Interactive Analog Layout Editing with Instant Placement Legalization

Xiaohan Gao
CECA, CS Department
Peking University
Beijing, China
xiaohangao@pku.edu.cn

Mingjie Liu
ECE Department
UT Austin
Austin, USA
jay_liu@utexas.edu

David Z. Pan
ECE Department
UT Austin
Austin, USA
dpan@ece.utexas.edu

Yibo Lin*
CECA, CS Department
Peking University
Beijing, China
yibolin@pku.edu.cn

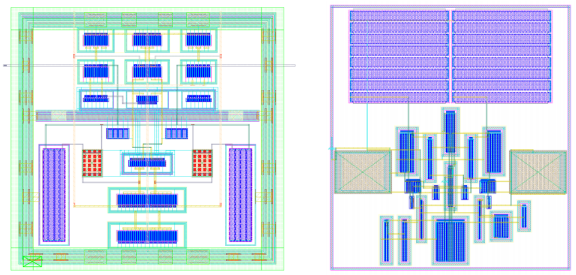
***Abstract*—Analog layout design still relies heavily on manual efforts. Current fully automated flows are not yet able to satisfy the demands of versatile customization and not compatible to the existing manual flows. Interactive layout editing has the potential to bridge the gap between the manual flows and fully automated flows shooting for both performance and productivity. In this paper, we propose an interactive editing framework with instructions for both topological editing and detailed customization. We also propose an effective instant legalization algorithm for fast layout update during the real-time interaction with users.**

I. Introduction

Analog layout design still relies heavily on manual efforts. Designers or layout engineers draw device placement and wire routing according to their design expertise and experience, considering various constraints like symmetry, matching, signal flow, and so on. However, increasing design complexity and complicated design rules are slowing down their productivity and design closure. Automation tools are desired to speedup analog design flows.

Recently, fully automated frameworks for analog layout generation have been proposed, such as the ALIGN [1] and MAGICAL [2]. They leverage both machine learning and algorithmic innovations to automatically generate analog layouts from circuit netlists, and aim at an end-to-end analog layout design flow without human-in-the-loop [3]–[6]. These frameworks usually consist of several stages, such as constraint generation for extracting the layout constraints from netlists, analog placement to determine the device locations, and routing to finish the wiring between devices. With the philosophy of no-human-in-the-loop, these fully automated flows finish layout synthesis with "one-button-click" and do not expect any intermediate interaction with designers or human involvement.

While fully automated layout generation can significantly speedup the prototyping time and provide good initial solutions on specific circuits, it alone is hard to achieve wide-adoption in the short term due to the following reasons. 1) Analog circuit topologies are rapidly evolving with new design practices. A fully automated flow is not flexible enough to satisfy the versatile customization demands. 2) Analog layout drawing is designer-specific. That is, two designers can have quite different ways to layout the same circuit according to their own experience and taste, which is hard to be implemented in a fully automated flow. 3) Layouts generated by a fully automated flow may not align with designers' intuition, like the example in Figure 1, leading to performance degradation or difficulties in post-silicon debugging after tape-out. Therefore, we argue that fully automated layout generation alone is not enough to completely satisfy the demands for lowering the design efforts and speeding up design closure.



(a) (b)

Fig. 1: Layouts for an OTA design. (a) The manual solution is very different from (b) the tool solution [2].

By analyzing the designers' workload and working patterns, we have following observations: 1) Designers often need to visually inspect and attempt different layout strategies before finalizing. 2) Designers are usually not willing to risk any unfamiliar layout topologies. 3) A significant portion of time is spent on cleaning design rule violations. In other words, designers expect a tool to improve their productivity but still have full control over the design process. Thus, interactive layout editing can be a promising option to bridge the gap between designers' expectation and existing fully automated flows. Based on such observations, to speedup the design closure and increase the productivity of designers, we propose an interactive analog layout editing framework. The idea of this framework is to free designers from dealing with detailed design rules and focus on layout topologies. Meanwhile, we still keep the freedom for full layout customization.

The main contributions of this paper are as follows.

- We propose an interactive analog placement framework that supports commands for fast layout editing. The set of instructions cover constraints for global perturbation and commands for local adjustment, enabling both topological editing and fine-grained customization.
- We propose an instant legalization algorithm for incremental layout update with linear time complexity, enabling real-time interaction upon users' input instructions.
- Experimental results on open-source analog circuits such as comparators, amplifiers, and data converters demonstrate that the framework can enable efficient layout editing and reduce the turn-around time for layout designers.

The rest of the paper is organized as follows. Section II introduces the limitation of fully-automated layout synthesis and the concept of interactive analog layout editing. Section III presents the workflow

*Corresponding author

978-1-6654-3274-0/21/$31.00

of layout editing. Section IV details the algorithms of the instant legalization technique. Section V demonstrates an interactive layout editing process and performance of the legalization algorithm. Section VI summarizes the paper.

II. Preliminaries

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

A. Fully-Automated Layout Synthesis

A fully-automated flow aims at end-to-end layout generation without human-in-the-loop. The flow takes analog circuit netlists and technology libraries as input, and outputs placed and routed layout solutions in GDSII format. The flow performs placement and routing internally with well-defined objectives and layout constraints to optimize. For example, the automated tools need to detect symmetric and matching devices, and handle these constraints in placement and routing.

So far, developing an end-to-end flow that can generate high-quality and stable solutions is still challenging, as it is hard to define a universal analytical objective for analog layout problems. Existing work like ALIGN and MAGICAL [1], [2] has incorporated machine learning based techniques to extract designers' experience from manual layouts. These tools demonstrate good initial solutions for specific circuits, but still need further improvement for wide adoption of 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. It takes the layout generated a fully-automated flow as input, and involves designers for customization. It provides high-level control over the layout topology but leaving the low-level design rule fixing and legalization to automated algorithms. Once designers are satisfied with the layout, the interactive layout editing tool outputs the eventual layout solution in GDSII format. Such a methodology can bridge the gap between designers' expectation and the current performance of automated tools through fast interaction.

In this work, we focus on interactive analog placement editing. 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 target is that users can progressively make changes to the layout topology with input commands. We also need a legalization backbone for real-time layout update upon input commands for interaction with users. We define 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.

We consider hierarchical designs with layout symmetry constraints in this work. The perturbation is evaluated with total displacement of devices. Runtime is very critical for smooth interaction when the designers make progressive changes.

C. Analog Placement

Layout editing is closely related to analog placement. A series of studies make efforts on improving the representation of analog placement [7], such as B*-tree [8], [9] and O-tree [10], and leveraging automated searching techniques like simulated annealing to find high-quality solutions. Recent work starts to investigate automatic prediction of layout constraints with deep learning [3], [4] as well as performance prediction and placement optimization [5], [11], [12]. Another line of work focuses on procedural based layout generation [13]–[15]. Although some of the works are able to produce silicon results, users have to write significant amount of codes, limiting the wide adoption of the solutions. After surveying the advantages and limitations of prior work, we see interactive layout editing as a potential bridge between the fully automated analog placement and the procedural based flow with instant layout visualization and capability of full customization.

Constraint graph is another common representation for layout topology [16]. In this representation, each placement topology corresponds to two directed acyclic graphs: a horizontal constraint graph (HCG) and a vertical constraint graph (VCG). Each graph specifies the relative locations of devices horizontally or vertically. For example, if a device B is located at the right of another device A, an edge is inserted to the HCG from vertex A to vertex B. Constraint graph can be adapted to handle multiple types of analog layout constraints [17]. We leverage the constraint graph for simplicity and propose a novel constraint graph representation with a single unified graph taking the advantage of topological-sort properties. Note that our layout editing scheme is compatible with different analog layout representation.

III. Interactive Layout Editing

In this section, we explain the workflow of interactive layout editing and our framework in detail.

A. Interactive Layout Editing Workflow

Figure 2 shows the software architecture of our interactive analog layout editing tool. In the frontend, we provide a command interface and GUI interface for displaying analog layout and real-time interaction. All events on the interface are encoded as commands. We adopt analog placement engine of MAGICAL [2] to generate the initial placement results shown to the users. The commands input by users will be interpreted to internal operations via a command interpreter. The internal operations are applied to update our data representation including constraint graph, constraint sets, and cell location information. If the user requests a legalized layout, the legalizer will do legalization sufficiently fast in the feedback phase, so that the users get the legalized results shown on the interaction interface instantly.

B. Layout Editing Command

Given an initial placement result generated by an automation tool, the user may not be satisfied with the solution and expect flexibility to adjust the placement results. We define a simple and extensible command set. The user can describe their demand in a sequence of commands. The interface mainly supports two types of commands: fine-grained topology-related commands and coarse-grained constraint-related commands. Table I lists six commands. Commands `move`, `spacing`, `resize` and `swap` are fined-grained and manipulate the placement topology or geometry. Command `arrayAdd` adds an array constraint which aligns the devices, and

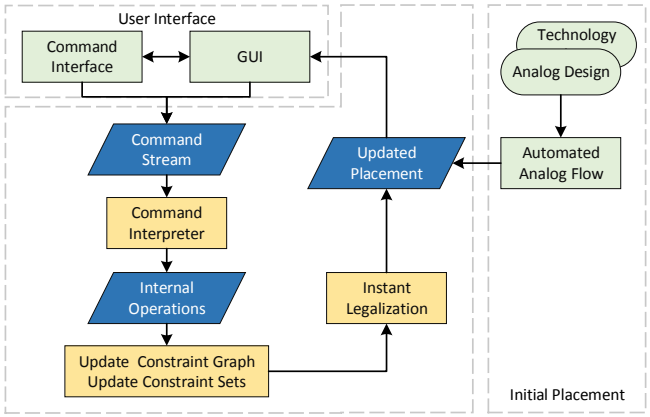


Fig. 2: Workflow of layout editing.

TABLE I: User command set.

Command	Parameters	Description
`move`	device v_i, location (x, y)	move a device to a location
`spacing`	devices v_i, v_j, spacing width W	add spacing between devices
`resize`	shape w, h	change the shape of a device
`swap`	devices v_i, v_j	swap two devices
`arrayAdd`	devices $\{v_i\}$	add array constraint
`symAdd`	devices v_i, v_j sym axis A_k	add symmetry constraint

command `symAdd` adds a symmetry constraint which enforce symmetric locations of devices. Naturally, the commands form a complete set for layout editing; i.e., for any two topologically different layouts L_i and L_j, we can find a command sequence converting L_i to L_j. The command sequence is usually short.

C. Command Interpretation

The layout is maintained in an internal representation with the constraint graph and a constraint set. All commands defined before can be realized with a sequence of internal operations on the internal representation. Similar to the command set, we define a group of standard internal operations on the constraint graph: {`Insert`, `Remove`}. Take command `swap` for example. We can implement command `swap` 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), `Remove`(v_j)} on the constraint graph, where p_i, p_j are the original locations of v_i, v_j. If inserting a vertex v_i to the position v_j being removed, the Insert operation will query the vertices which are adjacent to v_j before and determine the new edges for vertex v_i.

IV. Instant Legalization

In practical scenarios, the users conduct commands and run legalization frequently, so the legalization process is supposed to be sufficiently fast and consistent to ensure that users can see reliable legalized placement results immediately. In this section, we explain the detailed implementations of our linear-time legalization algorithm. We first give an overview of our proposed legalizer in Section IV-A. Then we introduce a novel placement topology representation named mixed constraint graph in Section IV-B. The mixed constraint graph data structure supports the internal operations aforementioned in constant time $\mathcal{O}(1)$ and legalization in linear time $\mathcal{O}(n)$ (n is the number of devices of the analog circuit). The detailed algorithm involves a layout partitioning technique and a topological sort based legalization scheme described in Section IV-C and Section IV-D, respectively. A brief analysis of time complexity is given in Section IV-E. Readers can refer to Table II for notations.

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

A. Legalizer Overview

Our legalizer can handle both flat circuits and hierarchical circuits. The legalization flow for a flat circuit is shown in Algorithm 1. The flow can be easily extended to hierarchical circuits by tackling the circuit hierarchy from the bottom up. For hierarchical circuits, the legalization process starts with the leaf-level subcircuits in the circuit hierarchy tree. At the leaf level, the algorithm takes the device-level constraint graph and device symmetry groups as inputs and completes legalization inside each subcircuit. For a higher-level subcircuit, we treat the child subcircuits as cells and repeat the legalization process.

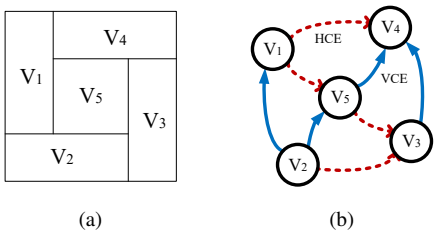
Algorithm 1 Instant Legalization

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: Blocks $\{B_i\} \leftarrow$ PartitionLayout($\{(b^{xl}, b^{xr}, b^{yb}, b^{yt})\}$)
4: construct global mixed constraint graph G^m
5: **for** B_i in $\{B_i\}$ **do**
6: do topo-sort based legalization inside B_i
7: do topo-sort based legalization on G^m

Algorithm 1 outlines three-step legalization. The first step is to partition the layout into blocks based on constraint groups (line 1-3). We can support multiple constraint types including symmetry constraint, matching constraint, and array constraint, and we take symmetry constraint in this paper for example. Each symmetry group contains a symmetry axis A_i and a symmetry group P_i. Symmetry Group P_i is a set of symmetry pairs (c_i, c_j), where c_i and c_j are cells (we generalize "cell" to denote a device or a subcircuit in hierarchical circuits if not specially mentioned). Given the symmetry groups $\{(A_i, P_i)\}$, we can traverse the cells c_i in the group to get the boundaries in four directions xl, xr, yb, yt (left, right, bottom, top) (line 1-2). Then, we add virtual lines to split the layout into grids and merge grids correlated to the same symmetry group to a block (line 3). We will go deeper in Section IV-C. The topology between blocks will be described as a mixed constraint graph(MCG) G^m (line 4). We will introduce the MCG in Section IV-B. The next step is to do legalization inside each block with a topological-sort-based approach described in Section IV-D (line 5-6). After all blocks are legalized, the final step is to apply a similar but coarse-grained approach to mixed graph G^m. Finally, we get a legalized layout L without overlaps.

B. Mixed Constraint Graph Representation

We derive mixed constraint graph (MCG) from constraint graph representation described in Section II-C. An MCG is a combination of HCG and VCG. Given a HCG $G^h = (V, E^h)$ and a VCG $G^v = (V, E^v)$, we can obtain the MCG $G^m = (V, E^h \cup E^v)$. Figure 3 illustrates how we extract the MCG from a layout.



(a) (b)

Fig. 3: Mixed constraint graph with horizontal constraint edges (HCEs) and vertical constraint edges (VCEs).

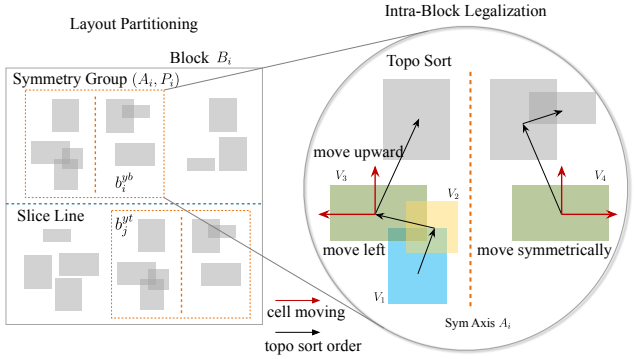


Fig. 4: Layout partitioning and intra-block legalization.

Note that G^m is a heterogeneous graph; namely, G^m contains two different types of edges, horizontal constraint edges from G^h and vertical constraint edges from G^v. There is a one-to-one correspondence between MCG and the pair of HCG, VCG. Because each layout can be uniquely represented as a horizontal graph and a vertical graph, we can represent the layout in a unique mixed constraint graph.

1) Topological Sort on MCG: The MCG keeps the information from the original constraint graph but also introduces a new property with topological sort.

For a directed graph $G = (V, E)$, the topological sort gives a linear ordering S of vertices in V. If there is a directed edge $\langle u, v\rangle$, then vertex u appears before vertex v in ordering S. If we regard the edges as dependencies among vertices, then the topological sort indicates an order that no vertex depends on its subsequent vertices. For example, we can give a topological order of the MCG in Figure 3(b) as $S = \{v_2, v_1, v_5, v_3, v_4\}$. If we move a vertex v_i towards the up right direction, the movement does not impact the vertices before v_i in S. Because of this property, we can develop an algorithm that traverses each cell only once and computes legal locations without backtracking. We will discuss the topological-sort-based algorithm in Section IV-D.

Our algorithm requires that the MCG has a topological ordering. Considering that MCG is a directed acyclic graph (DAG) and any DAG has at least one topological ordering, Theorem 1 holds.

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

C. Layout Partitioning Technique

We partition the layout according to the symmetry constraint groups. That is, we divide the legalization problem into subproblems for each symmetry constraint group. Figure 4 shows a layout partitioning example where there are only two symmetry groups.

Algorithm 2 details the layout partitioning process and introduce a slicing line procedure. As mentioned in Section IV-A, the boundaries $(b^{xl}, b^{xr}, b^{yb}, b^{yt})$ are calculated and passed to this step. We first sort the four boundary arrays in ascending order (line 11). Then a slicing rule is applied to x direction and y direction respectively (line 12). Taking x direction for example, we add a vertical slicing line at the middle of $b^{xr}[i]$ and $b^{xl}[j]$ if $b^{xr}[i]$ is less than $b^{xl}[j]$ and there is no other boundary value between $b^{xr}[i]$ and $b^{xl}[j]$ (line 1-8). Take the Figure 4 for example, the bottom boundary b_i^{yb} of symmetry group (A_i, P_j) follows the top boundary b_j^{yt} closely (there are no other boundaries between them), so we add a horizontal slice line between b_j^{yt} and b_i^{yb}.

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(b1, b2)
2:     i ← 0, j ← 0
3:     while i ≠ m do
4:         while b1[j] < b2[i] do
5:             j ← j + 1
6:         if b1[j] < b2[i + 1] then
7:             AddLine((b1[j] + b2[i])/2)
8:         i ← i + 1
9: end function
10: function PartitionLayout(b^xl, b^xr, b^yb, b^yt)
11:     Sort(b^xl), Sort(b^xr), Sort(b^yb), Sort(b^yt)
12:     AddSliceLine(b^xl, b^xr), AddSliceLine(b^yb, b^yt)
13:     traverse slice lines and get slices grids
14:     construct blocks {B_i} from sliced grids
15: end function
```

With the slicing lines added, the layout is divided into virtual grids. The grids associated with the same symmetry group are clustered into one block. The grids independent of symmetry groups are directly constructed as blocks. Shown in Figure 4, in the block B_i with symmetry group (A_i, P_i), there are some cells not belonging to P_i. For constraint groups other than symmetry constraint mentioned here, the divide and conquer methodology still works.

D. Topological-Sort-based Legalization

We propose an efficient legalization scheme based on topological sort. Algorithm 3 presents an implementation of symmetry-aware legalization based on two-fold topological sort. As shown in Figure 4, for an MCG G_l^m of a block with symmetry axis, the symmetry axis separates the block from the middle in geometry and also split the MCG into two subgraphs (the subgraph of MCG left to the symmetry axis and the subgraph of MCG right to the symmetry axis for a symmetry axis parallel to the y-axis). Imagine our legalization is a process of spreading the devices outward from the symmetry axis while eliminating the overlaps. As a horizontal constraint edge of MCG indicates a geometry constraint in the positive direction of the x-axis, we want the constraint edge to indicate the direction away from the symmetry axis. So we mirror the subgraph of MCG left to the symmetry axis, that is, we reverse the direction of horizontal constraint edges (every horizontal constraint edge switches its head and tail). We denote the mirrored left subgraph of MCG as G_l^m and the subgraph right to the symmetry axis as G_r^m. The legalizer maintains a queue of vertices with zero in-degrees for each MCG. At each step, the legalizer takes one vertex v_l out of queue Q_l (line 6). We find all the incoming edges $\{\langle u, v_l\rangle\}$ for vertex v_l. A horizontal edge $\langle u, v_l\rangle$ indicates that v_l is left to u, while a

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: **if** v_l in a symmetry pair (v_l, v_r) **then**
13: **while** $u_r \neq v_r$ **do**
14: $u_r \leftarrow Q_r.pop\_front$
15: update location of u_r
16: **for** each u adjacent to u_r **do**
17: $in_r[u] \leftarrow in_r[u] - 1$
18: **if** $in_r[u] = 0$ **then**
19: $Q_r.push(u)$
20: update location of v_l, v_r symmetrically
21: **while** $Q_r \neq \varnothing$ **do**
22: $v_r \leftarrow Q_r.pop\_front$
23: update location of v_r
24: **for** each v adjacent to v_r **do**
25: $in_r[v] \leftarrow in_r[v] - 1$
26: **if** $in_r[v] = 0$ **then**
27: $Q_r.push(v)$
28: **end function**

vertical edge indicates that v_l is above u. Then, we move v_l to $\min\{x_u - (w_{v_l} + w_u)/2\}$ in x direction which ensure v_l satisfies the horizontal constraints $\{\langle u, v_l\rangle\}$. We only move v_l to the left and move v_l upward to remove overlaps. For V_3 in Figure 4, we move it to the left and upward to resolve the overlaps with V_1 and V_2.

When the traversal reaches a v_l which is in a symmetry pair (v_l, v_r), the legalizer performs a similar traversal on G_r^m to reach v_r (line 13-19). We assign symmetric and legal coordinates for v_l and v_r, which means that the distance to symmetry axis takes $\max\{|x_{axis} - x_{v_l}|, |x_{v_r} - x_{axis}|\}$ and the y coordinate takes $\max\{y_{v_l}, y_{v_r}\}$ (line 20). Figure 4 shows that V_4 is moved to a symmetrical location with V_3. After the location of some v get legalized, we decrease the in-degrees of every vertex u with directed edge $\langle v, u\rangle$ and add the vertex to the corresponding queue if the in-degree becomes zero. The whole algorithm terminates when the two queues Q_l and Q_r are empty.

Suppose 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, claimed in Theorem 2. The property ensures that we can synchronize the traversals of two MCGs without skipping any vertex of another MCG.

Theorem 2. *For two symmetry pair (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.*

E. Time Complexity Analysis

We present a brief analysis of the time complexity. We will show that the flow of Algorithm 1 is $\mathcal{O}(n)$ (n is the number of devices of the analog circuits). Firstly, the layout partitioning stage is $\mathcal{O}(n)$. There are k symmetry groups. The slicing lines divide the layout to $\mathcal{O}(k^2)$ non-empty grids. The number of grids is always much less than the number of devices n, so $\mathcal{O}(k^2)$ is $\mathcal{O}(n)$. 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(line 13-14), which is $\mathcal{O}(k^2)$. Therefore, the first part (line 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 the block B. The time complexity of topological sort on graph $G = (V, E)$ is $\mathcal{O}(|V| + |E|)$. For MCG, we have a good property pointed in Lemma 1. Theorem 3 summarizes the linear time complexity.

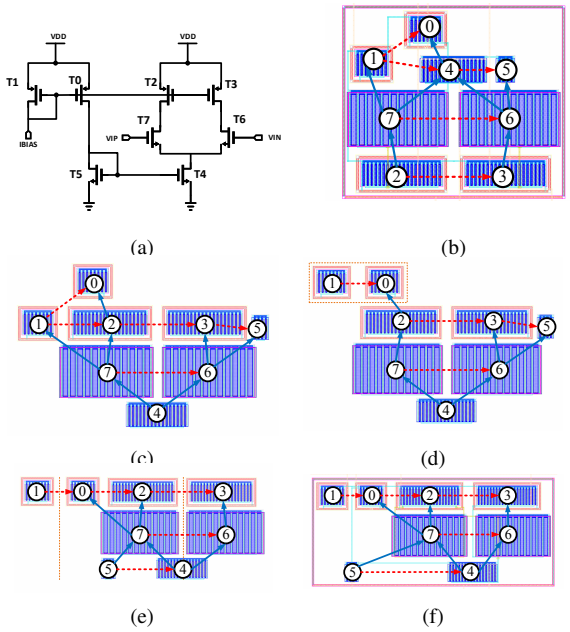


Fig. 5: An example of layout editing process with command sequence: {swap([4], [2, 3]), symAdd([1, 0], A_2), move([1, 0]), move([5]), symmAdd([5], A_2)}.

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

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

V. EXPERIMENTAL RESULTS

A. Interactive Editing Process

We illustrate the functionality of interactive layout editing with a simplified example. Figure 5 shows how to convert a layout generated by automation tool to a desired layout. The first layout in Figure 5 is a part of a placement layout generated by the automation tool MAGICAL [2] and the corresponding representation is shown in Figure 5(b). Assume that a designer gets the initial placement layout shown in Figure 5 from the automation tool, but the designer is not satisfied with the layout. We support an interactive process that the designer can conduct the commands defined in Table I and see the legalized placement layout immediately. At the first step, command `swap` is conducted between device 4 and devices {2, 3}. The command is interpreted as swapping v_4 and $\{v_2, v_3\}$ on constraint graph. With instant legalization, the user can see the layout shown in Figure 5(c). At the next step, command `symAdd` adds a new symmetry group with symmetry pair $\{v_1, v_0\}$. The symmetry constraint is presented in Figure 5(e). Then several `move` commands

TABLE III: Statistics of the circuits

Circuit	OTA1	OTA2	OTA3	COMP	ADC1	ADC2
Devices	25	49	42	17	114	211
Hierarchy	-	-	-	-	✓	✓

TABLE IV: Legalization performance

	TOPO			LP		
	T_{seq} ms	D_{max} %	D_{avg} %	T_{seq} ms	D_{max} %	D_{avg} %
OTA1	0.16	31.1	17.7	33.3	40.9	13.0
OTA2	0.23	43.4	5.1	33.4	43.4	5.1
OTA3	0.20	10.9	3.99	35.4	16.2	3.3
COMP	0.12	11.8	4.20	31.5	12.6	3.9
ADC1	0.84	14.1	10.1	101.7	16.4	8.0
ADC2	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	-	-

are conducted to re-assign device locations. After a `symAdd` which adds the device 5 to the left symmetry group, the user finally reaches a desired layout result.

B. Legalization Performance

Furthermore, we set up application scenarios to validate the instant legalization of our feedback phase. We implement the algorithms in C++ and Python and perform our experiments on a Linux server with 20-core Xeon(R) CPU @ 2.1GHz. We build a dataset to test the legalization algorithm as follows.

The circuits are from the open-source repository MAGICAL [2] with statistics in Table III. The dataset contains three operational transconductance amplifiers (OTAs), one comparator (COMP) and two analog to digital converters (ADCs). All circuits contain symmetry group constraints. The ADC circuits are relatively large and have a hierarchy. We prepare our dataset by perturbing the analog circuit layouts generated by the automation tool. First, we generate one initial placement result for each circuit by MAGICAL. Then a human command stream can be simulated by generating a random sequence of commands listed in Table I. For each initial placement layout, we obtain 10 feasible command streams to build 10 different pre-legalization layouts. Therefore, we get a dataset with 60 pre-legalization layouts to validate the performance of our legalization algorithm.

Also, we adopt the concept of linear-programming-based legalization from MAGICAL and implement an LP legalizer for comparison. The objective of the LP legalizer is to minimize the displacement. We use GUROBI as the LP solver. Table IV shows the performance of the legalizers. We evaluate our legalization algorithm with a runtime metric and two displacement metrics. The T_{seq} metric is the runtime of doing one-shot legalization after conducting the complete command stream.The D_{max} and D_{avg} metrics denote the maximum cell displacement and the average cell displacement in a layout, respectively. The D metrics are measured in percentage of Manhattan distance divided by half-perimeter of the layout bounding box. The T_{seq}, D_{max} and D_{avg} metrics take the maximum value over the 10 layout samples. Our legalizer produces legalized layout with displacement close to optimal value and the legalized layout is usually similar to the one desired. As shown in Table IV, our legalizer can give a legalized layout within about 1ms for each case, which indicates that there is almost no delay when the users edit the layout.

VI. Conclusion

In this paper, we propose an analog placement paradigm with interactive analog layout editing. We introduce the new concept, analog layout editing, to bridge the gap between existing analog layout automation tools and experienced designers. We define a complete process with command set, command interpretation, and instant legalization. We propose a novel layout representation, mixed constraint graph, and topological-sort-based legalization approach for layout editing in linear time complexity. The experimental results show the usability and efficiency of our algorithm. In future work, we will enhance the expressivity of our layout editing for better customization.

Acknowledge

This project is supported in part by the National Key Research and Development Program of China (No. 2019YFB2205000).

References

[1] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, "Align: Open-source analog layout automation from the ground up," in *Proc. DAC*, 2019, pp. 1–4.

[2] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, "Magical: Toward fully automated analog ic layout leveraging human and machine intelligence," in *Proc. ICCAD*. IEEE, 2019, pp. 1–8.

[3] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. DATE*, 2020.

[4] 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," 2020.

[5] M. Liu, K. Zhu, X. Tang, B. Xu, W. Shi, N. Sun, and D. Z. Pan, "Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis," in *Proc. DAC*. IEEE, 2020, pp. 1–6.

[6] 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. ICCAD*. IEEE, 2020.

[7] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *Proc. ASPDAC*. IEEE, 2016, pp. 617–622.

[8] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu, "B*-trees: a new representation for non-slicing floorplans," in *Proc. DAC*, 2000, pp. 458–463.

[9] P.-Y. Chou, H.-C. Ou, and Y.-W. Chang, "Heterogeneous b*-trees for analog placement with symmetry and regularity considerations," in *Proc. ICCAD*. IEEE, 2011, pp. 512–516.

[10] 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. DAC*, 2000, pp. 464–467.

[11] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, W. Xu, S. Sapatnekar, R. Harjani, and J. Hu, "Exploring a machine learning approach to performance driven analog ic placement," in *Proc. ISVLSI*. IEEE, 2020, pp. 24–29.

[12] M. Liu, K. Zhu, J. Gu, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "Towards decrypting the art of analog layout: Placement quality prediction via transfer learning," in *Proc. DATE*, 2020, pp. 496–501.

[13] J. Crossley, A. Puggelli, H.-P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja *et al.*, "Bag: A designer-oriented integrated framework for the development of ams circuit generators," in *Proc. ICCAD*. IEEE, 2013, pp. 74–81.

[14] E. Chang, J. Han, W. Bae, Z. Wang, N. Narevsky, B. NikoliC, and E. Alon, "Bag2: A process-portable framework for generator-based ams circuit design," in *Proc. CICC*. IEEE, 2018, pp. 1–8.

[15] 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 Journal Solid-State Circuits*, vol. 52, no. 7, pp. 1915–1926, 2017.

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

[17] Q. Ma, L. Xiao, Y. Tam, and E. F. Y. Young, "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE TCAD*, vol. 30, no. 1, pp. 85–95, 2011.