An effectiveness-oriented greedy heuristic for padding short paths in ultra-low supply voltage designs

Yan He¹, Jianyi Meng^{2a)}, Zhijian Chen¹, Xiaoyan Xiang²,
and Chen Chen²

¹ Institute of VLSI Design, Zhejiang University, Hangzhou, 310027 China

² State Key Laboratory of ASIC and System, Fudan University,
Shanghai, 201203 China

a) mengjy@fudan.edu.cn

Abstract: Among prior short paths padding algorithms, greedy heuristic based on the number of short paths had been proven to be area saving and fast. For ultra-low supply multi-voltage designs, however, delay buffers explode due to improper padding locations caused by unconsidered delay variations. To overcome this problem, we propose a new evaluation function “effectiveness” for greedy heuristic, to characterize the benefit of buffer insertion, thereby to optimize the allocation of padding delay and reduce buffer overhead. An improved algorithm to dynamically update slacks during delay assignment stage is also introduced. Experimental results show average area reduction of 91.6% compared to the prior method.

Keywords: short path padding, greedy heuristic, ultra-low supply voltage, multi-voltage

Classification: Integrated circuits

References

- [1] N. Pinckney, *et al.*: “Low-power near-threshold design: Techniques to improve energy efficiency,” IEEE Solid State Circuits Mag. **7** (2015) 49 (DOI: [10.1109/MSSC.2015.2418151](https://doi.org/10.1109/MSSC.2015.2418151)).
- [2] N. V. Shenoy, *et al.*: “Minimum padding to satisfy short path constraints,” Int. Conf. on Computer-Aided Design (1993) 156 (DOI: [10.1109/ICCAD.1993.580048](https://doi.org/10.1109/ICCAD.1993.580048)).
- [3] S.-H. Huang, *et al.*: “Minimum buffer insertions for clock period minimization,” Computer Communication Control and Automation (2010) (DOI: [10.1109/3CA.2010.5533776](https://doi.org/10.1109/3CA.2010.5533776)).
- [4] P.-C. Wu, *et al.*: “On timing closure: Buffer insertion for hold-violation removal,” Design Automation Conf. (2014) (DOI: [10.1145/2593069.2593171](https://doi.org/10.1145/2593069.2593171)).
- [5] Y. Liu, *et al.*: “Re-synthesis for cost-efficient circuit-level timing speculation,” Design Automation Conf. (2011) 158.
- [6] S. Yoshikawa: U.S. Patent 6 990 646 (2006).
- [7] T. Xiao, *et al.*: “Path aware event scheduler in Holdadvisor for fixing min timing violations,” Int. Conference on Computer Design (2011) (DOI: [10.1109/ICCD.2011.6081378](https://doi.org/10.1109/ICCD.2011.6081378)).

- [8] Y.-M. Yang, *et al.*: “PushPull: Short-path padding for timing error resilient circuits,” IEEE Trans. Comput.-Aided Design Integr. Circuits **33** (2014) 558 (DOI: [10.1109/TCAD.2014.2304681](https://doi.org/10.1109/TCAD.2014.2304681)).
- [9] A. B. Kahng: “New game, new goal posts: A recent history of timing closure,” Design Automation Conf. (2015) (DOI: [10.1145/2744769.2747937](https://doi.org/10.1145/2744769.2747937)).
- [10] *IC Compiler User Guide* (Synopsys, Mountain View, CA, 2014).
- [11] *Primetime User Guide* (Synopsys, Mountain View, CA, 2014).
- [12] IWLS Benchmarks, Online: <http://iwls.org/iwls2005/benchmarks.html>.

1 Introduction

NTC (near-threshold) or STC (sub-threshold) designs have been expounded better energy efficiency than that with standard supply voltage [1]. However, under these conditions, circuits need to tolerate astonishing PVT variations; hence huge timing derate has to be added to constraints to ensure circuit functionality. Meanwhile, clock skew under these conditions becomes irreparably large, especially for timing path across different voltage domains, as clock buffer’s delay varies significantly under different voltages. Under such circumstances, ultra-low supply designs suffer much more hold violations than usual, which lead to great area cost for padding short paths. Thus, it becomes critical to limit total introduced buffers when padding short paths.

Short paths padding algorithms have basically two directions: one is ILP (Integer Linear Programming) formulation, by solving equations constructed from timing constraints [2, 3, 4]. This method was first proposed in [2] and was preferred by later works because this mathematical approach had optimal solutions. However, compute cost grew exponentially with circuits scaling larger, and became impractical when dealing with VLSI circuits [5]. In [4] the authors proposed a graph reduction method for ILP formation in order to reduce runtime. Yet the experimental results did not show radical improvement. The other method is greedy heuristics, first referred to in [2, 6]. The method in [2] decided locations with the maximum setup margin to maintain the minimum clock period but was area consuming and abandoned in today’s nano-scale designs. [6] chose locations with the max number of violated paths to maximize the benefit of buffer insertions. This method was also adopted in [5, 7, 8]. [7] introduced a procedure first to size down buffers, then implemented the greedy method. While [5] introduced a procedure first to retime or relocate critical registers. This paper also did an exhaustive comparison of ILP and greedy approaches. [8] mainly focused on a new algorithm aimed at predicting the minimum period for resilient circuits. [5, 7, 8] did not actually change the original algorithm of greedy heuristic proposed in [6]. Despite greedy heuristics may not give optimal solution, the area overhead is small enough to be accepted compared to ILP, and runtime is linear dependent to circuit scales [5]. So far greedy heuristic is still the best paper algorithm both in area reduction and runtime.

Greedy heuristic shows one disadvantage when it comes to ultra-low supply multi-voltage designs: the buffer delay variations between different locations or

voltage domains were not taken into consideration. Actually, a buffer's delay shows drastic variations throughout the circuit with ultra-low supply voltage. This will cause inappropriate choices during padding allocation. Take the circuit in Fig. 1 as an example, path $R0$ to $vd_l/R0$ is setup critical, while $R1$ to both $vd_l/R1$ and $vd_l/R2$ are hold violated. The prior heuristic method will first pick point A for padding, which has two short paths through (path: $R1-A-B-vd_l/R1$ and path: $R1-A-C-D-vd_l/R2$). For instance, a buffer provides 0.1 ns delay under high voltage domain (typical under 1.1 V supply), and 1.0 ns delay under low voltage domain (typical under 0.6 V supply). Thus, padding on a point in high voltage domain is uneconomic, since padding on A requires 30 buffers, while padding on B and C , will require only 5 buffers in total. Besides, for points in the same low voltage domain, delay variations is also significant that some points may also become uneconomic. For instance, a buffer provides 1.0 ns on C , and 0.9 ns on D . If D is picked first, it will require more buffers than C . Based on our observation, prior padding methods, including commercial tools, introduced copious amounts of delay buffers when dealing with ultra-low supply designs.

We here propose a heuristic method to locate better sites for padding delay. Within our knowledge, this paper is the first to discuss short paths padding algorithms for NTC or STC designs. Innovations of our method are: we propose a new evaluation function “effectiveness”, which includes the consideration of buffer delay variation when choosing locations. With this function, the priority of points in high voltage domains, or in low voltage domains but supply less delay, is tuned lower, thus serves to find better locations for buffer insertion. We also introduced a slack-based padding spreading algorithm and proved its accuracy by formulas. Compared with the method described in [5, 7] and commercial tool, experimental results showed that our method remarkably reduced area overhead for NTC and STC designs.

Fig. 1. Demonstration for motivation and $G(V,E)$ graph

The following of this paper is organized as: Part 2 describes our modeling and implementation of the algorithm; Part 3 exhibits experimental results; finally Part 4 concludes this paper.

2 Algorithm

Fig. 2 is a brief look at our approach. The implementation starts with a post-route design, with static timing analysis (STA) done in Primetime. Unlike [8] in which slack on each vector is calculated by tracing arrival time and required time from start points or endpoints, necessary timing information is collected directly from a Primetime session, in order to maintain accuracy. A programmable graph model, $G(V, E)$ graph, is also constructed beforehand, with all possible buffer locations marked as available. With acquired information, we calculate the new evaluation function for all available vectors. Vector with the highest priority is padded first, then marked as unavailable. The assigned padding delay will be spread out for the associated vectors, whose evaluation function will also be updated. Iterations of delay assignment and spreading will be repeated until no vectors are available. After realizing padded delay to a buffer insertion list, we implement ECO-PR and STA, then check whether the padding result meets timing constraints.

Fig. 2. Flow of our approach.

2.1 $G(V, E)$ graph modeling

$G(V, E)$ graph, as described in [3, 4, 5, 6, 8], is the base for padding algorithm implementation. It's constructed from a set of vectors V , and a set of edges E that connects all vectors. As illustrated in Fig. 1, all pins in the data path are presented as vectors (otherwise like in [8], vectors present for gates), and all timing arcs as edges. The solid edges represent timing arcs from outputs to inputs (wires), while the dotted ones represent timing arcs from inputs to outputs inside gates ($E = E_w \cup E_g$). In this graph, padding delay will be determined on vectors, and then spread out along the edges.

Based on the $G(V, E)$ graph and basic concepts of timing analysis, we have following definitions and equations: Eq. (1, 2) represent the calculation of max and min slack of each vector, from prior vector's arrival time and following vector's required time. Eq. (3, 4) represent the calculation of max and min slack of each edge. Eq. (5, 6) represent the relationship between slack of a vector and slacks of its connected edges. Eq. (7) represent that the number of violated paths through a vector is the sum of violated paths through its connected edges. Eq. (8) represent that *ths* of a vector is the sum of *ths* of its connected edges. These equations are basis for our deduction and program:

R stands for setup require time for a vector, **r** for hold required time;

A stands for max arrival time for a vector, **a** for min arrival time;

S stands for setup slack for a vector or an edge, **s** for hold slack;

d stands for delay of an edge;

N stands for number of hold violated paths through a vector or an edge;

ths stands for total negative hold slack, sum of slacks of all hold violated paths through a vector or an edge;

δ stands for delay that will be introduced by a certain buffer on a vector;

Let $i \in \text{fanin of } v$, $x \in \text{fanout of } v$:

$$S(v) = R(v) - A(v) = \min(R(x) - d(v, x)) - \max(A(i) + d(i, v)) \quad (1)$$

$$s(v) = a(v) - r(v) = \min(a(i) + d(i, v)) - \max(r(x) - d(v, x)) \quad (2)$$

$$S(i, v) = R(v) - (A(i) + d(i, v)) \quad (3)$$

$$s(i, v) = a(i) + d(i, v) - r(v) \quad (4)$$

$$S(v) = \min(S(i, v)) = \min(S(v, x)) \quad (5)$$

$$s(v) = \min(s(i, v)) = \min(s(v, x)) \quad (6)$$

$$N(v) = \sum N(i, v) = \sum N(v, x) \quad (7)$$

$$ths(v) = \sum ths(i, v) = \sum ths(v, x) \quad (8)$$

2.2 Evaluation function

As mentioned in above, the prior methods showed the disadvantage of over padding during timing closure stage. Greedy heuristic based on the number of short paths does not count in the factor of buffer delay variation, then the locations chosen would be area consuming encountered with small buffer delay. We need a new evaluation function which characterizes the benefit in *THS* reduction of buffer insertions.

Lemma: Once a buffer is to be inserted on one vector v , the *THS* reduction to be introduced will satisfy:

$$\Delta ths(v) \leq \min\{|ths(v)|, N(v) * \delta(v)\}, \quad s(v) \geq \delta(v) \quad (9)$$

$$|ths(v)| \leq \Delta ths(v), \quad s(v) \leq \delta(v) \quad (10)$$

For example, for vector v_1 whose slack list is $\{-0.2, -0.2, -0.06, -0.04\}$, $N(v_1) = 4$, $\delta(v_1) = 0.1$: $\Delta ths(v_1) = 0.1 + 0.1 + 0.06 + 0.04 = 0.3$, while $|ths(v_1)| = 0.5$, $N(v_1) * \delta(v_1) = 0.4$, which satisfies Eq. (9). For vector v_2 whose slack list is $\{-0.1, -0.1, -0.06, -0.04\}$, $N(v_2) = 4$, $\delta(v_2) = 0.1$: $\Delta ths(v_2) = 0.3$, while $|ths(v_2)| = 0.3$, $N(v_2) * \delta(v_2) = 0.4$, which satisfies Eq. (10).

In real circuits, there can be thousands of short paths coming through a certain vector. It's not realistic to calculate actual $\Delta ths(v)$ for these vectors with too many short paths through. We propose a new evaluation function "effectiveness" that can be easily calculated and satisfy Eq. (9, 10), also take buffer delay into consideration:

$$E(v) = \frac{ths(v)}{s(v)} * \delta(v) \quad (11)$$

For example, for vectors in the prior context, $E(v_1) = (-0.5 / -0.2) * 0.1 = 0.25$, $E(v_2) = (-0.3 / -0.1) * 0.1 = 0.3$. Because $ths(v)/s(v) \leq N(v)$, $E(v) \leq N(v) * \delta(v)$ is always tenable. When $s(v) \geq \delta(v)$, $E(v) \leq |ths(v)|$; otherwise $E(v) \geq |ths(v)|$. It's thus clear that $E(v)$ satisfies Eq. (9, 10). Another example, for the circuit in Fig. 1, $E(A) = (-5 / -3) * 0.1 = 0.167$, $E(B) = (-2 / -2) * 1 = 1$, $E(C) = (-3 / -3) * 1 = 1$, and $E(D) = (-3 / -3) * 0.9 = 0.9$. With the proposed evaluation function, padding priority turns to $B = C > D > A$. So the priority of A is downgraded, avoiding to pad on A in the first place.

2.3 Padding and spreading

Once we have a vector with the highest priority, padding delay will be assigned:

$$P(v) = \min(|s(v)|, S(v)) \quad (12)$$

With this assignment, slacks of other vectors in common paths will be changed. Yet it would be time-consuming if we realize padding delay and re-update timing information whenever a new assignment occurs. It's more efficient to assign as much delays as possible on suitable vectors, and then realize them altogether. Thus we propose a padding delay spreading algorithm. Unlike spreading algorithm in [8] which needs to recalculate arrival time and required time, our approach starts from the padded vector and only consider slacks on each vector along. The data access in this stage is reduced hence. We use $\{i, j.. \}$ to denote fanins of v , and $\{x, y.. \}$ to denote fanouts of v , $\{u, v.. \}$ to denote fanins of x . Assume $P(i)$ occurred on vector i , the updated setup slack of v can be calculated as:

$$\begin{aligned} S'(v) &= \min \begin{cases} S'(i, v) \\ S(j, v), j \in \text{fanin of } v, j \neq i \end{cases} \\ &= \min \begin{cases} R(v) - (A(i) + P(i) + d(i, v)) \\ S(j, v), j \in \text{fanin of } v, j \neq i \end{cases} \end{aligned} \quad (13)$$

Here we introduce another definition, a virtual max padding delay $VP(v)$:

$$VP(v) = S(v) - S'(v) \quad (14)$$

Combined with Eq. (1), we get:

$$VP(v) = S(v) - S'(v) = R(v) - A(v) - (R(v) - A'(v)) = A'(v) - A(v) \quad (15)$$

As for vector x , the setup slack of x can be calculated as:

$$\begin{aligned}
 S'(x) &= \min \begin{cases} S'(v, x) \\ S(u, x), u \in \text{fanin of } x, u \neq v \end{cases} \\
 &= \min \begin{cases} R(x) - (A'(v) + d(v, x)) \\ S(u, x), u \in \text{fanin of } x, u \neq v \end{cases} \\
 &= \min \begin{cases} R(v) - (A(v) + VP(v) + d(v, x)) \\ S(u, x), u \in \text{fanin of } x, u \neq v \end{cases}
 \end{aligned} \tag{16}$$

Compare Eq. (13) and Eq. (16), it's clear that $VP(v)$ is all we need for calculation of $S(x)$: calculate $S(v)$ and $VP(v)$ from $P(i)$, and then calculate $S(x)$ and $VP(x)$ from $VP(v)$, and so on.

We also define a virtual min padding delay $vp(v)$ as:

$$vp(v) = s'(v) - s(v) \tag{17}$$

It can be similarly derived that for hold slack calculation, $vp(v)$ is enough for latter vectors. Furthermore, it can be derived that this pattern of spreading padding delay also suits the situation from latter vectors to prior ones. This conclusion is exemplified in Fig. 3: a small $G(V, E)$ graph, with the initial delay of each edge, and initial slacks of each vector presented along. Both i and j are on hold violated paths through v and have setup margin. Assume i is prior to other vectors, and $P(i) = 3$ is assigned to i for situation 1; and assume j is prior to other vectors, and $P(j) = 12$ is assigned for situation 2. Vector v is visited first, with its new slacks and virtual padding delays calculated. Then slacks of x will be updated based on v only, without the need to concern about i or j . The detailed spreading procedure is exhibited in Fig. 3. Further following vectors will be visited and updated similarly.

Fig. 3. Demonstration circuit for virtual padding delay.

Concurrently when padding delay is spread along edges, the “effectiveness” function also needs to be updated for visited vectors. Padding on one vector will solve part of its corresponding vectors’ *ths*, yet, not necessarily change their min slack, then “effectiveness” for them is possibly lowered. That is to say, the priority of vectors should be adjusted dynamically. However, the actual decrease in *ths*

cannot be acquired in this modeling: based on $G(V, E)$ graph, we cannot tell how much amount of ths is shared between distant vectors. Here we propose an approximate method. First, we define an equivalent number of short paths from the coefficient of function $E(v)$:

$$N_e(v) = \frac{ths(v)}{s(v)}, \quad N_e(i, v) = \frac{ths(i, v)}{s(i, v)} \quad (18)$$

According to Eq. (8), ths of a vector equals to the sum of ths of each of its fanin edges, or its fanout edges. It means $N_e(v)$ and $N_e(i, v)$ are interrelated with analogous tendency. Based on this observation, we can approximately pass on changes of ths to other associated vectors. For vector i , assume $P(i)$ has been assigned, $N_e(i)$ may remain the original value:

$$\begin{aligned} ths'(i) &= ths(i) + E(i) = s(i) * N_e(i) + N_e(i) * \delta(i) \\ &= (s(i) + \delta(i)) * N_e(i) = s'(i) * N_e(i) \end{aligned} \quad (19)$$

$$N'_e(i) = \frac{ths'(i)}{s'(i)} = \frac{s'(i) * N_e(i)}{s'(i)} = \begin{cases} 0, & \text{if } s'(i) = 0 \\ N_e(i), & \text{if } s'(i) \neq 0 \end{cases} \quad (20)$$

$$N'_e(i, v) = \frac{N_e(i, v)}{N_e(i)} * N'_e(i) = \begin{cases} 0, & \text{if } s'(i) = 0 \\ N_e(i, v), & \text{if } s'(i) \neq 0 \end{cases} \quad (21)$$

Though $N'_e(i)$ may not change, i is marked as unavailable and not to be visited again. For v , $ths(v)$ and $s(v)$ may all change, calculated as:

$$\begin{aligned} ths'(v) &= \sum_{i \in \text{fanin cone}} s(i, v) * N_e(i, v) \\ &= s'(i, v) * N_e(i, v) + \sum_{j \neq i} s(j, v) * N_e(j, v) \end{aligned} \quad (22)$$

$$N'_e(v) = \frac{ths'(v)}{s'(v)} \quad (23)$$

$$N'_e(v, x) = \frac{N_e(v, x)}{N_e(v)} * N'_e(v) \quad (24)$$

Because of delay padded on i , $ths(v)$ has a certain decrease (Eq. (22)). Yet $s(v)$ may not necessarily change, so it can be inferred that padding on i will lower $N_e(v)$. This consequence is consistent with the prior discussion. After $N'_e(v)$ is obtained, $E'(v)$ will also be recalculated and the priority be reordered. For example, in Fig. 1, $N_e(C) = N_e(D) = 1$ and $N_e(A) = 1.67$, if $P(C) = 3$ occurs, then we can calculate that $N'_e(C) = N'_e(D) = 0$ and $N'_e(A) = 1$, $E'(C) = E'(D) = 0$ and $E'(A) = 0.1$. D is no longer in the priority list.

In practice, the detailed program line is exhibited in Table I. For each selected vector with the highest E , the padding process (*Padding-Spreading*) will first pad the vector, then recursively call the spreading processes (*Spreading-Forward* and *Spreading-Backward*), until all fanin or fanout vectors are visited and updated. Then the process will iteratively turn to search for another available vector with the highest E until there are no more available vectors.

2.4 Realizing delay to buffers

Realizing delay on a vector is a mature question with many discussions. The majority of delay could be realized by inserting delay buffers and remaining by

Table I. Pseudo code for padding and spreading stage

Process: Spreading-Forward Input: $v, P(v), p(v), N_e(v)$ <pre> 1 foreach $\{i \in \text{fanin of } v\}$ do 2 $S'(i, v) = S(i, v) - P(v)$ 3 $s'(i, v) = s(i, v) + p(v)$ 4 update $S(i), s(i)$ 5 update $N'_e(i, v), N'_e(i), E'(i)$ 6 calculate $VP(i), vp(i)$ 7 if $\{i \text{ is a startpoint}\}$ {continue} 8 Spreading-Forward $(i, VP(i), vp(i), N_e(i))$ 9 end foreach End Process </pre>	Process: Spreading-Backward Input: $v, P(v), p(v), N_e(v)$ <pre> 1 foreach $\{x \in \text{fanout of } v\}$ do 2 $S'(v, x) = S(v, x) - P(v)$ 3 $s'(v, x) = s(v, x) + p(v)$ 4 update $S(x), s(x)$ 5 update $N'_e(v, x), N'_e(x), E'(x)$ 6 calculate $VP(x), vp(x)$ 7 if $\{x \text{ is an endpoint}\}$ {continue} 8 Spreading-Backward $(x, VP(x), vp(x), N_e(x))$ 9 end foreach End Process </pre>
Process: Padding-Spreading Input: $G(V, E)$ Graph <pre> 1 while $\{\text{there exists available vectors with positive } E\}$ 2 Search vector v with highest $E(v)$ and calculate $P(v); p(v) = P(v)$ 3 $S'(v) = S(v) - P(v); s'(v) = s(v) + p(v);$ if $\{s'(v) = 0\}$ {set } $N'_e(v) = 0\}$ 4 Mark v as not available 5 Spreading-Forward $(v, P(v), p(v), N_e(v));$ Spreading-Backward $(v, P(v), p(v), N_e(v))$ 6 end while End Process </pre>	

sizing cells, adding spare cells to loads, or bound dummy metal. The problem of deciding inserted buffers is transformed into ILP problem:

Let $\{\delta_1(v), \delta_2(v), \dots, \delta_n(v)\}$ denote the delay of candidate buffers,
 $\{a_1(v), a_2(v), \dots, a_n(v)\}$ denote the area of candidate buffers,
and $A(v)$ denotes the total introduced area, define:

$$P(v) = \sum_{i=1}^n N_i(v) * \delta_i(v), \quad A(v) = \sum_{i=1}^n N_i(v) * a_i(v) \quad (25)$$

The goal is to obtain a set of number $\{N_1(v), N_2(v), \dots, N_n(v)\}$ that tries to satisfy $P(v)$ and gives a minimum $A(v)$. For each of the assigned vectors, ILP procedures will be implemented successively and solved fleetly, because generally, there are no more than ten candidate buffers ($n \leq 10$ in Eq. (25)).

2.5 Time complexity

In this part, we'll be using following notations:

N : Number of vectors with negative hold slack in $G(V, E)$ diagram;

P : Number of vectors with assigned padding delay;

F : Total number of fanin and fanout vectors visited from padded vectors during process *Spreading-Forward* and *Spreading-Backward*.

In the stage of $G(V, E)$ construction, information gathering, and $E(v)$ calculation, every vector in the diagram needs to be visited once. The time complexity is $O(N)$. For padding and spreading, each vector with a padded delay triggers a procedure of spreading in its fanout and fanin cones, approximately every vector will be visited once. The time complexity is $O(F)$. In realizing delay stage, if it is assumed that ILP is solved in $O(1)$, the time complexity is $O(P)$ since there will be P times to

assign buffers. All three parameters are positively related to circuit scale and short paths constraints.

3 Experiment results

Our experiment is implemented at SMIC 55 nm tech node. The first reference method is greedy heuristics in [5, 7, 8] (shortened as “ref” in following content). As concluded in [9], “EDA tool innovation in the timing closure space has been impressive”. Unlike paper algorithms’ slow progress, commercial tools now can achieve complicated physical aware eco timing optimization with multi-scenario support, scales on to hundreds of threads to fasten runtime [11]. Under standard voltages, Primetime can also achieve minimum buffer insertion within a short runtime. So we choose Primetime as additional reference method (“pt”). Then comes our proposed greedy heuristic, which selects padding locations by the “effectiveness” function (“ours”). “ref” and “ours” are programmed using Tcl/Tk and built-in commands in Primetime.

The targeted circuits include several large-scale circuits in benchmark [12], and our industrial test chip, with an embedded microprocessor under ultra-low voltage, and remaining peripherals under standard voltage. Circuit *b12* is not divided into multiple voltage domains, in order to verify that our method makes differences under a single low voltage domain. The other circuits are divided into two domains. Low voltage domain is under *ff_0p33v_85c* for STC best condition and others under *lp32v* supply. Besides, our chip was implemented both under *ff_0p66v_85c* for NTC best condition (“*ind06*”) and *ff_0p33v_85c* (“*ind03*”). Based on our Monte-Carlo simulation, we set 100% derate NTC corners, and 300% derate for STC corners. All circuits are post-routed in IC-Compiler [10] and then analyzed in Primetime with parasitics extracted in StarRC. Three methods are all implemented first, with only one iteration. Then buffers are inserted back in IC-Compiler and re-analyzed in Primetime. At this moment *THS* may still be unclear, it’s because buffer delay may not be sufficient (if the smallest buffer delay is 0.2, 0.05 slack remains) and there are deviations between calculated buffer delay and realized delay. So a rounding-off stage with more buffers is implemented. Yet we don’t conclude this part in comparison because it’s not related to the methods. Finally, we record the results listed in Table II.

Setup constraints take effects in all cases to maintain the original period. The results show that the minimum setup slack did not degrade afterward. Under NTC and STC corners, inserted area by “ref” method increased exponentially. More concretely, too many padding locations were chosen in high voltage domain. Take case *ind06* as an example, illustrated in Fig. 4 (the inserted buffers are highlighted white), many of these locations required thousands of buffers to make up for hold slack. This lead to an explosion of total cell numbers. As for “pt”, under STC corner, it still inserted hundreds of unnecessary buffers on average in high voltage domain. Besides, though most selections did occur in low voltage domain, the tool was unable to make the best choice. Our proposed method chose the locations with the best benefit both in area reduction and hold fixing, thus forbade to insert buffers in high voltage domain. Also from case *b12*, our method achieved better area

Table II. Results for three short path padding methods

circuit	b12	b21	b17	b18	ind06	ind03
LV module	b12	P1	P1	P1	cpu	cpu
#total cells	1494	16062	25897	70161	57485	57485
THS (ns)	−2301.9	−12833	−21406	−55802	−370.8	−56435
S (ns)	−35.5	−15.7	−18.8	−79.3	−0.3	−11.3
ref	#B	663	51777	7152	128070	3837
	ΔA	1833.16	158997	21417.2	392389	10990
	THS'	−1.14	−50.12	−33.34	−346.38	−4.09
	%fixed	99.95%	99.61%	99.84%	99.38%	98.90%
	#RB	4	115	32	219	88
	THS''	0	0	0	0	0
	S'	−35.5	−15.7	−18.8	−79.3	−0.3
pt	#B	265	1088	1755	7371	422
	ΔA	720.44	3086.72	4741.8	20571	1168.44
	THS'	0	0	−10.12	0	0
	%fixed	100%	100%	99.95%	100%	100%
	#RB	0	0	15	0	0
	THS''	0	0	0	0	0
	S'	−35.5	−15.7	−18.8	−79.3	−0.3
ours	#B	163	1101	1412	4938	227
	ΔA	421.4	2858.24	3659.04	13016.9	519.68
	THS'	−12.36	−57.75	−112.14	−454.84	−0.94
	%fixed	99.46%	99.55%	99.48%	99.18%	99.75%
	#RB	16	35	50	181	19
	THS''	0	0	0	0	0
	S'	−35.5	−15.7	−18.8	−79.3	−0.3
	F	11967	1678 k	288 k	26858 k	2903 k
	rt_d2b	<1 s	<1 s	<1 s	10 s	2 s
	runtime	4 s	464 s	105 s	7889 s	749 s
%area saved compared to “ref”		77.01%	98.20%	82.92%	96.68%	95.27%
%area saved compared to “pt”		41.51%	7.40%	22.83%	36.72%	55.52%
		71.53%				

Annotations:

LV module: the module assigned with low supply voltage;

THS: total negative hold slack of all paths; #B: number of inserted buffers;

ΔA: total increased buffer area; THS': remained THS after implementation;

%fixed: percentage of fixed THS after implementation;

#RB: number of rounding-off buffers; THS'': final THS after rounding-off;

S: initial minimum setup slack; S': minimum setup slack after rounding-off;

F: one of the parameters mentioned in section-2.5 (Time complexity)

rt_d2b: runtime of single stage of realizing delay to buffers;

runtime: runtime of all stages including rt_d2b;

Fig. 4. Inserted buffers of *ref* and *ours* for case *ind06*.

Fig. 5. The correlation between F and runtime.

overhead for single voltage domain. The area of inserted buffers overall had been reduced 91.6% on average compared with “*ref*”, and had been reduced 39.2% on average compared with “*pt*”.

Fig. 5 provides the relevance between runtime and F . As $P \ll N \ll F$ (for example, in case *ind03*, $N = 148784$, $P = 544$, $F = 1273$ k), the runtime is mainly contributed by delay assignment stage and is linear dependent to F . Results show that runtime of realizing delay to buffers can be omitted.

4 Conclusion

In this paper, we proposed an effectiveness-oriented greedy heuristic to overcome the drawback of prior short paths padding methods, that they caused a considerable amount of buffer overhead dealing with NTC or STC designs. Compared to prior proposals, our algorithm had shown to be more effective under ultra-low-voltage conditions, with 91.6% lesser area dissipation on average. There are more timing closure topics worth exploring under ultra-low supply voltage, such as clock skew issue, minimum clock period exploration, short padding problem for resilient circuit and multi-scenario consideration. These topics all have different situations compared with standard voltages, thus needs more targeted algorithms.

Acknowledgments

This work was supported by the Ministry of Science and Technology of the People’s Republic of China under Grant 2015AA016601, Science and Technology Commission of Shanghai Municipality under Grant 15ZR1402700.