Interactive Analog Layout Editing With Instant Placement and Routing Legalization

Xiaohan Gao, Haoyi Zhang, Mingjie Liu, *Member, IEEE*, Linxiao Shen, *Member, IEEE*, David Z. Pan, *Fellow, IEEE*, Yibo Lin, *Member, IEEE*, Runsheng Wang, *Member, IEEE*, and Ru Huang, *Fellow, IEEE*

***Abstract*—Analog layout design is still primarily reliant on manual efforts. Current fully automated workflows are unable to meet the expectations for flexible customization and are incompatible with existing manual workflows. For both performance and productivity, interactive layout editing has the ability to bridge the gap between manual and automated flows. We present an interactive layout editing system in this study that includes well-defined commands for both placement and routing customization. This is a pioneering work that provides a holistic study on the interactive design methodology for analog layouts and its capability of speeding up design closure. Our framework comes up with the instant placement legalization and routing adjustment mechanism for rapid layout update and modification. The framework is capable of handling real-time user interaction and improving the performance of fully automated layout generators verified by post-layout simulation on real-world analog designs. Experimental results demonstrate the performance enhancement on real-world analog designs with only a few editing commands. As examples, on the low-dropout regulator, our framework can reduce the overshot down and up voltage to nearly 1/3 of layout generated by automation tool with two editing commands, and on the operational transconductance amplifier, it achieves 33.5% better common mode rejection ratio with only one command.**

***Index Terms*—Analog CAD, analog integrated circuits, physical design, placement, routing.**

I. Introduction

ANALOG layout design is primarily reliant on manual efforts. Designers or layout engineers create device location and wire routing based on their design experience, taking into account symmetry, matching, signal flow, and other constraints. However, as design complexity and design rules get more difficult, their productivity and design closure are severely impacted. To speed up the analog design flow, layout automation tools are required.

Fully automated analog layout tools, such as ALIGN [1], [2] and MAGICAL [3]–[5], have recently been presented. With the goal of creating an end-to-end analog layout design flow without a human in the loop, they use machine learning and analytic algorithms to automatically build analog layouts from circuit netlists [6]–[9]. Typical stages in these frameworks include constraint generation that extracts layout constraints from netlists, analog placement that determines the positions of devices, and routing that connects devices in the layout. These fully automated layout generation processes follow the no-human-in-the-loop concept, completing layout synthesis with "one-button-click" and without expecting any intermediary human involvement.

Although generating layouts automatically can greatly reduce prototyping time and give acceptable initial solutions for certain circuits, it is difficult to obtain widespread acceptance in the short term for the reasons listed as follows.

1) New analog circuit topologies are quickly developing. It is insufficient for a fully automated analog design flow to meet various customization requests.
2) Analog designers have their own styles to draw layouts. For example, two designers can arrange the same circuit in very different ways based on their individual expertise and taste, which is difficult to apply in a fully automated design flow.
3) Layouts created by a fully automated design flow, such as the example in Fig. 1, can be counterintuitive to designers, resulting in difficulties in post-silicon debugging or tape-out failure.

As a result, fully automated layout generation alone is insufficient to meet the needs for A faster analog design closure.

We make the following observations after investigating the workload and working routines of the designers.

1) Designers tend to visually inspect and try various layout drawing strategies before making a final decision.
2) Designers are often reluctant to take the risk of trying unfamiliar layout topologies.
3) Cleaning design rule violations consumes a large amount of time.

Manuscript received 11 January 2022; revised 11 May 2022; accepted 16 June 2022. Date of publication 12 July 2022; date of current version 20 February 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62141404; in part by Hisilicon; and in part by the 111 Project under Grant B18001. The preliminary version has been presented at the Design Automation Conference (DAC) in 2021 [DOI: 10.1109/DAC18074.2021.9586234]. This article was recommended by Associate Editor S. Mohanty. *(Xiaohan Gao and Haoyi Zhang contributed equally to this work.) (Corresponding author: Yibo Lin.)*

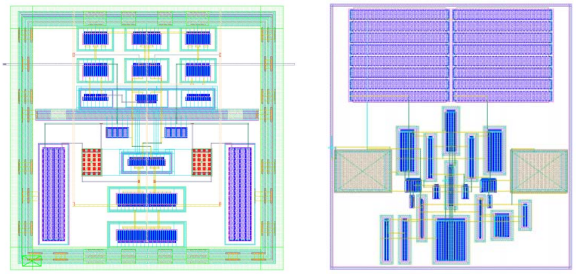
Xiaohan Gao is with the School of Computer Science and the School of Integrated Circuits, Peking University, Beijing 100871, China.

Haoyi Zhang is with the School of Integrated Circuits, Peking University, Beijing 100871, China.

Mingjie Liu and David Z. Pan are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA.

Linxiao Shen, Yibo Lin, Runsheng Wang, and Ru Huang are with the School of Integrated Circuits, Peking University, Beijing 100871, China, and also with the Beijing Advanced Innovation Center for Integrated Circuits, Beijing 100871, China (e-mail: yibolin@pku.edu.cn).

Digital Object Identifier 10.1109/TCAD.2022.3190234



(a) (b)

Fig. 1. Layouts for an OTA design. (a) Manual solution is very different from (b) tool solution [3].

In other words, designers want a tool that can help them increase their productivity while maintaining complete control over the layout design process. Therefore, interactive layout editing can be a promising solution for bridging the gap between the expectations of designers and the existing fully automated analog design flows. With such observations, we present an interactive analog layout editing framework in this work, to speed up the design closure and boost the productivity of layout drawing. Early analog layout automation tools require users' interaction with manually written specifications [10]–[12]. Different from those semiautomation works, our framework leverages the power of interactively editing the layout. The goal of this framework is to liberate designers from tedious design rule fixing and allow them to concentrate on designing layout topologies. Meanwhile, we retain the ability to fully customize layout designs.

The main contributions of this work are summarized as follows.

1) We propose an interactive analog layout generation framework supporting fast layout editing. We provide instructions for global perturbation and local adjustment, allowing for both topological editing and fine-grained customization.
2) We propose an instant legalization technique with linear time complexity for incremental placement updating, allowing real-time interactivity based on user input instructions. Experiments on legalization validate the capability of efficient placement editing and reducing the turn-around time of the layout design.
3) We propose routing kernels that can realize versatile interactive routing adjustments, such as spacing editing and topology adjustment.
4) Experiments on real-world analog designs show the capability of our framework to improve the post-layout simulation performance with simple commands.

The organization of this article is listed in the following. Section II presents the limitation of fully automated analog design flows and the basic idea of interactive analog layout editing. Section III illustrates the layout editing procedure. Section IV explains the instant legalization technique in detail. Section V introduces the critical algorithms involved in interactive routing. Section VI validates the effectiveness of the proposed interactive layout editing framework. Section VII concludes this article.

II. Preliminaries

In this section, we discuss the problem formulation and background of our work.

A. Fully Automated Layout Synthesis

A fully automated analog layout design flow targets end-to-end layout generation without any human involvement in the loop. The automated flow regards circuit netlists and technology libraries as input, and layout solutions with placement and routing as outputs. The automated flow defines optimization objectives and converts the layout generation problem to a constrained optimization problem. For instance, the automated flow detects symmetry or matching constraints to assist with placement and routing.

As a universal analytical objective is hard to be defined, generating a stable and high-performance layout within an end-to-end flow is still a challenging problem. There are works abstracting designers' experience from manually drawn layouts, such as ALIGN and MAGICAL [1], [3]. These works incorporate machine learning techniques into constraint detection. These works show good efficiency in generating layout solutions for several specific analog circuits, but still, need further refinement for a wider application of analog designers.

B. Interactive Analog Layout Editing

Interactive analog layout editing approaches the layout generation problem in an orthogonal but complementary way to the fully automated methodology. Layout editing employs an automated layout tool to generate the initial layout and engages designers in customizing the layout. Layout editing abstracts the topology from the layout and offers high-level control over the layout. We leave the low-level legalization and design rules fixing to automation algorithms. The interactive layout editing flow exports the final layout solution in GDSII format whenever the designers are satisfied with the layout result. The methodology of layout editing bridges the gap between the performance of existing automated layout tools and designers' expectations through an efficient interactive process.

This work focuses on interactive layout editing for analog designs. We formulate the problem as follows.

Problem 1 (Interactive Analog Placement Editing): Given an initial analog layout as input, define a set of commands and an interactive framework for users to efficiently edit the layout topology with the commands. The set of commands should be able to achieve any topological changes.

The goal is to enable designers to make progressive modifications to the layout topology with sequential commands. In order to interact with designers, we need a real-time legalization backbone to update the layout giving input commands. We formulate the problem as follows.

Problem 2 (Interactive Analog Placement Legalization): Given an input analog placement solution, constraint graphs, and layout constraints, legalize the placement subjecting to the constraints with minimum perturbation to the layout and minimum runtime.

In the same way as placement editing, we formulate the problem of interactive routing editing.

Problem 3 (Interactive Analog Routing Editing): Given a legalized placement solution, routing constraints, and command sequence for routing adjustment, modify the routing topology subjecting to both existing constraints and newly added constraints efficiently.

C. Analog Placement

Analog placement is closely related to layout editing. Previous researches consider enhancing how to represent analog placement [13], including using B*-tree [14], [15], and O-tree [16]. These researches often leverage heuristic search techniques like simulated annealing for high-quality placement results [17]. Other works share a similar idea with the analytical placement of digital circuits. Digital placement utilizes quadratic or nonlinear optimization to placement problem [18]–[22]. Some analog layout frameworks utilize analytical optimization to solve the analog placement [3], [9]. Recent works explore layout constraint detection with deep learning methods [6], [7]. Also, some works attempt to predict performance for optimizing placement [8], [23], [24]. Another line of work pays attention to generating layout with procedural form [25]–[27]. Although the prior works can produce silicon results, users are required to write a substantial amount of codes, which limits the application to brand new designs. Taking the advantages and limitations of previous works into consideration, we propose the interactive layout editing scheme to serve as an intermediary between procedural process and automated layout generation. The interactive layout editing supports visualization of layout and comprehensive customization.

Constraint graph is another mainstream method to represent the topology of analog layout [28]. Constraint graph describes a placement topology with two parts: 1) a horizontal constraint graph (HCG) and 2) a vertical constraint graph (VCG). The HCG and VCG are two directed acyclic graphs (DAGs). The HCG and the VCG specify the relative positions in the horizontal direction and in the vertical direction, respectively. For example, if a device u locates directly right to another device v, there will be a corresponding edge from vertex v to u in the HCG. Previous works explore variants of constraint graph for versatile layout constraints and layout dependent effects [29]–[31]. We propose a novel layout topology representation with one unified directed graph based on the constraint graph. It is worth noting that our layout editing scheme works with different types of layout representation methods.

D. Analog Routing

Analog routing is a critical concern in the layout editing process and has become increasingly complex as technology nodes advance. Most studies are devoted to handling various routing constraints. A number of studies focus on routing symmetry constraints, such as the symmetric net pair constraint [32]–[35], multilevel geometrical matching constraint [36], [37]. Recent works make efforts on design-rule-aware routing to handle end-of-line spacing and parallel run length constraint [38]. Although several state-of-art studies have made promising progress in diminishing the gap between manual and automated layouts, such as ALIGN [1], MAGICAL [3], etc., routing solutions given by the fully automated tools still cannot integrate designers' expertise in practice. Based on such observations, we propose interactive routing which allows users to adjust routing solutions given by automated tools with simple commands. Interactive routing combines the basic constraints and the expertise of analog designers to guide routing algorithms, which can bridge the gap between automated tools and designers' requirements.

III. Interactive Layout Editing

In this section, we elaborate the interactive layout editing workflow in detail.

A. Interactive Layout Editing Workflow

Fig. 2 illustrates the software architecture of our interactive analog layout editing tool. We provide both command interface and GUI interface in the frontend, aiming to perform real-time interaction and display analog layout. We encode operations on the interface as commands. We leverage the automatic analog layout tool MAGICAL [3] to generate the initial layout results displayed in GUI interface for users. The interactive layout editing consists of two stages, interactive placement, and interactive routing. The automated analog flow generates an initial routing solution with the placed layout which is produced by interactive placement. Through the command interpreter, the commands entered by users will be interpreted to either internal placement or routing operations for interactive placement and interactive routing, respectively. We implement the internal operations to update our data depiction which contains constraint graph, constraint set, and cell placement information. The routing solutions are updated by the routing operations directly with local adjustments. If the user requires a legalized placement, the legalization procedure will be performed extremely rapidly by legalizer in the feedback phase, so that the users obtain the legalized layout placement displayed on the interactive interface instantly.

B. Placement Editing Commands

The user is likely to be unsatisfied with the primary placement outcome generated by a fully automatic tool and desire flexibility to adjust the placement results due to their experience. We present a concise and extensible command set. The user is able to use a sequence of commands to describe the demands. Two major types of commands are supported: 1) fine-grained topology-related commands and 2) coarse-grained constraint-related commands. Table I illustrates six commands for interactive placement editing. Fined-grained commands includes `move`, `spacing`, `resize`, and `swap` which interoperate with the topology or geometry of the placement. Command `arrayAdd` introduces an array constraint that aligns the devices, as well as command `symAdd` introduces a symmetry constraint that enforces symmetric locations of devices. In line with expectations, the commands constitute

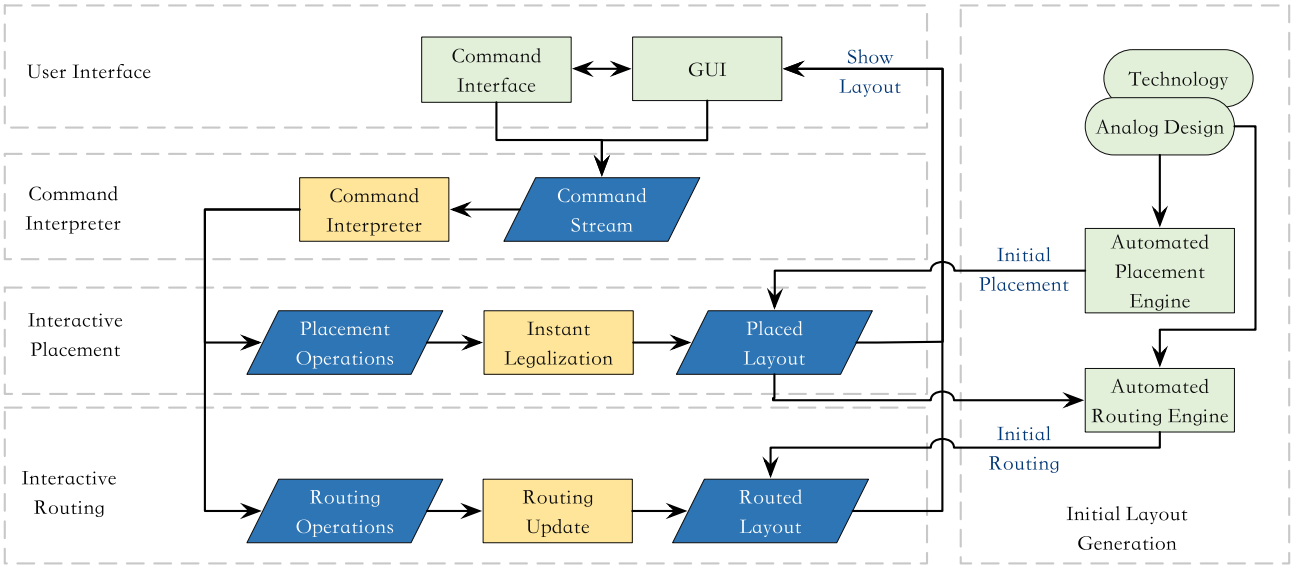


Fig. 2. Workflow of layout editing.

TABLE I
INTERACTIVE COMMAND SET

	Command	Parameters	Description
Placement	`move`	device v_i, destination location (x, y)	move a device to a destination location
	`spacing`	devices v_i, v_j, spacing width W	add spacing between devices
	`resize`	new shape w, h	change the shape of a device
	`swap`	devices v_i, v_j	swap two devices
	`arrayAdd`	a list of devices v_i	add array constraint
	`symAdd`	devices v_i, v_j, symmetry axis A_k	add symmetry constraint
Routing	`remove`	net n_i	remove wires of a single net
	`reroute`	net n_i	route a single net
	`wireWidth`	wire w_i of net n_j, new wire width W	set wire width
	`wireSpacing`	wire w_i of net n_m, wire w_j of net n_t, spacing width S wire w_i of net n_m, device v_j, spacing width S	adjust spacing between two wires adjust spacing between a wire and a device
	`netPriority`	a list of nets n_i with their priorities	designate routing priority for nets
	`netTopology`	net n_i, rough guide with points p_i	set topology for particular net

a comprehensive set for layout placement editing, i.e., for arbitrary two topologically distinct layouts L_i and L_j, we are able to find a command sequence converting L_i to L_j. The command sequence is usually concise.

C. Routing Editing Commands

As the performance of analog design can be sensitive to routing, it is hard for automatic tools to encode all the constraints explicitly into routing algorithms. To support flexible routing adjustments, we design routing commands for both wire-level and net-level adjustments, where a wire is a segment of a net and a net consists of multiple wire segments. Table I provides six commands for routing. Command `remove` and `reroute` provide straightforward manipulation for routing of a net. Command `wireWidth` assigns a new wire width. Command `wireSpacing` adjusts the spacing between two designated wires or between a wire and a device. Command `netPriority` modifies the routing priority of nets such that designers can specify to route critical nets first. Command `netTopology` further allows designers to specify rough routing topologies for nets and the routing algorithm will generate solutions following the topology guidance.

D. Command Interpretation

The layout is maintained in an internal representation including the constraint graph as well as a constraint set. All of the defined previously placement commands can be implemented with a sequence of internal operations. We identify a group of standard internal placement operations on the constraint graph: {`Insert`, `Remove`}, which is similar to the command set. Take command `swap` as example. Command `swap` is implemented for swapping the locations of two devices v_i, v_j with {`Insert`(v_i, p_j), `Remove`(v_i), `Insert`(v_j, p_i), and `Remove`(v_j)} on the constraint graph, where p_i and p_j are the former locations of v_i and v_j. If a vertex v_i is inserted to the position v_j being removed, the Insert operation will query the vertices adjacent to v_j originally and determine the new edges for vertex v_i.

Each routing command is converted into combinations of routing operations, such as {`remove_wires`, `reroute`, `add_route_obstacle`, `route_with_guide`}. Operation `remove_wires` removes a group of wires for new routing. Operation `add_route_obstacle` supports to insert an obstacle region for routing. Note that when combined with other operations, the magnitude of the

TABLE II
NOTATION

Symbol	Description
L	layout
G^h, G^v	horizontal/vertical constraint graph
G^m	mixed constraint graph
$\{(A_i, P_i)\}_{i=1,\cdots,k}$	symmetry axes with pair groups
$\{B_i\}_{i=1,\cdots,t}$	blocks
$b^{xl}, b^{xr}, b^{yb}, b^{yt}$	left, right, bottom and top boundaries
N_i	routing area node
n_i	routing net
P_i	coordinate of routing area node
GP_i	segement of the routing path
$\{GP_i\}$	whole routing path

obstacle is automatically computed to satisfy the design rules. Operation `reroute` implements A-star-based rerouting for a net. Operation `route_with_guide` performs routing following a sketchy topology guidance. A routing operation can be used in conjunction with other operations to implement a routing command. For instance, the routing command `wireSpacing` will be interpreted to `remove_wires` of the two involved wires, `add_route_obstacle` to ensure the new spacing between wires, and `reroute` for rerouting.

IV. Instant Legalization

In real-world application scenarios, the users execute commands and perform legalization constantly, so the legalization process is ought to be fast and consistent enough to make sure that users are able to see reliable legalized placement results instantly. In this section, we elaborate on the implementations of our linear-time legalization algorithm in detail. First, we outline the framework of our legalizer in Section IV-A. We then propose an innovative placement topology representation method, called the mixed constraint graph (MCG), in Section IV-B. The MCG data structure is the foundation of our algorithm, which supports the internal operations aforementioned in constant time $\mathcal{O}(1)$ and placement legalization in linear-time $\mathcal{O}(n)$ (n is the number of devices of the analog circuit). The primary algorithm includes a layout partitioning technique and a topological sort-based legalization scheme outlined in Sections IV-C and IV-D, respectively. A succinct analysis of temporal complexity is demonstrated in Section IV-E. Readers can consult Table II for notations.

A. Legalizer Overview

The proposed legalizer can deal with both flat circuits and hierarchical circuits. We demonstrate the legalization flow for a flat circuit in Algorithm 1. It is simple for the flow to be extended to hierarchical circuits by tackling the circuit hierarchy from the bottom up. When dealing with hierarchical circuits, the legalization procedure starts from the leaf-level subcircuits in the circuit hierarchy tree. At the leaf-level, the legalizer takes the device-level constraint graph and device symmetry groups as inputs as well as implements legalization instantly inside each subcircuit. We consider the child subcircuits as cells and duplicate the legalization process for a higher level subcircuit.

Algorithm 1 Instant Legalization

Input: initial constraint graphs G^h and G^v, symmetry axes with their corresponding symmetry pair groups $\{(A_i, P_i)\}$
Output: legalized layout L
1: **for** A_i, P_i in $\{(A_i, P_i)\}$ **do**
2: traverse P_i and update boundary $(b_i^{xl}, b_i^{xr}, b_i^{yb}, b_i^{yt})$
3: **end for**
4: Blocks $\{B_i\} \leftarrow$`PartitionLayout`$\{(b^{xl}, b^{xr}, b^{yb}, b^{yt})\}$
5: construct global mixed constraint graph G^m
6: **for** B_i in $\{B_i\}$ **do**
7: do topo-sort based legalization inside B_i
8: **end for**
9: do topo-sort based legalization on G^m

Algorithm 1 outlines three-step legalization. Partitioning the layout into blocks based on constraint groups is the first step (lines 1–3). We are capable of dealing with multiple constraint types including symmetry constraint, matching constraint, and array constraint, and we take symmetry constraint as an example in this article. A symmetry axis A_i and a symmetry group P_i is necessary for each symmetry group. Symmetry group P_i is made up of a series of symmetry pairs (c_i, c_j), where c_i and c_j are cells (we broaden the meaning of "cell" to indicate a device or a subcircuit in hierarchical circuits if not specially mentioned). We are able to traverse the cells c_i in the particular group to get the boundaries in four directions xl, xr, yb, and yt (left, right, bottom, and top) (lines 1 and 2) according to the given symmetry groups $\{(A_i, P_i)\}$. We then partition the layout into grids by inserting virtual lines as well as merging the grids relevant to the same symmetry group into a single block (line 3). We will have more in-depth discussions in Section IV-C. We depict the topology between blocks as an MCG G^m (line 4). The MCG will be introduced in Section IV-B. The following step is to perform placement legalization for each block with a topological-sort-based method depicted in Section IV-D (lines 5 and 6). After all blocks are successfully legalized, we implement a similar but coarse-grained method to mixed graph G^m. Finally, we obtain a legalized layout L without overlaps.

B. Mixed Constraint Graph Representation

We deduce MCG from the constraint graph representation depicted in Section II-C. The MCG is made up of HCG and VCG. We are able to obtain the MCG $G^m = (V, E^h \cup E^v)$ according to the given a HCG $G^h = (V, E^h)$ and a VCG $G^v = (V, E^v)$. Fig. 3 illustrates exactly how we extract the MCG from a layout.

Note that G^m is a heterogeneous graph; namely, G^m includes two distinct types of edges: 1) horizontal constraint edges (HCEs) from G^h and 2) vertical constraint edges (VCEs) from G^v. Between MCG and the pair of HCG, VCG there exists a one-to-one correspondence. Since each legalized layout can be distinctively depicted as a combination of a horizontal graph and a vertical graph, we are able to represent the layout in a particular MCG.

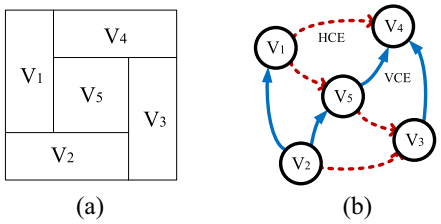


Fig. 3. MCG with HCEs and VCEs. (a) Example placement. (b) Corresponding MCG.

1) Topological Sort on MCG: The MCG not only maintains the information from the original constraint graph but also introduces a new property with topological sort.

The topological sort generates a linear ordering S of vertices in V for a directed graph $G = (V, E)$. If there is a directed edge $\langle u, v\rangle$, then vertex u appears before vertex v in ordering S. If we consider the edges as dependencies between vertices, then topological sort guarantees an order that no vertex is dependent to its subsequent vertices. For instance, the topological order of the MCG shown in Fig. 3(b) can be given as $S = \{v_2, v_1, v_5, v_3, v_4\}$. If we move a vertex v_i toward the up right direction, the movement does not affect any vertices before v_i in S. Due to this distinctive property, we can design an algorithm that only traverses every particular cell once and computes their legal positions without backtracking. The detail of the topological-sort-based algorithm is discussed in Section IV-D.

The proposed algorithm requires that there exists a topological ordering for the MCG. Since MCG is a DAG and it is known that any DAG has at least one topological ordering, Theorem 1 holds.

Theorem 1: Any MCG has at least one topological ordering.

Proof: If a directed graph has no circle, then for any two vertices v_ia and v_j of the graph, path from v_i to v_j and path from v_j to v_i will not exist at the same time. Hence, for any two vertices in a DAG, there always exists an order without contradiction. Therefore, we only need to prove that MCG contains no circle. Assume that there exists a circle $\{v_1, \ldots, v_t, v_1\}$ in an MCG. Consider the center location (x_{v_1}, y_{v_1}) of v_1. There must exist at least an HCE or a VCE in the circle, so we get either $x_{v_1} < x_{v_1}$ or $y_{v_1} < y_{v_1}$, which leads to a contradiction. ■

C. Layout Partitioning Technique

We partition the layout according to the symmetry constraint groups. In other words, we break the legalization problem into subproblems for each particular symmetry constraint group. Fig. 4 illustrates a layout partitioning example with only two symmetry groups.

Algorithm 2 demonstrates the layout partitioning procedure in detail and shows a slicing line scheme. As aforementioned in Section IV-A, the boundaries $(b^{xl}, b^{xr}, b^{yb}, b^{yt})$ are calculated and transferred to this step. First, we sort the four boundary arrays in ascending order (line 11). Then, we apply a slicing rule to x and y directions, respectively (line 12).

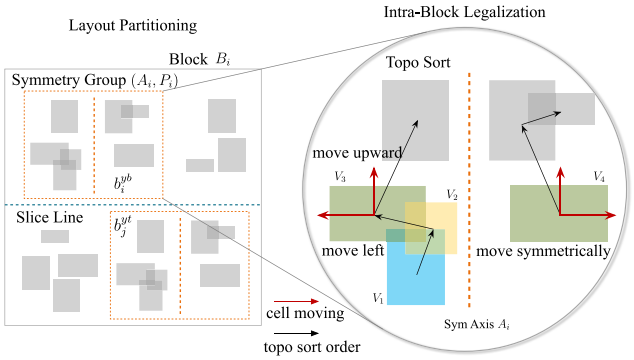


Fig. 4. Layout partitioning and intrablock legalization.

Algorithm 2 Layout Partitioning

Input: boundaries $b^{xl}, b^{xr}, b^{yb}, b^{yt}$ of symmetry groups
Output: partitioned layout blocks $\{B_i\}$

1: **function** ADDSLICELINE(b_1, b_2)
2: $i \leftarrow 0, j \leftarrow 0$
3: **while** $i \neq m$ **do**
4: **while** $b_1[j] < b_2[i]$ **do**
5: $j \leftarrow j + 1$
6: **end while**
7: **if** $b_1[j] < b_2[i+1]$ **then**
8: `AddLine`$(b_1[j] + b_2[i])/2$
9: **end if**
10: $i \leftarrow i + 1$
11: **end while**
12: **end function**
13: **function** `PartitionLayout`($b^{xl}, b^{xr}, b^{yb}, b^{yt}$)
14: `Sort`b^{xl}, `Sort`b^{xr}, `Sort`b^{yb}, `Sort`b^{yt}
15: `AddSliceLine`b^{xl}, b^{xr}, `AddSliceLine`b^{yb}, b^{yt}
16: traverse slice lines and get slices grids
17: construct blocks $\{B_i\}$ from sliced grids
18: **end function**

Take x-direction as an example. If $b^{xr}[i]$ is less than $b^{xl}[j]$ and there is no other boundary value between $b^{xr}[i]$ and $b^{xl}[j]$, we add a vertical slicing line in the middle of $b^{xr}[i]$ and $b^{xl}[j]$ (lines 1–8). In the example of Fig. 4, as the bottom boundary b_i^{yb} of symmetry group (A_i, P_j) follows the top boundary b_j^{yt} closely, i.e., no other boundaries between them, we add a horizontal slice line between b_j^{yt} and b_i^{yb}.

With the slicing lines inserted, the layout is divided into virtual grids. The grids relative to the same symmetry group are clustered into one block. The grids independent of symmetry groups are directly constructed as blocks. As depicted in Fig. 4, in the block B_i with symmetry group (A_i, P_i), there are several cells not belonging to P_i. For constraint groups other than the symmetry constraint mentioned here, the divide and conquer methodology is still valid.

D. Topological-Sort-Based Legalization

We propose an efficient topological-sort-based legalization strategy. Algorithm 3 sketches the procedure of the symmetry-aware legalization leveraging a two-fold topological sort. As

Algorithm 3 Topological-Sort-Based Legalization

Input: MCGs G_l^m, G_r^m
Output: Legalized block
1: **function** `TopSortLegalizer`(G_l^m, G_r^m)
2: $Q_l \leftarrow$ empty queue, $Q_r \leftarrow$ empty queue
3: in-degree in_l, in_r
4: push v with 0 in_l (or in_r) into Q_l (or Q_r)
5: **while** $Q_l \neq \varnothing$ **do**
6: $v_l \leftarrow Q_l.pop\_front$
7: update location of v_l
8: **for** each v adjacent to v_l **do**
9: $in_l[v] \leftarrow in_l[v] - 1$
10: **if** $in_l[v] = 0$ **then**
11: $Q_l.push(v)$
12: **end if**
13: **end for**
14: **if** v_l in a symmetry pair (v_l, v_r) **then**
15: **while** $u_r \neq v_r$ **do**
16: $u_r \leftarrow Q_r.pop\_front$
17: update location of u_r
18: **for** each u adjacent to u_r **do**
19: $in_r[u] \leftarrow in_r[u] - 1$
20: **if** $in_r[u] = 0$ **then**
21: $Q_r.push(u)$
22: **end if**
23: **end for**
24: **end while**
25: update location of v_l, v_r symmetrically
26: **end if**
27: **end while**
28: **while** $Q_r \neq \varnothing$ **do**
29: $v_r \leftarrow Q_r.pop\_front$
30: update location of v_r
31: **for** each v adjacent to v_r **do**
32: $in_r[v] \leftarrow in_r[v] - 1$
33: **if** $in_r[v] = 0$ **then**
34: $Q_r.push(v)$
35: **end if**
36: **end for**
37: **end while**
38: **end function**

Fig. 4 illustrates, for an MCG G_l^m of a block with a symmetry axis, this axis separates the block from the middle in geometry and at the same time split the MCG into two subgraphs (the subgraph left to the axis and the other subgraph right to the axis for such a vertical symmetry axis). We can regard the legalization as a process of dispersing the devices outward from the symmetry axis and, meanwhile eliminating the overlaps. Considering that an HCE of MCG indicates a geometry constraint in the positive direction of the x-axis, we want the constraint edge to denote the direction away from the symmetry axis. Therefore, we mirror the subgraph of MCG left to the symmetry axis; i.e., we reverse the direction of HCEs (every HCE switches its head and tail). Let the mirrored left subgraph of MCG be G_l^m and the subgraph right to the symmetry axis be G_r^m. We maintain a queue of vertices that have zero in-degrees for each MCG. At each step, we take one vertex v_l out of the queue Q_l (line 6) and check all the incoming edges $\{\langle u, v_l\rangle\}$ for vertex v_l. A horizontal edge $\langle u, v_l\rangle$ represents that v_l is left to u and a vertical edge represents that v_l is above u. We then move v_l to $\min\{x_u - (w_{v_l} + w_u)/2\}$ in the x direction, ensuring v_l satisfies the horizontal constraints $\{\langle u, v_l\rangle\}$. We only move v_l to the left and move v_l upward for overlap removal. For V_3 in the example of Fig. 4, we move it to the left and upward to remove the overlaps with V_1 and V_2.

When the traversal reaches v_l that is in a symmetry pair (v_l, v_r), the legalizer performs a similar traversal on G_r^m to reach v_r (lines 13–19). We set symmetric and legal locations for v_l and v_r, meaning that the distance to symmetry axis is $\max\{|x_{axis} - x_{v_l}|, |x_{v_r} - x_{axis}|\}$ and the y coordinate is $\max\{y_{v_l}, y_{v_r}\}$ (line 20). We also move V_4 to a symmetrical location with V_3 in Fig. 4. After legalizing some vertices v, we decrease the in-degrees of every vertex u with directed edge $\langle v, u\rangle$ and push the vertex to the corresponding queue if its in-degree goes to zero. The algorithm continues until the two queues Q_l and Q_r become empty.

Assume that we have the topological orders S_l for G_l^m and S_r for G_r^m. S_l and S_r preserve the order for symmetry groups, as claimed in Theorem 2. This property guarantees that the traversals of two MCGs can be synchronized without skipping any vertices of another MCG.

Theorem 2: For two symmetry pairs (u_i, v_i) and (u_j, v_j), if u_i appears before u_j in a topological sort of G_l^m, then v_i appears before v_j in any topological sort of G_r^m.

Proof: Assume that there are two symmetry pairs (u_i, v_i) and (u_j, v_j) that u_i appears before u_j but v_j appears before v_i in topological sort of G_l^m and G_r^m. As u_i appears before u_j in topological sort of G_l^m, u_j is left and up to u_i. Similarly, v_i is right and up to v_j. As u_j and u_i have the same y-coordinate, v_i cannot be up to v_j. Consider the x-coordinate x_{sym} of the symmetry axis. u_i and v_i is symmetric about the symmetry axis, which indicates $x_{u_i} + x_{v_i} = 2x_{\text{sym}}$. We have $x_{u_i} + x_{v_i} = 2x_{\text{sym}}$ and $x_{u_j} + x_{v_j} = 2x_{\text{sym}}$, so v_i cannot be right to v_j. The contradiction means that the topological sort preserves the order of symmetry pairs in G^m. ■

E. Time Complexity Analysis

We now analyze the time complexity of the legalization algorithm. We show that the time complexity of Algorithm 1 is $\mathcal{O}(n)$, where n is the number of devices of the analog circuits. First, the layout partitioning stage is $\mathcal{O}(n)$. Suppose we have k symmetry groups. The slicing lines divide the layout to $\mathcal{O}(k^2)$ nonempty grids. As the number of grids is always much smaller than the number of devices n, $\mathcal{O}(k^2)$ is bounded by $\mathcal{O}(n)$. The sorting process is $\mathcal{O}(k \log k)$ (line 11 of Algorithm 2). Adding slicing lines only scans the boundaries once (line 12), which is $\mathcal{O}(k)$. Building blocks and MCG requires a traversal of all grids (lines 13 and 14), which is $\mathcal{O}(k^2)$. Thus, the first part (lines 1–4 of Algorithm 1) is $\mathcal{O}(n)$. The topological-sort-based legalization is $\mathcal{O}(n_B)$, where n_B is the number of devices in block B. The time complexity of topological sort on graph $G = (V, E)$ is $\mathcal{O}(|V| +$

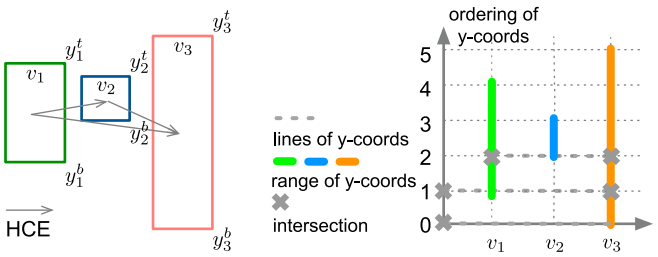


Fig. 5. Example for Lemma 1.

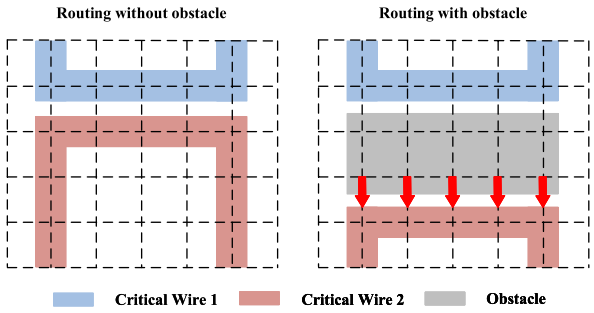


Fig. 6. Routing obstacle insertion.

$|E|$). To demonstrate the linear time complexity, we have a nice property shown in Lemma 1 for MCG, and eventually, Theorem 3 summarizes the overall time complexity.

Lemma 1: For an MCG $G^m = (V, E)$, $|E|$ is $\mathcal{O}(|V|)$.

Proof: We give the proof in the case of the number of HCEs. If there is a horizontal edge from vertex u to v, then the corresponding device v is right to device u directly. To prove that $|E|$ is $\mathcal{O}(|V|)$, we sort the bottom and top y-coordinates of all devices and we get a sequence of $2|V|$ coordinates. Fig. 5 gives an example. The left figure of Fig. 5 is the layout example of three devices. To illustrate that the number of HCEs is bounded by the number of devices, we show a special property of the ordering of y-coordinates in the right figure of Fig. 5. We sort the y-coordinates (abbreviated to y-coords) of the three vertices $\{v_1, v_2, v_3\}$, and we can represent their y-coordinates with $\{0 : y_3^b, 1 : y_1^b, 2 : y_2^b, 3 : y_2^t, 4 : y_1^t, 5 : y_3^t\}$. The three devices can be represented as the corresponding line segments in the right figure. For example, the line segment of device v_1 starts with 1 and ends with 4, which means the bottom y_1^b of device v_1 ranks 1 of the y-coords. The MCG of this example has three HCEs: 1) (v_1, v_2); 2) (v_1, v_3); and 3) (v_2, v_3). Taking device v_2 for instance, there exists an HCE (v_1, v_2) because y_2^b is between y_1^b and y_1^t. Draw a horizontal line crossing the bottom point of each line segment, and we get intersections with at most two other line segments. The number of HCEs related to device v_2 equals the number of the intersections between the dotted line ($y = y_2^b$) and other vertical line segments (v_1 and v_3). As dotted lines of a specific device and intersections portray which device has direct left-to- or right-to- relationship with this device, the number of total horizontal edges equals the number of all those intersections. Therefore, $|E|$ is $\mathcal{O}(|V|)$ for an MCG. ■

Theorem 3: The proposed legalization process runs in $\mathcal{O}(n)$.

Proof: Each stage of the legalization process runs in $\mathcal{O}(n)$. As mentioned above, the layout partitioning is $\mathcal{O}(n)$. Legalization inside each block requires topological sort on the corresponding MCG. Indicated by Lemma 1, the time complexity of legalization is $\sum_{G^m=(V,E)} |E|$, which is linear to the number of vertices. The whole process of legalization runs in $\mathcal{O}(n)$. ■

V. Interactive Routing

Due to a variety of sophisticated constraints, such as end-of-line spacing, parallel run length spacing, manual routing is tedious and time consuming. Although automated routing tools can generate basic routing solutions, it is difficult to integrate all the expertise from designers, such as extra space for critical wires, routing obstacles, and preferred routing topologies, which can highly impact the eventual performance. In this section, we explain how to realize the routing operations defined in Section III-D. For operation `add_route_obstacle`, we propose a routing obstacle insertion algorithm to reserve enough space for critical wires and sensitive devices. For operation `route_with_guide`, we propose a routing algorithm that can follow the topology guidance provided by designers. Note that the proposed interactive router is based on the MAGICAL router which could handle multiple routing layers in practice. The proposed interactive router, similar to the MAGICAL router, also takes several concise DRC rules into account aiming to ensure the routing legalization during the layout editing process.

A. Routing Obstacle Insertion

Automatic layout routing tools hardly consider special spacing constraints for critical paths and devices, which leads to a decrease in the performance of analog circuits. In order to adjust the spacing of critical paths and devices, we propose a concise method which is defined in Section III-D as `add_route_obstacle`. Fig. 6 demonstrates the obstacle insertion process. The initial routing result is shown on the left of Fig. 6. Spacing between two critical wires in the same routing layer is particularly narrow, which may increase parasitic capacitance and affect signal transmission. The adjustment procedure outlines three steps. First, select one of the critical wires as a reference and remove another wire. Then, determine the boundaries of the obstacle in the same routing layer according to the desired spacing constraints. Finally, reroute the removed wire such that enough spacing can be maintained by avoiding the obstacle.

B. Routing With Guidance

In more practical scenarios, users may desire to be capable of controlling the routing topology. Given a net and a rough routing topology, we propose an effective algorithm to generate routing solutions following the topology guidance. Algorithm 4 details our method. Function `route_with_guide` demonstrates the top flow of our algorithm modified from the conventional A-star-based routing procedure. We adjust the cost of each node according to

Algorithm 4 Routing With Guidance

Input: Guiding path $\{GP_i\}$, routing net n_i with source node v_{src} and target node v_{tgt}, cost-scaled parameter α
Output: Routing solution considering routing guidance

1: **function** `route_with_guide`(n_i, $\{GP_i\}$, α)
2: Priority queue $Q \leftarrow \varnothing$
3: Initialize all costs to ∞
4: $Q.push(v_{src})$
5: **while** $Q \neq \varnothing$ **do**
6: $v \leftarrow Q.pop()$
7: **if** $v = v_{tgt}$ **then**
8: trace_back(v)
9: **break**
10: **end if**
11: **for** $u \in$ neighbors of v **do**
12: $cost \leftarrow$ A-Star_routing_cost(u)
13: $+ \alpha \times$ `Distance`$u.coord, \{GP_i\}$
14: **if** $cost < u.cost$ **then**
15: $u.cost \leftarrow cost$
16: **if** $u \notin Q$ **then**
17: $Q.push(u)$
18: **end if**
19: **end if**
20: **end for**
21: **end while**
22: **end function**
23: **function** `Distance`(p, $\{GP_i\}$)
24: $d_{min} \leftarrow \infty$
25: **for** $GP_i \in \{GP_i\}$ **do**
26: $d \leftarrow$ `Point_Path_Distance`p, GP_i
27: **if** $d < d_{min}$ **then**
28: $d_{min} \leftarrow d$
29: **end if**
30: **end for**
31: **end function**

$p - \{GP_i\}$ (lines 12 and 13). Take this cost as penalty in the A-star-based routing algorithm. The higher the cost, the less likely the node is to be selected as a routing path. Furthermore, we also set a cost-scaled parameter α to adjust the tightness of the routing path following the routing guidance. Empirically, α is set to 5 in the experiments.

To illustrate the guidance cost, we define two specific distances for convenience.

Definition 1: $p - GP_i$ represents the Manhattan distance from an objective node to a particular segment of the guiding path.

Definition 2: $p - \{GP_i\}$ represents the Manhattan distance from an objective node to the entire guiding path.

More specifically, $p - GP_i$ corresponds to the dotted line in three different colors in Fig. 7. $\{GP_i\}$ is made up of several end-to-end GP_i, so we take the minimum of $p - GP_i$ as the final $p - \{GP_i\}$ (lines 23–31). As Fig. 7 demonstrated, the final $p - \{GP_i\}$ is Distance ② in this case and the ideal wire is routed following the routing guidance precisely without passing through obstacles.

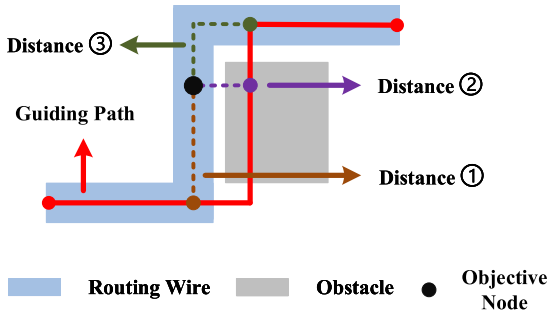


Fig. 7. Example of routing with guidance, where the distance between point p and the guiding path is the minimum of distance ①, ②, and ③.

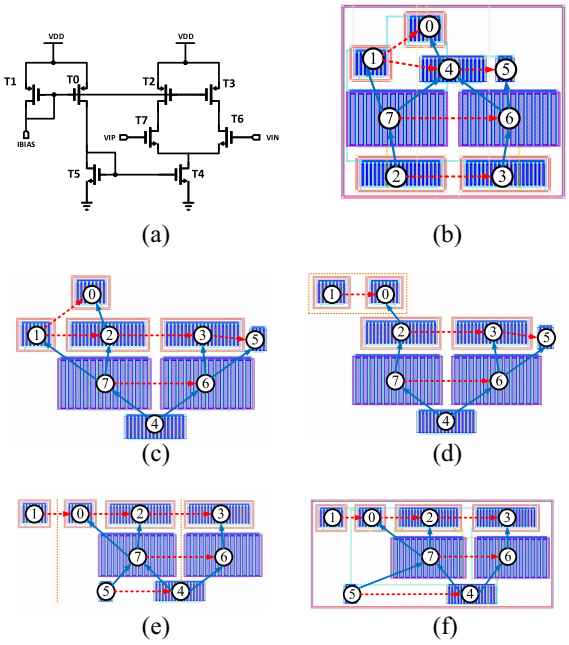


Fig. 8. Example of layout editing process with command sequence: {`swap`([4], [2, 3]), `symAdd`([1, 0], A_2), `move`([1, 0]), `move`([5]), `symmAdd`([5], A_2)}. (a) Schematic. (b) Initial placement. (c) After `swap`. (d) After `symAdd`. (e) After `move`. (f) Legalized placement.

VI. Experimental Results

In this section, we first show an example of the interactive layout editing process. Then, we verify the performance of instant placement legalization on a synthetic dataset. Eventually, we validate the post-layout performance of the entire interactive layout editing workflow, including placement and routing on real designs.

A. Demonstration of Interactive Editing

We first demonstrate the capability of interactive layout editing with a simple example, as shown in Fig. 8, where an automation-tool-generated layout is converted to the desired layout. We adopt the layout automation tool MAGICAL [3] to generate the first placement layout, as shown in Fig. 8(b). It is worth noting that our framework cares less about where the initial layout comes from. As our interactive placement and routing are independent with the automation engine, the initial layout can be given by other automation tools or even

TABLE III
LEGALIZATION PERFORMANCE

	STATISTICS			TOPO			LP		
	Devices	Symmetry	Hierarchy	T_{seq} ms	D_{max} %	D_{avg} %	T_{seq} ms	D_{max} %	D_{avg} %
case1	25	✓	-	0.16	31.1	17.7	33.3	40.9	13.0
case2	49	✓	-	0.23	43.4	5.1	33.4	43.4	5.1
case3	42	✓	-	0.20	10.9	3.99	35.4	16.2	3.3
case4	17	✓	-	0.12	11.8	4.20	31.5	12.6	3.9
case5	114	✓	✓	0.84	14.1	10.1	101.7	16.4	8.0
case6	211	✓	✓	1.06	21.7	11.1	253.7	19.6	6.1
Avg.	-	-	-	-	22.2	8.7	-	24.9	6.6
Ratio	-	-	-	1.00	-	-	192.2	-	-

another manual result. Assume a designer receives the first placement layout depicted in Fig. 8, but is unsatisfied with it. We enable an interactive process in which the designer is able to run the commands in Table I and can instantly see the legalized layout. In step 1, we conduct command `swap` between devices 4 and {2, 3}, which denotes to swap v_4 and $\{v_2, v_3\}$ on the constraint graph. The designer is able to view the layout in Fig. 8(c) after instant legalization. In step 2, we conduct command `symAdd` to add a new symmetry pair $\{v_1, v_0\}$ and Fig. 8(e) shows the symmetry constraint. Then, we conduct several `move` commands to reassign the locations of certain devices. In the last step, we conduct command `symAdd` to add device 5 to the left symmetry group and achieve the eventual placement layout.

B. Legalization Performance

We then set up experiments to verify the performance of the instance legalization in the feedback phase. The algorithms are implemented in C++ and Python and the experiments are conducted on a Linux server with 20-core Xeon CPU @ 2.1 GHz.

We construct a dataset from circuits in the opensource MAGICAL repository [3] to validate the legalization algorithm. The STATISTICS column in Table III lists the properties of the test cases. All cases contain symmetry group constraints and two cases with more devices have hierarchical structures. We prepare the cases by perturbing the initial layouts from MAGICAL [3]. Then, we generate a random sequence of placement commands in Table I to simulate multiple synthetic command streams. We obtain ten feasible command streams for every initial placement layout to construct ten prelegalization synthetic layouts. In this way, we synthesize a dataset with 60 prelegalization placement layouts to verify the legalization algorithm.

In addition, we implement the legalization algorithm in MAGICAL based on linear programming (LP) by using GUROBI as the LP solver. The algorithm aims at minimizing the displacement. The performance of different legalizers is compared in Table III. We use two displacement metrics and a runtime metric to assess the legalizers, where T_{seq} denotes the runtime of one-shot legalization after applying a complete command stream, and D_{max} and D_{avg} represent the maximum and average cell displacements in a layout, respectively. Both D metrics are evaluated as the ratio (in percentage) between the Manhattan distance and half-perimeter of the layout bounding box. We take the maximum values over ten layout samples for the three metrics. Our legalizer can generate legalized layouts with nearly optimal displacement, and the layout solutions are aligned with designers' expectations. Table III demonstrates that the legalization can be finished within about 1ms for each case, indicating the capability of real-time interaction with designers.

TABLE IV
POST-LAYOUT PERFORMANCE OF LDO

Method	Gain (dB)	CURRENT (uA)	OD (mV)	OU (mV)	PM (degree)
`Pre-Simu`	73.69	16.82	539.6	540.2	89.59
`MAGICAL`	73.06	16.18	1937.0	1422.0	89.61
`Interactive`	**73.65**	**16.19**	**544.8**	**541.5**	89.61

C. Post-Layout Simulation Results

To verify the circuit performance after interactive layout editing, we establish interactive processes and conduct post-layout simulation on two real-world analog designs, i.e., an operational transconductance amplifier (OTA) and a low-dropout regulator (LDO), under TSMC 40-nm technology node. In our interactive layout editing processes, we first generate initial placement and routing layout results using MAGICAL [3]. Then, we present the initial layout results to an experienced analog designer and collect command sequences to conduct editing directly on the layout results. We employ Cadence Virtuoso and Mentor Graphics Calibre for post-layout simulation. We compare the post-layout performance results between initial layouts and interactively edited layouts.

Fig. 9(a) shows the schematic of LDO circuit. We generate the initial layout with MAGICAL [3], shown in Fig. 9(b). We obtain the command sequence {`wireSpacing(`①, ②), `wireSpacing(`③, ④)} from an experienced analog designer. Layout editing following the command sequence produces the layout shown in Fig. 9(c). Table IV shows the post-layout simulation performance of the LDO circuit. `MAGICAL` represents the initial layout generated by MAGICAL [3], and `Interactive` represents the layout interactively edited with our framework. Table IV summarizes the post-layout performance with close-loop gain (Gain), dc power current

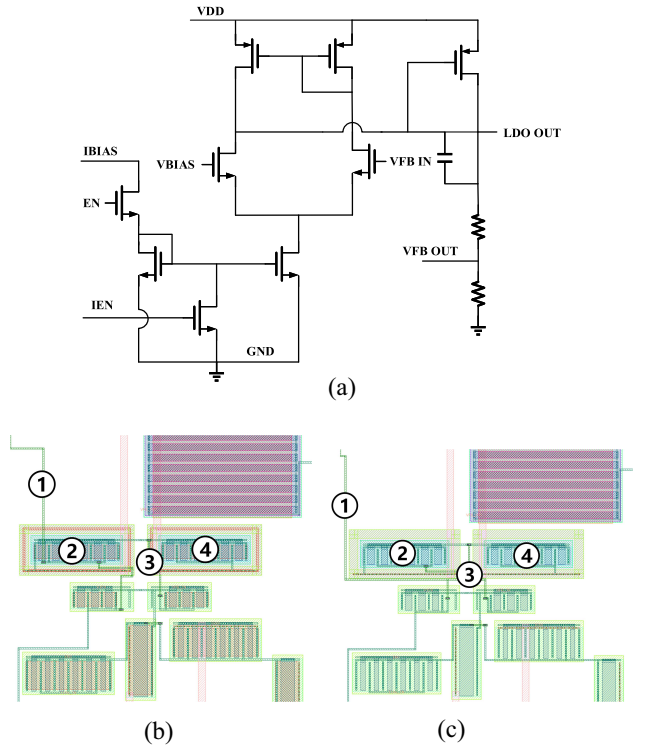


Fig. 9. LDO: (a) schematic, (b) part of layout generated by MAGICAL [3], and (c) corresponding part of layout interactively edited with {wireSpacing(①, ②), wireSpacing(③, ④)}.

TABLE V
POST-LAYOUT PERFORMANCE OF OTA

Method	Gain (dB)	UGB (MHz)	CMRR (ratio)	PM (degree)
Pre-simu	15.40	6.85	-	70.98
MAGICAL	14.15	5.11	55.79	70.48
Interactive	**14.49**	**5.24**	**74.47**	69.23

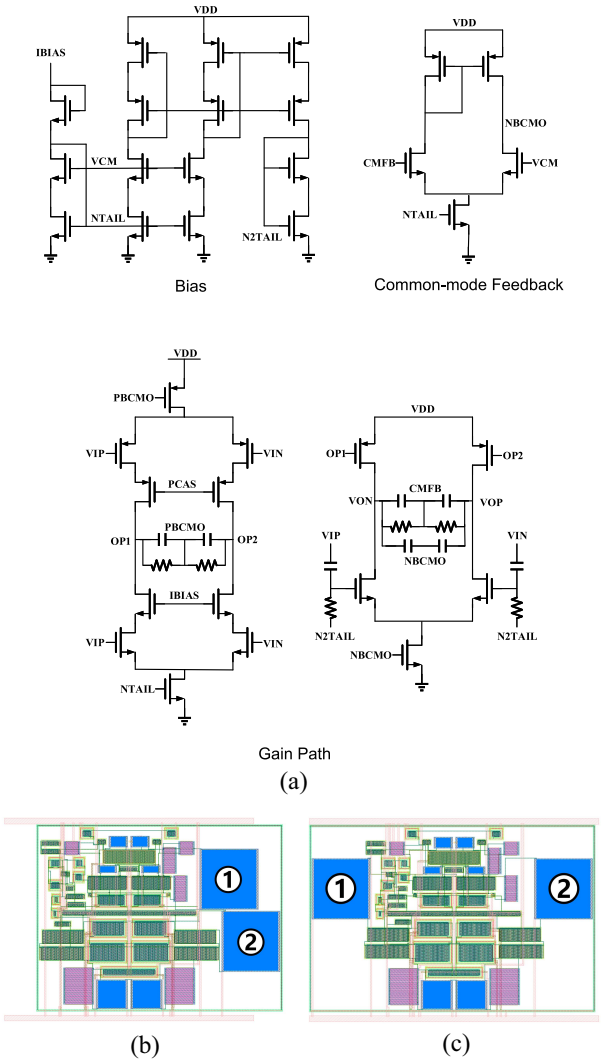


Fig. 10. OTA: (a) schematic, (b) layout generated by MAGICAL [3], and (c) layout interactively edited with {symAdd(①, ②)}.

(CURRENT), overshot down voltage (OD), overshot up voltage (OU), and phase margin (PM). We can achieve better results in Gain, CURRENT, OD, and OU, with the same PM, compared with the initial layout from MAGICAL. With only two editing commands, the interactive editing brings about a noteworthy improvement on OD and OU. The total time for the two editing commands is about 1 s, basically achieving the purpose of real-time layout editing.

Fig. 10 presents the schematic of an OTA circuit, the layout generated by MAGICAL [3], and the layout interactively edited with our framework. We obtain the command sequence symAdd(①, ②) from an experienced analog designer, perform on the initial layout [Fig. 10(b)], and obtain the edited layout [Fig. 10(c)]. We finish the layout editing process in 0.1 s and rerun the routing process with 5.5 s. Considering that manual layout drawing may take hours to days, the framework is very promising to reduce the turn-around time. We compare the post-layout performance of the initial layout (MAGICAL) and the edited layout (Interactive) in Table V. The performance metrics listed in Table V include close-loop gain (Gain), unity gain bandwidth (UGB), common-mode rejection ratio (CMRR), and PM. The interactively edited layout achieves significant improvements in Gain, UGB, and CMRR, meanwhile satisfying the PM requirement. The simulation performance demonstrates that our layout editing framework can effectively enhance the existing fully automated layout generator with compact command streams.

D. Application Scope of the Framework

Our interactive framework can effectively reduce the turn-around time for circuit designs with heavy manual efforts. As analog layouts are quite custom and the design methodology is mostly manual, we choose analog layouts as a target application. The framework may also be applicable to other custom designs like the memory cell circuits and the customized digital parts in mixed-signal circuits, which can be explored in the future.

VII. Conclusion

We presented an analog placement and routing framework with interactive analog layout editing in this study. To bridge the gap between fully automated analog layout tools and skilled designers, we proposed the novel idea of interactive analog layout editing. We proposed a complete workflow for interactive placement and routing modification with a novel placement representation leveraging MCG and linear-time topological-sort-based legalization as well as flexible routing algorithms. This is a pioneering work that provides a holistic study on the interactive design methodology for analog layouts and its capability of speeding up design closure. We validated the effectiveness and efficiency of the proposed framework on real-world analog designs with post-layout simulation. Our framework achieves quick response to users' commands, with most commands not exceeding 1 s. Fast response ensures a real-time interaction with users and, thus, our framework can speed up the whole design closure. The experiments demonstrated that we can improve post-layout performance with only a few simple commands on real-world analog designs. In the future, we plan to performance and layout dependent effects during the legalization, and extend the framework to other custom designs like memory cell circuits, customized digital parts in mixed-signal circuits, and printed circuit boards (PCBs) designs.

References

[1] K. Kunal *et al.*, "ALIGN: Open-source analog layout automation from the ground up," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–4.

[2] T. Dhar *et al.*, "ALIGN: A system for automating analog layout," *IEEE Design Test*, vol. 38, no. 2, pp. 8–18, Apr. 2021.

[3] B. Xu *et al.*, "MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[4] H. Chen *et al.*, "MAGICAL: An open- source fully automated analog IC layout system from Netlist to GDSII," *IEEE Design Test*, vol. 38, no. 2, pp. 19–26, Apr. 2021.

[5] H. Chen *et al.*, "MAGICAL 1.0: An open-source fully-automated AMS layout synthesis framework verified with a 40-nm 1GS/s $\Delta\Sigma$ ADC," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2021, pp. 1–2.

[6] K. Kunal *et al.*, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. IEEE/ACM Proc. Design Autom. Test Eurpoe (DATE)*, 2020, pp. 55–60.

[7] K. Kunal, J. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. S. Sapatnekar, "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–8.

[8] M. Liu *et al.*, "Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[9] K. Zhu, H. Chen, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Effective analog/mixed-signal circuit placement considering system signal flow," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[10] J. Rijmenants, J. B. Litsios, T. R. Schwarz, and M. G. R. Degrauwe, "ILAC: An automated layout tool for analog CMOS circuits," *IEEE J. Solid-State Circuits*, vol. 24, no. 2, pp. 417–425, Apr. 1989.

[11] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.

[12] R. Martins, N. Lourenco, and N. Horta, "LAYGEN II—Automatic layout generation of analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 11, pp. 1641–1654, Nov. 2013.

[13] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. IEEE/ACM Asia South Pacific Design Autom. Conf. (ASPDAC)*, 2016, pp. 617–622.

[14] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2000, pp. 458–463.

[15] P.-Y. Chou, H.-C. Ou, and Y.-W. Chang, "Heterogeneous B*-trees for analog placement with symmetry and regularity considerations," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2011, pp. 512–516.

[16] Y. Pang, F. Balasa, K. Lampaert, and C.-K. Cheng, "Block placement with symmetry constraints based on the O-tree non-slicing representation," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2000, pp. 464–467.

[17] P.-H. Lin, Y.-W. Chang, and S.-C. Lin, "Analog placement based on symmetry-island formulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 6, pp. 791–804, Jun. 2009.

[18] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with Preplaced blocks and density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, Jul. 2008.

[19] J. Lu *et al.*, "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM Trans. Design Autom. Electron. Syst.*, vol. 20, no. 2, p. 17, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2699873

[20] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and Routability validation in global placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1717–1730, Sep. 2019.

[21] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[22] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," *Proc. IEEE*, vol. 103, no. 11, pp. 1985–2003, Nov. 2015.

[23] Y. Li *et al.*, "Exploring a machine learning approach to performance driven analog IC placement," in *Proc. IEEE Annu. Symp. VLSI (ISVLSI)*, 2020, pp. 24–29.

[24] M. Liu *et al.*, "Towards decrypting the art of analog layout: Placement quality prediction via transfer learning," in *Proc. IEEE/ACM Design, Autom. Test Eurpoe (DATE)*, 2020, pp. 496–501.

[25] J. Crossley *et al.*, "BAG: A designer-oriented integrated framework for the development of AMS circuit generators," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2013, pp. 74–81.

[26] E. Chang *et al.*, "BAG2: A process-portable framework for generator-based AMS circuit design," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2018, pp. 1–8.

[27] C. Wulff and T. Ytterdal, "A compiled 9-bit 20-MS/s 3.5-fJ/conv.step SAR ADC in 28-nm FDSOI for Bluetooth low energy receivers," *IEEE J. Solid-State Circuits*, vol. 52, no. 7, pp. 1915–1926, Jul. 2017.

[28] B. Yao, H. Chen, C.-K. Cheng, and R. Graham, "Floorplan representations: Complexity and connections," *ACM Trans. Design Autom. Electron. Syst.*, vol. 8, no. 1, pp. 55–80, 2003.

[29] Q. Ma, L. Xiao, Y. Tam, and E. F. Y. Young, "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 1, pp. 85–95, Jan. 2011.

[30] M. P.-H. Lin, Y.-T. He, V. W.-H. Hsiao, R.-G. Chang, and S.-Y. Lee, "Common-centroid capacitor layout generation considering device matching and parasitic minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 7, pp. 991–1002, Jul. 2013.

[31] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu, and Y.-W. Chang, "Layout-dependent effects-aware analytical analog placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1243–1254, Aug. 2016.

[32] L. Xiao, E. F. Y. Young, X. He, and K. P. Pun, "Practical placement and routing techniques for analog circuit designs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2010, pp. 675–679.

[33] P.-C. Pan, H.-M. Chen, Y.-K. Cheng, J. Liu, and W.-Y. Hu, "Configurable analog routing methodology via technology and design constraint unification," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2012, pp. 620–626.
[34] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 8, pp. 923–942, Aug. 1996.
[35] A. Patyal, P.-C. Pan, K. A. Asha, H.-M. Chen, and W.-Z. Chen, "Exploring multiple analog placements with partial-monotonic current paths and symmetry constraints using PCP-SP," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5056–5068, Dec. 2020.
[36] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang, "Nonuniform multilevel analog routing with matching constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1942–1954, Dec. 2014.
[37] H. Y. Chi, C. N. J. Liu, and H. M. Chen, "Wire load oriented analog routing with matching constraints," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 6, p. 55, Aug. 2020.
[38] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Toward silicon-proven detailed routing for analog and mixed-signal circuits," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–8.

Xiaohan Gao received the B.S. degree in computer science from Peking University, Beijing, China, in 2020, where she is currently pursuing the Ph.D. degree, advised by Prof. Y. Lin, with the School of Computer Science.

Her research interests include layout design automation for analog and mixed-signal circuits, incorporating machine learning techniques with physical design and design manufacturability, and abstracting description from layout designs.

Haoyi Zhang received the B.S. degree in microelectronics from Beihang University, Beijing, China, in 2022. He is currently pursuing the Ph.D. degree in microelectronics with Peking University, Beijing.

His research interests include analog design automation and mixed-signal circuit design.

Mingjie Liu (Member, IEEE) received the B.S. degree from Peking University, Beijing, China, in 2016, and the M.S. degree from the University of Michigan, Ann Arbor, MI, USA, in 2018. He is currently pursuing the Ph.D. degree in electrical and computer engineering with The University of Texas at Austin, Austin, TX, USA.

His current research interests include applied machine learning for design automation and physical design automation for analog and mixed-signal-integrated circuits.

Linxiao Shen (Member, IEEE) received the B.S. degree from the School of Microelectronics, Fudan University, Shanghai, China, in 2014, and the Ph.D. degree from the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA, in 2019.

He is currently an Assistant Professor with the School of Integrated Circuits, Peking University, Beijing, China. He was a Design Engineer with Silicon Laboratories, Austin, where he was involved in analog circuits design, mainly focusing on relaxation oscillators and high-speed synchronizer. His current research interests include analog mixed-signal and RF integrated circuits, intelligent sensor interfaces, imaging systems, and analog circuit design automation.

Dr. Shen was the recipient of the IEEE Solid State Circuits Society Predoctoral Achievement Award in 2019, the UT Austin Cockrell School of Engineering Fellowship in 2019, the Samsung Fellowship in 2011, and the National Scholarship in 2012. He serves for the Technical Program Committee of ACM/IEEE Design Automation Conference. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS. He serves as a Reviewer for numerous IEEE journals, including the IEEE JOURNAL OF SOLID-STATE CIRCUITS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM—PART II: EXPRESS BRIEFS, and IEEE SENSORS JOURNAL.

David Z. Pan (Fellow, IEEE) received the B.S. degree from Peking University, Beijing, China, in 1992, and the M.S. and Ph.D. degrees from The University of California at Los Angeles, Los Angeles, CA, USA, in 1998 and 2000, respectively.

He was a Research Staff Member with IBM T. J. Watson Research Center, Cambridge, MA, USA, from 2000 to 2003. He is currently an Engineering Foundation Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. He has published over 340 technical papers, and holds eight U.S. patents. He has graduated 25 Ph.D. students who are now holding key academic and industry positions. His current research interests include cross-layer nanometer IC design for manufacturability, reliability, security, physical design, analog design automation, and CAD for emerging technologies, such as 3-D-IC and nanophotonics.

Dr. Pan was a recipient of numerous awards for his research contributions, including the SRC 2013 Technical Excellence Award, the DAC Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASP-DAC Frequently Cited Author Award, 16 best paper awards at premier venues, including GLSVLSI 2018, VLSI Integration 2018, HOST 2017, SPIE 2016, ISPD 2014, ICCAD 2013, ASPDAC 2012, ISPD 2011, the IBM Research 2010 Pat Goldberg Memorial Best Paper Award, ASPDAC 2010, DATE 2009, ICICDT 2009, and SRC Techcon in 1998, 2007, 2012, and 2015, 14 additional best paper award nominations at DAC/ICCAD/ASPDAC/ISPD, the Communications of the ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award (thrice), the IBM Faculty Award (four times), the UCLA Engineering Distinguished Young Alumnus Award in 2009, the UT Austin RAISE Faculty Excellence Award in 2014, and several international CAD contest awards. He has served as a Senior Associate Editor for *ACM Transactions on Design Automation of Electronic Systems* and an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS, IEEE DESIGN & TEST, *Science China Information Sciences*, *Journal of Computer Science and Technology*, and *IEEE CAS Society Newsletter*. He has served for the Executive and Program Committees of several major conferences, including DAC, ICCAD, ASPDAC, and ISPD. He is the ASPDAC 2017 Program Chair, the ICCAD 2018 Program Chair, the DAC 2014 Tutorial Chair, and the ISPD 2008 General Chair. He is a Fellow of SPIE.

Yibo Lin (Member, IEEE) received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 2018 advised by Prof. David Z. Pan.

He worked as a Postdoctoral Researcher with the University of Texas at Austin from 2018 to 2019. He is currently an Assistant Professor with the School of Integrated Circuits, Peking University, Beijing, China. His research interests include physical design, machine learning applications, and heterogeneous computing in VLSI CAD.

Dr. Lin is a recipient of the Best Paper Awards at premier EDA/CAD journals/conferences, such as TCAD, DAC, DATE, and ISPD.

Runsheng Wang (Member, IEEE) received the B.S. and Ph.D. (Highest Hons.) degrees from Peking University, Beijing, China, in 2005 and 2010, respectively.

He was a Visiting Scholar with Purdue University, West Lafayette, IN, USA, from November 2008 to August 2009. He joined Peking University in 2010, where he is currently a Professor with the School of Integrated Circuits and is serving as the Associate Dean of the School of EECS. He has authored/coauthored one book, four book chapters, and about 200 scientific papers, including more than 40 papers published in the International Electron Devices Meeting (IEDM) and Symposium on VLSI Technology. He has been granted 19 U.S. patents and 38 Chinese patents. His current research interests include nanoscale CMOS devices and reliability, design automation, and new-paradigm computing.

Dr. Wang was awarded the IEEE EDS Early Career Award by the IEEE Electron Device Society, the National Distinguished Young Scholars by the National Natural Science Foundation of China, the Natural Science Award (First Prize) by the Ministry of Education of China, and many other awards. He serves for the editorial board of the IEEE TRANSACTIONS ON ELECTRON DEVICES and *SCIENCE CHINA: Information Sciences*, and has served for the Technical Program Committee of many IEEE conferences, including IEDM and IRPS.

Ru Huang (Fellow, IEEE) received the B.S. (Hons.) and M.S. degrees in electronic engineering from Southeast University, Nanjing, China, in 1991 and 1994, respectively, and the Ph.D. degree in microelectronics from Peking University, Beijing, China, in 1997.

Since 1997, she has been a Faculty Member with Peking University, where she is currently a Boya Chair Professor. She has authored or coauthored five books, five book chapters, and more than 300 papers, including more than 100 papers in IEDM (46 IEDM papers from 2007 to 2021), VLSI Technology Symposium, IEEE EDL, and IEEE T-ED. She has been granted over 300 patents, including 49 U.S. patents. Her research interests include nano-scaled CMOS devices, ultra-low-power new devices, new device for neuromorphic computing, emerging memory technology, and device variability/reliability.

Dr. Huang is an Elected Academician of the Chinese Academy of Science and an Elected Member of TWAS Fellow. She has delivered over 50 keynote/invited talks at conferences and seminars.