

APCG: Automatic Placement Constraint Generation for Analog Circuits

Binfeng Fan¹, Rongliang Fu^{3*}, Wenxing Li^{4*}, Huilong Jiang⁴, Minglei Zhou^{2,4}, Zhixuan Zhang⁴,
Da Wang⁴, Fei Wang⁵, Xiaochun Ye⁴, Dongrui Fan², Junying Huang⁴

¹Hangzhou Institute for Advanced Study, UCAS, Hangzhou, China

²School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

³Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

⁴SKLP, Institute of Computing Technology, CAS, Beijing, China

⁵School of Integrated Circuits, Shandong University, Jinan, China

Abstract

Automatic generation of placement constraints is a key step in analog layout automation. Prior studies mainly focus on inferring symmetry or matching relations from circuit netlists, or on performing layout realization under a given constraint set, but rarely treat executable constraint-form assignment and physical-feedback-driven constraint-set optimization in a unified flow. Consequently, the generated constraints may be logically incompatible or physically suboptimal. To address this limitation, this paper proposes APCG, a physically closed-loop framework for automatic placement constraint generation. First, a graph attention network is used to extract high-confidence candidate symmetry constraints. Second, a conflict hypergraph captures high-order incompatibilities and defines a logically feasible search space. Third, Monte Carlo Tree Search over the hypergraph selects area-optimized constraint subsets, guided by physical feedback from layout results. Experiments on five benchmark circuits across the ASAP7 Predictive PDK, FinFET14nm Mock PDK, and SkyWater SKY130 PDK demonstrate that APCG consistently outperforms the baseline, achieving average area reductions of 21.28%, 21.74%, and 37.68%, respectively, while preserving correct functionality. APCG also consistently reduces HPWL, and post-layout evaluation on SkyWater SKY130 further confirms correct functionality together with lower delay and power.

1 Introduction

The growing demand for high-performance analog chips, driven by automotive electrification, AI servers, and 5G communications, has intensified the need for efficient analog design automation. Although analog integrated circuits are widely used in communication, automotive electronics, medical devices, and industrial control, their design automation lags significantly behind digital circuits [1]. A key reason is that analog layouts must carefully account for parasitics, device mismatch, and process variations, requiring precise control over geometric and electrical constraints (e.g., symmetry, matching, proximity, routing topology) during placement and routing. Consequently, analog layout synthesis remains a time-consuming,

Table 1. Comparison with representative prior works.

Method	Constraint extraction		Layout coupling	
	Symmetry inference	Symmetry orientation	Layout realization	Layout-feedback optimization
DAC'19 [5]	✓	✓	✓	✗
ASPDAC'21 [6]	✓	✗	✗	✗
MLCAD'23 [7]	✓	✗	✗	✗
TCAS-I'24 [8]	✓	✗	✗	✗
APCG (ours)	✓	✓	✓	✓

error-prone task relying heavily on expert experience [2–4], thereby limiting further design productivity improvements.

Among various analog layout synthesis challenges, generating placement constraints is particularly critical as it encapsulates expert design knowledge and directly governs subsequent automation steps. However, constraint specification remains largely manual, serving as a primary bottleneck in the automation pipeline. This motivates the need for automatic techniques capable of extracting high-quality placement constraints directly from circuit netlists.

To generate high-quality layouts, substantial research focuses on generating analog placement constraints. The three most prevalent layout constraint categories are symmetry, matching, and proximity, with symmetry constraints receiving considerable attention. Depending on the abstraction level, existing symmetry constraint generation methods are classified into device-level and group-level. Group-level constraints apply to subcircuit blocks or hierarchical modules [9–12], while device-level constraints directly govern individual device pairs (e.g., two transistors forming a differential pair). Given that device-level symmetry forms the foundation of analog circuit performance—directly mitigating mismatch-induced degradation in key metrics like offset and common mode rejection ratio—this work concentrates on device-level symmetry constraint extraction.

Existing device-level methods can be categorized into heuristic and learning-based approaches. Heuristic approaches (e.g., graph-similarity-based symmetry detection) [5] rely on manually tuned thresholds, limiting their generalization capability. Learning-based methods have also been widely explored. Gao et al. [6] formulate symmetry detection as a GNN-based binary classification task; Xu et al. [8] adopt an edge-augmented graph attention network for device-level symmetry detection; Zhang et al. [13] propose a heterogeneous graph convolution network with a three-class classifier for matching constraints. Wu et al. [7] combine VF2 subgraph isomorphism for candidate generation with relational graph convolutional networks for false-positive filtering and functional classification. Table 1 compares existing device-level methods in terms of constraint

* Corresponding authors: rlfu@cse.cuhk.edu.hk, liwenxing@ict.ac.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License. ICCAD '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2873-0/2026/11

<https://doi.org/10.1145/3831252.3834248>

extraction and layout coupling, revealing three limitations. First, most methods formulate symmetry inference as binary classification, only identifying the existence of symmetry without specifying its layout realization. Second, methods predicting symmetry orientations (e.g., horizontal or vertical) generally couple orientation selection with symmetry inference, although orientation directly affects placement and routing. Third, existing methods lack layout feedback mechanisms for iterative constraint refinement, making it difficult to identify constraint sets optimized for final layout quality.

These limitations raise three fundamental questions regarding layout design trade-offs: (1) For each device pair, which symmetry orientation (e.g., horizontal or vertical) should be applied? (2) How can conflicts and couplings among constraints (e.g., device reuse, orientation exclusivity, spatial cyclic dependencies) be handled to ensure the generated constraint set's logical consistency? (3) How can the constraint set minimizing layout area be selected?

To address these questions, we propose APCG, formulating analog placement constraint generation as a combinatorial optimization problem over constraint subsets. First, we employ EGAT [8] to generate high-confidence candidate constraints from the netlist. Second, we construct a constraint conflict hypergraph uniformly modeling candidate relationships, capturing device reuse, orientation exclusivity, and spatial cyclic dependencies to define a logically feasible search space. Third, we perform Monte Carlo Tree Search (MCTS) on the hypergraph, using layout area feedback to guide the search toward superior constraint subsets. By unifying learning-based candidate priors, hypergraph-based logical reasoning, and layout area evaluation within a closed loop, APCG identifies logically consistent and area-efficient constraint sets. In summary, this work makes the following contributions:

- We propose APCG, an automatic placement constraint generation framework that produces area-efficient constraint sets for analog circuits.
- We introduce a constraint conflict hypergraph explicitly modeling candidate constraint conflicts and dependencies, enabling unified handling of structural incompatibilities (e.g., device reuse, orientation exclusivity, spatial cyclic dependencies).
- We develop an MCTS-based constraint generation framework integrating candidate prediction, logical reasoning, and layout feedback into a closed-loop search.
- Experimental results on multiple benchmark circuits across three technology nodes (ASAP7 Predictive PDK, FinFET14nm Mock PDK, SkyWater SKY130 PDK) demonstrate APCG consistently improves layout quality over the baseline, achieving average area reductions of 21.28%, 21.74%, and 37.68%, respectively.

2 Background

2.1 Symmetry Constraints in Analog Layout

In analog layout design, symmetry constraints require matched device pairs or modules to be placed and routed symmetrically across a common axis [14]. These constraints are extensively used to mitigate mismatch from process variation, parasitic imbalance, and layout-dependent effects, thereby effectively preserving key metrics like offset, current-mirror accuracy, and common-mode rejection ratio (CMRR) [9, 12, 15]. These constraints may appear at both the

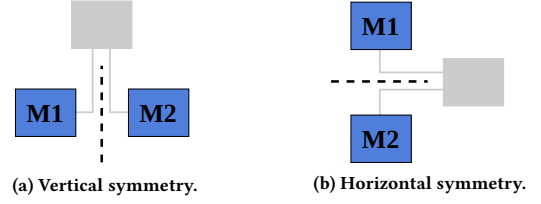


Figure 1. Axis-symmetry constraints considered in this work.

module level (e.g., OTAs, comparators) and device level (e.g., matched transistors, resistors, capacitors) [10].

It should be noted that matching-related layout constraints in analog design come in various forms, including axis symmetry, common-centroid layout, and general matching requirements [16]. Among these, the common-centroid layouts suit array-like placements of split devices around a shared centroid [17], axis symmetry requires device or module pairs to be placed as mirror images across a common axis [13]. This work focuses exclusively on pairwise axis-based symmetry, instantiated into vertical and horizontal forms (Fig. 1).

Based on this symmetry model, we define a candidate constraint as an a priori hypothesis derived from the circuit netlist, indicating that a pair of devices should be placed symmetrically. Thus, each candidate is fundamentally associated with a potentially matched device pair. A candidate is valid if the paired devices reside in the same circuit hierarchy, share the same device type (with consistent extra-pin potentials where required), and exhibit similar local topological roles. Device pairs spanning different hierarchies or having mismatched types are excluded [10]. Rather than being directly applied during layout generation, these candidates serve as prior hypotheses for subsequent hypergraph reasoning and layout-in-the-loop search.

2.2 Symmetry Constraint Estimation

To automatically estimate device symmetry likelihood from a SPICE netlist, Xu et al. proposed the edge-augmented graph attention network (EGAT) [8]. EGAT represents the netlist as a directed graph, treating devices and I/O ports as nodes, and inter-device connections as bidirectional edges. This graph representation allows the model to capture not only the attributes of individual devices but also the connection semantics carried by different pin relations.

Specifically, EGAT exploits both node and edge features to characterize circuit structure. Node features include device type, normalized size, and gate-connectivity information, while edge features encode pin-level connections (e.g., gate, drain, and source). Unlike conventional GNNs that rely primarily on node embeddings, EGAT incorporates node-edge interactions through an edge-aware attention mechanism during message passing, yielding representations better aligned with the structural regularities of device-level symmetry in analog circuits.

Based on the learned node embeddings, EGAT computes a confidence score for each candidate device pair using cosine similarity, indicating the likelihood of a symmetric match. Higher scores imply greater similarity in local topology, device attributes, and connection roles. The scores are then thresholded and post-processed to yield high-confidence symmetric pairs.

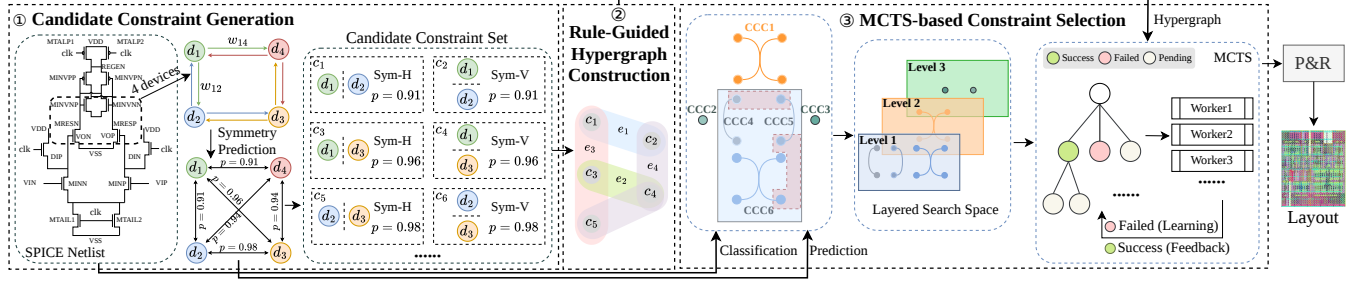


Figure 2. Overall framework of APCG.

In our work, EGAT serves only as a candidate generator rather than a full constraint generation solution. That is, it provides netlist-level symmetry priors, while constraint instantiation, conflict resolution, and physical-feedback-guided optimization are handled by the downstream APCG stages.

3 Task Definition

3.1 Terminology

An analog circuit can be represented by a SPICE netlist $N(D, W)$, where $D = \{d_1, d_2, \dots, d_n\}$ denotes the set of devices and W denotes the set of nets. Based on the symmetry constraints introduced above, the set of candidate constraints $C = \{c_1, c_2, \dots, c_m\}$ for devices can be generated from N . Each candidate constraint $c_i = \langle \tau_i, D_i, p_i \rangle$, where $\tau_i \in \{H, V\}$ denotes the symmetry orientation (horizontal or vertical), $D_i \subseteq D$ denotes the involved device subset (typically a pair), and $p_i \in [0, 1]$ is the confidence score predicted by EGAT. To formalize the constraint generation from C , we define a binary vector $\mathbf{x} = [x_1, x_2, \dots, x_m]^T \in \{0, 1\}^m$, where $x_i = 1$ denotes that c_i is selected and $x_i = 0$ otherwise.

Logical Feasibility via Conflict Hypergraph. Candidate constraints may conflict through device reuse, orientation exclusivity within a device pair, and cyclic spatial dependencies induced by relative-position relations. Since these conflicts can involve more than two candidates, we model them using a conflict hypergraph for unified reasoning. The constraint conflict hypergraph is defined as:

$$\mathcal{H} = (V, E, \sigma), \quad (1)$$

where the node set $V \equiv C$ is in one-to-one correspondence with the candidate constraints, and each hyperedge $e \in E$ represents a subset of conflicting candidate constraints. The mapping $\sigma(e)$ specifies the Boolean semantics associated with hyperedge e , such as `not_all` or `at_most_k`. Specifically, `not_all` represents that not all candidates within e can be simultaneously selected, which is formulated as:

$$\sum_{v_i \in e} x_i \leq |e| - 1. \quad (2)$$

`at_most_k` denotes that at most k candidates within e can be selected, which is formulated as:

$$\sum_{v_i \in e} x_i \leq k. \quad (3)$$

In this work, a frequently used special case is `at_most_1`, which arises when multiple candidates correspond to mutually exclusive

realizations of the same underlying relation, such as different symmetry orientations assigned to the same device pair.

For each hyperedge $e \in E$, let $\phi_e(\mathbf{x})$ denote the satisfaction of $\sigma(e)$ by $\mathbf{x}$, where $\phi_e(\mathbf{x}) = 1$ indicates that the vector $\mathbf{x}$ satisfies the Boolean semantics associated with hyperedge e . The conflict hypergraph enables the computation of the logical feasible region Ω_{logic} comprising all vectors that satisfy the Boolean semantics of each hyperedge, formulated as:

$$\Omega_{\text{logic}} = \{\mathbf{x} \in \{0, 1\}^m \mid \phi_e(\mathbf{x}) = 1, \forall e \in E\}. \quad (4)$$

For implementation, these Boolean semantics are encoded in an incremental SAT-style feasibility model, which is queried to validate logical feasibility during search.

Physical Feasibility via Layout Evaluation. Logical feasibility does not guarantee physical feasibility. In practice, a set of logically satisfiable constraints may still fail during detailed placement and routing in the downstream physical design flow, because the physical engine must simultaneously resolve geometric packing, routing access, and technology-specific realizability constraints. Therefore, physical feasibility must be evaluated through actual layout generation rather than inferred from logic constraints alone.

For a given vector $\mathbf{x}$, physical feasibility is determined by whether the layout engine returns a legal layout, denoted by $r(\mathbf{x}) \in \{0, 1\}$. In addition, for physically realizable constraints, we record the realized layout area $A(\mathbf{x})$ and the corresponding wirelength $W(\mathbf{x})$ for subsequent optimization. Accordingly, the physically feasible region is defined as:

$$\Omega_{\text{phys}} = \{\mathbf{x} \in \Omega_{\text{logic}} \mid r(\mathbf{x}) = 1\}. \quad (5)$$

3.2 Problem Formulation

Given a SPICE netlist $N(D, W)$ and a set of candidate symmetry constraints C , our objective is to generate the optimal constraints subset $\mathbf{x}^*$ for devices to produce a layout with minimum area and wirelength. Based on the physically feasible region Ω_{phys} , we first define the area-optimal solution set as:

$$X_A^* = \arg \min_{\mathbf{x} \in \Omega_{\text{phys}}} A(\mathbf{x}). \quad (6)$$

Then, $\mathbf{x}^*$ with minimum wirelength in X_A^* can be formulated as:

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in X_A^*} W(\mathbf{x}). \quad (7)$$

4 Automatic Placement Constraint Generation

To solve the above constraint generation problem, we propose APCG, an automatic placement constraint generation framework that produces area-efficient constraint sets for analog circuits. The framework consists of three main steps: 1) candidate constraint generation, 2) rule-guided hypergraph construction, and 3) MCTS-based constraint selection. Fig. 2 provides an overview of APCG. The details of each step are presented in the following subsections.

4.1 Candidate Constraint Generation

Because conducting a physical feedback search directly within the complete constraint combination space is computationally prohibitive, we first employ a graph learning model to generate high-quality candidate priors, thereby narrowing the subsequent search space. As shown in Fig. 2, this stage contains two steps: pairwise symmetry prediction and candidate orientation assignment.

Using the EGAT framework introduced in Section 2.2, we process the input SPICE netlist to predict which device pairs are likely to form symmetry relations and assign a confidence score to each predicted pair. However, these pairwise predictions are not yet executable layout constraints, as the layout engine requires explicit symmetry orientation information. Moreover, the preferred symmetry orientation for a symmetric device pair cannot be determined from pairwise similarity alone, because horizontal and vertical realizations may lead to different compatibility relations in the downstream constraint set. To address this issue, APCG expands each predicted device pair into two candidate orientations: horizontal symmetry (H) and vertical symmetry (V). As shown in Fig. 2, both candidates act on the same device pair, share the same confidence score, but correspond to different symmetry orientations during layout realization. Because this work focuses on pairwise axis-based symmetry, only these two orientations are considered. Formally, for a predicted device pair D_i with EGAT confidence p_i , APCG instantiates two orientation-conditioned candidates, $\langle H, D_i, p_i \rangle$ and $\langle V, D_i, p_i \rangle$, so that all subsequent reasoning is carried out over a unified candidate representation.

Through the above two steps, the prediction results are transformed into the set of candidate constraints C . This set is then fed into the rule-guided hypergraph construction stage, which determines logical compatibility among candidates.

4.2 Rule-Guided Hypergraph Construction

Even when each candidate constraint is individually plausible, they may still be topologically incompatible and cannot be satisfied simultaneously. Therefore, in this stage, we instantiate the candidate constraint set into a logically self-consistent search space. Given the high-order coupling inherent in constraint conflicts, we model these conflicts by constructing a hypergraph.

Fig. 3 illustrates why a standard pairwise conflict graph is insufficient for modeling such high-order interactions. Suppose three candidate constraints c_2 , c_4 , and c_5 cannot be selected simultaneously. The intended semantics is not pairwise mutual exclusion: any two of them may still be feasible, while only the full combination $\{c_2, c_4, c_5\}$ must be excluded. An ordinary pairwise conflict graph (Fig. 3(a)) cannot capture this relation exactly. Adding binary conflict edges between every pair would incorrectly forbid all two-candidate subsets, leading to over-pruning of the feasible region. Omitting some

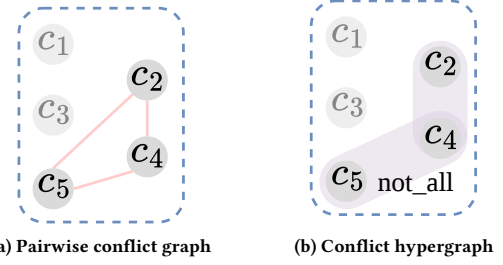


Figure 3. Comparison between a pairwise conflict graph and a conflict hypergraph.

edges, on the other hand, would lose the true three-way conflict. In contrast, a hypergraph (Fig. 3(b)) directly encodes this semantics by introducing a single `not_all` hyperedge over $\{c_2, c_4, c_5\}$, preserving feasible partial combinations while excluding the jointly invalid one.

Based on this insight, APCG adopts a conflict hypergraph to model candidate incompatibilities. We employ a rule-driven methodology to convert the candidate set into a conflict hypergraph governed by explicit logical statements. This stage primarily addresses the following core conflicts:

- **Mutually exclusive orientations.** When a device pair is expanded into multiple orientation candidates (e.g., H and V), these variants cannot be selected simultaneously. Since this work considers only two possible symmetry orientations for each device pair and at most one orientation can be assigned in the final constraint set, we instantiate an `at_most_1` hyperedge over the corresponding candidates to prevent conflicting assignments.
- **Cyclic spatial dependencies.** If several candidates collectively form a closed loop of spatial relative relationships, enabling all of them would trap the placer in a circular dependency. We construct a dependency directed graph and encapsulate the nodes of any detected directed cycle into a `not_all` hyperedge.
- **Device reuse conflict.** A physical device can only belong to a finite number of compatible structures within a layout. When multiple candidates compete for the same device, we bundle the competing constraint nodes and construct an `at_most_k` hyperedge, where k reflects the structural capacity of the device.

Through the above rule instantiation, we construct a conflict hypergraph that embeds node priors and high-order logical semantics. This hypergraph prunes logically inconsistent candidate combinations while retaining potentially viable ones. Moreover, these SAT feasible-region constraints progressively shrink the action space during search: *mutually exclusive orientation* hyperedges eliminate alternative orientations after each selection, while *cyclic spatial dependency* hyperedges remove multiple candidates simultaneously. Consequently, structurally invalid subsets are filtered out before invoking the expensive physical evaluator, providing a logically feasible search space for the layout feedback search process and improving the efficiency of the subsequent layout-in-the-loop search.

4.3 MCTS-based Constraint Selection

Given the logically feasible search space defined by the candidates and their high-order conflicts, we further apply a layout-in-the-loop MCTS to search for the optimal constraint subset for downstream

physical design. Unlike most traditional automated constraint generation, which simply predicts the presence of a constraint, our core objective is to discover a constraint combination that not only satisfies the hypergraph/SAT logic but also successfully generates a legal layout, outperforming the baseline in physical metrics like area and wirelength. Because the number of constraint combinations grows exponentially and a single ALIGN invocation is significantly more expensive than tree expansions, performing a flat search directly on the complete space is intractable and generates excessive invalid physical evaluations. Therefore, APCG adopts a closed-loop MCTS that combines hierarchical search-space decomposition, reward-guided tree policy, parallel physical evaluation, and failure-driven pruning.

4.3.1 CCC-Guided Hierarchical Search Space Construction. To reduce search complexity, APCG partitions the candidate search space into multiple hierarchical levels solved progressively. As shown in Fig. 2, level construction is jointly determined by Channel-Connected Components (CCCs) and the symmetry predictions produced by EGAT. The netlist is first converted into a switch-level graph, where vertices represent transistors and internal nodes, and edges represent channel connectivity. Each connected component is treated as a CCC [18], i.e., a set of transistors and internal nodes connected through conduction paths. Since devices within the same CCC are strongly coupled, CCCs provide a natural basis for hierarchical decomposition. EGAT then predicts candidate symmetric device pairs. If a predicted pair spans two CCCs, the corresponding CCCs are assigned to the same search level, ensuring that cross-CCC symmetry constraints are resolved consistently.

After the levels are constructed, the levels are ordered by the number of involved devices, from larger to smaller, and searched sequentially. Larger levels, which typically have greater influence on the final layout, are explored first. The selected partial solution at each level is fixed and propagated to subsequent levels as context. Assume that the hierarchical search space contains M levels, denoted by $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_M$. For each level $\mathcal{L}_i$, let $V_i \subseteq V$ denote the subset of candidate constraints assigned to that level, where V is the set of all candidate constraints in the full search space. Within this formulation, the original exponential search over V is decomposed into a sequence of restricted subproblems over $V_1, V_2, \dots, V_M$, reducing the overall time complexity from $O(2^{|V|})$ to $O(2^{|V_1|} + 2^{|V_2|} + \dots + 2^{|V_M|})$. Moreover, partitioning the hypergraph into independent connected components bounds the branching factor within each level, mitigating combinatorial explosion as the number of devices increases.

To enable legality checking during tree expansion, each hypergraph node $v \in V$ is mapped to a Boolean selection variable $x_v \in \{0, 1\}$. All hyperedge semantics are encoded in an incremental SAT feasibility model queried throughout the search. A partial search state assigns $x_v = 1$ to selected nodes while leaving all other variables unconstrained. Let $\mathcal{L}_i$ denote the current search level, V_i the corresponding candidate set, and $\mathcal{S}_{<i}$ the fixed context inherited from previous levels. Within $\mathcal{L}_i$, a tree node represents a partial constraint set $\mathcal{S}_i \subseteq V_i$, whose legal action space is defined as:

$$\mathcal{A}_i(\mathcal{S}_i) = \{v \in V_i \setminus \mathcal{S}_i \mid \text{SAT}(\mathcal{S}_{<i} \cup \mathcal{S}_i \cup \{v\}; \mathcal{H}) = 1\}. \quad (8)$$

An action is allowed only if the added candidate satisfies the SAT constraints of the conflict hypergraph. Thus, the tree search is

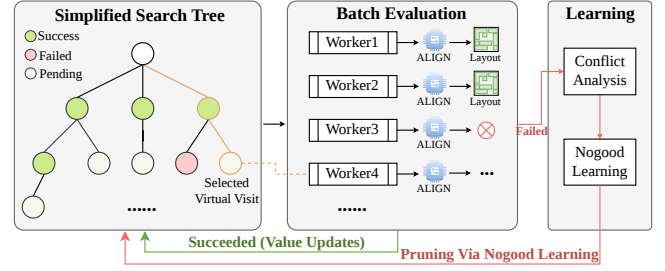


Figure 4. Parallel layout-in-the-loop MCTS.

restricted to the logically feasible domain, preventing candidates violating orientation exclusivity or spatial dependencies from reaching physical evaluation.

4.3.2 Physics-Aware Reward-Guided Tree Policy. At each search level, MCTS explores partial subsets of the constraints. Each tree node represents a partial selection $\mathcal{S}_i$, the root corresponds to an empty set or a fixed inherited context, and each edge represents the decision to activate one additional candidate. Because the quality of a state cannot be determined analytically and must instead be returned by the layout engine, APCG adopts physical reward: a leaf contributes to the tree only after the corresponding constraint subset has been instantiated and evaluated by ALIGN.

During selection, APCG evaluates each node s using a physics-aware reward that jointly considers feasibility, area improvement, and exploration. Let $N(s)$, $N_{\text{succ}}(s)$, and $N_{\text{virt}}(s)$ denote the numbers of completed evaluations, successful layouts, and assigned but unfinished jobs, respectively, with $N_{\text{tot}}(s) = N(s) + N_{\text{virt}}(s)$ as the effective visit count. Let $A(s)$ and $A_{\text{min}}(s)$ represent the average and minimum layout areas in the subtree rooted at s , and A_{base} the baseline layout area. The score is defined as:

$$\Phi_{\text{succ}}(s) = \frac{N_{\text{succ}}(s)}{N_{\text{tot}}(s)}, \quad (9)$$

$$\Phi_{\text{avg}}(s) = \max\left(0, 1 - \frac{A(s)}{A_{\text{base}}}\right), \quad (10)$$

$$\Phi_{\text{min}}(s) = \max\left(0, 1 - \frac{A_{\text{min}}(s)}{A_{\text{base}}}\right). \quad (11)$$

The selection reward is then defined as:

$$R(s) = w_f \cdot \Phi_{\text{succ}}(s) + w_a \cdot \Phi_{\text{avg}}(s) + w_m \cdot \Phi_{\text{min}}(s) + w_e \cdot \sqrt{\frac{\ln(N_{\text{tot}}(pa(s)) + 1)}{N_{\text{tot}}(s) + 1}}, \quad (12)$$

where w_f , w_a , w_m , and w_e control feasibility, average area improvement, best area improvement, and exploration, respectively, while $pa(s)$ denotes the parent node of s in the current search tree. This reward steers the search toward branches that are more likely to generate valid layouts and smaller layout area, while still preserving sufficient exploration of under-sampled candidates. Since area improvement quickly saturates after applying key constraints, the dominant best-area term ($w_m = 0.7$) naturally limits effective search depth by yielding diminishing returns for deeper expansions. During parallel execution, the effective visit count includes virtual visits, so a branch that has already been dispatched to a worker temporarily

receives a lower selection reward, thereby discouraging repeated selection by other workers.

4.3.3 Parallel Physical Evaluation and Failure-Driven Learning. As shown in Fig. 4, the physical layout evaluation of a selected leaf state is executed asynchronously by multiple workers, since ALIGN runtime dominates the overall search cost. Before dispatching a selected root-to-leaf path $\mathcal{P}$ to a worker, APCG temporarily assigns a virtual visit to every node on that path:

$$N_{virt}(u) \leftarrow N_{virt}(u) + 1, \quad \forall u \in \mathcal{P}. \quad (13)$$

Because $N_{virt}(u)$ contributes to $N_{tot}(u)$ in the selection reward, this update immediately decreases the attractiveness of the active branch and encourages idle workers to explore sibling subtrees instead of repeatedly choosing the same node.

Before invoking ALIGN, the selected constraints are serialized using a lightweight structure-aware order. Constraints forming strongly coupled backbone patterns, such as chain-like and star-like structures, are applied first, while weakly coupled or independent constraints are placed later. This ordering improves implementation stability but does not alter the logical search space itself.

Once a worker finishes, APCG performs physical backpropagation along the same path:

$$N_{virt}(u) \leftarrow N_{virt}(u) - 1, \quad N(u) \leftarrow N(u) + 1, \quad \forall u \in \mathcal{P}. \quad (14)$$

If layout generation succeeds and a legal layout is obtained, APCG updates N_{succ} and records the returned physical statistics, including layout area and HPWL, into the corresponding subtree statistics. If the layout fails, APCG converts the failed subset S_{fail} into a blocking clause and injects it into the same SAT model used during expansion:

$$\sum_{a \in S_{fail}} x_a \leq |S_{fail}| - 1. \quad (15)$$

This nogood learning step transforms an expensive physical failure into reusable logical pruning knowledge by permanently excluding failed combinations and their supersets, thereby progressively reducing the effective search breadth and preventing redundant exploration.

5 Experimental Results

5.1 Experiment Setup

The experiments were conducted on a Linux server with an Intel Xeon Platinum 8352Y CPU @ 2.20GHz, two NVIDIA A800 80GB GPUs, and 128GB RAM. The proposed APCG framework is implemented in Python, with the open-source ALIGN framework [5] used as the layout-in-the-loop evaluator. To address the high cost of analog physical evaluation, MCTS was executed in an asynchronous multi-threaded environment with 16 parallel workers, balancing evaluation efficiency and search diversity.

To minimize layout area, the weights in the MCTS formula are set as follows: feasibility weight $w_f = 0.2$, average area weight $w_a = 0.1$, minimum area weight $w_m = 0.7$, and exploration factor $w_e = 1.0$. This weight configuration emphasizes minimum area while preserving feasibility and average area improvement.

To verify that the proposed method does not rely on specific design rules of technology, we performed experiments on three technology nodes: ASAP7 7nm Predictive PDK [19], FinFET14nm

Table 2. Summary of Benchmark Circuits.

Case	Circuit	#Device	#Net	#Sym-pair	#Hyperedge	#Search Level
case1	Five-transistor OTA	5	6	1	1	1
case2	Telescopic OTA	10	10	5	5	5
case3	Cascode Current Mirror OTA	20	9	5	6	3
case4	Double-tail Sense Amplifier	14	10	19	61	3
case5	High-speed Comparator	15	7	19	86	3

Table 3. Layout Quality Comparison on ASAP7 Predictive PDK.

Case	Area (μm^2)		Area Ratio	HPWL		HPWL Ratio	Runtime (s)
	Baseline	Ours		Baseline	Ours		
case1	10.78	8.98	0.833	54452	51768	0.951	26.1
case2	7.55	6.99	0.926	60870	46924	0.771	112.4
case3	32.95	18.55	0.563	151110	111906	0.741	254.5
case4	19.11	17.77	0.930	101420	89356	0.881	3730.8
case5	25.36	17.35	0.684	143636	96252	0.670	973.7

Table 4. Layout Quality Comparison on FinFET14nm Mock PDK.

Case	Area (μm^2)		Area Ratio	HPWL		HPWL Ratio	Runtime (s)
	Baseline	Ours		Baseline	Ours		
case1	25.12	20.99	0.836	39535	39320	0.995	26.1
case2	17.67	16.34	0.924	47725	35697	0.748	170.4
case3	76.42	45.30	0.593	117929	86937	0.737	306.6
case4	47.53	41.34	0.870	81870	67034	0.819	3766.5
case5	58.44	40.35	0.690	109680	73498	0.670	1112.3

Table 5. Layout Quality Comparison on SkyWater SKY130 PDK.

Case	Area (μm^2)		Area Ratio	HPWL		HPWL Ratio	Runtime (s)
	Baseline	Ours		Baseline	Ours		
case1	466.99	392.72	0.841	187420	165070	0.881	28.1
case2	499.84	260.06	0.520	179970	131360	0.730	138.4
case3	1310.25	653.31	0.499	515485	370980	0.720	424.7
case4	1079.49	694.14	0.643	399760	321620	0.805	4132.7
case5	1116.38	684.42	0.613	466500	366620	0.786	1546.8

mock PDK [5], and SkyWater SKY130 PDK [20]. These nodes span advanced FinFET and mature planar technologies with substantially different design rules and routing constraints, enabling evaluation of the method's cross-technology generalization.

5.2 Benchmark Circuits

To comprehensively evaluate the candidate constraint filtering and physical optimization capabilities of APCG across different structural complexities, we selected five representative analog and mixed-signal benchmark circuits from the ALIGN open-source dataset: the Five-transistor OTA, the telescopic OTA, the Cascode current mirror OTA, the Double-tail Sense Amplifier, and the High-speed comparator. These circuits span diverse topologies, from compact differential/current mirror structures to complex comparators with regenerative feedback, cross-branch coupling, and multiple symmetry candidates, enabling systematic evaluation under both simple and highly conflicting scenarios.

TABLE 2 summarizes the key characteristics of the five benchmark circuits, including device count, net count, number of candidate symmetry pairs, number of hyperedges, and the resulting number of search levels after hierarchical decomposition based on CCC. These benchmark circuits reflect increasing levels of constraint search difficulty, spanning a wide range of candidate-pair density, hypergraph complexity, and search-level depth. The OTA circuits mainly evaluate whether APCG preserves essential symmetry under a small

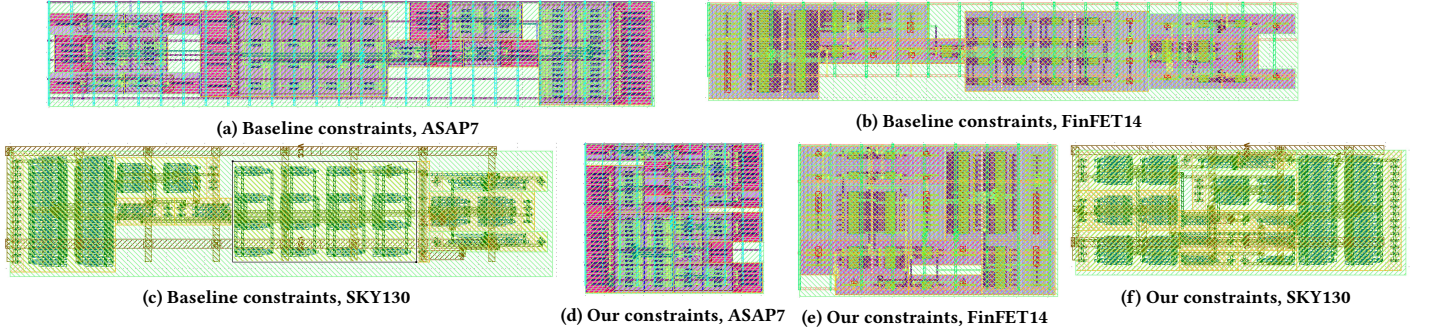


Figure 5. Layout comparison of the High-speed Comparator across three technology nodes.

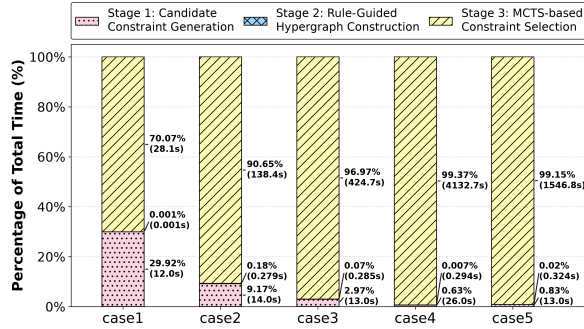


Figure 6. Runtime Breakdown of APCG.

candidate space. In contrast, the Double-tail Sense Amplifier and High-speed Comparator exhibit substantially denser conflicts and a larger number of feasible-yet-competing symmetry choices, which tend to generate more mutually competing candidate constraints and thus are more prone to triggering high-order conflicts during physical implementation. Consequently, these two circuits provide challenging cases for evaluating whether the conflict hypergraph effectively prunes the logical search space and whether layout-in-the-loop MCTS can identify higher-quality constraint subsets.

5.3 Multi-Technology Layout Evaluation

Regarding physical implementation, to ensure that performance differences arise solely from constraint generation and selection strategies rather than backend implementation variations, all experiments use identical SPICE netlists, ALIGN version, and PDK settings. The constraint generation method in ALIGN serves as the primary baseline. We first report quantitative results by comparing the resulting layouts in terms of area and HPWL. Area measures layout compactness and silicon overhead, while HPWL serves as a proxy for interconnect complexity and parasitic sensitivity.

The results presented in TABLE 3 to 5 demonstrate that the proposed method consistently achieves superior layout quality across all three technologies. On ASAP7 PDK, the area ratios for the five benchmark circuits range from 0.563 to 0.930, and HPWL ratios range from 0.670 to 0.951; on the FinFET14nm Mock PDK, the corresponding ranges are 0.593–0.924 and 0.670–0.995; on the SkyWater SKY130 PDK, the area ratios further reach 0.499–0.841, with HPWL ratios between 0.720 and 0.881. Averaged over the five benchmark circuits, APCG reduces layout area by 21.28%, 21.74%, and 37.68%

on ASAP7 PDK, FinFET14nm Mock PDK, and SkyWater SKY130 PDK, respectively; the corresponding average HPWL reductions are 19.72%, 20.62%, and 21.56%. Overall, the improvements are not limited to a specific class of circuits but show consistency across different topological complexities and technology rules.

Furthermore, the improvements are more significant on circuits with complex structures and stronger candidate conflicts. For example, the Cascode Current Mirror OTA and High-speed Comparator achieve substantial area reductions across all three technologies. This is consistent with our design objective: under intensive candidate competition and strong global coupling, prediction alone is often insufficient, whereas hypergraph-guided physical feedback search more effectively eliminates locally feasible yet globally suboptimal constraint combinations.

To further examine how these gains manifest in the actual generated layouts, we qualitatively compare a High-speed Comparator across the three PDKs, whose dense candidate conflicts make the geometric impact of constraint selection more apparent.

Fig. 5 presents a qualitative layout comparison of the High-speed Comparator across three PDKs. Compared with the default ALIGN flow, our generated constraints produce more compact and regular core layouts in all technologies. The relative positioning of the differential branches and the regenerative structure is more stable, and long cross-branch wires as well as local empty regions are significantly reduced. Consequently, the overall bounding box is smaller, and critical interconnects are shorter. These layout improvements align with the area and HPWL gains reported in TABLE 3–5, demonstrating that our method optimizes constraint combinations with tangible physical benefits rather than abstract constraint scores.

It is worth noting that this improvement is not simply due to adding more constraints. The candidates identified by the baseline may appear locally reasonable, but often introduce global congestion or suboptimal placement when combined. In contrast, APCG leverages the conflict hypergraph and layout-in-the-loop search to select physically consistent constraint subsets, balancing layout compactness and interconnect complexity. For the High-speed Comparator, this improved global consistency leads to stable area reduction across all three technologies, with the most significant gain observed in the SkyWater SKY130 PDK.

The runtime of APCG consists of three parts (Fig. 6), where Stage 3 accounts for over 90% of the total runtime. This bottleneck is due to ALIGN layout evaluations, taking 12–99 seconds each. However,

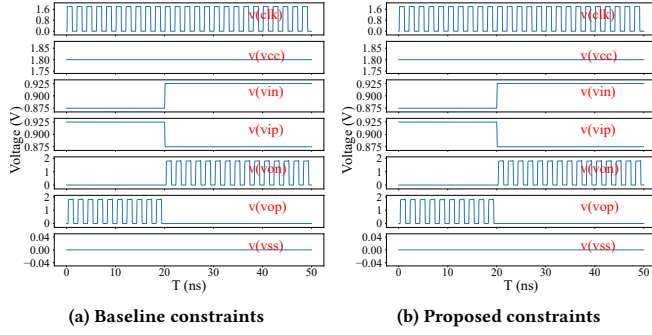


Figure 7. Post-layout functional waveforms of the High-speed Comparator.

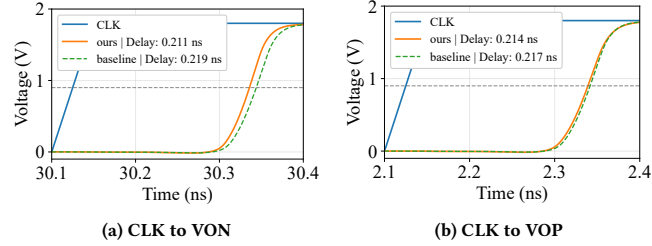


Figure 8. Post-layout delay comparison of the High-speed Comparator under baseline and the proposed constraint set.

Table 6. Post-Layout Electrical Comparison on SkyWater SKY130 PDK.

Metric	Cycle	Baseline	Ours
Delay $T_{clk \rightarrow VOP}$ (ns)	Cycle 1	0.2166	0.2141
	Cycle 2	0.2166	0.2141
	Cycle 3	0.2166	0.2141
	Cycle 4	0.2166	0.2141
	Cycle 5	0.2166	0.2141
Delay $T_{clk \rightarrow VON}$ (ns)	Cycle 1	0.2189	0.2108
	Cycle 2	0.2189	0.2108
	Cycle 3	0.2189	0.2108
	Cycle 4	0.2189	0.2108
	Cycle 5	0.2189	0.2108
Power (mW)	—	0.3943	0.3789

our 16-way parallel framework reduces the amortized cost to 3.4–14 seconds per iteration. Moreover, evaluations are bounded by CCC partitioning, SAT pruning, and the MCTS budget. Consequently, APCG’s runtime remains highly manageable and scalable.

5.4 Post-Layout Simulation

Layout geometry improvements ultimately need to be reflected in electrical behavior. To this end, we conduct post-layout electrical evaluation on the SkyWater SKY130 PDK. We use Magic [21] for parasitic extraction of generated layouts, and the extracted netlists are fed into Ngspice [22] for transient simulation. To rigorously evaluate the impact of layout parasitics, we set the clock frequency of the high-speed comparator to 500MHz; as the clock frequency rises, the available regeneration time decreases, which significantly degrades the resolution and timing margin of the comparator. Therefore, 500MHz achieves a stable balance between maintaining functional correctness and maximizing parasitic observability.

Fig. 7 compares the post-layout functional waveforms of the High-speed Comparator with baseline and APCG constraints. Both layouts maintain correct functionality, demonstrating that APCG achieves more compact layouts without compromising functional correctness.

Delay is measured at the intersection of the CLK rising edge and the corresponding rising edge of the output node VOP/VON at the 50% voltage level (Value/2). For the comparator, delay and power are critical post-layout metrics since the decision path is highly sensitive to branch imbalance and parasitic effects. Therefore, improved symmetry preservation and interconnect organization are reflected in clock-to-output delay and dynamic power. Specifically, we measure $T_{clk \rightarrow VOP}$ and $T_{clk \rightarrow VON}$. Fig. 8 presents the delay results, where the delay points are determined by the intersections of the dashed line with the CLK rising edge and the output waveform. Compared to the baseline, the output nodes in our method reach the decision level earlier, indicating a smaller clock-to-output propagation delay. The average power consumption is calculated as $P_{ave} = V_{CC} \times I_{ave}$, where I_{ave} is the average current of the V_{CC} port within the observation time window. To mitigate the impact of random fluctuations, we sample 5 consecutive clock cycles in the same transient simulation.

TABLE 6 summarizes the post-layout comparison. Compared to the baseline, the $T_{clk \rightarrow VOP}$ of our proposed scheme decreases from 0.2166 ns to 0.2141 ns, a reduction of 1.15%; $T_{clk \rightarrow VON}$ decreases from 0.2189 ns to 0.2108 ns, a reduction of 3.70%; the average power consumption decreases from 0.3943 mW to 0.3789 mW, a reduction of 3.91%. The consistent results over 5 consecutive clock cycles demonstrate that these improvements are attributed to optimized layout parasitics and interconnect organization rather than random fluctuations. Overall, our method achieves a more compact layout with reduced post-layout delay and power overhead.

6 Conclusion

This paper presented APCG, a physically closed-loop framework for automatic analog placement constraint generation. By combining EGAT-based candidate prediction, hypergraph-based modeling of high-order constraint incompatibilities, and MCTS driven by real ALIGN layout feedback, APCG identifies a subset of highly executable constraints that yield superior physical results. Extensive experiments on five benchmark circuits across three technology environments demonstrate that APCG consistently reduces layout area and HPWL relative to the default ALIGN flow, while post-layout evaluations on SkyWater SKY130 further confirm the preservation of correct functionality alongside improved delay and power. Ultimately, these findings demonstrate that integrating learning priors, logical reasoning, and physical feedback in a closed loop represents an effective direction for practical analog layout automation.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62302477); in part by the State Key Lab of Processors, Institute of Computing Technology, CAS (Grant No. CLQ202402, No. CLQX202604); in part by the Shandong Province Natural Science Foundation, China (Grant No. ZR2024MF073); and in part by the Shandong Province Major Scientific and Technological Innovation Projects, China (Grant No. 2025S0101-01954).

References

- [1] G. G. E. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.
- [2] P. Xu, J. Li, T.-Y. Ho, B. Yu, and K. Zhu, "Performance-driven analog layout automation: Current status and future directions," in *Proc. ASPDAC*. IEEE, 2024, pp. 679–685.
- [3] H. Chen, M. Liu, X. Tang, K. Zhu, N. Sun, and D. Z. Pan, "Challenges and opportunities toward fully automated Analog layout design," *Journal of Semiconductors*, vol. 41, no. 11, p. 111407, 2020.
- [4] J. Scheible and J. Lienig, "Automation of analog IC layout: Challenges and solutions," in *Proc. ISPD*, 2015, pp. 33–40.
- [5] 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.
- [6] X. Gao, C. Deng, M. Liu, Z. Zhang, D. Z. Pan, and Y. Lin, "Layout symmetry annotation for analog circuits with graph neural networks," in *Proc. ASPDAC*, 2021, pp. 152–157.
- [7] Z. Wu, I. Song, and I. Savidis, "Hybrid utilization of subgraph isomorphism and relational graph convolutional networks for analog functional grouping annotation," in *Proc. MLCAD*. IEEE, 2023, pp. 1–6.
- [8] Q. Xu, L. Wang, J. Wang, L. Cheng, S. Chen, and Y. Kang, "Graph attention-based symmetry constraint extraction for analog circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 8, pp. 3754–3763, 2024.
- [9] 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. ICCAD*, 2020, pp. 1–8.
- [10] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Universal symmetry constraint extraction for analog and mixed-signal circuits with graph neural networks," in *Proc. DAC*, 2021.
- [11] M. Eick, M. Strasser, H. E. Graeb, and U. Schlichtmann, "Automatic generation of hierarchical placement rules for analog integrated circuits," in *Proc. ISPD*, 2010, pp. 47–54.
- [12] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. ASPDAC*. IEEE, 2020, pp. 193–198.
- [13] Y. Zhang, Y. Yin, N. Xu, and B. Jia, "MCE-HGCN: Heterogeneous graph convolution network for analog IC matching constraints extraction," *Micromachines*, vol. 16, no. 6, p. 677, 2025.
- [14] Y.-C. Tam, E. F. Young, and C. Chu, "Analog placement with symmetry and other placement constraints," in *Proc. ICCAD*. IEEE, 2006, pp. 349–354.
- [15] K. Zhu, H. Chen, M. Liu, and D. Z. Pan, "Automating Analog constraint extraction: From heuristics to learning: (invited paper)," in *Proc. ASPDAC*, 2022, pp. 108–113.
- [16] M. Eick, M. Strasser, K. Lu, U. Schlichtmann, and H. E. Graeb, "Comprehensive generation of hierarchical placement rules for Analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 180–193, 2011.
- [17] N. Karmokar, M. Madhusudan, A. K. Sharma, P. Jariwala, J. Burns, J. Hu, S. S. Sapatnekar, and R. Harjani, "Common-centroid layout for active and passive devices: A review and the road ahead," in *Proc. ASPDAC*, 2022, pp. 114–121.
- [18] M. Hubner and H. T. Vierhaus, "Efficient partitioning and analysis of digital CMOS-circuits," in *Proc. ICCAD*. IEEE, 1992, pp. 280–283.
- [19] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
- [20] S. Technology and Google, "SKY130 process design kit," Available: <https://github.com/google/skywater-pdk>, accessed: 2026-04-14.
- [21] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, and G. Taylor, "Magic VLSI layout tool," Available: <http://opencircuitdesign.com/magic/>, accessed: April 01, 2026.
- [22] N. contributors, "Ngspice: Open source mixed signal/mixed level circuit simulator," Available: <https://ngspice.sourceforge.io/>, accessed: April 01, 2026.