Minimum Implant Area-Aware Gate Sizing and Placement

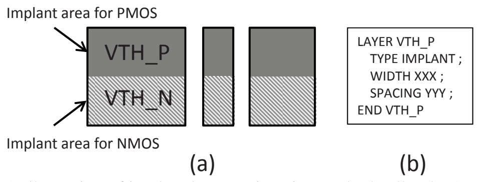
Andrew B. Kahng[†‡] and Hyein Lee[†]
[†]ECE and [‡]CSE Departments, University of California at San Diego
abk@ucsd.edu, hyeinlee@ucsd.edu

ABSTRACT

With reduction of minimum feature size, the *minimum implant area* (MinIA) constraint is emerging as a new challenge for the physical implementation flow in sub-22nm technology. In particular, the MinIA constraint induces a new problem formulation wherein gate sizing and V_t-swapping must now be linked closely with detailed placement changes. To solve this new problem, we propose heuristic methods that fix MinIA violations and reduce power with gate sizing while minimizing placement perturbation to avoid creating extra timing violations. Compared to recent versions of commercial P&R tools, our methodologies achieve significant reductions (up to 100%) in the number of MinIA violations under timing/power constraints.

1. INTRODUCTION

As minimum feature sizes decrease, physical design rules have become tighter. Geometric constraints for layout that arise from limits of patterning technology are described as design rules in technology files such as LEF [17]. Each layer has width, spacing, minimum-area, etc. rules which can affect the legality of standard-cell placement. Implant (active) layers, which indicate regions for ion implantation, determine the threshold voltage (V_t) of transistors. Figure 1 shows the implant region in standard cells and an example *minimum implant area* (MinIA) layer rule in the LEF file. As shown in the figure, the polygons of implant layers have drawn dimensions typically matched to the width of standard cells.



```
LAYER VTH_P
  TYPE IMPLANT ;
  WIDTH XXX ;
  SPACING YYY ;
END VTH_P
```

Figure 1: (a) Illustration of implant layer regions in standard cells. (b) An example of an implant layer rule in a LEF5.7 file.

MinIA layer rules affect multi-V_t standard cell-based designs; the multi-V_t regime is essential to achieve low-leakage, high-performance design implementations [4]. Traditional timing- and routability-driven placement of cells with multiple V_t values can result in a small island of a given V_t that cannot meet the MinIA layer rule. A small cell that cannot meet the MinIA rule must be abutted with cells having the same V_t, so as to form a wider implant layer polygon. That is, a narrow cell cannot be sandwiched between different-V_t cells. Figure 2 shows an example of a MinIA violation. The dotted line indicates the minimum width of the V_{t_2} implant layer. A narrow cell c_2 with V_{t_2} is surrounded by V_{t_1} cells, and this violates the MinIA (equivalently, width) constraint. We note that the impact of the MinIA rule can be huge when the proportion of small-width cells in the netlist is large; this is a common scenario especially in cost-driven, low-power mobile IC products. We have studied how "thin" a netlist can be. For example, in a *jpeg* netlist synthesized from OpenCores [19] using a 28nm FDSOI foundry library, the smallest and second-smallest cell widths comprise ∼18% and ∼23% of the total number of instances, respectively.

The (minimum width and spacing) design rules for implant layers have not been critical before, as cell sizes have been large enough to cover these minimum rules. Hence, up until recent technology generations, placement and gate sizing/V_t-swapping methods have not had to consider these rules, since any legal cell placement would be correct with respect to the MinIA criterion. However, as cell sizes have continued to shrink in advanced process nodes, even as the wavelength used in 193*i* optical lithography remains constant, the MinIA rule has become larger than the minimum width of standard cells (e.g., INV×1 cell). MinIA rules constrain cell placement starting with the foundry "20nm" (64nm minimum metal pitch) node, due to the minimum pattern size limits and cell library (diffusion layer layout) strategies.

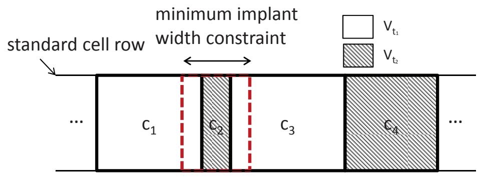


Figure 2: An example of the minimum implant area violation. The dotted line indicates the minimum width constraint of the implant layer. The cell instance c_2 (V_{t_2}) violates the constraint as it is narrow and sandwiched by two cells (c_1 and c_3) that have a different V_t (V_{t_1}).

The minimum width constraint of implant layers changes the traditional placement and post-layout gate sizing problems. That is, beyond existing constraints such as timing, power and area, additional *geometric* information of cells must be considered. A major change to the traditional sizing problem formulation is that even downsizing and V_t-swapping operations must comprehend spatially adjacent cell instances and whitespace in order to avoid creation of sandwiched narrow cells. Further, placement algorithms must comprehend V_t assignment of spatially adjacent cell instances, as well as whitespace, for the same reason. Therefore, the two problems cannot be solved independently. A unified method that considers both cell size/V_t and placement together is needed.

Case study of P&R tools. Recently, commercial P&R tools have claimed that minimum implant area rules are considered during the implementation, given that filler cells with active layers can be used to improve active density and manage STI stress effects [15]. Commercial P&R tools apparently fix the minimum implant area violations by inserting filler cells at the final design stage. For example, Mentor Graphics *Olympus* [18] has a utility to define an implant layer group for filler cells, so that each narrow cell can be padded filler cells having the same implant type. Cadence *Encounter Digital Implementation System (EDI)* [16] checks and fixes implant area violations according to the rules specified in LEF, during placement and filler cell insertion. Synopsys *IC Compiler (ICC)* [21] offers a *Voltage-threshold-aware filler cell insertion* flow according to which users can define the V_t filler cells to be inserted between different V_t regions. For example, users can

GLSVLSI'14, May 21–23, 2014, Houston, Texas, USA.
Copyright 2014 ACM 978-1-4503-2816-6/14/05 ...$15.00..

insert NVT filler cells between NVT and HVT cells, and LVT filler cells between LVT and NVT cells. In our case studies of P&R tools, we assign HVT filler cells to HVT-NVT and HVT-LVT pairs, and NVT filler cells to NVT-LVT pairs. We have tested commercial P&R tools using two small netlists (*dma* and *aes*) synthesized from OpenCores [19] RTL, and a 45nm foundry library along with intentionally tightened MinIA constraints. The minimum implant layer rules are defined in LEF. The portion of each V_t type in the testcases is evenly distributed so as to heighten the number of MinIA rule violations. The utilizations in the *dma* and *aes* implementations are 77% and 82%, respectively. Figure 3(c) shows example commands used with current P&R tools. After routing, the tools' built-in filler cell insertion flows are applied. The results shown in Figure 3 (a) and (b) show that commercial P&R tools cannot fix all of the MinIA violations by simply inserting filler cells.

This work. We have discussed how MinIA rules change the gate sizing and placement problems, and we have observed that commercial P&R tools can handle this new issue only to a limited extent. To our knowledge, no work in the research literature has tried to solve MinIA violations systematically, for both gate sizing and placement. The main contributions of our work are summarized as follows.

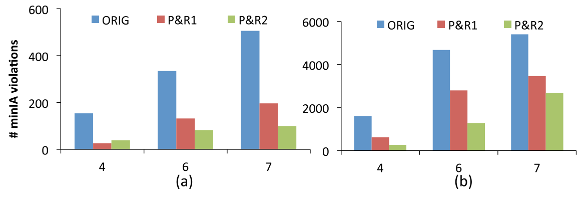
- We redefine traditional placement and gate sizing problems in view of the minimum implant layer constraint.
- We propose methods to minimize MinIA violations and optimize power under the MinIA constraint with placement perturbation and gate sizing/V_t-swapping.
- Our proposed methods are implemented with C++ programs and incorporated into a standard P&R flow. Our methods are validated with a commercial tool and 45nm foundry library with a range of MinIA constraints.

The remainder of this paper is organized as follows. In Section 2, we review relevant prior work. Section 3 describes various problem formulations that consider both gate sizing and placement under minimum implant layer constraints. To address these problems, we introduce a heuristic approach for sequential sizing and placement considering minimum implant layer rules. The overall flow of our optimization is described in Section 4. Section 5 provides experimental results and analysis. We give conclusions and ongoing research directions in Section 6.

2. PREVIOUS WORK

Gate sizing, and co-optimization with placement. Traditional gate sizing, which optimizes power and delay of circuits using size (gate width), V_t, and gate length of cells, has been extensively reviewed in [12]. Co-optimization of placement and gate sizing has had limited previous consideration. Lee and Gupta [11] suggest LP-based gate sizing considering an ECO cost, which is computed with respect to power and area and modeled as a linear function of several parameters. The objective is to minimize "power + ECO cost". Luo et al. [14] minimize power by combining placement, sizing and V_t-swapping optimizations using a slack management technique. The placement, sizing and V_t-swapping phases are performed sequentially.

Linear (1-D) placement. Since MinIA violations do not occur between standard cell rows, the associated problem of obtaining a MinIA-legal placement can be treated as a type of linear (1-D) placement. In the related literature, Kahng et al. [10] and Brenner et al. [1] consider the problem of placing a set of cells in a single row with a fixed horizontal ordering, with the objective of minimizing the (weighted) sum of net bounding box perimeters. The paper [8] evaluates several techniques to legalize cell overlaps in row-based placements, so as to improve metrics of routability and routed wirelength. In [3], Gupta et al. describe a dynamic programming (DP) based technique for Assist-Feature Correctness (AFCorr) in



P&R tool script examples	
verifyGeometry addFiller {cell_list}	set_cell_vt_type set_vt_filler_rule "VT1 VT2" {cell_list} insert_stdcell_filler

(c)

Figure 3: Minimum implant area violation fix results from two commercial tools (P&R1 and P&R2) through filler cell insertion, for two testcases: (a) *dma* and (b) *aes*. The x-axis shows the minimum implant width constraints in the number of grids. (c) The commands used for filler cell insertion flow.

detailed placement of standard-cell designs. Their implementation achieves improved depth of focus and CD control, and subsequent works from the same group use similar DP methods to address leakage and other objectives.

Layout effect-aware placement. STI stress-aware placement has tangential similarity to our MinIA placement, in the sense that solutions can seek to increase implant area; however, the main objective is different. Joshi et al. [5] propose stress as a means to achieve an optimal power-performance tradeoff. They study how stress-induced performance enhancements are affected by layout properties, and they improve standard-cell layouts to achieve maximum performance gain with dual-V_t assignment. Kahng et al. [9] combine detailed placement and active-layer fill insertion to exploit STI stress for performance improvement. Chakraborty et al. [2] propose an active area sizing aware cell-level delay model which forms the basis of linear programming to achieve either maximum performance, or target performance under a resource budget. Li et al. [13] present a methodology to determine optimal STI well width; using geometric programming, they also solve a STI stress aware placement optimization formulation.

3. PROBLEM FORMULATION

Given a placed standard-cell design[1] with potential MinIA rule violations, our ultimate goal is to find location and sizing/V_t assignment for each cell, so as to achieve minimum power without any violation of design constraints – including timing, placement legality, and MinIA rule constraints. Implicitly, our problem formulation assumes the following. (1) We do not consider violations that occur inside of a cell; individual cells are assumed correct by construction. (2) We assume that the minimum width of an implant region inside a cell is always the same as the cell width. (3) We do not consider the height of implant regions within a cell; again, correctness by construction is assumed. To optimize power considering both timing and geometry information, we combine gate sizing and ECO placement to address MinIA-aware gate sizing and placement problem:

Problem: ***MinIA*-aware sizing and placement**

Minimize: $\sum_{\forall c} P(c)$

Subject to:

$$S(c) > 0 \quad \text{(T1)}$$

$$T_r(c) < T_{r_{max}} \quad \text{(T2)}$$

$$W_e(c) < I_{min}(V_t(c)), \ \ \forall c \quad \text{(P1)}$$

No overlap in placement (P2)

Here, (T1) and (T2) are the timing-related constraints: (T1) is the setup (max path delay) slack constraint, and (T2) is the maximum transition time constraint. (P1) and (P2) are the placement-related

[1]Possibly, the design is routed as well, depending on preferences regarding parasitic estimation accuracy versus turnaround time.

constraints: (P1) is the minimum implant area constraint, and (P2) is the placement legality (non-overlapping) constraint. Additional constraints such as max capacitance, hold time checks, and various other physical design rules can be transparently considered within the approaches that we propose. We omit discussion of these for simplicity of exposition.

Table 1: Notations used in this work. Note that $W_e(c)$ is the maximum width of any contiguous region of an implant layer in c. Also note that $B(c)$ includes minimum implant area violations.

Notation	Meaning
$P(c)$	leakage power of cell c
$V_t(c)$	threshold voltage of cell c
$S(c)$	timing slack of cell c
$T_r(c)$	transition time of cell c
T_{rmax}	max transition time constraint
$W_e(c)$	effective implant layer width of cell c
$B(c)$	violations caused by cell c in placement
$\{N(c)\}$	neighbor cells of cell c
$I_{min}(V_t)$	minimum implant area constraint for V_t
m_k	a potential sizing solution

When we optimize both sizing and ECO placement via any sequential/iterative procedure, it is clear that gate sizing will minimize power at the cost of potential violations (in particular, with respect to MinIA placement rules), and that these violations must be fixed by an ECO placement step. Further, the sizing and placement problems naturally have different objectives and constraints. How we combine the two problems in a single framework will induce any of several basic heuristic approaches:

Heur1. Size cells freely (enforcing only the (T1)(T2) constraints, as is traditional in gate-sizing formulations), and fix placement at a later stage (enforcing all of (T1)(T2)(P1)(P2)).

Heur2. Constrain sizing to enforce all placement and design rules (i.e., with enforcement of (T1)(T2)(P1)(P2)).

Heur3. Size cells with partial constraints of placement (enforcing (T1)(T2) constraints, and relaxed (P1)(P2) constraints), such that only a small number of violations require fixing at the ECO placement stage.

We observe that **Heur1** may achieve the best power reduction with its sizing optimization, but may also result in a large number of MinIA violations in placement. At the other extreme, **Heur2** may achieve only limited power reduction since all potential sizing moves for each cell are restricted by placement. **Heur3** may be viewed as a compromise between the two methods. We note that **Heur3** gives the best results in our experiments reported in Section 5 below.

A truly simultaneous optimization of sizing and placement for MinIA fixing is not obvious to us at this point: it appears difficult to explore the entire solution space since there are so many combinations of sizes, locations, V_t assignments, and filler cell assignments. For now, we have pursued sequential optimization of sizing and placement, with (i) a sizing heuristic that considers placement, and (ii) placement perturbation heuristics to fix post-sizing MinIA violations. For the placement optimization stage, we define a ***MinIA*-aware placement** problem, derived from the ***MinIA*-aware sizing and placement** problem, in which the objective is to minimize the number of violations:

Problem: ***MinIA*-aware placement**

$$\textbf{Minimize:} \quad \sum_{\forall c} B(c)$$

$$\textbf{Subject to:} \quad S_c > 0 \qquad \text{(T1)}$$

$$T_r(c) < T_{r_{max}} \qquad \text{(T2)}$$

The **Experiment 1** discussion in Section 5 below assesses our placement algorithms in the context of this ***MinIA*-aware placement** problem.

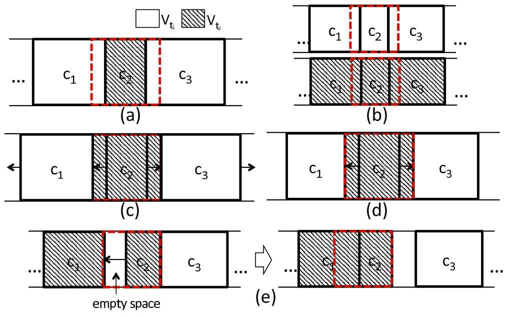


Figure 4: Available fixing approaches for MinIA rule violations. A given violation, depicted in (a), can be fixed by using (b) V_t-swapping, or (c) moving a neighbor cell, or (d) downsizing a neighbor, or (e) moving the narrow cell.

4. OUR APPROACHES

We now discuss our gate sizing and placement approaches considering MinIA constraints.

4.1 Minimum implant area-aware placement

Levers to solve MinIA rule violation. If violators of MinIA constraints have enough spacing around them, then the violations can be easily fixed by inserting same-V_t filler cells, thereby increasing the width of the implant area. However, when the spacing is insufficient, we must change cell size/V_t or perturb the placement, such that the implant layer region of the narrow cell can match up with an adjacent (abutting) implant layer region. Figure 4 illustrates four ways to fix a given violation. Suppose that cell c_2 with V_{t_2} violates the MinIA constraint, and that c_1 and c_3 are neighbor cells with V_{t_1}, as shown in Figure 4(a). First, we can fix the violation either by swapping V_t of c_2 to V_{t_1} or by swapping V_t of c_1 and c_3 to V_{t_2}; this is shown in Figure 4(b). Second, we can push out (i.e., shift) c_1 and c_3 to create spacing for filler cell insertion, as shown in Figure 4(c). Third, c_1 and c_3 can be downsized to create spacing around c_2, as shown in Figure 4(d). Fourth, the violator can be relocated so that it becomes abutted to a same-V_t cell, c_1 in Figure 4(e).

Algorithm 1 *MinIA*-aware Placement Heuristic

```
Procedure $fixMinImpVio(c, \{I_{min}\}, D)$
Input : a cell $c$, a set of min implant constraints $\{I_{min}\}$, a netlist $D$, a placement of $D$
Output : a sizing/location solution for $c \in D$
1: for all cell $c \in D$ do
2:     $s \leftarrow 0$;
3:     for all cell $n \in \{N(c)\}$ do
4:         $s \leftarrow s +$ spacing of $c$ to $n$;
5:     end for
6:     $MinImpSlack \leftarrow c.width - I_{min}(V_t(c))$;
7:     if $s + MinImpSlack \geq 0$ then insert filler cells;
8: end for
9: if #violations is zero then return success;
10: for all cell $c \in D$ do
11:     if cell $c$ violates then $ChangeVtCell(c)$;
12: end for
13: if #violations is zero then return success;
14: for all cell $c \in D$ do
15:     if cell $c$ violates then $MoveNCell(c)$;
16: end for
17: if #violations is zero then return success;
18: for all cell $c \in D$ do
19:     if cell $c$ violates then $DownSizeNCell(c)$;
20: end for
21: if #violations is zero then return success;
```

Heuristic approach for fixing minimum implant area violations. Algorithm 1 shows the overall flow of our heuristic approach, which is based on the four MinIA violation-fixing approaches noted above. First, whitespace around violating cells is calculated (Line 3). If there is enough spacing, sufficient width of same-V_t filler cells is inserted to fix the violation (Line 8). In the next step, V_t-swapping is performed for the violating cell or its neighbor cells with *ChangeV$_t$Cell* (Line 13). For any violations remaining after this step, there is no space for insertion of filler cells and V_t-swapping is unavailable due to timing constraints. Thus, we try to create spacing by changing the placement. We first try to move neighbor cells (Line 17).[2] Then, downsizing of neighbor cells (Line 21) can be tried if those cells are not timing-critical. Note that these steps can be performed in a different order – e.g., downsizing of neighbor cells can be performed before moving cells. The particular sequence of optimizations of steps used in our flow has been experimentally determined. Filler insertion is performed first, since it does not require any cost. Our studies of the permutations of V_t*-swapping*, *moving* and *downsizing cells* indicate that downsizing of cells occurs very rarely, since many cells are small and timing violations can result. With respect to V_t-swapping and moving cells, the results are better when V_t-swapping is performed first (e.g. #MinIA violations of V_t*-swapping-first* and *moving-first* are 155 and 44, respectively, for the *jpeg* testcase). We also note that our heuristic flow executes steps occur in an order that minimizes perturbation of placement or sizing.

Subroutine 1 Functions for *MinIA*-aware Placement Heuristic

```
1:  Procedure $ChangeVtCell(c)$
2:  Input : a cell $c$
3:  Output : $V_t$ solutions for $c$ and its neighbor cells
4:  // Change $V_t$ of the violator
5:  $V_{t_{orig}} \leftarrow V_t(c)$;
6:  for all cell $n \in \{N(c)\}$ do
7:      $V_t(c) \leftarrow V_t(n)$;
8:      $w \leftarrow$ total width of $c$ and $\{N(c)\}$;
9:      if $w \geq I_{min}(V_t(c))$ && $c$ does not violate timing then return success;
10:     else $V_t(c) \leftarrow V_{t_{orig}}$;
11: end for
12: // $V_t$-swapping for neighbors
13: for all cell $n \in \{N(c)\}$ do
14:     $V_{t_{orig}} \leftarrow V_t(n)$; $V_t(n) \leftarrow V_t(c)$;
15:     if $n$ violates timing then
16:         $V_t(n) \leftarrow V_{t_{orig}}$; continue;
17:     end if
18:     $w \leftarrow$ total width of $c$ and its neighbor cells;
19:     if $w \geq I_{min}(V_t(c))$ then return success;
20: end for
1:  Procedure $MoveNCell(c)$
2:  Input : a cell $c$
3:  Output : location solutions for $c$'s neighbor cells
4:  $n_{cl} \leftarrow$ the left neighbor cell; $n_{cr} \leftarrow$ the right neighbor cell;
5:  $s \leftarrow$ spacing of $c$ to $n_{cl}$ and $n_{cr}$;
6:  $MinImpSlack \leftarrow c.width - I_{min}(V_t(c))$;
7:  if $n_{cl}$ has leftside space $\Delta$ then
8:      $d \leftarrow max(-(MinImpSlack+s),\Delta)$; Move $n_{cl}$ by $d$; $s \leftarrow s+d$;
9:  end if
10: if $s+MinImpSlack < 0$ then
11:     if $n_{cr}$ has rightside space $\Delta$ then
12:         $d \leftarrow max(-(MinImpSlack+s),\Delta)$; Move $n_{cr}$ by $d$; $s \leftarrow s+d$;
13:     end if
14: end if
15: if $s+MinImpSlack \geq 0$ then
16:     insert filler cells; return success;
17: else
18:     return fail;
19: end if
```

Details of two levers are described in Subroutine 1. *ChangeV$_t$Cell* tries V_t-swapping for c, using any V_t in $\{N(c)\}$, and checks timing and MinIA violations. If there is any violation, the V_t-swapping of c is reverted (Line 10). In a similar way, V_t-swapping of neighbor cells is tried. *MoveNCell*(c) moves neighbor cells to create additional space. It first checks whitespace around the left and right neighbor cells and moves these cells if possible, similar to [9]. We limit the movement of cells to avoid large perturbation of placement, in light of the timing impact of cell placement.[3]

Our minimum implant area-aware placement can be used standalone, in an ECO methodology, to fix MinIA violations. We evaluate our approach with various minimum implant area constraints in Section 5, **Experiment 1**.

4.2 Minimum implant area-aware gate sizing

Our gate sizing method is based on sensitivity-guided gate sizing [6] [7]. In addition to timing constraints, we consider the placement constraints (P1)(P2) described in Section 3. Our objective is to minimize power without creating any additional (P1)(P2) violations.

Algorithm 2 *MinIA*-aware Gate Sizing Heuristic

```
    Procedure $GSMinImp(\{I_{min}\},D)$
    Input : minimum implant constraints $\{I_{min}\}$, a netlist $D$, a placement of $D$, a set of timing constraints
    Output : a sizing solution
1:  $M \leftarrow \emptyset$
2:  for all cell $c$ in the netlist $D$ do
3:      if $c$ is downsizable then
4:          $m_k.c \leftarrow c$; $m_k.m \leftarrow downsize$; $m_k.cost \leftarrow \Delta TNS$;
5:          $m_k.sensitivity \leftarrow \Delta Leakage/m_k.cost$;
6:          $M \leftarrow M \cup m_k$;
7:      end if
8:      if $c_i$ is not a highest $V_t$ cell then
9:          $m_k.c \leftarrow c$; $m_k.m \leftarrow V_t\ upscaling$; $m_k.cost \leftarrow \Delta TNS$;
10:         $V_{t_{orig}} \leftarrow V_t(c_i)$;
11:         $V_t(c) \leftarrow$ higher $V_t$; // placement cost calculation
12:         $\{N(c)\} \leftarrow \{N(c)\} \cup c$
13:         for all $n \in \{N(c)\}$ do
14:             if $n$ violates $I_{min}(V_t(n))$ then
15:                 if fixable by $V_t$-swapping/sizing/move of neighbors $n$ then
16:                     $m_k.cost \leftarrow m_k.cost + CalCost(n)$;
17:                 else
18:                     break; continue to the next $c_i$;
19:                 end if
20:             end if
21:         end for
22:         $V_t(c) \leftarrow V_{t_{orig}}$; $m_k.sensitivity \leftarrow \Delta Leakage/m_k.cost$; $M \leftarrow M \cup m_k$;
23:     end if
24: end for
25: while $M \neq \emptyset$ do
26:     Pick a $m_k$ with maximum $sensitivity$ in $M$; Commit $m_k$; $M \leftarrow M \setminus m_k$;
27:     $STA()$;
28:     Fix the extra MinIA violations;
29:     if $!feasible()$ then restore $m_k$;
30: end while
```

For each gate, we check whether the potential sizing produces violations that require placement perturbations or sizing of neighbor cells to be resolved. If the neighbor cells need to be changed or relocated, we estimate the timing impact on neighbor cells and add this to the sensitivity function. During gate sizing to recover power, a gate can be downsized or its V_t can be swapped to a higher threshold voltage. Downsizing a gate can produce violations, but if neighbor cells do not consume the whitespace created by downsizing, the violations can be easily cured by inserting filler cells. However, when a gate is V_t-swapped and creates violations by itself or in relation to its neighbor cells, possible fixing methods should be carefully explored. Algorithm 2 describes our sizing flow. $\Delta Leakage/cost$ is used as our sensitivity function. The default cost is the change in total negative slack

[2] We do not need to try moving the target cell (Figure 4(e)) as this case could have be fixed at the filler cell insertion stage.

[3] We observe that up to 10 placement grids of movement will change timing by less than 2 ps with the 45nm foundry library. We allow perturbation of cell location by up to 10 grids.

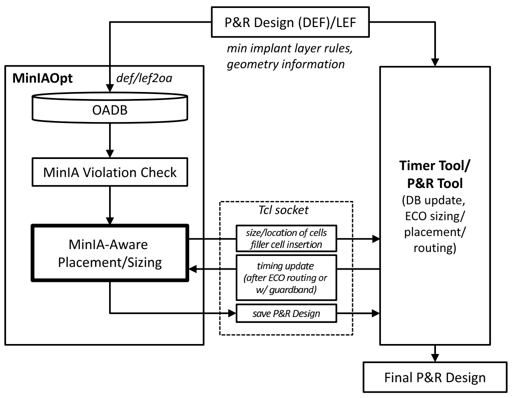

P&R Design (DEF)/LEF
min implant layer rules, geometry information
MinIAOpt
def/lef2oa
OADB
MinIA Violation Check
MinIA-Aware Placement/Sizing
Tcl socket
size/location of cells filler cell insertion
timing update (after ECO routing or w/ guardband)
save P&R Design
Timer Tool/ P&R Tool
(DB update, ECO sizing/ placement/ routing)
Final P&R Design

Figure 5: Overall flow of our optimizer, *MinIAOpt*.

(TNS). When MinIA violations occur, we additionally calculate potential decrease (worsening) of TNS from changing neighbor cells to fix the violations ($CalCost()$, Line 16). This calculated cost is then added to $cost$ so that the $sensitivity$ decreases.

4.3 Overall flow

Figure 5 shows the overall flow of our optimizer, *MinIAOpt*. A DEF file of a routed netlist, and LEF files for geometry information of standard cells and technology information including the minimum implant layer rules, are converted into *OpenAccess* [20] DB using *def/lef2oa* parsers. The minimum implant area-aware gate sizing is performed to reduce leakage power with considerations of geometry information. Further minimum implant area-aware placement optimization can be performed to fix MinIA violations without creating new timing violations. For both stages, a *Tcl* socket interface is used to enable the communication between the P&R tool and/or timer tool and our optimizer. Via this interface, *MinIAOpt* can send commands to insert filler cells, size/V_t-swap cells and change the locations of cells, as well as obtain updated timing information after ECO routing. We also can use a quick timing estimation from timer tools without ECO routing to achieve faster runtime while sacrificing some accuracy.[4]

5. EXPERIMENTAL RESULTS

5.1 Experimental setup

Our program is written in C++, and the interface to support DEF/LEF [17] is implemented using the *OpenAccess 2.6* [20] API. We use a *Tcl*-socket interface (*Tcl/Tk 8.4* [23]) to communicate with P&R and timer tools similar to Trident2.0 [7] and SensOpt [24]. We have applied our proposed method to a set of open-source designs [19], which we synthesize from RTL using *Synopsys Design Compiler H-2013.03-SP3* [22]. For P&R, we use *Cadence Encounter Digital Implementation System XL 13.1* [16]. Implementations in all experiments are with a 45nm foundry technology and library.

Table 2 shows the testcases used in our experiments. We compare to the result of a simple filler cell insertion performed by a commercial P&R tool with high-utilization testcases that are synthesized with a standard implementation flow. We test various MinIA constraints to understand the scaling of algorithm performance with instance difficulty (e.g., reflecting MinIA constraints in future technology nodes). The utilization, the distribution of cell V_t values[5], and the percentage of smaller cells (% Mincells) with width less than the minimum implant area constraints all affect the difficulty of a given testcase. We use various minimum implant width constraints $Const1$, $Const2$ and $Const3$, corresponding to four, six and seven, respectively. In the 45nm library, 3%, 12% and 28% of standard cells are narrower than these constraints. Also, to study the sensitivity of the results to the difficulty of problem instances, we intentionally tweak the V_t cell distribution of *aes* and generate *aes_var** so that the same placement will have various numbers of MinIA violations. All experiments are performed on a 2.5GHz Intel Xeon Linux workstation.

[4]For any change of cells, ECO routing should be performed and timing should be updated accordingly. But, performing ECO routing for each cell change takes too much time to be practically feasible. To compensate for the inaccuracy, a timing guardband can be used.

[5]The percentage of each V_t type relative to the total number of cells. H/N/L indicates HVT, NVT and LVT %, respectively.

Table 2: Testcases used in the experiments. *WS, Period, Leak* are worst slack, clock period and leakage power, respectively, after routing, before filler cell insertion stage.

Bench	#Inst	Util	Mincells	V_t(H/N/L)	WS	Period	Leak
		(%)	(%)	(%)	(ps)	(ns)	(mW)
dma	1168	78	59	78/4/16	137	1.0	0.050
mpeg	7121	82	64	83/6/9	39	1.25	0.363
aes	9611	75	84	95/1/2	21	1.5	0.238
jpeg	44911	81	68	82/8/8	36	1.6	2.413
aes_var1	9611	75	84	40/30/30	39	2.1	0.129
aes_var2	9611	75	84	60/20/20	82	2.4	0.106
aes_var3	9611	75	84	80/10/10	137	2.5	0.080

5.2 Experimental results

Experiment 1: Evaluation of MinIA-fix algorithms. Our first experiment evaluates the MinIA-fix algorithm under power and timing constraints. In Table 3, *Commercial P&R* indicates the result of simple filler cell insertion performed by a commercial P&R tool. *Δ#Vio.(%)* shows the absolute (relative) change in number of MinIA violations compared to the original number of violations (negative numbers indicate that the number of violations is reduced). *Commercial P&R* does not change the design, and the runtime is almost zero. However, it fixes only 36% of MinIA violations in the worst case (i.e., 64% of violations remain), and 74% on average across all testcases. By contrast, our heuristic substantially reduces the number of MinIA violations (97% in the worst case, and by 99% on average). Note that the WS of all testcases is positive even though ΔWS is negative for the case of *mpeg* with $Const3$, *aes_var2* and *aes_var3*. *Fill (%)* indicates the portion of the total area occupied by filler cells. We see that the numbers are very small, which means that whitespace is not all consumed by filler cells. The *CPU total/tool* shows the total runtime, and the time consumed by the socket interface between external tools (i.e., P&R and timer tool) and our optimizer. Nearly all of the total runtime is consumed by external tools, since the operations used during the optimization such as adding filler cells, moving and/or sizing a cell take *O(few seconds per operation)*.

Experiment 2: MinIA-aware sizing. Our second experiment evaluates three approaches – free sizing (**Heur 1**), restricted sizing (**Heur 2**) and MinIA-aware sizing (**Heur 3**) with our heuristic approach (Algorithm 2) in terms of leakage power, timing and the number of MinIA violations. *Const3* is used for the minimum implant width constraint. Table 4 shows **Heur 1**, **Heur 2** and **Heur 3** results after sizing. Although the leakage power values are smallest with **Heur 1**, the increase in number of MinIA violations is up to 151% of the original number of MinIA violations. With **Heur 2**, the leakage power values are high since the sizing is prevented from creating any violation for the target cell. The increase in number of violations comes from the impact of sizing on neighbor cells. **Heur 3** shows near-zero or small increase in MinIA violations with less leakage power than **Heur 2**. In the **Heur 3** results, we see that solution quality in terms of leakage and

Table 3: Results for a simple filler insertion and our heuristic method.

Bench	**MinIA Const**	**Orig. #Vio.**	**Commercial P&R**	**Heuristic**					
			Δ #Vio. (%)	**Δ WS (ps)**	**Δ Leak (mW)**	**Fill (%)**	**Δ #Vio. (%)**	**CPU total / tool (min)**
dma	*Const1*	71	-53 (-75)	0	0.000	0.4	-71 (-100)	1.9 / 1.0
	Const2	128	-93 (-73)	0	0.000	1.7	-128 (-100)	2.1 / 1.1
	Const3	193	-151 (-78)	0	0.000	2.5	-193 (-100)	2.0 / 1.1
mpeg	*Const1*	183	-155 (-85)	0	0.000	0.1	-183 (-100)	4.6 / 3.7
	Const2	453	-322 (-71)	-1	0.000	0.7	-451 (-100)	5.9 / 5.0
	Const3	693	-515 (-74)	-18	0.000	1.0	-689 (-99)	6.7 / 5.8
aes	*Const1*	338	-327 (-97)	0	0.000	0.5	-338 (-100)	5.1 / 4.1
	Const2	978	-868 (-89)	0	0.000	2.6	-978 (-100)	8.2 / 7.2
	Const3	1146	-1005 (-88)	0	0.000	3.4	-1146 (-100)	8.7 / 7.8
jpeg	*Const1*	1341	-1186 (-88)	0	0.001	0.3	-1341 (-100)	25.3 / 24.3
	Const2	3865	-3079 (-80)	0	0.004	1.5	-3850 (-100)	73.8 / 72.9
	Const3	7864	-6077 (-77)	0	-0.002	2.5	-7820 (-99)	168.7 / 167.7
aes_var1	*Const3*	2955	-1069 (-36)	28	0.001	10.8	-2863 (-97)	32.3 / 31.4
aes_var2		2558	-1106 (-43)	-51	-0.002	8.5	-2492 (-97)	25.0 / 24.1
aes_var3		1816	-1014 (-56)	-84	-0.002	4.1	-1792 (-99)	15.6 / 14.7

Table 4: Results for our heuristic sizing algorithm. *Const*3 is used for the minimum implant width constraint in this experiment.

Benchmarks	**WS (ps)**			**Leak (mW)**			**Δ #Vio. (%)**			**CPU total / tool (min)**		
	Heur1	**Heur2**	**Heur3**	**Heur1**	**Heur2**	**Heur3**	**Heur1**	**Heur2**	**Heur3**	**Heur1**	**Heur2**	**Heur3**
dma	3	131	**4**	0.046	0.048	**0.047**	64 (33)	50 (26)	**0 (0)**	7 / 5	5 / 3	**7 / 5**
mpeg2	3	19	**19**	0.298	0.315	**0.309**	376 (54)	164 (24)	**9 (1)**	43 / 41	27 / 26	**34 / 33**
aes	6	18	**18**	0.184	0.213	**0.203**	1734 (151)	814 (71)	**412 (36)**	98 / 96	51 / 50	**69 / 68**
jpeg	-5	9	**10**	1.898	2.109	**1.954**	4861 (62)	2921 (37)	**659 (8)**	1209 / 1207	663 / 662	**1093 / 1091**

timing is nearly maintained, while the number of MinIA violations is greatly reduced.

Additionally, we have applied our placement heuristic to the results of sizing. For *aes*, we observe that 22% of MinIA violations still remain in the result of **Heur 1**, while 3-4% of those are left for **Heur 2** and **Heur 3**. Note that the MinIA violations may increase the total area and power even though the initial leakage power might be less with **Heur 1**.

6. CONCLUSIONS

In this work, we have addressed a new gate sizing/V_t-swapping and placement problem with the *minimum implant area* (MinIA) constraint. The MinIA constraint presents a new challenge to the physical implementation flow in sub-22nm technology, and requires true co-optimization of placement and gate sizing/V_t-swapping. We have proposed sizing and placement heuristics that optimize power and fixes MinIA violations while minimizing placement perturbation. Compared to commercial P&R tools, our methods achieve significant reductions in the number of MinIA violations under timing/power constraints.

Our current heuristics cannot guarantee to minimize the perturbation of placement and/or the number of violations, though they are straightforward and easy to apply. Hence, similar to [3], we intend to study the use of dynamic programming to solve single-row placement with MinIA fixing. Procedure *fixRowMinImpVio()* sketches such an approach. Our ongoing work includes implementation of, and analysis of results from, dynamic programming based optimizations.

```
Procedure $fixRowMinImpVio(\{I_{min}\}, R, \{T\})$
Input : minimum implant constraints $\{I_{min}\}$, a placement of a standard-cell row $R$, a set of timing constraints $\{T\}$
Output : a sizing/placement solution for $R$
$\{c_i.sol\} \leftarrow \emptyset$;
// $\{c_i.sol\}$ = sizing/placement solutions from the left-most cell to the $i$th cell
for $i = 1$ to $k$ do
    for all $\{l, v, s\}$, where $l = -W$ to $W$, $v$ = LVT, NVT, HVT, $s = -S$ to $S$ do
        // $l$ = Δ cell location, $v = V_t$, $s$ = Δ cell size
        $Cost(c_{i,l,v,s}) \leftarrow \min_{j \in \{c_{i-1}.sol\}} Cost(\{c_{i,l,v,s}, j\})$
        // $Cost(*)$ = cost of power and minIA violations
        $\{c_i.sol\} \leftarrow \{c_i.sol\} \cup c_{i,l,v,s}.sol$;
    end for
end for
$sol \leftarrow$ the minimum cost solution in $\{c_k.sol\}$;
return $sol$;
```

7. REFERENCES

[1] U. Brenner and J. Vygen, "Faster Optimal Single-Row Placement with Fixed Ordering", *Proc. DATE*, 2000, pp. 117-121.

[2] A. Chakraborty, S. X. Shi and D. Z. Pan, "Stress Aware Layout Optimization Leveraging Active Area Dependent Mobility Enhancement", *IEEE Trans. on CAD* 29(10) (2010), pp. 1533-1545.

[3] 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.

[4] ITRS Low-Power Design Technology Roadmap, Design Chapter Table DESN14, 2011. `http://public.itrs.net/reports.html`

[5] V. Joshi, B. Cline, D. Sylvester, D. Blaauw and K. Agarwal, "Leakage Power Reduction Using Stress-Enhanced Layouts", *Proc. DAC*, 2008, pp. 912-917.

[6] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim and I. L. Markov, "Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing", *Proc. ICCAD*, 2012, pp. 233-239.

[7] A. B. Kahng, S. Kang, H. Lee, I. L. Markov and P. Thapar, "High-Performance Gate Sizing with a Signoff Timer", *Proc. ICCAD*, 2013, pp. 450-457.

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

[9] A. B. Kahng, P. Sharma and R. O. Topaloglu, "Exploiting STI Stress for Performance", *Proc. ICCAD*, 2007, pp. 83-90.

[10] A. B. Kahng, P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites", *Proc. ASPDAC*, 1999, pp. 241-244.

[11] J. Lee and P. Gupta, "Incremental Gate Sizing for Late Process Changes", *Proc. ICCD*, 2010, pp. 215-221.

[12] J. Lee and P. Gupta, "Discrete Circuit Optimization", *Foundations and Trends in Electronic Design Automation* 6(1) (2012), pp. 1-120.

[13] J. Li, B. Yang, X. Hu, Q. Dong and S. Nakatake, "STI Stress Aware Placement Optimization Based On Geometric Programming", *Proc. GLSVLSI*, 2009, pp. 209-214.

[14] T. Luo, D. Newmark and D. Z. Pan, "Total Power Optimization Combining Placement, Sizing and Multi-Vt Through Slack Distribution Management", *Proc. ASPDAC*, 2008, pp. 352-357.

[15] L. Remy, P. Coll, F. Picot, P. Mico and J.-M. Portal, "Definition of an Innovative Filling Structure for Digital Blocks: the DFM Filler Cell", *Proc. ICECS*, 2009, pp. 73-76.

[16] Cadence SOC Encounter User Guide. `http://www.cadence.com/products/di/first_encounter/pages/default.aspx`

[17] LEF DEF reference. `http://www.si2.org/openeda.si2.org/projects/lefdef`

[18] Mentor Graphics Olympus-SoC. `http://www.mentor.com/products/ic_nanometer_design/place-route/olympus-soc`

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

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

[21] Synopsys IC Compiler User Guide. `http://www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/ICCompiler.aspx`

[22] Synopsys Design Compiler User Guide. `http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages`

[23] Tcl/Tk Built-in Commands Manual. `http://www.tcl.tk/man/tcl8.4/TclCmd`

[24] UCSD SensOpt Leakage Optimizer (A. B. Kahng and S. Kang, 2010-2011), `http://vlsicad.ucsd.edu/SIZING/optimizer.html`