fig. 9. Increase in ΔBUFs according to the number of corners (in circuit b18).

TABLE III
THE NUMBER OF BUFFERS VIA *Sequential* WITH DIFFERENT NUMBER OF CORNERS

Circuit	ΔBUFs (%)		
	2 corners	4 corners	6 corners
s35932	4.4	5.8	6.6
b15	13.4	19.5	30.2
b18	12.5	17.6	28.3
s38417	4.1	5.2	5.8
ac97_ctrl	6.8	10.1	16.6

increases, the difference of the number of buffers between intuitive method and 'Proposed' becomes larger, as shown in Fig. 9; the increase in the number of corners makes larger proportion of the edges with negative hold slack at multiple corners which is strongly correlated with the difference of the number of buffers. In circuit b18, the difference of the number of buffers between *Parallel* and 'Proposed' is 12%, 18%, and 26% in 2, 4, and 6 corners, respectively.

The amount of the variation of the number of buffers according to the number of corners is small in circuits s35932 and s38417, while it is large in b15 and b18, as shown in Table III. This difference is also related to the percentage of the edges with negative hold slack at multiple corners: the percentage is only changed from 16% to 18% between 2 corners and 6 corners in s35932; while it is changed from 33% to 55% in b15.

C. Runtime

Computation times for heuristic methods and ILP are shown in Table IV. Runtime of ILP tends to grow exponentially as the number of cells increases, and it is 10∼1,000 times larger than runtime of heuristic methods, as expected.

Runtime of heuristic methods, on the other hand, depends on the number of times that timing information is updated. *Sequential* has larger runtime compared with the others, because timing information is serially updated at each corner. In *Parallel*, timing information at all corners are updated simultaneously, but *Buffer_Insertion* and *Buffer_Removal* are repeated until buffers do not incur setup violation. Due to this iteration, *Parallel* consumes 30% larger runtime than 'Proposed'.

TABLE IV
RUNTIMES (IN SECONDS) OF BUFFER INSERTION METHODS

Circuit	Proposed	*Sequential*	*Parallel*	ILP
ac97_ctrl	8	13	10	302
b19	10	14	12	916
b17	7	11	9	2779
RC4_dec	24	29	27	25840
des3_perf	31	57	47	Timeout
Average	1.0	1.6	1.3	-

V. CONCLUSION

Buffer insertion with multiple process corners into consideration has been addressed. It has been formulated as ILP which serves as a reference; it has been combined with circuit partitioning so that some larger circuits can also be handled. A heuristic algorithm has been proposed and compared to two additional intuitive algorithms to emphasize the need of new algorithm.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A2A01008037).

REFERENCES

[1] D. Harris, M. Horowitz, and D. Liu, "Timing analysis including clock skew," *IEEE Trans. on Computer-Aided Design*, vol. 18, no. 11, pp. 1608–1618, Nov. 1999.
[2] Y. Yang, I. Jiang, and S. Ho, "PushPull: Short-path padding for timing error resilient circuits," *IEEE Trans. on Computer-Aided Design*, vol. 33, no. 4, pp. 558–570, Apr. 2014.
[3] P. Wu *et al.*, "On timing closure: Buffer insertion for hold-violation removal," in *Proc. Design Automation Conf.*, Jun. 2014, pp. 1–6.
[4] T. Xiao, H. Bagga, G. Chen, R. Cheung, and R. Pattipati, "Path aware event scheduler in holdadvisor for fixing min timing violations," in *Proc. Int. Conf. on Computer Design*, Oct. 2011, pp. 71–77.
[5] Y. Liu, F. Yuan, and Q. Xu, "Re-synthesis for cost-efficient circuit-level timing speculation," in *Proc. Design Automation Conf.*, Jun. 2011, pp. 158–163.
[6] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1993, pp. 156–161.
[7] S. Huang *et al.*, "Clock period minimization with minimum delay insertion," in *Proc. Design Automation Conf.*, Jun. 2007, pp. 970–975.
[8] S. Onaissi, F. Taraporevala, J. Liu, and F. Najm, "A fast approach for static timing analysis covering all PVT corners," in *Proc. Design Automation Conf.*, Jun. 2011, pp. 777–782.
[9] G. Scott and P. Watson, *ARM Cortex iRM: CPF-driven low-power functionality in a high-performance design flow*. Power Forward, 2009.
[10] *PrimeTime User Guide*, Synopsys, Mountain View, CA, Dec. 2014.
[11] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com
[12] OpenCores. [Online]. Available: http://www.opencores.org/
[13] *Design Compiler User Guide*, Synopsys, Mountain View, CA, Jun. 2015.
[14] *IC Compiler User Guide*, Synopsys, Mountain View, CA, Jun. 2015.