PushPull: Short-Path Padding for Timing Error Resilient Circuits

Yu-Ming Yang, Iris Hui-Ru Jiang, *Member, IEEE,* and Sung-Ting Ho

***Abstract*—Modern IC designs are exposed to a wide range of dynamic variations. Traditionally, a conservative timing guardband is required to guarantee correct operations under the worst-case variation, thus leading to performance degradation. To remove the guardband, resilient circuits are proposed. However, the short-path padding (hold time fixing) problem in resilient circuits is far severer than conventional IC design. Therefore, in this paper, we focus on the short-path padding problem to enable the timing error detection and correction mechanism of resilient circuits. Unlike recent prior work adopts greedy heuristics with a local view, we determine the padding values and locations with a global view. Moreover, we utilize spare cells and a dummy metal to further achieve the derived padding values at physical implementation. Experimental results show that our method is promising to validate timing error-resilient circuits.**

***Index Terms*—Delay padding, engineering change order, hold time fixing, linear programming, resilient circuits, timing analysis.**

I. Introduction

DUE TO a wide range of dynamic variations, e.g., supply voltage droops, process variations, temperature fluctuations, soft errors, and transistor aging degradation, the timing characterization is extremely difficult in modern IC designs. Therefore, in conventional design, designers conservatively (pessimistically) reserve a timing guardband to ensure correct functionality even under the (rare) worst-case circumstance. However, this reserved guardband may severely degrade circuit performance, i.e., limiting the clock frequency.

Recently, several resilient circuits have been proposed to eliminate the guardband by error detection and correction [1]–[5]. For example, Fig. 1 illustrates one error-detection circuit, the Razor flip–flop proposed in [1]. One extra storage element, the shadow latch, is augmented to sample the output of a combinational logic by a delayed clock. The main flip–flop and shadow latch outputs are compared to generate a timing-error signal. If the output of the combinational logic transitions is late, a timing error (discrepancy) is detected. Error correction is then performed through instruction replay.

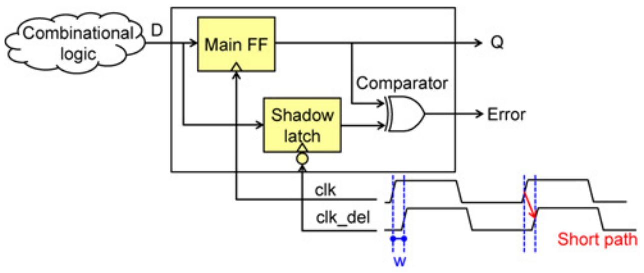


Fig. 1. Error-detection part of a Razor flip–flop proposed in [1].

However, these resilient circuits require a significant hold time margin for short paths. The resilient circuit may detect a false timing error if the result of the next computation is propagated through a short path and sampled by the delayed clock. To avoid false error detection, short paths should exceed the error detection window, i.e., the phase difference between the delayed clock and the normal clock (w in Fig. 1). The error detection window induces an extra hold time margin requirement. This issue also exists in new forms of resilient circuits [2]–[5]. In fact, short-path padding (hold time fixing) is an inevitable and essential task in conventional IC design. A circuit with hold violations cannot operate correctly. This short path issue becomes even more challenging in resilient circuits due to this extra hold time margin, typically, approximately 20% of the clock period. In order to validate the error detection and correction mechanism of resilient circuits, we focus on short-path padding in this paper.

To resolve this padding problem, prior works typically insert buffers to lengthen short paths, see [6]–[13]. Among them, the conventional delay padding is combined with clock skew scheduling to minimize the clock period at the logic resynthesis stage [7]–[9]. Their goal is to determine the padding delay for each path rather than to decide where to insert the delay. In contrast, several short-path padding methods determine the positions to insert the delay [6], [10]–[13]. Shenoy *et al.* [6] solve this problem by linear programming. Lin and Zhou [10] transform this problem into a network flow problem (but resulting in larger padding delay). Liu *et al.* [13] reveal that linear programming is time consuming and not applicable to large-scale circuits by empirical data in [12]. Hence, recently,

Manuscript received June 24, 2013; revised October 1, 2013 and November 26, 2013; accepted January 15, 2014. Date of current version March 17, 2014. This work was supported by the NSC of Taiwan under Grant NSC 101-2628-E-009-012-MY2, Grant NSC-101-2220-E-009-044, and Grant NSC 102-2220-E-009-028. An earlier version of this paper was published at ACM International Symposium on Physical Design 2013 [17]. This paper was recommended by Associate Editor C. Sze.

Y.-M. Yang and I. H.-R. Jiang are with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: yuming.yyang@gmail.com; huiru.jiang@gmail.com).

S.-T. Ho was with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan. He is currently with CMSC, Inc (e-mail: gsy2i7y14@hotmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2014.2304681

two greedy heuristics are proposed in [11]–[13]. One greedy rule is to pad the gate with the largest setup slack, trying not to hurt the longest path delay. The other is to pad at the gate passed by most hold violating paths, trying to reduce the total padding delay.

However, we observe that these greedy heuristics based on local views may not pad short paths well. Fig. 2(a) gives an input design, where gates g_1, g_2, and g_3 incur hold violations. After iteratively padding delay on the gate either with the largest setup slack [Fig. 2(b)] or with most hold violating paths [Fig. 2(c)], we still have an unresolved hold violation at gate g_2. In fact, all hold violations can be cleaned as shown in Fig. 2(d). It can be seen that padding with local views may consume all setup slacks thus leaving some hold violations unfixed [Fig. 2(b) and (c)]. Moreover, even if we find an optimal padding solution [6], we may still fail at physical implementation because the available buffer delays are discrete.

Based on the above observations, we develop a three-stage short-path padding algorithm, named PushPull, to overcome these difficulties: Stage 1 decides a feasible clock period; Stage 2 tries to minimize the total padding delay with a global view and determines padding locations; Stage 3 allocates load/buffers to attain the padding at postlayout to handle the discrete cell library. Our features include the following.

1) Adjusting the target clock period dynamically: In some resilient circuit design flow [4], designers verify whether the target clock period is feasible after short paths are padded. Unlike them, we check the feasibility of a specific clock period at the early stage and adjust the target clock period dynamically.
2) Finding the padding values with a global view: The greedy heuristics proposed by prior works may fail to fix all hold violations due to local views. Instead, in Stage 2, we compute the padding flexibility of the fanout cone of each gate. With this global view, we determine the padding value for each gate accordingly.
3) Delay padding at postlayout: As the available resource of padding is uncertain at early stages, unlike prior work determines the padding values at logic resynthesis, we further realize delay padding at the postlayout stage. As the amount of delay offered by a cell library is discrete, and the dummy metal offers an abundant and tunable resource of capacitance [14], we simultaneously allocate spare cells and the dummy metal to match the delay padding determined in Stage 2.

Experiments are conducted on the IWLS 2005 benchmark circuits [20] through the resilient circuit design flow. Our results show that we can clean all hold violations with the shortest runtime, while prior work may either fail to clean all violations or incur long runtime. In addition, the spare cell and dummy metal coallocation can successfully achieve the derived padding values at postlayout.

The remainder of this paper is organized as follows. Section II briefly introduces the resilient circuit design flow, describes the timing model and gives the problem formulation. Section III presents the overview of our short-path padding framework, PushPull. Section IV describes $S_{\mathrm{th}}/H_{\mathrm{th}}$ decision and derives setup/hold slack properties. Section V details padding value determination. Section VI presents load/buffer allocation. Section VII extends PushPull to adopt composite current source (CCS) timing model. Section VIII shows experimental results. Finally, Section IX concludes this paper.

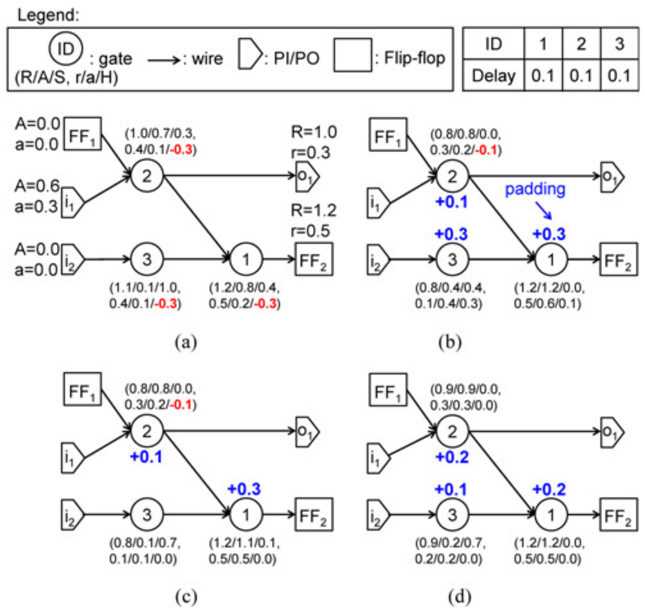


Fig. 2. Short-path padding. (a) Input design. (R/A/S, r/a/H) indicates setup required time (R), setup arrival time (A), setup slack (S), hold required time (r), hold arrival time (a), and hold slack (H) of a gate. (b) Padding from gates with largest setup slacks: g_3 $(+0.3) \rightarrow g_1$ $(+0.3) \rightarrow g_2$ $(+0.1)$, total padding delay $= +0.7$, unfixed. (c) Padding from gates with most hold time violating paths: g_1 $(+0.3) \rightarrow g_2$ $(+0.1)$, total padding delay $= +0.4$, unfixed. (d) Optimal short-path padding: g_1 $(+0.2)$, g_2 $(+0.2)$, and g_3 $(+0.1)$, total padding delay $= +0.5$.

II. Preliminaries and Problem Formulation

In this section, we briefly introduce the design flow for resilient circuits, describe the timing model, and give the problem formulation.

A. *Resilient Circuit Design Flow*

Fig. 3 shows a sample design flow to integrate timing error-resilient circuits into a design. After logic synthesis and timing analysis based on a conservative clock period (determined by the worst case delay), the target clock period and the error detection window w are determined. S_{th} (respectively, H_{th}) means the ratio of the target clock period (respectively, the error detection window) over the conservative clock period. The timing suspicious flip–flops, whose longest path delays exceed the target clock period, are replaced by resilient circuits. After the replacement, placement and routing are applied. Because of the significant hold time margin, hold violations may still exist in a placed and routed resilient design. Finally, short-path padding is performed. The target clock period is adjusted according to how many hold violations remain.

The short-path issue is not only inevitable for conventional IC design but also more challenging in resilient circuits due to

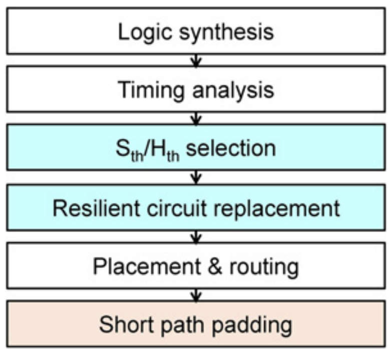


Fig. 3. Design flow of resilient circuits.

this extra hold time margin. Therefore, the short-path padding is a must to validate timing error-resilient circuits.

B. Timing Model

The cell timing model used in this paper is based on Synopsys' Liberty library [15]. The calibrated delay values of each library cell are stored in the lookup tables and indexed by its input slew and output capacitance. The wire delay of each net is lumped into the delay of its driving gate. The output capacitance of a gate includes wire loading, the input capacitance of its fanout gates, and its output pin capacitance. In addition, the output capacitance of each cell is bounded by the maximum load capacitance defined in the cell library. Chen *et al*. [16] observe loading dominance phenomenon: The change on the gate delay is dominated by output capacitance. Later, the experimental results also show that the impact of input slew on the gate delay is quite small. In Section VI-B, we shall discuss how to handle CCS timing model.

C. Problem Formulation

In order to validate the error detection and correction mechanism of resilient circuits, we focus on the short-path padding problem which is formulated as follows.

The Short Path Padding Problem: Given a placed and routed resilient design, the cell library, spare cells, and dummy metal information, our goal is to determine the target clock period and pad all short paths such that the target clock period is minimized, the padding overhead is minimized, and setup/hold timing constraints are satisfied.

As the reported timing is somewhat inaccurate and the available resource for padding is uncertain at early stages, we perform the short-path padding (hold time fixing) at the post-layout stage. To lengthen short paths, we may insert buffers [Fig. 4(a)] or introduce extra load capacitance [Fig. 4(b)]. The inserted delay can be provided by cells and metal. A design is usually sprinkled with the spare cells (redundant cells) at placement. In addition, the dummy metal offers an abundant resource of capacitance [14] and can be tuned. Hence, padding at the postlayout stage can then be done by rewiring spare cells and dummy metal. Because of loading dominance, the amount of delay increment and the corresponding amount of load/buffers inserted can be directly converted to each other by the lookup table. Later, the experimental results show that the impact of input slew on the padding delay capacitance conversion is quite small.

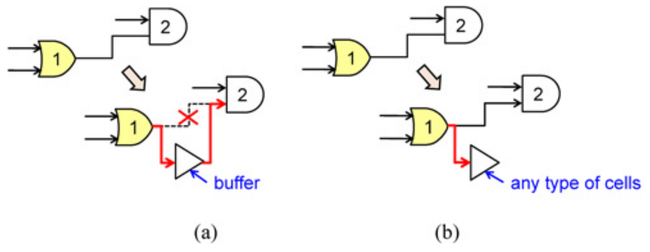


Fig. 4. Delay padding. (a) Buffer insertion (padding wire between gates g_1 and g_2). (b) Extra load hook-up (padding gate g_1).

III. PushPull: Delay Padding Framework

In this section, we present the overview of our short-path padding framework, PushPull, as shown in Fig. 5. Our framework consists of three stages: $S_{\text{th}}/H_{\text{th}}$ decision; padding value determination; and load/buffer allocation. Finally, the timing analysis is applied to verify our framework.

At the $S_{\text{th}}/H_{\text{th}}$ decision stage, we first select a target clock period and define $S_{\text{th}}/H_{\text{th}}$ accordingly. Then, we collect the available padding resource, including spare cells and dummy metal. With the physical information, we check the feasibility of selected $S_{\text{th}}/H_{\text{th}}$.

At the padding value determination stage, the calculated fanout padding flexibility estimates the padding delay that could be applied on its whole fanout cone, and thus a hold violating gate is padded to fix the remaining negative hold slack (push). The fanout padding flexibility calculation and padding value decision steps are repeated until all hold violations are resolved or no more violations can be eliminated. Then, we further reduce the total padding delay on gates (pull) and resolve unfixed hold violations by padding wires. If still there are unfixed hold violations, we return to Stage 1 to adjust the padding resource of unfixed gates and the selected $S_{\text{th}}/H_{\text{th}}$.

At the load/buffer allocation stage, the padding values on gates/wires are realized by introducing extra load, inserting buffers, as well as allocating the dummy metal. To achieve the assigned padding values, we first give a mixed integer linear programming (MILP) formulation and then propose a network-flow-based heuristic to allocate the spare cells and the dummy metal simultaneously. The network-flow-based heuristic starts from a maximum flow solution (initial allocation). We further propose spare cell allocation refinement to fix the infeasible spare cell assignment. If the flow network is unsolved or has unfixed infeasible flow, we return to the $S_{\text{th}}/H_{\text{th}}$ decision stage and adjust the $S_{\text{th}}/H_{\text{th}}$ and padding resources of unfixed gates.

The major challenges of the short-path padding problem are determining the padding delay and finding padding resource at the postlayout stage. For the former, the padding overhead is the total padding delay. For the latter, the padding overhead is the used padding resource. The more padding delay, the more padding capacitance. In our framework, the padding value

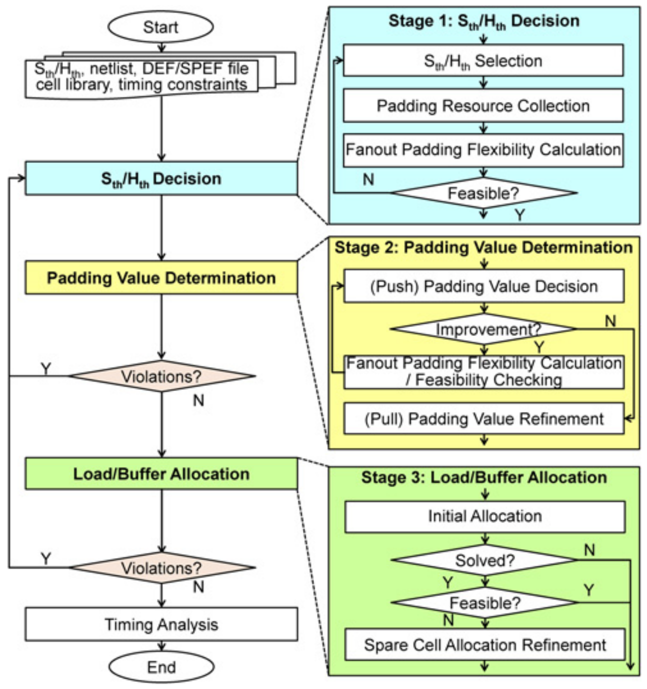

Start
S_{th}/H_{th}, netlist, DEF/SPEF file cell library, timing constraints
S_{th}/H_{th} Decision
Padding Value Determination
Violations?
Y
N
Load/Buffer Allocation
Violations?
Timing Analysis
End
Stage 1: S_{th}/H_{th} Decision
S_{th}/H_{th} Selection
Padding Resource Collection
Fanout Padding Flexibility Calculation
Feasible?
Stage 2: Padding Value Determination
(Push) Padding Value Decision
Improvement?
Fanout Padding Flexibility Calculation / Feasibility Checking
(Pull) Padding Value Refinement
Stage 3: Load/Buffer Allocation
Initial Allocation
Solved?
Feasible?
Spare Cell Allocation Refinement

Fig. 5. Overview of our short-path padding framework.

determined at Stage 2 will be transformed to the corresponding padding capacitance at Stage 3.

IV. S_{th}/H_{th} DECISION

In this stage, we select S_{th}/H_{th} dynamically and check the feasibility based on the fanout padding flexibility.

A. S_{th}/H_{th} Selection

As mentioned in Section II-A, S_{th} is defined as the ratio of the target clock period over the conservative clock period, while H_{th} is defined as the ratio of the error detection window over the conservative clock period. There is a tradeoff between S_{th} and H_{th}. At the first short-path padding iteration, we set S_{th}/H_{th} based on the user-defined target clock period (from the S_{th}/H_{th} selection step in Fig. 3). When the selected S_{th} (H_{th}) is flagged as infeasible later, we increase (decrease) S_{th} (H_{th}) and apply our short path padding again.

B. Padding Resource Collection

As our short-path padding method is applied at the post-layout stage, first of all, the available padding resource is collected. The available resource to pad each gate (wire) includes the spare cells and dummy metal located within the bounding box of its fanout net (the investigated wire). We have the following definition to constrain the maximum padding capacitance.

Definition 1: The maximum padding capacitance $C_{max}(i)$ of gate g_i is the minimum of the maximum output capacitance defined in the cell library and its available padding resource. $C_{max}(i)$ is 0 for a primary output or a flip–flop input.

The maximum padding capacitance $C_{max}(i, j)$ of the wire between gates g_i and g_j is defined similarly. $C_{max}(i)$ and $C_{max}(i, j)$ give upper bounds but still preserve flexibilities to set padding values. In some cases, the bounding boxes of padding gates/wires heavily overlap, or the padding value cannot be fulfilled at Stage 3, $C_{max}(i)$ and $C_{max}(i, j)$ can be adjusted.

C. Fanout Padding Flexibility Calculation and Feasibility Checking

To check the feasibility of selected S_{th}/H_{th}, we first calculate the padding flexibility of the whole fanout cone of each hold violating gate.

A design is represented by a directed graph $K = (G, E)$, where each node $g_i \in G$ represents a gate associated with gate delay $D(i)$, and each edge $e(i, j) \in E$ represents the wire between gates $g_i, g_j \in G$. In the following, we derive the slack properties used in this paper. The setup constraints indicate the timing requirement on long paths, while the hold constraints indicate that for short paths.

The notations used in the slack properties include:

1) $D(i)$: Gate delay;
2) $A(i)$: Setup arrival time;
3) $a(i)$: Hold arrival time;
4) $R(i, j)$: Setup edge required time;
5) $r(i, j)$: Hold edge required time;
6) $R(i)$: Setup node required time;
7) $r(i)$: Hold node required time;
8) $S(i, j)$: Setup edge slack;
9) $H(i, j)$: Hold edge slack;
10) $S(i)$: Setup node slack;
11) $H(i)$: Hold node slack.

Based on the above notations, we have the following properties.

Definition 2: In [18], the setup arrival time $A(i)$ of the output signal of node $g_i \in G$ is computed as

$$A(i) = \max_j \{A(j) \,|\, e(j, i) \in E\} + D(i) \tag{1}$$

while the setup required time $R(i)$ of g_i is computed as

$$R(i) = \min_k \{R(i, k) \,|\, R(i, k) = R(k) - D(k), e(i, k) \in E\}. \tag{2}$$

Definition 3: In [18], the hold arrival time $a(i)$ of the output signal of node $g_i \in G$ is computed as

$$a(i) = \min_j \{a(j) \,|\, e(j, i) \in E\} + D(i) \tag{3}$$

while the hold required time $r(i)$ of node g_i is computed as

$$r(i) = \max_k \{r(i, k) \,|\, r(i, k) = r(k) - D(k), e(i, k) \in E\}. \tag{4}$$

Definition 4: In [18], the setup edge slack $S(i, j)$ is the slack of edge $e(i, j) \in E$ contributed from node g_j back to node g_i

$$S(i, j) = R(i, j) - A(i). \tag{5}$$

The setup node slack $S(i)$ of node $g_i \in G$ is the slack of node g_i

$$S(i) = \min_j \{S(i, j) \,|\, e(i, j) \in E\} = R(i) - A(i). \tag{6}$$

Definition 5: In [18], the hold edge slack $H(i, j)$ is the slack of edge $e(i, j) \in E$ contributed from node g_j back to node g_i

$$H(i, j) = a(i) - r(i, j). \tag{7}$$

The hold node slack $H(i)$ of node $g_i \in G$ is the slack of node g_i

$$H(i) = \min_j \{H(i, j) \,|\, e(i, j) \in E\} = a(i) - r(i). \tag{8}$$

For example, as shown in Fig. 2(a), $H(2, 1) = 0.1 - 0.4 = -0.3$, $H(2, o_1) = 0.1 - 0.3 = -0.2$, and $H(2) = -0.3$.

Definition 6: The maximum padding delay $P_{\max}(i)$ of gate g_i is the padding delay converted from $C_{\max}(i)$. $P_{\max}(i)$ is 0 for a primary output or a flip–flop input. The safe padding value $P_{saf}(i)$ of gate g_i is computed as

$$P_{saf}(i) = \min\{S(i), |\min\{0, H(i)\}|, P_{max}(i)\}. \tag{9}$$

Lemma 1: The setup constraint is satisfied when the delay of a node g_i on a short path is increased by t, $t \le P_{saf}(i)$.

We define the fanout padding flexibility $P_F(i)$ for each gate to estimate the padding delay that can be applied on its whole fanout cone without violating setup constraints. For a hold satisfying gate or a primary output (a flip–flop input is considered as a pseudo primary output), the flexibility is zero. For a hold violating gate g_i, the fanout padding flexibility $P_F(i)$ is the hold slack difference between hold node slack $H(i)$ and the minimum updated hold edge slack $H(i, j)$ over all fanout edges if each gate in its fanout cone is padded with its safe padding value.

Definition 7: The fanout padding flexibility $P_F(i)$ of node $g_i \in G$ is computed as

$$P_F(i) = \begin{cases} 0, g_i \in PO \text{ or } H(i) \ge 0; \\ \min_{e(i,j)\in E}\{H'(i, j)\} - H(i), \text{otherwise} \end{cases} \tag{10}$$

where

$$H'(i, j) = H(i, j) + P_F(j) + P_{saf}(j).$$

$H'(i)$ and $S'(i)$ represent the updated slacks if each gate in g_i's fanout cone is padded with its safe padding value. $P_{saf}(i)$ is dynamically updated accordingly

$$H'(i) = \min_j \left\{H'(i, j) \,|\, e(i, j) \in E\right\} \tag{11}$$

$$S'(i) = \min_j \left\{S(i, j) - \left(S'(j) - S(j)\right) - P_{saf}(j) \,|\, e(i, j) \in E\right\}. \tag{12}$$

By definition, the fanout padding flexibility is thus calculated from primary outputs (and flip–flop inputs) toward primary inputs (and flip–flop outputs). Consider the case shown in Fig. 2(a). Assume $P_{\max}(2) = 0.5$, $P_{\max}(3) = 0.4$, and $P_{\max}(1) = 0.4$, respectively. According to (10)–(12), we have

$P_F(o_1) = 0.0$,
$P_F(FF_2) = 0.0$,
$P_F(1) = \min\{(-0.3 + 0.0 + 0.0)\} - (-0.3) = 0.0$,
$H'(1) = -0.3$,
$S'(1) = 0.4$,
$P_F(2) = \min\{(-0.2 + 0.0 + 0.0), (-0.3 + 0.0 + 0.3)\}$
$-(-0.3) = 0.1$,

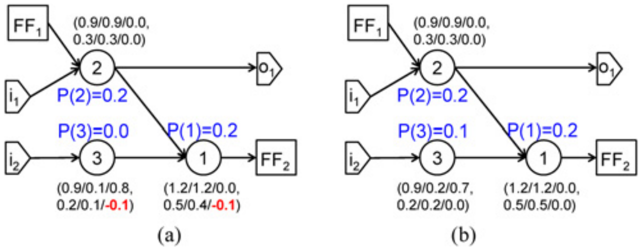


Fig. 6. Padding values of gates in Fig. 2. (a) Padding value $P(i)$ of node $g_i \in G$ after the first iteration of padding value decision: g_3 (+0.0) $\rightarrow$ g_2 (+0.2) $\rightarrow$ g_1 (+0.2). (b) Padding value $P(i)$ of node $g_i \in G$ after the second iteration of padding value decision: g_3 (+0.1).

$H'(2) = \min\{(-0.3 + 0.0 + 0.3), (-0.2 + 0.0 + 0.0)\} = -0.2$,
$S'(2) = \min\{(0.4 - 0.0 - 0.3), (0.3 - 0.0 - 0.0)\} = 0.1$,
$P_F(3) = \min\{(-0.3 + 0.0 + 0.3)\} - (-0.3) = 0.3$,
$H'(3) = \min\{(-0.3 + 0.0 + 0.3)\} = 0.0$,
$S'(3) = \min\{(1.0 - 0.0 - 0.3)\} = 0.7$.

Moreover, we have the following lemma to check the padding feasibility.

Lemma 2: If $|\min\{0, H(i)\}| > P_F(i)$, $g_i \in$ PI, the hold violations cannot be resolved by padding on gates. If $S(i, j) < |\min\{0, H(i, j)\}|$, $e(i, j) \in E$, the hold violations cannot be resolved by padding on wires.

V. Padding Value Determination

We propose a padding value determination algorithm to determine the padding values and locations with a global view in this section.

Basically, the more total padding delay, the more total padding overhead. Hence, we first target to minimize the total padding delay and then convert the padding delay of each gate/wire to padding load/buffers. However, the challenges are twofold: One is to find good locations to pad delay; the other is not to hurt the setup time.

Conceptually, padding on gates close to primary inputs can easily satisfy the timing constraints, but may increase the total padding values. Padding on gates shared by many short paths can lower total padding values, but may violate the timing constraints. As shown in Fig. 2, if we individually pad gates g_2 and g_3 with 0.3-unit delay, the timing constraints are satisfied, but the total padding value is somewhat large (+0.6). If we pad 0.3-unit delay on gate g_1 first, the short path through g_2 to primary output o_1 is unresolved [Fig. 2(b) and (c)]. Thus, to tackle these challenges, we shall determine the padding values and locations with a global view. Our idea is first to push the padding values toward the gates as close to outputs as possible and then to pull the values back to the gates with forked paths.

A. Padding Value Decision

After the fanout padding flexibility is calculated with a global view in Section IV-C, the padding value is decided accordingly. The padding value of each hold violating gate is derived in the topological order [19]. For each hold violating gate, the fanout padding flexibility is an estimated padding delay applied on its whole fanout cone. Then, the hold violating gate only needs to be padded to fix the remaining negative hold slack, i.e., the difference between the safe padding value and the fanout padding flexibility

$$P(i) = \max\left\{P_{saf}(i) - P_F(i), 0\right\} \tag{13}$$

where $P_{saf}(i)$ represents the safe padding value after gate g_i's fanin gates are padded. When the padding value of a gate is decided, the increased delay affects the arrival time of its fanout gates. The fanout edge slack of the padding gate should be updated accordingly

$$S(i, j) = R(i, j) - A(i) - P(i) \tag{14}$$

$$H(i, j) = P(i) + a(i) - r(i, j). \tag{15}$$

After updating the fanout edge slacks of each padding gate, the setup and hold node slacks of its fanout gates are also updated by (6) and (8).

Fig. 6(a) gives an example of padding value decision. Assume $P_{\max}(2) = 0.5$, $P_{\max}(3) = 0.4$, and $P_{\max}(1) = 0.4$, respectively. Based on the fanout padding flexibilities, we have

$P_{saf}(2) = \min\{0.3, |-0.3|, 0.5\} = 0.3$,
$P(2) = 0.3 - 0.1 = 0.2$,
$S(2,1) = 1.1 - 0.7 - 0.2 = 0.2$,
$H(2,1) = 0.2 + 0.1 - 0.4 = -0.1$,
$P_{saf}(3) = \min\{0.3, |-0.3|, 0.4\} = 0.3$,
$P(3) = 0.3 - 0.3 = 0.0$,
$S(3,1) = 1.1 - 0.1 - 0.0 = 1.0$,
$H(3,1) = 0.0 + 0.1 - 0.4 = -0.3$,
$S(1) = \min\{1.0, 0.2\} = 0.2$,
$H(1) = \min\{-0.3, -0.1\} = -0.3$,
$P_{saf}(1) = \min\{0.2, |-0.3|, 0.4\} = 0.2$,
$P(1) = 0.2 - 0.0 = 0.2$.

After the above padding value decision, the short path from g_3 to g_1 still has a negative hold slack, -0.1, because of the overestimated fanout padding flexibility. This short path can be resolved by applying another iteration of fanout padding value calculation plus padding value decision. The fanout padding flexibility calculation step and the padding value decision step are repeated until all hold violations are resolved or no more violations can be eliminated. With the iterative procedure, this procedure can determine the padding values and locations with a global view. As shown in Fig. 6(b), all short paths are resolved, and the result is same as the optimal solution [Fig. 2(d)].

B. Padding Value Refinement

Now, we further reduce the total padding delay on gates and resolve unfixed hold violations by padding wires.

In the padding value decision step, the padding locations are decided as close to primary outputs as possible (push). For a circuit with forked short paths, the total padding value is increased if the padding location is not determined on the gate where two or more short paths fork. Fig. 7(a) gives an example; gate g_4 has forked paths. After our padding value decision (Section IV-C), the padding values and locations are indicated beside each gate, and the total padding delay is 0.5. In fact, the total padding delay can be further reduced to 0.4 by changing the padding values and locations as shown in Fig. 7(b).

At refinement, we further reduce the padding values by pulling the padding values backward the gates where two or more short paths fork. To accomplish this task, we define the reverse padding value, the added safe padding value, and the refined padding value as follows.

Definition 8: The reverse padding value $P_{\text{rev}}(i)$ of gate g_{i} is computed as

$$P_{rev}(i) = \begin{cases} P(i), \textit{if } g_i \textit{ has only one hold violating fanin}; \\ \qquad\qquad 0, \textit{other wise}. \end{cases} \tag{16}$$

The reverse padding value of each gate g_i is to record how much padding can be propagated backward to its fanin gate. To avoid propagating padding values to joined paths, we consider the case that g_i has only one fanin with a hold violation. A fanin of g_i has a hold violation if the hold edge slack is smaller than the padding value of g_i. Furthermore, the padding value can be fully propagated in this case. The refined padding value of gate g_i is constrained by its setup slack and its maximum padding delay $P_{\max}(i)$.

Definition 9: The added safe padding value $P_{add}(i)$ of gate g_i is computed as

$$P_{add}(i) = \min\{S(i), P_{max}(i) - P(i)\}. \tag{17}$$

Definition 10: The refined padding value P_{ref}(i) of gate g_i is computed as

$$P_{ref}(i) = P(i) + \min\{P_{add}(i), \min_j\{P_{rev}(j)|H(i, j) < P(i)\}\}. \tag{18}$$

For each of g_i's fanout gate g_j, we have

$$P_{ref}(j) = P(j) - \min\{P_{add}(i), \min_j\{P_{rev}(j)|H(i, j) < P(i)\}\}. \tag{19}$$

Based on the above definitions, the refined padding values are calculated in the reverse topological order, and thus the total padding delay can be reduced.

Sometimes, the hold violations cannot be fully cleaned by padding on gates (extra load hook-up) due to insufficient setup slacks or maximum output capacitance constraints. In this case, we may further apply padding on wires (buffer insertion) after the above refinement.

Definition 11: The wire padding value $P(i, j)$ of $e(i, j)$ is

$$P(i, j) = \min\{S(i, j), |\min\{0, H(i, j)\}|\}. \tag{20}$$

The wire padding value is determined in the topological order. According to the timing library, the final padding delay of each gate/wire is then converted to an amount of padding load/buffers.

In some critical cases, the fanout padding flexibility may be overestimated. These cases may pass the feasibility checking by Lemma 2, but the hold violations cannot be completely resolved by padding gates/wires. In these cases, we return to Stage 1 to adjust the selected $S_{\text{th}}/H_{\text{th}}$.

C. Time Complexity

For each step in the padding value determination stage, the algorithm takes O(G) time to visit all gates in forward or backward topological order. When setup and hold slacks are changed, the algorithm takes O(E) time to update timing. Thus, the padding value determination stage can be done in O(GE) time.

VI. Load/Buffer Allocation

In this section, we realize the padding delay of each padding gate/wire at physical implementation. Because the available cell capacitances/delays are discrete for a given cell library; spare cells may not match the required padding load/buffer for a padding gate/wire. On the other hand, the dummy metal accommodates an abundant resource of capacitance [14] and can be tuned. The required padding is converted to an amount of capacitance and fixed by spare cell allocation and dummy metal insertion. For example, if spare cells offer either 0.15-unit or 0.25-unit delay, a 0.2-unit padding delay can be done by a spare cell of 0.15-unit delay plus a dummy metal of 0.05-unit delay.

To simultaneously allocate spare cells and dummy metal, we first give a mixed integer linear programming (MILP) formulation (see Section VI-A) and then propose a network-flow-based heuristic. The network-flow-based heuristic starts from a maximum flow solution (see Section VI-B). Then, infeasible spare cell assignments of the initial allocation are fixed (see Section VI-C). If the flow network is unsolved or has unfixed infeasible flow, we return to Stage 1 to adjust the padding resources of unfixed gates and selected $S_{\text{th}}/H_{\text{th}}$.

A. Mixed Integer Linear Programming Formulation

First of all, we collect available spare cells and the amount of available dummy metal resource for each padding gate/wire. For each padding gate (respectively, wire), the available spare cells located within the bounding box of its fanout net (respectively, the investigated wire) are extracted. If there is no spare cell located in the bounding box of its fanout net, we expand the bounding box with a user-defined row height. As shown in Fig. 8(a), s_1 and s_2 are the available spare cells of padding gate g_1, while s_2 is available for gate g_2. The amount of available dummy metal of each padding gate (respectively, wire) is the unoccupied routing resource upon the bounding box of its fanout net (respectively, the investigated wire). Different padding gates/wires may compete for the same metal resource if their corresponding bounding boxes overlap. For example, as shown in Fig. 8(b), the amount of dummy metal within independent bounding boxes of g_1 and g_2 is 0.1 and 0.15, respectively, and the amount of dummy metal within the overlapping region is 0.1.

The notations used in the MILP formulation are listed as follows.

1) G: Set of padding gates.
2) W: Set of padding wires.
3) S: Set of spare cells.
4) M: Set of dummy metal regions.

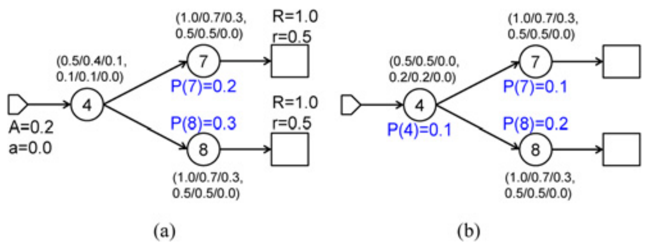


Fig. 7. Padding refinement. (a) Padding delay after padding value decision: g_7 (+0.2), g_8 (+0.3). Total padding delay is 0.5. (b) Ideal padding value: g_4 (+0.1), g_7 (+0.1), g_8 (+0.2). Total padding delay is 0.4.

5) $c_{i,j}$: Rewiring capacitance and the input capacitance if spare cell s_j is assigned to gate/wire g_i /w_i.
6) d_j: Capacitance of independent/overlapping dummy metal region m_j.
7) $x_{i,j}$: 0–1 integer variable that denotes if spare cell s_j is assigned to padding gate/wire g_i /w_i.
8) $y_{i,j}$: Floating variable that denotes the amount of dummy metal u_j assigned to padding gate/wire g_i /w_i.
9) C_i: The required padding capacitance of gate gate/wire g_i /w_i.

Based on the above notation, we formulate the MILP as follows:

$$\text{minimize} \sum_{i \in G+W} \left(C(i) - \sum_{j \in S} x_{i,j} c_{i,j} - \sum_{j \in M} y_{i,j} \right)$$

$$\text{subject to} \sum_{i \in G} x_{i,j} \leq 1, \forall j \in S \tag{21}$$

$$\sum_{i \in G} y_{i,j} \leq d_j, \forall j \in M \tag{22}$$

$$\sum_{j \in S} x_{i,j} c_{i,j} + \sum_{j \in M} y_{i,j} \leq C(i), \forall i \in M \tag{23}$$

$$x_{i,j} \in \{0.1\}, \forall j \in S \tag{24}$$

$$y_{i,j} \geq 0, \forall j \in M. \tag{25}$$

The objective function is to minimize the total difference between the padding values and padded capacitance. Constraint (20) ensures each spare cell input is assigned to at most one padding gate. Constraint (21) guarantees that the assigned amount of dummy metal does not exceed the available capacitance. Constraint (22) states that the total padded capacitance should not exceed the padding value of each padding gate.

In addition to the MILP formulation given here, we further propose a network-flow-based heuristic in Sections VI-B and VI-C. Later, our results show that compared with MILP, the network-flow-based heuristic is very efficient and effective.

B. Initial Allocation

The initial allocation of the network-flow-based heuristic is done by maximum network flow [19]. The flow network is composed of a node set and an edge set. The node set contains a source node s, a sink node t, node g_i represents a padding gate/wire, node s_i represents a spare cell within the bounding box of some padding gate/wire, and node m_i represents a dummy metal of some divided region. Similar to

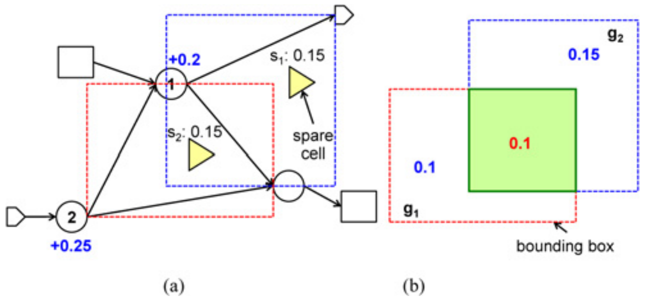


Fig. 8. Available space cell and dummy metal. (a) For padding gates g_1 and g_2, spare cells inside the bounding boxes of their fanout nets are extracted. (b) Available dummy metal of g_1 and g_2 are divided into three regions.

Section VI-A, if there is no spare cell located in the bounding box of its fanout net, we expand the bounding box with a user-defined row height. The edge set N is defined as follows. An edge connects s to each spare cell s_i, and its capacity $c(s, s_i)$ is the capacitance offered by s_i. An edge connects s to each dummy metal region m_i, and its capacity $c(s, m_i)$ is the capacitance offered by m_i. An edge connects s_i and g_i if g_i's bounding box covers s_i, and its capacity $c(s_i, g_i)$ is infinite. Similarly, an edge connects m_i and g_i if g_i's bounding box covers m_i, and its capacity $c(m_i, g_i)$ is infinite. An edge connects g_i and t, and its capacity $c(g_i, t)$ is the corresponding padding capacitance. Fig. 9(a) shows the corresponding flow network and the maximum flow of Fig. 8.

Based on maximum network flow, we can assign spare cells and dummy metal simultaneously to fix the required padding capacitance. If the obtained flow does not match the required padding values (i.e., the flow of the edge between g_i and t is unequal to its capacity), the padding resource is insufficient and we return to the $S_{\text{th}}/H_{\text{th}}$ decision stage to adjust $C_{\max}(i)/C_{\max}(i,j)$. Moreover, the obtained flow may contain infeasible assignments even when the maximum flow is achieved. The flow through a spare cell may be split to more than one gate [see s_2 in Fig. 9(b)], or the flow through a spare cell is less than its capacity [see s_1 in Fig. 9(b)]. For these cases, we propose a spare cell allocation refinement to fix the infeasible assignment (see Section VI-C).

C. Spare Cell Allocation Refinement

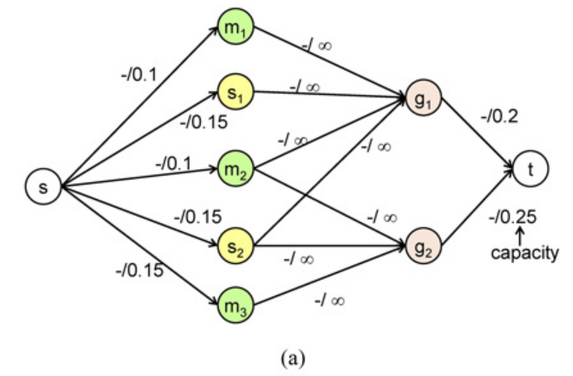
In this section, we refine the flow to fix infeasible spare cell assignments.

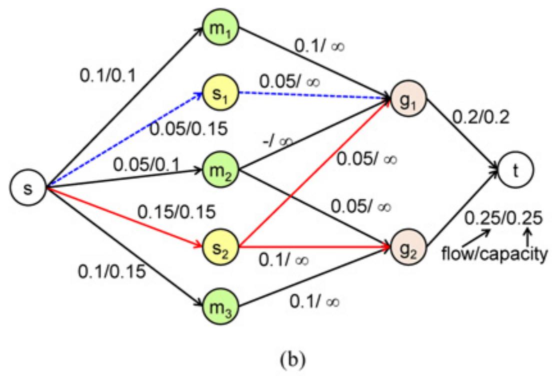
1) *Types of spare cell assignment:* One spare cell can be assigned to at most one padding gate/wire, and the corresponding flow is with full capacity. Due to the unsplit flow requirement of spare cells, we categorize the spare cell assignments into four types.

Type 1: The flow $f(s, s_i)$ from source s to a spare cell s_i is equal to its capacity $c(s, s_i)$, and s_i is assigned to exactly one padding gate/wire.

Type 2: The flow $f(s, s_i)$ from source s to a spare cell s_i is less than its capacity $c(s, s_i)$, and s_i is assigned to exactly one padding gate/wire.

Type 3: The flow $f(s, s_i)$ from source s to a spare cell s_i is equal to its capacity $c(s, s_i)$, and s_i is assigned to two or more padding gates/wires.





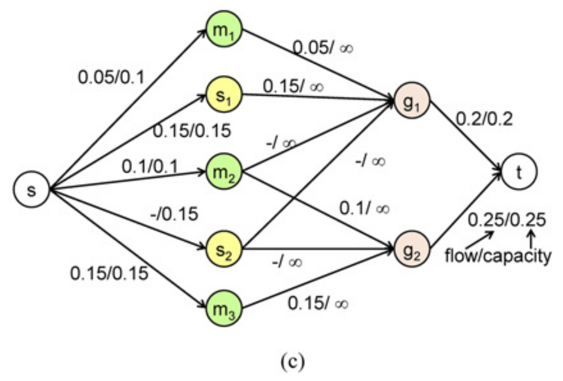


Fig. 9. Network-flow-based heuristic. (a) Flow network. (b) Initial allocation of Fig. 8. (c) Flow network after spare cell allocation refinement.

Type 4: The flow $f(s, s_i)$ from source s to a spare cell s_i is less than its capacity $c(s, s_i)$, and s_i is assigned to two or more padding gates/wires.

Only type 1 assignment is feasible. Fig. 9(b) has two infeasible assignments. Spare cell s_1 belongs to type 2 assignment: The flow from source s to spare cell s_1 is less than to its capacity, and spare cell s_1 is assigned to only padding gate g_1. Spare cell s_2 belongs to type 3 assignment: The flow from source s to spare cell s_2 is equal to its capacity, but spare cell s_2 is assigned to padding gates g_1 and g_2. We shall fix type 2, 3, and 4 infeasible assignments.

2) *Fill and Return Operations:* We introduce two basic operations, fill and return, to fix an infeasible assignment. Fill operation tries to fill padding gate/wire g_i with an extra flow F_{Fill}, and return operation tries to take off flow F_{Return} from padding gate/wire g_i.

Definition 12: For a padding gate/wire g_i connected with an infeasibly assigned spare cell s_j, the amount of dummy metal used $M_{used}(g_i)$ are defined as

$$M_{used}(g_i) = \sum \{ f(m_k, g_i) | (m_k, g_i) \in N, m_k \in M \}. \quad (26)$$

Definition 13: For a padding gate/wire g_i connected with an infeasibly assigned spare cell s_j, the amount of free dummy metal $M_{free}(g_i)$ are defined as

$$M_{free}(g_i) = \sum \{ c(s, m_k) - f(s, m_k) | (m_k, g_i) \in N, m_k \in M \}. \quad (27)$$

Definition 14: For a padding gate/wire g_i connected with an infeasibly assigned spare cell s_j, the amount of spare cells used $S_{used}(g_i)$ are defined as

$$S_{used}(g_i) = \sum \{ f(s_k, g_i) | (s_k, g_i) \in N, s_k \neq s_j, s_k \in S \}. \quad (28)$$

Definition 15: For a padding gate/wire g_i connected with an infeasibly assigned spare cell s_j, the amount of free spare cells $S_{free}(g_i)$ are defined as

$$S_{free}(g_i) = \sum \{ c(s, s_k) | f(s, s_k) = 0, (s_k, g_i) \in N, s_k \in S \}. \quad (29)$$

For example, as shown in Fig. 9(b), $M_{used}(g_1) = 0.1$, $M_{free}(g_1) = 0.05$, $S_{used}(g_1) = 0.1$ and $S_{free}(g_1) = 0.0$.

Definition 16: For a padding gate/wire g_i connected to an infeasibly assigned spare cell s_j, flow F_{Fill} is defined as

$$F_{Fill} = c\left(s, s_j\right) - f\left(s_j, g_i\right) \quad (30)$$

For fill operation, we cannot find another augmenting path to fill flow F_{Fill} because the flow is already maximum in this flow network. Alternatively, we collect other used resources of padding gate/wire g_i with flow F_{Fill} and return them. Then, the flow is still maximum after flow F_{Fill} is filled to the edge between the infeasibly assigned spare cell s_j and padding gate/wire g_i, successfully. As the capacitances offered by spare cells are discrete, fill operation collects used spare cells followed by the dummy metal.

Lemma 3: If a set of used spare cells $s_i \in S$ of padding gate/wire g_i and the related capacitances satisfy

$$F_{Fill} - M_{used}(g_i) \leq S_{used}(g_i) \leq F_{Fill} + M_{free}(g_i) + S_{free}(g_i). \quad (31)$$

Flow F_{Fill} can be filled to padding gate/wire g_i successfully.

Definition 17: For a padding gate/wire g_i connected with an infeasibly assigned spare cell s_j, flow F_{Return} is defined as

$$F_{Return} = f\left(s_j, g_i\right). \quad (32)$$

For return operation, we find other free resources of padding gate/wire g_i with flow F_{Return} and take F_{Return} off the edge between the infeasiblely assigned spare cell s_j and padding gate/wire g_i. Then, the flow is still maximum after flow F_{Return} is taken off from the edge between spare cell s_j and padding gate/wire g_i. Similarly, the return operation collects free spare cells followed by free dummy metal.

```
FixType2Infeasible( $F$, $s_i$, $g_i$)
// ( $s_i$, $g_i$ ) $\in$ N and $f(s, s_i) = f(s_i, g_i) > 0$
1. fill $g_i$ with remaining flow $c(s, s_i) - f(s, s_i)$
2. if ( fill operation fails on edge ($s_i$, $g_i$) )
3.     return existing flow $f(s_i, g_i)$ from $g_i$
4.     if ( return operation fails on edge ($s_i$, $g_i$) )
5.         go to the $S_{th}/H_{th}$ decision stage
```

(a)

```
FixType3and4Infeasible( $F$, $s_i$)
1. for each edge ($s_i$, $g_i$) and $f(s_i, g_i) \neq 0$
2.     return existing flow $f(s_i, g_i)$ from $g_i$
3.     if ( return operation fails on edge ($s_i$, $g_i$) )
4.         add $g_i$ to fail list
5. if ( only one gate fails to return flow )
6.     fill $g_i$ with remaining flow $c(s, s_i) - f(s, s_i)$
7.     if ( fill operation fails on edge ($s_i$, $g_i$) )
8.         go to the $S_{th}/H_{th}$ decision stage
9. else if ( more than one gate fails to return flow )
10.    go to the $S_{th}/H_{th}$ decision stage
```

(b)

Fig. 10. Spare cell allocation refinement. (a) Algorithm for fixing type 2 infeasible flow. (b) Algorithm for fixing type 3 and 4 infeasible flow.

Lemma 4: If a set of free spare cells $s_i \in S$ of padding gate/wire g_i and the related capacitances satisfy

$$F_{Return} - M_{free}(g_i) \leq S_{free}(g_i) \leq F_{Return} + M_{used}(g_i) + S_{used}(g_i). \quad (33)$$

Flow F_{Return} can be returned from padding gate/wire g_i successfully.

3) *Refinement Methods:* Using these two operations, fill and return, we can fix the infeasible flow efficiently.

As type 2 infeasible flow of some spare cell only fills one padding gate, finding used resources is easier than finding free resources, i.e., applying fill operation is easier than applying return operation. The refinement algorithm is listed in Fig. 10(a). The fill operation fills the remaining capacity of an infeasible spare cell to the padding gate. When fill operation fails, we further apply return operation to return the existing flow. If the infeasibly assigned spare cell cannot be fixed by neither fill operation nor return operation, we go back to padding resource collection step to adjust $C_{\max}(i)/C_{\max}(i,j)$ and redo PushPull algorithm.

For type 3 and type 4 infeasible flows, applying return operation is more efficient than applying fill operation because the flow from some spare cell fills to two or more padding gates. The refinement algorithm is listed in Fig. 10(b). First of all, we return all existing flow from the padding gates/wires which are filled by the infeasible spare cell. If all flows through the infeasible spare cell are returned, the refinement is finished. If all flows from the spare cell can be returned except only one padding gate, fill operation is applied to fill all capacity to this gate. However, if both return and fill operations fail, or there are two or more unreturned flows, we return to Stage 1 to adjust padding resources of unfixed gates and the selected $S_{\text{th}}/H_{\text{th}}$. Fig. 9(c) shows the flow network after spare cell allocation refinement.

```
CCSDelayLoadTranslator($S_{input}$, $C_{load}$, $C_{max}$, $D_{target}$)
//$S_{input}$: input slew
//$C_{load}$: present output loading
//$C_{max}$: maximum output loading
//$D_{target}$: target delay
1. $C_{middle} \leftarrow (C_{load} + C_{max})/2$
2. $D_{middle} \leftarrow$ Get delay from CCS model with $S_{input}$, $C_{middle}$
3. if ( $D_{target} > D_{middle}$ ) //target load is in upper subset
4.     return CCSDelayLoadTranslator($S_{input}$, $C_{middle}$, $C_{max}$, $D_{target}$)
5. else if ( $D_{target} < D_{middle}$ ) //target load is in lower subset
6.     return CCSDelayLoadTranslator($S_{input}$, $C_{load}$, $C_{middle}$, $D_{target}$)
7. else  //targe load is found
8.     return $C_{middle}$
```

Fig. 11. Algorithm to transform delay and output loading with CCS model.

D. Time Complexity

The notations used to analyze the time complexity of Stage 3 are listed as follows.

1) N_{GW}: The number of padding gates and wires.
2) N_{SM}: The number of padding resources, including spare cells and dummy metal.
3) N_{SM}': The number of padding resources whose flow will be filled (returned) during the fill (return) operation.
4) N_E: The number of edges in the flow network.
5) N_E': The number of edges whose flow will be changed during the fill (return) operation.
6) T_2: The number of type 2 infeasible assignments.
7) T_3: The number of type 3 infeasible assignments.
8) T_4: The number of type 4 infeasible assignments.

The proposed load/buffer allocation heuristic consists of two steps: Initial allocation and spare cell allocation refinement. The initial allocation step takes $O(N_{GW} + N_{SM} + N_E)$ time to construct the flow network, and we use the improved shortest augmenting path (ISAP) algorithm [21], [22] to solve the maximum flow problem. Because of the special structure of our flow network, ISAP takes only $O(N_{GW}N_{SM})$ time to obtain maximum flow, instead of $O((N_{GW} + N_{SM})^2 N_E)$ time. In the spare cell allocation refinement step, the fill/return operation takes $O(N_{SM}' + N_E')$ to fill/return flow. Type 2 refinement takes $O(T_2(N_{SM}' + N_E'))$ time, while types 3 and 4 take $O((T_3 + T_4)(N_{SM}' N_E'))$ time. If the number of infeasible assignments is much smaller than the number of nodes in the flow network, the proposed load/buffer allocation eventually takes $O(N_{GW}N_{SM})$ time.

VII. Extension To CCS Models

The CCS models may be adopted instead of nonlinear delay models (NLDMs) for advanced technology.

Unlike NLDMs, the CCS models consist of a driver model and a receiver model. The driver model is a time and voltage dependent current source. The receiver model is composed of two capacitances. The two models are also dependent on input slew and output loading. The main discrepancy between NLDMs and CCS models is timing characterization. The CCS models the cell response as a current waveform, not a specific delay value.

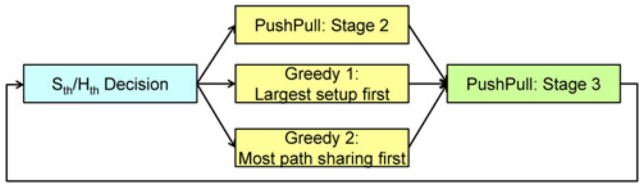

PushPull: Stage 2
S_{th}/H_{th} Decision
Greedy 1: Largest setup first
Greedy 2: Most path sharing first
PushPull: Stage 3

Fig. 12. Experimental flow of evaluating the effect of S_{th}/H_{th} decision.

TABLE I
Benchmark Statistics

Circuit	#Gate	#FF	#SFF	Conservative clock period (ns)	THS (ps)
s1196	301	19	1	1.0	152.5
s1423	486	74	45	1.0	4,916.9
s5378	739	162	37	1.0	3,852.8
s9234	555	132	24	1.0	1,929.0
s13207	748	213	14	1.0	371.2
s15850	428	128	29	1.0	2,114.6
s38584	7,890	1,159	194	3.4	108,759.0
des_perf	51,349	8,808	1,190	2.9	583,579.0
b19	72,872	5,541	2,737	3.8	1,481,800.0

TABLE II
Selected S_{th}/H_{th}

Circuit	PushPull		Greedy 1		Greedy 2	
	S_{th}	H_{th}	S_{th}	H_{th}	S_{th}	H_{th}
s1196	70%	30%	70%	30%	70%	30%
s1423	70%	30%	87%	13%	81%	19%
s5378	70%	30%	84%	16%	85%	15%
s9234	70%	30%	77%	23%	77%	23%
s13207	70%	30%	81%	19%	81%	19%
s15850	70%	30%	73%	27%	78%	22%
s38584	72%	28%	94%	6%	94%	6%
des_perf	71%	29%	92%	8%	81%	19%
b19	75%	25%	93%	7%	92%	8%

To replace NLDMs with CCS models in PushPull, we construct a translator to transform from a specified delay to a relative output loading. Given the input slew, the present output loading, the maximum output loading, and a target cell delay, we can easily find the target padding load by binary search. The delay-load translator algorithm is listed in Fig. 11.

VIII. Experimental Results

We implemented our algorithm in the C++ programming language and executed the program on a platform with an Intel Xeon 3.8 GHz CPU and with 16 GB memory under CentOS 5.5. Experiments are conducted on the IWLS 2005 benchmark circuits [20] through the resilient circuit design flow (Section II-A). Table I lists benchmark statistics, where Circuit indicates the circuit name, #Gate lists the combinational logic gate count, #FF means the number of flip–flops, #SFF is the number of timing suspicious flip–flops, Conservative clock period means the clock period considering a timing guardband, and THS represents the total negative hold slack contributed from suspicious flip-flops. Each circuit is synthesized, placed, and routed based on 55-nm technology using state-of-the-art commercial tools [23], [24]. We also use these tools to verify the circuit timing.

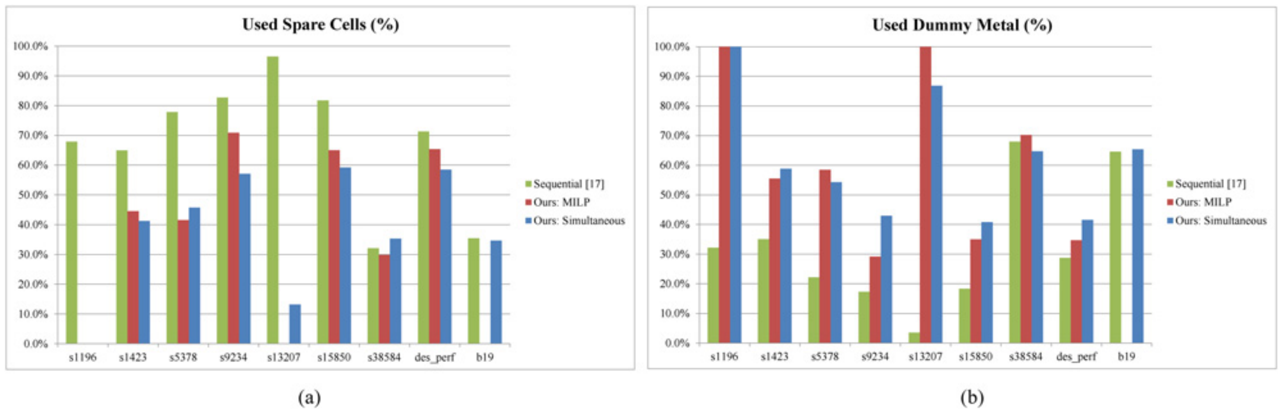


Fig. 13. Resource usage comparison. (a) Percentage of used spare cells. (b) Percentage of used dummy metal.

TABLE III
STAGE 2: PADDING VALUE DETERMINATION (S_{th} = 75%, H_{th} = 25%)

Circuit	LP [6]				Greedy 1: Largest setup first [12]					Greedy 2: Most path sharing first [11][12][13]					Ours: PushPull				
	TNS_1 (ps)	THS_1 (ps)	Padding delay (ps)	Runtime (s)	TNS_1 (ps)	THS_1 (ps)	Padding delay (ps)	#Ite.	Runtime (s)	TNS_1 (ps)	THS_1 (ps)	Padding delay (ps)	#Ite.	Runtime (s)	TNS_1 (ps)	THS_1 (ps)	Padding delay (ps)	#Ite.	Runtime (s)
s1196	0.0	0.0	152.5	0.03	0.0	0.0	338.4	5	0.02	0.0	0.0	173.8	3	0.01	0.0	0.0	152.5	1	0.02
s1423	0.0	0.0	4,126.8	0.15	0.0	43.5	4,968.7	56	0.45	0.0	43.5	4,802.8	35	0.24	0.0	0.0	4,746.9	2	0.05
s5378	0.0	0.0	3,661.3	0.06	0.0	79.3	4,678.1	60	0.79	0.0	79.3	4,555.4	43	0.58	0.0	0.0	3,722.7	2	0.08
s9234	0.0	0.0	1,459.6	0.05	0.0	12.0	1,878.3	33	0.30	0.0	12.0	2,158.5	27	0.26	0.0	0.0	1,647.5	1	0.05
s13207	0.0	0.0	371.2	0.03	0.0	34.5	762.3	22	0.21	0.0	34.5	621.5	18	0.21	0.0	0.0	371.2	2	0.07
s15850	0.0	0.0	1,305.1	0.03	0.0	0.0	1,662.8	43	0.25	0.0	0.0	2,161.5	41	0.27	0.0	0.0	1,510.8	2	0.04
s38584	0.0	0.0	108,476.0	6.96	0.0	0.0	225,026.0	780	115.19	0.0	85.8	130,703.0	512	80.37	0.0	0.0	143,764.1	2	1.04
des_perf	0.0	0.0	583,579.0	60.59	0.0	19,270.5	795,022.0	3,414	6,401.86	0.0	3,864.1	595,088.0	2,257	4,202.46	0.0	0.0	583,579.0	2	9.46
b19	NA	NA	NA	timeout	0.0	108,078.0	6,128,710.0	21,902	37,473.30	0.0	65,746.7	1,729,230.0	6,220	10,978.40	0.0	0.0	1,675,688.0	4	21.68

TNS1: Total negative setup slack after padding value determination.
THS1: Total negative hold slack after padding value determination. timeout: Runtime exceeds 12 hours.
#Ite.: Number of iterations. Greedy 1 and Greedy 2 pad one hold violating gate at a time; #Ite. means how many times the greedy method run. For PushPull, the padding value determination and fanout padding flexibility calculation will be applied iteratively until all hold violations are resolved or no more violations can be eliminated; #Ite. means how many times these steps are applied.

TABLE IV
RUNTIME BREAKDOWN OF STAGE 2

Circuit	PushPull Stage 2		
	Padding Value Decision (s)	Padding Value Refinement (s)	Total Runtime (s)
s1196	0.01	0.01	0.02
s1423	0.02	0.03	0.05
s5378	0.03	0.05	0.08
s9234	0.02	0.03	0.05
s13207	0.02	0.05	0.07
s15850	0.02	0.02	0.04
s38584	0.51	0.53	1.04
des_perf	1.90	7.56	9.46
b19	8.76	12.92	21.68

For evaluating the impact of PushPull on dynamic S_{th}/H_{th} decision, we compare Stage 2 of PushPull with state-of-the-art work [11]–[13], as shown in Fig. 12. Greedy 1 greedily pads from the gate with the largest setup slack [12], while Greedy 2 greedily pads from the gate passed by most hold violating paths [11]–[13]. The initial selected S_{th} and H_{th} are 70% and 30%, respectively. All methods pad short paths without hurting setup time. Our load/buffer allocation method realizes delay padding at postlayout. Table II lists the resulting S_{th}/H_{th} based on the experimental flow in Fig. 12. It can be seen that PushPull achieves the minimum clock period (i.e., least S_{th} and greatest H_{th}) at each case.

Table III compares our padding value determination method with optimal [6] and two greedy methods. S_{th} = 75% and H_{th} = 25%, the values used in this experiments are typical settings in modern designs. Padding delay indicates the total assigned padding delay (including gates and wires). LP uses linear programming (we implemented [6] using CPLEX [25]). Because of the global view, our method cleans all hold violations for each case. LP is time consuming for large-scale circuits, but interestingly LP is efficient for small cases. In contrast, Greedy 1 and Greedy 2 may either fail to clean all hold violations or suffer from long runtime. At each iteration, Greedy 1 and Greedy 2 fix the selected gate and update the timing for the entire circuit. Thus, two traversals are performed at each iteration: One traversal updates arrival times (forward), while the other updates required times (backward). Because of the local view, Greedy 1 and Greedy 2 may induce inefficient padding and thus require more iterations (longer runtime). In some cases, although the two greedy methods clean all hold violations at the padding value determination stage, they may still fail in delay padding at the postlayout stage, such as s38584 for Greedy 1 and s15850 for Greedy 2. In addition, although not listed here, in our experiments, the refinement in

TABLE V
STAGE 3: LOAD/BUFFER ALLOCATION

Circuit	LP+Mapping [6]				Sequential [17]			Ours: MILP			Ours: Simultaneous			
	TNS_2 (ps)	THS_2 (ps)	Padded area (um^2)	Runtime (s)	TNS_2 (ps)	THS_2 (ps)	Runtime (s)	TNS_2 (ps)	THS_2 (ps)	Runtime (s)	TNS_2 (ps)	THS_2 (ps)	Padded area (um^2)	Runtime (s)
s1196	0.0	0.3	5.12	0.03	0.0	0.0	0.33	0.0	0.0	0.30	0.0	0.0	20.0	0.02
s1423	0.0	826.4	145.28	0.09	0.0	0.0	0.43	0.0	0.0	0.41	0.0	0.0	2,256.9	0.04
s5378	0.0	302.3	156.48	0.05	0.0	0.0	0.54	0.0	0.0	0.41	0.0	0.0	774.4	0.04
s9234	0.0	631.3	38.72	0.04	0.0	0.0	0.41	0.0	0.0	0.34	0.0	0.0	476.9	0.02
s13207	0.0	161.2	16.96	0.03	0.0	0.0	0.61	0.0	0.0	0.33	0.0	0.0	68.1	0.04
s15850	0.0	352.0	60.16	0.02	0.0	0.0	0.38	0.0	0.0	0.33	0.0	0.0	406.7	0.04
s38584	0.0	29,970.7	3,175.36	0.58	0.0	0.0	1.96	0.0	0.0	33.15	0.0	0.0	51,150.4	1.14
des_perf	0.0	51,146.0	19,789.40	41.99	0.0	0.0	12.36	0.0	0.0	33.45	0.0	0.0	151,246.7	8.80
b19	NA	NA	NA	timeout	0.0	0.0	43.33	NA	NA	timeout	0.0	0.0	530,484.6	98.12

TNS2: Total negative setup slack after load/buffer allocation.
THS2: Total negative hold slack after load/buffer allocation.

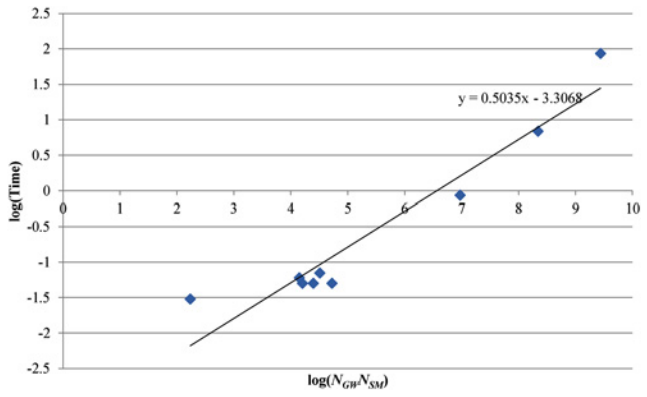


Fig. 14. Log-log graph of the runtime (y-axis) versus $N_{GW}N_{SM}$ (x-axis). Empirically, PushPull can be done in O((N_{GW} N_{SM})$^{0.5}$) time.

the padding value determination step averagely improves the padding delay by 4.78%.

Furthermore, Table IV lists the runtime breakdown of Stage 2 (padding value decision and padding value refinement). We also conducted one experiment to test the impact of input slew on padding delay capacitance conversion. Consider the error rate of the converted padding capacitance with and without input slew consideration. For each padding gate/wire, error rate = |1 − (padding capacitance without slew)/(padding capacitance with slew)| × 100%. The average error rate over all cases is quite small, only 0.56%.

Table V compares our load/buffer allocation method with [6] and [17]. LP + Mapping is the heuristic proposed in [6] to map the LP results to available cells. As mentioned in Section I, even Shenoy *et al*. [6] finds the optimal padding, the directly mapped results may still incur hold violations. Sequential is the heuristic proposed in [17], where coarse-grained delay padding allocates spare cells first and then fine-grained padding allocates a dummy metal. During coarse-grained delay padding, spare cells are assigned without dummy metal consideration. Hence, it consumes more spare cells. MILP means the results of the MILP formulation given in Section VI-A. Although it can achieve the padding values assigned by Stage 2, the runtime might be too long for large cases. Simultaneous means our network-flow-based heuristic. Our network-flow-based heuristic assigns spare cells and the dummy metal simultaneously. Because of flexible and tunable dummy metal, we can successfully achieve the padding values assigned by Stage 2 for all cases. Padded area means the total area over all layers contributed by selected spare cells and the dummy metal. The combination of spare cells and the dummy metal provides the flexibility to allocate load/buffers, and thus dummy metal can indeed solve the discrete delay problem well. Fig. 13 shows the resource usage of three methods. Sequential uses much more spare cells than the other two methods. Our network-flow-based heuristic obtains similar results as MILP, but shorter runtimes.

Based on Tables I–V, it can be seen that our short-path padding framework, PushPull, can select a small target clock period and solve the short-path padding (hold time fixing) problem for resilient circuits. Moreover, we reduce 25∼30% of the clock period for all cases with 4.81% and 3.06% increment on the average HPWL and power, respectively.

The time complexity of PushPull is O($GE + N_{GW}N_{SM}$). Since the flow network is larger than the timing graph, PushPull eventually takes O($N_{GW}N_{SM}$) time. After regression analysis, Fig. 14 shows that empirically, PushPull can be done in only O(($N_{GW}N_{SM}$)$^{0.5}$) time.

With adopting CCS model in PushPull, we resynthesize our benchmark circuits with the Nangate 45 nm Open Cell Library [26]. To illustrate the error rate of our proposed delay-load translator, we compare the path required time with the path arrival time with hooking up the translated load. For each hold violating path, the error rate = |1 – (path arrival time with hooking up the translated load)/(path required time)| × 100%. Empirically, the average error rate is only 0.36%.

IX. CONCLUSION

Resilient circuits are recently proposed to mitigate dynamic variations. In this paper, to enable the timing error detection and correction mechanism of resilient circuits, we focused on target clock period selection and the severe short-path padding problem in resilient circuits. Unlike greedy heuristics adopted by recent prior work, we determined the padding values and locations with a global view. Moreover, to further realize the determined padding values at physical implementation, we proposed an MILP formulation and a network-flow-based load/buffer allocation heuristic to assign spare cells and dummy metal simultaneously. Experimental results showed the efficiency and effectiveness of our method. Our network-flow-based allocation method can successfully achieve the derived padding values which may be infeasible when only discrete delays/capacitances are used. In addition, we can extend PushPull to adopt CCS timing model in advanced technology.

REFERENCES

[1] D. Ernst, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, *et al*., "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. MICRO.*, 2003, pp. 7–18.

[2] D. Blaauw, S. Das, C. Tokunaga, S. Pant, S. Kalaiselvan, K. Lai, *et al*., "RazorII: In situ error detection and correction for PVT and SER tolerance," in *Proc. ISSCC*, 2008, pp. 400–622.

[3] K. A. Bowman, J. W. Tschanz, J. C. Lee, C. B. Wilkerson, S. L. Lu, T. Karnik, *et al*., "Energy-efficient and metastability-immune timing-error detection and instruction-replay-based recovery circuits for dynamic-variation tolerance," in *Proc. ISSCC*, 2008, pp. 402–623.

[4] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32 bit ARM processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," *IEEE JSSC*, vol. 46, no. 1, pp. 18–31, Jan. 2011.

[5] K. Bowman, J. W. Tschanz, S. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, *et al*., "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE JSSC*, vol. 46, no. 1, pp. 194–208, Jan. 2011.

[6] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proc. ICCAD*, 1993, pp. 156–161.

[7] J. P. Fishburn, "Clock skew optimization," *IEEE TC*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[8] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proc. ISCAS*, 1994, pp. 407–410.

[9] S.-H. Huang, C.-H. Cheng, C.-M. Chang, and Y.-T. Nieh, "Clock period minimization with minimum delay insertion," in *Proc. DAC*, 2007, pp. 970–975.

[10] C. Lin and H. Zhou, "Clock skew scheduling with delay padding for prescribed skew domains," in *Proc. ASP-DAC*, 2007, pp. 541–546.

[11] S. Yoshikawa, "Hold time error correction method and correction program for integrated circuits," U.S. Patent 6 990 646, Jan. 2006.

[12] Y. Sun, J. Gong, and C.-T. Chen "Method and apparatus for fixing hold time violations in a circuit design," U.S. Patent 7 278 126, Oct. 2007.

[13] Y. Liu, F. Yuan, and Q. Xu, "Re-synthesis for cost-efficient circuit-level timing speculation," in *Proc. DAC*, 2011, pp. 158–163.

[14] Y. Kim, D. Petranovic, and D. Sylvester, "Simple and accurate models for capacitance increment due to metal fill insertion," in *Proc. ASP-DAC*, 2007, pp. 456–461.

[15] (2006, Dec.). *Liberty library modeling: The semiconductor industry's most widely used library modeling standard*, [Online]. Available: http://www.opensourceliberty.org/

[16] Y.-P. Chen, J.-W. Fang, and Y.-W. Chang, "ECO timing optimization using spare cells," in *Proc. ICCAD*, 2007, pp. 530–535.

[17] Y.-M. Yang, I. H.-R. Jiang, and S.-T. Ho, "PushPull: Short path padding for timing error resilient circuits," in *Proc. ISPD*, 2013, pp. 50–57.

[18] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Berlin, Germany: Springer-Verlag, 2006.

[19] J. Kleinberg and E. Tardos, *Algorithm Design*. Reading, MA, Addison Wesley, 2006.

[20] (2005, Jun.). *IWLS Benchmarks*, [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*. Englewood Cliffs, NJ, USA: Prentice Hall, 1993.

[22] R. K. Ahuja and J. B. Orlin, "Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems," *NRL*, vol. 38, no. 3, pp. 413–430, 1991.

[23] (2009, Nov.). *Synopsys Design Compiler*, [Online]. Available: http://www.synopsys.com

[24] (2012, Apr.). *Cadence SoC Encounter*, [Online]. Available: http://www.cadence.com

[25] (2009, May). *IBM ILOG CPLEX Optimizer*, [Online]. Available: http://www.ilog.com/products/cplex/

[26] (2008, Feb.). *Nangate 45 nm Open Cell Library*, [Online]. Available: http://www.nangate.com

Yu-Ming Yang received the B.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2008, where he is currently pursuing the Ph.D. degree from the Institute of Electronics.

His current research interests include large-scale integration physical designs, with emphases on timing optimization and low power design.

Iris Hui-Ru Jiang (M'07) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1995 and 2002, respectively.

From 2002 to 2005, she was with VIA Technologies, Inc., New Taipei, Taiwan. She is currently an Associate Professor with the Department of Electronics Engineering and the Institute of Electronics, NCTU. Her current research interests include physical design optimization and interaction between logic design and physical synthesis.

Dr. Jiang is a member of the Phi Tau Phi Scholastic Honor Society.

Sung-Ting Ho received the B.S. degree in electronics engineering from National Central University, Jhongli, Taiwan, in 2008 and the M.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2010. He is currently with CMSC, Inc.

His current research interests include physical design automation.