Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints

Kwangsoo Han[‡], Andrew B. Kahng[†‡] and Hyein Lee[‡]
[†]CSE and [‡]ECE Departments, UC San Diego, La Jolla, CA 92093
{kwhan, abk, hyeinlee}@ucsd.edu

***Abstract*—Technology scaling to 10nm and below introduces complex intra-row and inter-row constraints in standard-cell detailed placement. Examples of such constraints are found in rules for drain-drain abutment, minimum implant region area and width, oxide diffusion (OD) notching and jogging, etc. Typically, these rules are too complex for the normal global-detailed placement flow to fully consider. On the other hand, guardbanding the library cell design so that arbitrary cell placement adjacencies are all "correct by construction" has increasingly high area cost. This motivates the introduction of a final legalization phase for standard-cell placement tools in advanced (particularly 10nm and 7nm) foundry nodes. In this work, we develop a mixed integer-linear programming (MILP)-based placer, called DFPlacer, for final-phase design rule violation (DRV) fixing. DFPlacer finds (near-)DRV-free solutions considering various complex layout constraints including minimum implant width, drain-drain abutment, and oxide diffusion jogs. To overcome the runtime limitation of MILP-based approaches, we implement a distributable optimization strategy based on partitioning of the block layout into windows of cells that can be independently legalized. Using layouts in an abstracted 7nm library, we find that DFPlacer fixes 99% of DRVs on average with minimal impacts on area and timing. We also study an area-DRV tradeoff between two types of standard-cell library strategies, namely, with and without dummy poly gates.**

I. Introduction

Continued technology scaling to the foundry 10nm node (42nm minimum metal pitch, 36nm fin pitch) and below leads to more constraints in physical implementation. Not only do new metal-layer (back end of line, or BEOL) ground rules arise from multi-patterning techniques, but rules for device layers (front end of line, or FEOL) also become considerably more complex and restricted. For example, at the foundry 10nm node (henceforth referred to as **N10**), there are minimum width and area constraints for implant regions, as well as notch and jog width constraints for oxide diffusion (OD) regions. In older technology nodes, such layer rules were fairly benign: while of concern to the library cell designer, once the library cells were correctly designed, design rule violations (DRVs) could not occur during placement due to the correctness by construction of any non-overlapping cell placement.

Unfortunately, correctness by construction no longer holds for detailed placement at N10 and below. Cell sizes and minimum metal pitches have continued shrinking to stay on the Moore's Law density curve. However, patterning resolution in device (FEOL) layers has not kept pace due to challenges in device definition (e.g., ion implant) or lithographic variation (e.g., corner rounding). Thus, placing several 'legal' standard-cell layouts next to each other may cause violations of FEOL layer rules such as minimum implant width or area [4] rules. Such violations could in theory be prevented with larger cell area budgets (similar in spirit to how BEOL colorability, especially on the M1 layer, can be preserved) that permit correct-by-construction cell layout styles. However, this runs counter to a core purpose of shrinking to the next node, and reduces the return on investment from enabling that node. Our present work proposes a new, final phase of detailed cell placement that can potentially maintain placement legality in the face of new N10 FEOL rules – without loss of density, routability or performance metrics.

A. N10 FEOL and Cell Placement Constraints

Figure 1 illustrates the layout of an inverter cell in the N10 node. The figure shows two fins each for PMOS and NMOS.[1] Source nodes of PMOS and NMOS are connected to M2 power/ground rails with M1. The input ***A*** is connected to the PMOS and NMOS gates using middle-of-line (MOL), a complementary metal layer below M1 that is used for intra-cell routing. The output ***Y*** is connected to the drain nodes of PMOS and NMOS. The FEOL layers which affect legal placement (i.e., in the context of other cells' placements) include implant layer, OD layer and poly, as follows.

- Implant layers, which indicate regions for ion implantation, decide the threshold (V_t) of transistors. Regions of the implant layer are typically aligned to the boundaries of standard cells.
- Oxide diffusion (OD) defines the active region of transistors.
- Dummy poly gates are inserted at the (vertical) standard cell boundaries to avoid edge device variability.

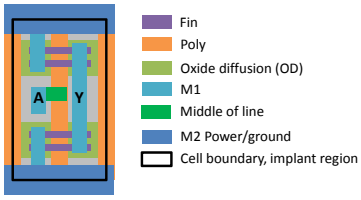

Fig. 1. Illustration of inverter cell layout in N10 node.

Minimum implant width constraints (IW). Minimum implant width (IW) constraints induce placement illegalities due to both inter- and intra-row IW violations, as shown in Figure 2(a). Below, we refer to the inter-row IW violation as being of type IW1. We refer to the intra-row IW violation as being of type IW2. An example of IW1 is shown in the figure, where two same-V_t cells are misaligned vertically and thus result in a narrow, "staircase" implant layer shape. IW2 occurs when a narrow cell is sandwiched between different-V_t cells, which results in a narrow implant region. Interestingly, the IW rules cause interactions between placement and sizing optimizations (e.g., V_t-swapping) that compromise the notion of, e.g., "post-route leakage optimization". This interaction has been recently studied in [4].

Minimum OD jog length (OW). Standard cells can have different oxide diffusion (OD) region heights according to functionality, drive strength, etc. When cells with different OD heights abut, OD jogs can result as shown in Figure 2(b). This is forbidden in N10 and below due to lithographic corner rounding, and the resulting device performance variability, e.g., under misalignment. In N10, a minimum OD jog length rule is violated if the jog length is less than

[1] A more typical library in N10 might have 9-track (M2 tracks) cell height, and three fins each for PMOS and NMOS, with a gear ratio of M2:fin pitch anywhere from 7:6 to 4:3.

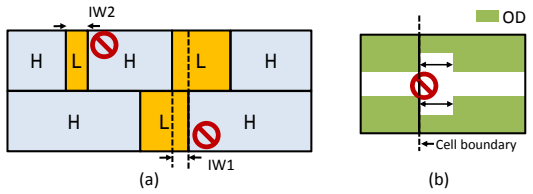

Fig. 2. (a) Examples of minimum implant width violations [14]. (b) The design rule for OD jogs.

a given minimum value. Introducing sufficient spacing between the violating cells can cure the OD jog violation.

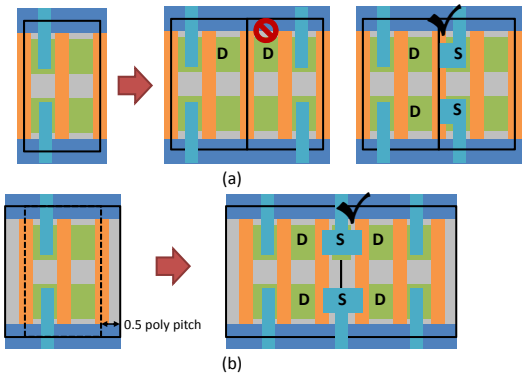

Fig. 3. (a) Drain-drain abutment violation with an example standard cell layout. (b) Use of dummy poly gates in the library design style can avoid DDA violation in a correct-by-construction manner.

Drain-drain abutment (DDA). Dummy poly gates create extra dummy transistors connected to logic transistors within standard cells. The dummy transistors can induce leakage power and logic failure if they are not fully turned off. Hence, gate and source nodes of dummy transistors must be tied off to power/ground rails; in particular, if two drain nodes are abutted, an extra dummy poly gate is needed to create an additional source node to be tied up with power/ground rails. The recent work of Du and Wong [6] studies cell instance flipping as a way of mitigating this issue in detailed placement. Figure 3(a) depicts the DDA problem. The leftmost diagram shows an example inverter layout, and the middle and right diagrams respectively show DDA and no-DDA cases. To avoid the DDA problem, we can consider two approaches [11]: (i) a smart detailed placement with comprehension of DDA; and (ii) standard cells with embedded dummy poly gates as shown in Figure 3(b). With approach (ii), the width overhead for each cell is one poly pitch. (We study the area-DRV tradeoff between approaches (i) and (ii) in Section IV below.)

B. This Work

As noted above, [4] and [6] have respectively made initial studies of IW- and DDA-induced placement issues. However, to our knowledge, there is no existing work that addresses all the issues above simultaneously in detailed placement. Popular techniques used for conventional placement legalization, including graph-based, dynamic programming-based, etc. appear ill-suited to handling of complex FEOL layer rules at N10 and below. For example, previous techniques have focused on removing overlaps between cells while maintaining the ordering of cells within a row, while minimizing half-perimeter wirelength or placement perturbation. Such previous works are largely single-row-based, and are applied row by row. Thus, they do not capture *inter-row* constraints such as IW1 that arise in N10. Furthermore, a number of implicit assumptions made by placement legalizers are broken when placement correctness by construction no longer holds, e.g., when more than two cells can interact and create DRVs. This challenges the use of dynamic programming frameworks, since decomposition into independent placement subproblems is no longer obvious. Finally, filler cell insertion has not previously been a concern of placement legalizers, but in N10 the filler cells can cause additional implant layer rule violations.

In this work, we propose a mixed integer-linear programming (MILP)-based placement legalization that considers complex N10 FEOL-layer design rules including minimum implant width, minimum oxide diffusion jogs and drain-drain abutment. We also propose a distributable optimization approach based on partitioning a given placement into many windows of cells, with each window being independently optimizable. The main contributions of our work are summarized as follows.

- We formulate as an MILP a placement problem that addresses new DRVs caused by complex N10 design rules. In contrast to previous approaches, our formulation captures new *inter-row* violation types. We further implement our solution approach in a prototype tool, **DFPlacer**.
- DFPlacer handles whitespace in the problem formulation and determines filler cell insertions to solve implant width constraint violations.
- We propose a distributable optimization based on partitioning of an input placement into windows of cells, and demonstrate that our optimization is scalable via this mechanism.
- We implement our proposed methods in C++ with OpenAccess 2.2.43 [15] and incorporate them into a commercial tool-based placement and routing (P&R) flow for evaluation.
- A further study provides insight into timing and area impacts of the dummy poly gate library cell strategy, using two kinds of libraries: (i) standard cells with dummy poly gates (drain-drain abutment violation free) and (ii) standard cells without dummy poly gates.

The remainder of this paper is organized as follows. In Section II, we review relevant prior work. To address N10 rules, we formulate an MILP in Section III-A and describe our distributable optimization strategy in Section III-B. Section IV provides experimental results and analysis. We give conclusions and future research directions in Section V.

II. Previous Work

We now summarize relevant previous works on detailed placement and placement legalization.

Dynamic programming-based approaches. Dynamic programming (DP), typically for a single cell row, has been used by a number of authors. Kahng et al. [5] use DP to legalize placement of a single row with various minimization objectives: total perturbation, maximum perturbation, and wirelength. A shortest-path algorithm is applied to a directed acyclic graph constructed from the input ordering of cells. Gupta et al. [1] perform detailed placement optimization to enable sub-resolution assist feature insertion for improved manufacturability. A DP-based single row placement achieves this *assist-feature correctness* (AFCorr) while minimizing (timing criticality-weighted) perturbations of cell locations. Subsequent work addresses a 2D formulation that considers both horizontal and vertical interactions between adjacent cells [2]. The *2D AFCorr* approach uses DP in which vertical and horizontal costs are calculated with restricted perturbations. Hur and Lillis [7] propose *optimal interleaving* for intra-row optimization in

detailed placement. Their work splits the cells of a single row into two groups with a given window size, and the two sequences are optimally interleaved via DP while preserving the initial relative ordering of cells in each group. At the global placement level, cells are assigned to bins and optimized via *relaxation-based local search*.

Integer Linear Programming (ILP)-based approaches. Another important class of previous methods is based on integer linear programming. Ramachandaran et al. [10] apply branch-and-price for improved scaling of the placement optimization. Li and Koh [8] propose ILP-based detailed placement approaches using *placement site variables*. Dantzig-Wolfe decomposition is applied to improve scalability, and single-cell-placement (SCP) variables enable grouping and mapping of placement site variables into patterns. The extension [9] supports mixed-size circuits and improves runtime by bounding solution spaces. In our present work, we begin with the MILP model of [8] [9], extending it to provide the first-ever comprehensive support (to our knowledge) of N10-relevant design rules such as minimum implant width, diffusion jogs and drain-drain abutment.

N10 design rules-aware placement. Du and Wong [6] address the abutment of source and drain in FinFET-based cell placement (i.e., for the foundry $14nm$ node onward), where the DDA constraint becomes prominent. The authors use cell flipping and adjacent-cell swapping as underlying operations for detailed placement perturbation that minimizes drain-drain abutments. As in [5], the authors of [6] apply a shortest-path algorithm with their proposed graph model, in which each operation and the violations are modeled as nodes and node/edge costs, respectively. However, the approach only swaps and flips cells within a single row, and does not handle interactions between placement rows. Hence, the optimization is made with respect to a highly restricted portion of the overall detailed placement solution space. Moreover, DDA-related optimization cannot be performed in isolation at the N10 node: many other neighborhood-related constraints (e.g., constraints for implant and oxide diffusion (OD) layers) have interactions with, and constrain, the drain-drain abutment solution. In our present work, we handle neighborhood-related constraints along with drain-drain abutment, with a larger solution space that includes multiple rows.

III. Our Approach

A. Problem Formulation

We now formulate a MILP for our detailed placement problem to address N10 related design rules including IW, DDA and OW in Section I-A. Our notation is described in Table I.

TABLE I
Notations.

Notation	Meaning
C, R, Q	sets of cells, rows, columns
f_c	a binary indicator of whether cell c is flipped
$x(y)_{c,init}$	initial x (y) coordinate of cell c
s_{crq}	a binary indicator of whether cell c occupies site (r, q)
K_c	a set of candidate states of cell c
λ_c^k	a binary indicator of the k^{th} candidate state for cell c
x_c^k, y_c^k	x and y coordinates corresponding to λ_c^k
f_c^k	f_c corresponding to λ_c^k
s_{crq}^k	s_{crq} corresponding to λ_c^k
m_{rq}	inter-row variable for IW1
h_{rq}	intra-row variable for IW2
W	minimum implant width (unit: site)

$$\textbf{Minimize:} \quad \sum_{c \in C} (|x_c - x_{c,init}| + |y_c - y_{c,init}|) \tag{1}$$

Subject to:

$$\sum_{k \in K_c} \lambda_c^k = 1, \qquad \forall c \in C, \quad \lambda_c^k \in \{0,1\} \tag{2}$$

$$f_c = \sum_{k \in K_c} f_c^k \lambda_c^k \tag{3}$$

$$x_c = \sum_{k \in K_c} x_c^k \lambda_c^k, \quad y_c = \sum_{k \in K_c} y_c^k \lambda_c^k, \quad \forall c \in C \tag{4}$$

$$s_{crq} = \sum_{k \in K_c} s_{crq}^k \lambda_c^k, \quad \forall c \in C \tag{5}$$

$$\sum_{c \in C} s_{crq} \leq 1, \qquad \forall q \in Q, r \in R \tag{6}$$

For a given input layout, our objective is to minimize the sum of cell displacements while achieving a legal placement with respect to given N10 design rules. We assume a given perturbation range for each cell g ($g.l$, $g.r$, $g.t$ and $g.b$ are respecting the maximum allowed displacements of the cell in the left, right, top and bottom directions, respectively); a cell cannot move beyond its given perturbation range. Thus, we have a limited number of possible states (locations and orientations) within g, for each cell. To represent each candidate state for a cell, we adopt the single-cell-placement (SCP) model of [9]. The binary SCP variable λ_c^k represents a candidate state k for a cell c. The variable λ_c^k is associated with the location and orientation of cell c, e.g., x_c^k, y_c^k, f_c^k, which are pre-defined values. Also, s_{crq}^k, where $r \in |R|$ and $q \in |Q|$, is pre-defined for λ_c^k.

From Constraint (2), exactly one state is chosen for cell c among multiple candidate states $\lambda_c^k, k \in K_c$, which determines the location and orientation of c. Constraints (3), (4) and (5) determine the final x, y of cell c and s_{crq} for $r \in |R|, q \in |Q|$ from a selected candidate site for cell c. To ensure a legal placement (no overlap), Constraint (6) forces a site at (r, q) to be occupied by at most one cell. In addition to the basic formulation, we add extra constraints to address OW, DDA, IW1 and IW2 rules, as follows.

OW and DDA constraints. To handle OW and DDA constraints, we pre-characterize all adjacency conditions which violate OW and/or DDA for each library cell pair. We note that our pre-characterization considers the orientations of cells (i.e., the adjacency conditions change depending on the orientations of cells). We then generate a set P of forbidden pairs of λ_c^k. Based on P, we formulate Constraint (7) for every forbidden pair $(\lambda_{c_1}^i, \lambda_{c_2}^j)$.

$$\lambda_{c_1}^i + \lambda_{c_2}^j \leq 1 \qquad \text{where } c_1, c_2 \in C, (\lambda_{c_1}^i, \lambda_{c_2}^j) \in P \tag{7}$$

IW1 and IW2 constraints. IW1 violations occur across rows when vertically-adjacent same-V_t layers form a narrow staircase shape with width less than the minimum implant width (see Figure 2(a)). To handle IW1, we define a 0-1 *inter-row variable,* m_{rq}, that indicates whether the site at (r, q) (row r and column q) and the site at ($r + 1$,q) have the same V_t ($m_{rq} = 1$) or not ($m_{rq} = 0$). Figure 4(a) illustrates the m_{rq} variables and IW1 constraints. As shown in the figure, if a 0-1 sequence of m values is found (e.g., m_{12}, m_{13}), the implant region has a staircase shape, and hence $(W - 1)$ consecutive m variables must be one. Thus, we formulate constraints that, if $m_{r(q-1)} = 0$ and $m_{rq} = 1$, force at least W consecutive inter-row variables $m_{rq} = \ldots = m_{r(q+W-1)} = 1$, so as to satisfy IW1 (e.g., $m_{13}, m_{14}, m_{15} = 1$ where $W = 3$, in Figure 4(a)).

IW2 violations occur when small-width cells are sandwiched in between different-V_t cells in the same row. Similar to how we

handle IW1, we define a 0-1 *intra-row variable*, h_{rq}, that indicates whether the site at (r, q) and the site at $(r, q+1)$ have the same V_t ($h_{rq} = 1$) or not ($h_{rq} = 0$), as shown in Figure 4(b). If $h_{rq} = 0$, i.e., sites (r, q) and $(r, q+1)$ have different V_t, we force $(W-1)$ consecutive binary variables $h_{r(q+1)} = \ldots = h_{r(q+W-1)} = 1$, so as to have at least W consecutive same-V_t sites. Figure 4(b) shows the case when $h_{rq} = 0$, where $r = 1$, $q = 2$, $W = 3$.

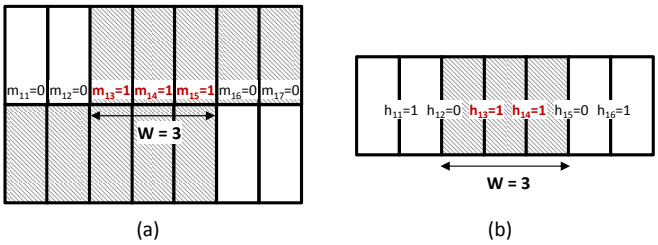

Fig. 4. (a) Inter-row variable m_{rq} for IW1. (b) Intra-row variable h_{rq} for IW2. The color (gray and white) of regions indicates V_t.

The generalized constraints for IW1 and IW2 are as follows:

$$m_{r0} = 0, h_{r0} = 0 \qquad 1 \le r < |R| \tag{8}$$

$$m_{rq} + (1 - m_{r(q+1)}) + y_{r(q+2)} \ge 1 \qquad 0 \le q < |Q| - W,\ 1 \le r < |R| \tag{9}$$

$$y_{rq} \le m_{r(q+w)} \qquad 2 \le q \le |Q| - 2,\ 1 \le r < |R|,\ 0 \le w < W - 1 \tag{10}$$

$$h_{rq} + z_{rq} \ge 1 \qquad 0 \le q < |Q| - W,\ 1 \le r < |R| \tag{11}$$

$$z_{rq} \le h_{r(q+1+w)} \qquad 2 \le q \le |Q| - 2,\ 1 \le r < |R|,\ 0 \le w < W - 1 \tag{12}$$

- Constraint (8) initializes the leftmost m and h variables where $q = 0$.
- Constraint (9) detects the condition of $m_{rq} = 0$ and $m_{r(q+1)} = 1$, and forces $y_{r(q+2)} = 1$.
- When $y_{r(q+2)} = 1$, Constraint (10) forces $(W-1)$ consecutive binary variables $m_{r(q+2)} = \ldots = m_{r(q+2-(W-1))} = 1$.
- Constraint (11) detects the condition of $h_{rq} = 0$ and forces $z_{rq} = 1$.
- When $z_{rq} = 1$, Constraint (12) forces $(W-1)$ consecutive binary variables $h_{r(q+1)} = \ldots = h_{r(q+(W-1))} = 1$.

We now describe our method of obtaining inter- and intra-row variables (m_{rq} and h_{rq}). We first set the V_t of cell c as the binary vector $\vec{k}_c$. The length of $\vec{k}_c$ is determined by $\lceil log_2(n_{V_t} + 1) \rceil$ where n_{V_t} is the number of available V_t options. For example, if we have three V_t options, then $\vec{k}_c$ is $\{k_c^1\ k_c^2\}$. Concretely, the binary vectors $\{0\ 1\}$, $\{1\ 0\}$ and $\{1\ 1\}$ represent HVT, NVT and LVT, respectively. We then define the V_t *variable* $\vec{v}_{rq}$ as a binary vector variable $\{v_{rq}^1\ v_{rq}^2\}$ indicating the V_t of the site (r, q). Given that $m_{rq} = 1$ if $\vec{v}_{rq} = \vec{v}_{(r+1)q}$, we add the following constraint to obtain m_{rq}:

$$m_{rq} = \overline{(v_{rq}^1 \oplus v_{(r+1)q}^1) + (v_{rq}^2 \oplus v_{(r+1)q}^2)} \tag{13}$$

Constraint (13) is rewritten in our MILP formulation, using binary variables u_1, u_2 and $\overline{m_{rq}}$, as follows:

$$\begin{aligned} &\overline{m_{rq}} + m_{rq} \le 1; \\ &\overline{m_{rq}} \le u_1 + u_2;\ \ \overline{m_{rq}} \ge u_1;\ \ \overline{m_{rq}} \ge u_2; \\ &u_1 \le v_{rq}^1 + v_{(r+1)q}^1;\ \ u_1 \ge v_{rq}^1 - v_{(r+1)q}^1; \\ &u_1 \ge v_{(r+1)q}^1 - v_{rq}^1;\ \ u_1 \le 2 - v_{rq}^1 - v_{(r+1)q}^1 \\ &u_2 \le v_{rq}^2 + v_{(r+1)q}^2;\ \ u_2 \ge v_{rq}^2 - v_{(r+1)q}^2; \\ &u_2 \ge v_{(r+1)q}^2 - v_{rq}^2;\ \ u_2 \le 2 - v_{rq}^2 - v_{(r+1)q}^2 \end{aligned} \tag{14}$$

Similarly, h_{rq} can be formulated as follows:

$$h_{rq} = \overline{(v_{rq}^1 \oplus v_{r(q+1)}^1) + (v_{rq}^2 \oplus v_{r(q+1)}^2)} \tag{15}$$

We also consider whitespace (empty sites), which can be filled with filler cells. We have the freedom to choose V_t of filler cells to satisfy IW1 and IW2 constraints. To exploit this flexibility, we define a binary vector variable $\vec{e}_{rq} = \{e_{rq}^1 e_{rq}^2\}$ which indicates V_t of the site at (r, q). From variables $\vec{k}_c$, s_{crq} and $\vec{e}_{rq}$, v_{rq} is defined as follows:

$$\vec{v}_{rq} = \sum_{c \in C} \vec{k}_c \cdot s_{crq} + \vec{e}_{rq} \tag{16}$$

Thus, $\vec{v}_{rq}$ is determined by either $\sum_{c \in C} \vec{k}_c \cdot s_{crq}$ or $\vec{e}_{rq}$. We add a constraint below for $\vec{e}_{rq}$:

$$e_{rq}^1 \le 1 - \sum_{c \in C} s_{crq};\quad e_{rq}^2 \le 1 - \sum_{c \in C} s_{crq} \tag{17}$$

Constraint (17) states that if a site is occupied by any cell, $\vec{e}_{rq} = 0$. Then, Constraint (16) becomes independent of $\vec{e}_{rq}$. Otherwise, Constraint (16) becomes $\vec{v}_{rq} = \vec{e}_{rq}$.

Analysis of the number of variables and constraints. The number of variables and constraints depends on the number of sites in a target window ($|R| \cdot |Q|$), the number of instances ($|C|$) and the size of the perturbation range ($g.size = (g.l + g.r) \cdot (g.t + g.b)$).

- The number of variables s_{crq} is $|C| \cdot |R| \cdot |Q|$.
- The number of variables x_c, y_c is (each) $|C|$; the number of variables $x_c^k, y_c^k, \lambda_c^k$ is (each) $g.size \cdot |C|$.
- The number of inter-/intra-row variables m, h is (each) $|R| \cdot |Q|$.
- The number of variables v and e is (each) $n \cdot |R| \cdot |Q|$, where n is $\lceil log_2(n_{V_t} + 1) \rceil$.
- The numbers of Constraints (2), (4), (5) and (6) are $|C|$, $|C|$, $|C| \cdot |R| \cdot |Q|$ and $|R| \cdot |Q|$, respectively.
- The number of Constraint (7) is $g.size \cdot |C|^2$.
- The number of Constraints (9), (10), (11) and (12) is (each) $|R| \cdot |Q|$.

B. Overall Flow

We implement our flow in C++ with *OpenAccess 2.2.43* [15] to support LEF/DEF [14], and with *CPLEX 12.5.1* [13] as our MILP solver. Figure 5 shows the overall flow of our tool, which we call DFPlacer. DFPlacer has two optimization stages: global and local optimization. In the global optimization, we split the given routed layout T uniformly into a set of windows D and optimize each of the windows $d \in D$ in parallel. We use a fixed boundary margin b for each window to enable independent optimization among windows. In the local optimization, we generate a new window for each remaining violation $\gamma \in \Gamma$ such that the violation is located at the center of the window. We then remove overlapping windows so that no window affects another. With the new set of windows D', we optimize each window $d' \in D'$ again in parallel. The output cell location solution is saved in DEF file format, which can be fed into

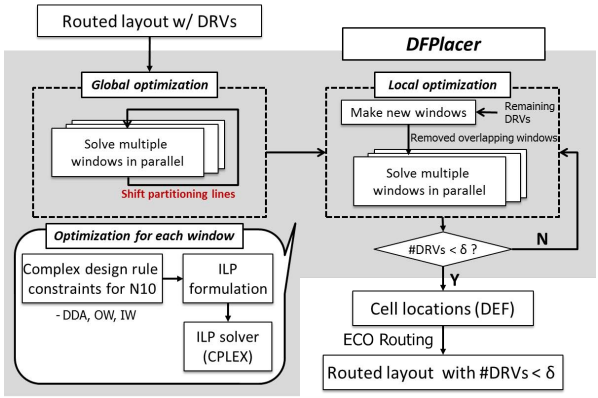

Fig. 5. Overall flow of detailed placement legalization.

a commercial P&R tool. We perform ECO routing with the solution and finally obtain a new layout T_{opt} with number of violations $|\Gamma|$ less than the given target number δ.

Algorithm 1 Overall flow of DFPlacer.

Procedure $DFPlacer(T, U, z, b, g, \delta)$
Input : Layout T, set of design rule constraints U, window size z, boundary margin b, perturbation range g, target number of DRVs δ
Output : Layout T_{opt} with $|\Gamma| < \delta$

1: // Global optimization
2: **for** $i = 1$ to 3 **do**
3: A set of windows $D \leftarrow Partition(T, i, z, b, g)$;
4: Solve all MILP instances for windows D in parallel;
5: Update MILP solutions to T;
6: **end for**
7: // Local optimization
8: $\Gamma \leftarrow getDRV(T, U)$;
9: **while** $|\Gamma| < \delta$ **do**
10: $D \leftarrow \emptyset$;
11: **for all** $\gamma \in \Gamma$ **do**
12: $d \leftarrow MakeNewWindows(T, \gamma, z, b, g)$;
13: $D \leftarrow D \cup d$;
14: **end for**
15: $D' \leftarrow NonOverlapWindows(D)$
16: Solve all MILP instances for windows D' in parallel;
17: Update MILP solutions to T;
18: $\Gamma \leftarrow getDRV(T, U)$;
19: **if** $|\Gamma|$ is the same as $|\Gamma|$ in the previous iteration **then**
20: $IncreaseWindow(z)$;
21: $IncreasePerturb(g)$;
22: **end if**
23: **end while**
24: $T_{opt} \leftarrow T$;
25: return T_{opt};

Algorithm 1 gives further details of our optimization flow. In Lines 2-6, the global optimization phase solves D in parallel with a small perturbation range g (e.g., $g.l = 4$, $g.r = 4$, $g.t = 1$ and $g.b = 1$ sites) of cells. This distributable method overcomes the runtime limitation of MILP-based approaches and fixes more than 90% of initial $|\Gamma|$ (see Figure 7). In Line 3, we first partition a given routed T into D, and we solve each $d \in D$ in parallel using OpenMP [17] in Line 4. We set a window width $z.w$ as 47 sites, and a window height $z.h$ as nine cell rows in our experiments. [2] When running optimizations for the windows in parallel, we set the vertical (resp. horizontal) boundary margin $b.v$ (resp. $b.h$) so that the solution of one window can be isolated from the solutions of neighbor windows. We set $b.v$ as the minimum implant width W and $b.h$ as two cell row heights. Figure 6 shows the boundary margin in green color. We then update the MILP solutions to the layout T.

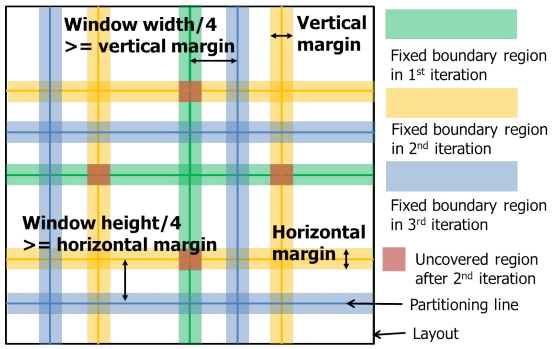

Fig. 6. Partitioning of layout for parallel global optimization.

Since the fixed boundary cells corresponding to b of the first iteration can contain DRVs which are not fixed in the first iteration, we perform a second iteration with a new partitioning that is shifted by half of $z.w$ and $z.h$ in the x- and y-directions, respectively; these are shown in yellow color in Figure 6. We then partition the current T into a new D and solve the corresponding MILP instances to fix the violations Γ remaining from the first iteration. We then update the MILP solutions to T. Even after the first and second iterations, DRVs could still exist in the intersection of the fixed boundary region (red color in Figure 6). To fix the Γ in the uncovered intersection region, we perform a third iteration that has new partitioning lines shifted by a quarter of $z.w$ and $z.h$ in x-direction and y-direction. Note that a quarter of $z.w$ and of $z.h$ should respectively be larger than or equal to $b.v$ and $b.h$. This ensures that the windows of the third iteration contain the uncovered regions, such that the fixed boundary region of the third iteration is not overlapped with the uncovered intersection region.

The small window size and perturbation range used in global optimization restricts the solution space, potentially leading to infeasible solutions for certain windows. To fix the remaining Γ, we perform the local optimization in Lines 8-24. For each DRV, the function $MakeNewWindows()$ creates a new window whose center is the DRV point. In Line 15, $NonOverlapWindows()$ picks a set of disjoint windows D' to process in parallel. We then update the solutions to the current T and check the remaining Γ (Lines 16-17). In Lines 19-22, if the current $|\Gamma|$ is the same as the $|\Gamma|$ of the previous iteration, we increase $z.w$ by 10 sites and $z.h$ by one cell height. For perturbation range, we increase $g.l$, $g.r$, $g.t$ and $g.b$ by 2, 2, 1 and 1 sites, respectively. When the current $|\Gamma|$ is less than the target number of DRVs δ, we save the current T as T_{opt} and terminate the optimization. In our experiment, we set δ as 1% of initial $|\Gamma|$.

[2]Window size affects the tradeoff between the number of remaining violations $|\Gamma|$ after global optimization and the runtime of global optimization. Our studies of different window sizes (i.e., $z.w$ ranging from 40 sites to 55 sites and $z.h$ ranging from five cell row heights to 11 cell row heights) find that for a sample design (jpeg) a width of 47 sites and a height of nine cell row heights empirically achieves a good outcome ($< 10\%$ of initial $|\Gamma|$) with relatively small runtime (< 30 minutes). We therefore use this window size in all of our reported experiments.

IV. Experimental Setup and Results

A. Experimental Setup

We evaluate *DFPlacer* using two open-source designs (aes_*, jpeg_*) [16], an ARM Cortex M0 without memories (m0_*) and a 3×ARM Cortex M0 without memories (m0x3_*). We synthesize these testcases from RTL, and perform P&R with an abstracted $7nm$ dual V_t library. Our RTL-to-layout flow uses *Synopsys Design Compiler H-2013.03-SP3* [18] and *Cadence Encounter Digital Implementation System XL 13.1* [12] for logic synthesis and P&R, respectively. All experiments are performed with 40 threads on a $2.6GHz$ Intel Xeon E5-2690 dual-CPU server. In principle, the number of threads could be as large as the number of layout windows.

TABLE II
Testcases used in the experiments.

Design	#Inst	LVT (%)	Util. (%)	WL (μm)	Area (μm^2)	WSS (ps)	WHS (ps)
m0_nd	8260	52	77	114685	7668	38	0
aes_nd	12147	54	78	142294	8894	90	0
m0x3_nd	27248	56	80	392540	24463	126	0
jpeg_nd	47948	51	77	694624	49629	12	0
m0_d	8238	51	77	116866	8668	93	1
aes_d	12491	54	80	150632	10596	58	0
m0x3_d	26690	55	79	409579	27400	107	0
jpeg_d	48317	52	77	764738	55824	13	0

Libraries and design rules. We use a prototype $7nm$ standard-cell library from a leading IP provider. Since our design enablement for the $7nm$ technology is missing detailed BEOL technology information such as RC values and BEOL stack options, we scale the library to use a $28nm$ BEOL stack, following the methodology described in [3]. The site width and height are $0.136\mu m$ and $0.9\mu m$; these values correspond to of the $28nm$ BEOL information. For design rules, we set the OW, IW1 and IW2 rules as four site widths. To check for DDA and OW violations, we pre-characterize all pairs of standard cells in the $7nm$ library. The library has 62 standard cells and the total number of pairs is 15376 (= 62×62×2×2), including cell flipping. For the standard cells without dummy poly gate, 7172 pairs out of the 15376 pairs violate the DDA constraint, and these pairs require at least one site space. Similarly, with the 4 site widths for OW, 280 out of the 15376 pairs violate the OW constraint, and such pairs also require one site space.[3]

Tradeoff between area/wirelength and DRVs. Table II shows the testcases used in our experiments. LVT, WSS, WHS and WL respectively indicate the portion of LVT cells, the worst hold and setup slacks, and wirelength. We assign V_t to cells uniformly to create more IW1 and IW2 violations. We use two kinds of libraries: (i) without dummy poly gates (CWOD) and (ii) with dummy poly gates (CWD). CWD is designed with dummy poly gates inserted to avoid interactions between cells which create DDA violations. For the CWD library, cell width is increased by one poly pitch compared to the CWOD library. The suffixes *_*d* and *_*nd* indicate that the designs are implemented with CWD and CWOD libraries, respectively. The same initial netlists are used for both *_*d* and *_*nd*. While comparing *_*d* and *_*nd* designs, we observe that the average wirelength and area overhead of designs implemented using libraries with dummy poly gates are 7% and 14%, respectively. In terms of DRVs, *_*nd* testcases have 134%∼176% more DRVs as reported in the fourth column of Table III.

[3]Based on our OW rule and library, all pairs of standard cells that violate OW constraints require only one site space. However, depending on the OW rule and library, some pairs of standard cells could require two or more site spaces.

B. Experimental Results

Table III summarizes the number of DRVs, the worst setup slack, worst hold slack, Δwirelength, maximum Δlocation, average Δlocation, the number of moved cells and runtime. Our DFPlacer fixes more than 99% of initial violations in runtime that is reasonable for practical contexts. From a timing perspective, ΔWSS (i.e., final WSS - init WSS) ranges from $-19ps$ to $68ps$, but all final designs have no negative WSS. Similar to WSS, ΔWHS ranges from $-2ps$ to $0ps$. The timing impact is small since most of the cells are moved within a given small perturbation range. Some cells can be moved more than 20 sites (i.e., $0.136 \times 20 = 2.72\mu m$) from their initial locations due to the accumulated displacement in the iterative local optimization. However, those cells are less likely to be in the most critical path, which is how the WSS or WHS would worsen. Also, the positive ΔWSS implies that there is room to improve timing, and that we could potentially co-optimize the timing along with DRV fixing in detailed placement legalization. This is a direction of ongoing work.

On the other hand, DFPlacer increases wirelength up to 3%. The accumulated displacements of cells and the limited pin access for the standard cells in N10 could be causes of this wirelength increase. Between *_nd and *_d testcases, the ΔWL% of *_nd cases is similar or slightly larger. We believe that this is because IW violations are harder to fix compared to the OW and DDA violations, since the constraints are more complex. The rate at which the number of IW violations reduces is slower than that for OW and DDA violations. Also, since the CWD library cells are larger than the CWOD library cells, the displacement of cells in *_d cases might have more impact on the wirelength increase. Therefore, % WL increase of *_d cases is smaller in general, but not necessarily always less than that of *_nd cases.

Columns *Max.* Δ*loc.* and *Avg.* Δ*loc.* show maximum and average cell displacement, respectively. We observe that the average cell displacement for all designs is up to $0.70\mu m$, which is ∼5 sites' width. The maximum displacement is up to $8.99\mu m$ for $jpeg\_nd$. For other designs, the maximum displacement is similar to the half-perimeter of the perturbation range used in the global optimization ($2.888 = 0.9 \cdot 2 + 0.136 \cdot 8$ microns).

When we compare the results of designs with CWOD and CWD, we see a tradeoff between area and the number of DRVs (and runtime). We observe that the area overhead of using cells with dummy poly gate is 14% on average (up to 19%). However, the number of DRVs decreases by 61% on average (up to 64%). This affects the runtime of detailed placement legalization.

Figure 7 shows the remaining number of DRVs (%) versus runtime (sec). Each dot stands for an iteration of the optimization and the third iteration points are marked with diamond-shaped markers. During the global optimization, which includes first, second and third iterations, the remaining violations drop quickly; ∼90% of DRVs are fixed in most of the designs during the global optimization. The runtime of the global optimization phase still increases with the problem size. However, with added computing resources to run windows of cells in parallel, the runtime can be further reduced. After the third iteration, when entering into local optimizations, the rate of decrease of the number of DRVs becomes much lower, implying that DFPlacer spends considerable time to fix the last few DRVs. This is because these last DRVs cannot be solved with small window sizes and perturbation ranges in the global optimization; thus, DFPlacer tries to resolve them in the local optimization by increasing window sizes and perturbation ranges. The poor scaling of MILP solution versus instance size leads to the observed run times.

Figures 8(a) and 8(b) respectively show layout snapshots from the

TABLE III

RESULTS WITH #VIOLATIONS, WORST SETUP SLACK, WORST HOLD SLACK, ΔWIRELENGTH, MAXIMUM ΔCELL LOCATION, AVERAGE ΔCELL LOCATION, #CHANGED CELLS AND RUNTIME.

Design	IW #Vio.		DDA/OW #Vio.		WSS (ps)		WHS (ps)		ΔWL (%)	Max. Δloc. (μm)	Avg. Δloc. (μm)	#Changed cells (%)	CPU total (sec)	
	Init	Final	Init	Final	Init	Final	Init	Final					Global	Total
m0_nd	926	11	1611	14	38	83	0	0	2.79	2.89	0.56	4489 (54%)	768	2820
aes_nd	1771	16	1900	18	90	71	0	-1	3.42	2.89	0.52	5939 (49%)	787	2992
m0x3_nd	3514	17	4230	48	126	113	0	0	2.90	3.02	0.51	12752 (47%)	957	6897
jpeg_nd	4056	29	12024	135	12	22	0	0	2.30	8.99	0.70	24169 (50%)	1788	11983
m0_d	988	10	0	0	93	85	1	0	3.04	2.89	0.57	2996 (36%)	161	434
aes_d	1566	11	0	0	58	80	0	0	3.10	2.89	0.54	3852 (31%)	425	1207
m0x3_d	2810	27	0	0	107	105	0	-2	2.14	2.89	0.58	9340 (35%)	517	1336
jpeg_d	6296	43	0	0	13	81	0	0	-0.57	3.02	0.49	12244 (27%)	954	1401

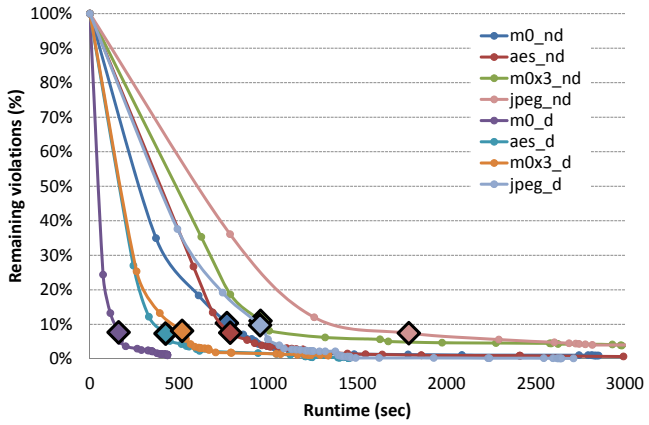

Fig. 7. Remaining violations vs. runtime. Each dot indicates an iteration; after the third iteration, local optimization is performed. The diamond-shaped markers represent third-iteration points.

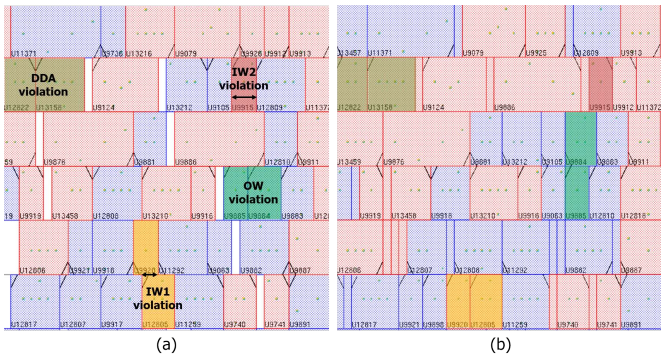

Fig. 8. (a) Layout with DRVs before optimization. (b) Layout without DRVs after optimization.

pre- and post-detailed placement legalization phases. In Figure 8(a), we highlight the cells that violate OW (green color), DDA (light green color), IW1 (yellow color) and IW2 (brown color) rules. Figure 8(b) shows the displacement of corresponding cells in post-detailed placement legalization. We observe that our DFPlacer fixes the DDA violation by flipping one of the violating cells; the IW1, IW2 and OW violations are all resolved by moving the violating cells or their neighbor cells within and/or across rows.

V. CONCLUSIONS

In this work, we have proposed a scalable detailed placement legalization flow for complex FEOL constraints arising at the N10 foundry node. These include drain-drain abutment, minimum implant width, and minimum OD jogging rules. Given initial (timing-driven) placements, our $DFPlacer$ fixes 99% of DRVs with 3% increase in wirelength and minimal impact on timing. We feel that our use case of fixing all but a few tens of violations, with a highly parallelizable two-iteration strategy, is a good practical tradeoff between runtime complexity and DRV fixing. Further, the level of DRV fixing achieved by DFPlacer is encouraging, given that our default experimental configuration makes no attempt at "correctness by construction". Using OpenMP, we confirm that our flow is scalable via a distributed optimization strategy. Additionally, we study an area-DRV tradeoff between two types of standard-cell library strategies, namely, with and without dummy poly gates.

Our future work includes (i) timing and wirelength-driven placement legalization, which we believe can be enabled by more compact optimization formulations along with a more restricted perturbation range for each cell; (ii) a "smart ECO" method for the few DRVs that remain after global placement legalization; and (iii) further investigation of the scalability of our partitioning-based distributed optimization approach. Finally, we believe that our present placement-centered work may converge with such recent routing-centered works as [3], leading eventually to an "optimal detailed P&R" that can shield physical design teams from impacts of increasing ground rule complexity at N10 and beyond.

REFERENCES

[1] P. Gupta, A. B. Kahng and C.-H. Park, "Detailed Placement for Improved Depth of Focus and CD Control", *Proc. ASPDAC*, 2005, pp. 343-348.

[2] P. Gupta, A. B. Kahng and C.-H. Park, "Manufacturing-Aware Design Methodology for Assist Feature Correctness", *Proc. SPIE*, 2005, pp. 131-140.

[3] K. Han, A. B. Kahng and H. Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router", *Proc. DAC*, 2015.

[4] A. B. Kahng and H. Lee, "Minimum Implant Area-Aware Gate Sizing and Placement", *Proc. GLSVLSI*, 2014, pp. 57-62.

[5] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *Proc. GLSVLSI*, 2004, pp. 214-219.

[6] Y. Du and M. D. F. Wong, "Optimization of Standard Cell Based Detailed Placement for 16nm FinFET Process", *Proc. DATE*, 2014, pp. 1-6.

[7] S.-W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement", *Proc. ICCAD*, 2000, pp. 165-170.

[8] S. Li and C.-K. Koh, "Mixed Integer Programming Models for Detailed Placement", *Proc. ISPD*, 2012, pp. 87-94.

[9] S. Li and C.-K. Koh, "MIP-based Detailed Placer for Mixed-size Circuits", *Proc. ISPD*, 2014, pp. 11-18.

[10] P. Ramachandaran, A. R. Agnihotri, S. Ono, P. Damodaran, K. Srihari and P. H. Madden, "Optimal Placement by Branch-and-Price", *Proc. ASPDAC*, 2005, pp. 337-342.

[11] R. Aitken, *personal communication*, March 2015.

[12] Cadence SOC Encounter User Guide, http://www.cadence.com

[13] IBM ILOG CPLEX. www.ilog.com/products/cplex/

[14] LEF DEF reference 5.7. http://www.si2.org/openeda.si2.org/projects/lefdef

[15] Si2 OpenAccess. http://www.si2.org/?page=69

[16] OpenCores: Open Source IP-Cores, http://www.opencores.org

[17] OpenMP Architecture Review Board, "OpenMP Application Program Interface, Version 3.1".

[18] Synopsys Design Compiler User Guide, http://www.synopsys.com