This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2022.3172869, IEEE Transactions on Circuits and Systems II: Express Briefs

Tutorial and Perspectives on MAGICAL: A Silicon-Proven Open-Source Analog IC Layout System

Keren Zhu *Student Member, IEEE*, Hao Chen, Mingjie Liu *Student Member, IEEE*, and David Z. Pan* *Fellow, IEEE*

***Abstract*—MAGICAL is an open-source system for analog and mixed-signal (AMS) circuit layout synthesis. Using custom place-and-route and constraint extraction algorithms, MAGICAL provides a fully-automated layout implementation flow. MAGICAL 1.0 has been proven in silicon with a 40-nm 1GS/s $\Delta\Sigma$ ADC. The source code has been released to enable broad usage. Recently, MAGICAL has also been extended to cover more circuit classes such as SAR-ADC. This tutorial/perspective paper describes the overall MAGICAL framework and algorithms. We also provide a tutorial on how to use and extend MAGICAL and discuss future research directions for AMS layout design automation.**

I. Introduction

Analog/mixed-signal (AMS) integrated circuits (ICs) are essential in many emerging applications, including the Internet of Things (IoT), 5G networks, advanced computing, and healthcare electronics. The layout design for AMS IC has been a heavily manual and error-prone task. Therefore, automating layout synthesis is desired to enhance design productivity.

Layout automation for AMS IC has been an active research area for years [1]–[4]. However, automatic AMS layout tools have not been widely adopted in industrial flow as AMS layout strategies usually do not have a standardized design methodology like that in a digital design flow. An AMS circuit often requires custom layout implementations, and its performance is sensitive to its layout designs. These factors impose challenges on analog design automation. Therefore, specialized physical design algorithms are needed to consider additional requirements in the analog domain. Researchers have proposed alternative methods to handle the problem, such as using existing digital tools for specific digital-like netlists and adopting procedure-based layout generation [3]. For example, FASoc [5] creates many analog standard cells to assist the AMS layout synthesis. BAG [6] develops a standardized interface for procedure-based layout generation [7], [8]. However, these methodologies either are not general-purpose or require a significant amount of manual effort to program the layout templates. On the other hand, optimization-based layout synthesis obtains better flexibility by applying advanced placement and routing techniques on AMS circuits. Recently, researchers have developed optimization-based AMS layout synthesis frameworks, such as MAGICAL [9]–[11] and ALIGN [12]. Leveraging both human and machine intelligence, they have shown very promising results and pushed AMS layout synthesis closer to real-world applications.

MAGICAL is an open-source end-to-end framework for AMS layout synthesis, aiming to generate GDSII layouts from netlists. MAGICAL can be run with no-human-in-the-loop, i.e., it will automatically perform layout constraint generation, then placement and routing. MAGICAL can also be run with user-specified layout constraints and guidance. It could also be run with intelligent simulations in the loop. Over the last few years, we have made tremendous progress on the MAGICAL system, from the initial version with limited capability [9] to the silicon-proven MAGICAL 1.0 release [10]. The MAGICAL 1.0 flow has been verified in silicon tapeout with a TSMC 40nm 1GS/s $\Delta\Sigma$ ADC design. The source codes of MAGICAL 1.0 have been released on Github[1]. Recently, MAGICAL 1.0 has been extended to support SAR-ADC [13] and recently verified in silicon.

This paper aims to give a comprehensive and updated tutorial with perspectives on the MAGICAL system. Although there have been publications related to MAGICAL, they may focus on individual components or algorithms [14]–[17], or outdated [9], [11] or lacking details [10]. Furthermore, existing publications do not provide a tutorial on the MAGICAL release from a user's perspective. This paper further provides guidance on how to use and extend the MAGICAL release.

The rest of this paper will be organized as follows. Section II reviews the key MAGICAL components and algorithms. Section III gives a tutorial on how to use and extend MAGICAL. Section IV provides our perspectives on future research directions in general AMS layout and design automation, followed by conclusion in Section V.

II. MAGICAL Algorithms

This section reviews the MAGICAL algorithms, including the overall flow (Section II-A), automatic constraint extraction (Section II-B), placement engine (Section II-C), and routing algorithms (Section II-D).

*The corresponding author.

K. Zhu, H. Chen, M. Liu and D. Z. Pan are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, TX, USA.

This work is supported in part by DARPA under the IDEA program, NSF under grants 1704758 and 2112665, Nvidia, and Synopsys.

[1]https://github.com/magical-eda/MAGICAL

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2022.3172869, IEEE Transactions on Circuits and Systems II: Express Briefs

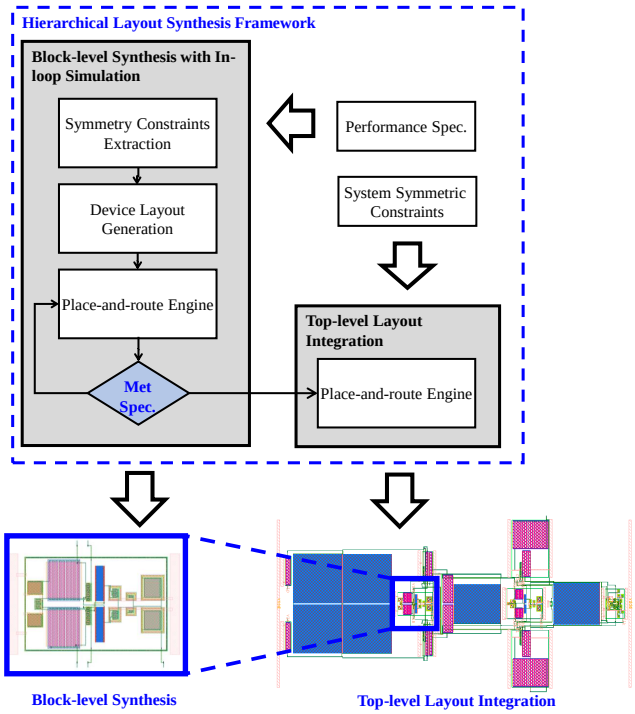


Fig. 1: The MAGICAL hierarchical layout synthesis framework.

A. Overall Flow

MAGICAL is a hierarchical layout synthesis framework. It starts from the bottom-level building blocks (e.g., comparator, amplifier) in a hierarchical circuit and completes the entire layout level by level. Figure 1 illustrates an example of the flow. MAGICAL first synthesizes the block-level sub-circuits. An automatic constraint extractor first analyzes its topology and generates the symmetry constraints for a sub-circuit. Then the sub-circuit is placed and routed under the extracted constraints. After routing the sub-circuit, MAGICAL verifies its post-layout performance if its simulation scripts are given. MAGICAL repeats the layout generation process when the performance specifications are not met. Once all block-level layouts are implemented, they are then integrated into the system-level circuit using placement and routing techniques. The details of MAGICAL's hierarchical layout implementation flow are described in [14].

B. Automatic Constraint Extraction

MAGICAL has two automatic constraint extractors: one for device-level symmetry constraints and one for system-level symmetry constraints. Device-level symmetry constraints target primitive devices (e.g., transistors and capacitors), while system-level constraints are applied to sub-circuit. Automatic constraint extraction for analog circuits has been a fast-growing area in recent years. [18] gives an overview of recent developments in automatic constraint extraction research.

MAGICAL adopts a heuristic algorithm using pattern library and signal flow-based graph traversal for device-level

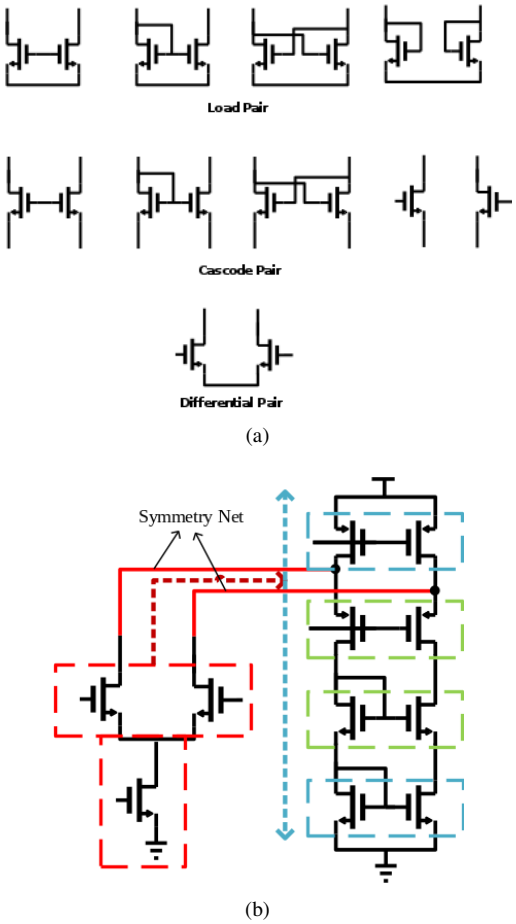


Fig. 2: Device-level constraint extraction illustration. (a) Pattern library. (b) Signal flow-based graph traversal [9].

symmetry constraint extraction. First, an input circuit is converted into a graph. Then, its local structures are compared with a pattern library to generate the seed patterns for symmetry pairs. Finally, graph traversals are performed from the seed patterns to expand the recognized constraints and reduce constraint ambiguity. Figure 2 illustrates the device-level constraint generation algorithms, where Figure 2(a) gives examples in the pattern library, and Figure 2(b) depicts the signal flow-based graph traversal method. More details can be found in [9].

On the other hand, system-level constraint extraction is based on measuring the graph similarity between two sub-circuits. The entire netlist is first converted to a graph. Sub-circuit pairs in the same circuit hierarchy are then checked one by one. The sub-circuits and their neighboring structures are extracted into sub-graphs from the graph. The constraint extractor compares the two sub-graphs and measures their topological similarity. A symmetry constraint is labeled if the graph similarity is larger than a threshold. The graph similarity metric is computed by a statistical method. The eigenvalues of the graph Laplacians are computed, and Kolmogorov-Smirnov (K-S) tests are used to score the graph similarity between two eigenvalue vectors. More details of the MAGICAL device-level symmetry constraints can be found in [15].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2022.3172869, IEEE Transactions on Circuits and Systems II: Express Briefs

C. Analytical AMS Placement

MAGICAL's placement engine is used for building-block-level layout generation and system-level integration. It includes a non-linear programming-based global placer and a linear programming-based legalizer. It honors the symmetry and self-symmetry constraints while optimizing for other objectives.

Global placement determines a rough position for each instance and overall floorplan. During global placement, a non-linear objective function is iteratively optimized over the positions of instances. Optimizing the cost function minimizes the objectives, such as wirelength, and the penalties, such as overlapping. The objectives include wirelength and system-signal flow. The penalties include overlapping, asymmetry, and violation of direct current flow. These different costs are optimized together in a weighted cost function. The weights of penalties are increased over the iterations to ensure the global placement results are closed to legal solutions.

After global placement, the resulting placement is further legalized and optimized. With a linear programming-based legalization procedure, instance positions are adjusted to enforce non-overlapping and symmetry constraints. The linear programming-based legalization maintains the relative positions between instances from the global placement results. The legalization process is performed twice for each placement: The first execution legalizes the placement to minimize the area. The second execution further optimizes the wirelength without increasing the area.

In MAGICAL 1.0, system-signal flow is an optional input to the placement engine. Parasitic RC and couplings from the layout on the critical system signals can change the behavior of the circuits and degrade post-layout performance. A typical layout strategy to mitigate this issue is to maintain the regularity of system signal flow in the placement stage. Figure 3 illustrates an example of system signal flows on a continuous-time $\Delta\Sigma$ modulator (CTDSM) design. There are critical forward and feedback signal paths in the schematic design (Figure 3(a)). In an ideal floorplan, the signal paths are regular in a "schematic-like" manner (Figure 3(b)). The placement engine considers the system signal flow in the global placement.

More details of the MAGICAL placement algorithm can be found in the paper [17].

D. AMS Detailed Routing

AMS detailed routing finishes the interconnections while considering geometrical and electrical constraints. Geometrical constraints in the MAGICAL router include four variants of symmetry constraints: mirror-symmetry, cross-symmetry, self-symmetry, and partial symmetry. Figure 4 illustrates these symmetry constraint variants.

The core detailed routing process consists of three phases. First, pre-processing is applied to detect the unspecified symmetry constraints and model the geometric metal shapes into access point sets. Then, the core routing engine connects the nets. Finally, the remaining design rule violations are corrected in the post-refinement phase.

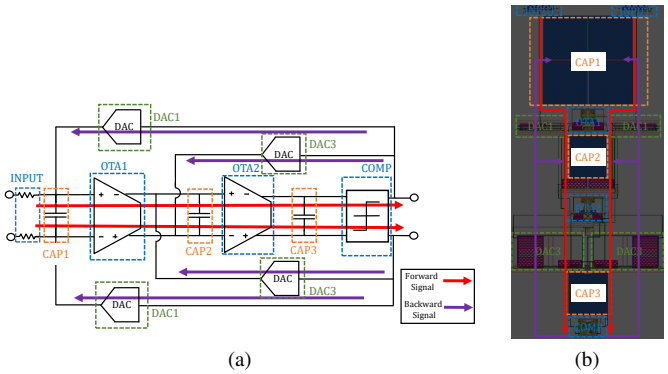


Fig. 3: An illustration of system-signal flows in a CTDSM circuit. (a) The schematic. (b) A layout implementation with regular system-signal flows [17].

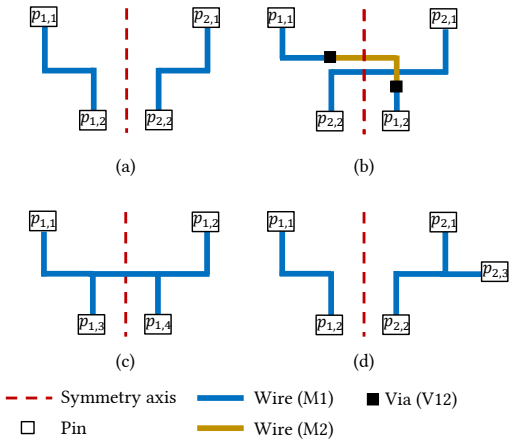


Fig. 4: Examples of the four symmetry constraint variants. (a) Mirror-symmetry. (b) Cross-symmetry. (c) Self-symmetry. (d) Partial-symmetry [16].

The pre-processing phase first analyzes the pins in the placed layouts. In addition to the given symmetry constraints, it also identifies the symmetric pins in the layouts. Those symmetric pins are also routed symmetrically later. The pre-processing phase also analyzes the pin shapes and determines the access points, which benefit the pin access in the later path-searching procedure.

The core routing phase is based on an obstacle-aware path-finding algorithm. It searches the feasible routing paths connecting the pins. Symmetry is considered by simultaneously performing obstacle-aware path-finding on both sides of the symmetric axis. Design rules, including parallel run spacing, end-of-line spacing, min-area, and min-step, are also checked in the path-finding process. When no feasible routing solution is found, the detailed router starts a rip-up and reroute optimization loop until reaching a feasible solution.

Finally, post-processing checks the remaining design rule violations of the routing solution. Min-step violations are fixed by adding patches to the metal shapes. More details are described in [16].

III. How to Use and Extend MAGICAL

This section gives a tutorial on using and further extending the open-sourced MAGICAL software. Selected results of the MAGICAL 1.0 system are presented in Section III-C. The following tutorial is based on the MAGICAL 1.0 release [2].

A. Using MAGICAL

This section covers the tutorial for MAGICAL from a user's perspective.

Setting-up the software: MAGICAL is developed on the Linux operating system. To the best of our knowledge, the necessary dependencies of the MAGICAL system are all open-sourced. To compile MAGICAL from scratch on a Linux system, a user can refer to `README` and `Dockerfile` files under the root of the MAGICAL directory. An alternative way is to use the Docker image [19]. We provide a complete Docker image installed with the MAGICAL tool and its dependencies. A user can use the tool out-of-box using the Docker image. The `README` file contains the instruction to use the Docker image.

Configuring the parameters: MAGICAL 1.0 is currently developed for TSMC 40nm technology. Due to the non-disclosure agreement, the publicly released MAGICAL is sanitized and uses some fake parameters related to the technology. A user can edit `flow/python/Params.py` and `device_generation/glovar.py` to correct those parameters.

Running the examples: MAGICAL 1.0 release contains several example circuits under the directory `examples` for users' reference. After installing the MAGICAL tool, a user can execute the `run.sh` script to run an example. The example benchmark can generate a layout from the input netlist.

Applying to new designs: A user to refer to the examples for the input files. The MAGICAL takes a JSON file for the input parameters and files. The essential input files include a spectre or hspice netlist and several technology files. The MAGICAL release provides an example set of technology files in the `examples` directory for the technology files. There are several optional input files. A `sym` file specifies the placement symmetry constraints. A `symnet` file specifies the routing symmetry net constraints. The MAGICAL tool will automatically generate the `sym` and `symnet` files if the user does not give them. The user can use the optional symmetry constraint files to override the automatically extracted constraints. On the other hand, the system signal flow can not be automatically extracted. The user can use a `sigpath` file to specify the system signal flow. There are examples of these optional input files in the examples named `adc1` and `adc2`.

Standard cells or other IP cells can also be used in MAGICAL instances. To add a new standard cell, the user needs to add the name of standard cells to `flow/python/Params.py`. Then the GDS layout and its pin locations need to be given to the tool. There are examples of using the digital standard cells in the examples of `adc1` and `adc2`. Note that those standard cells in the public release are sanitized.

[2]https://github.com/magical-eda/MAGICAL/releases/tag/v1.0.2

B. Extending MAGICAL

This section intends to give software developers a tutorial on extending the MAGICAL tool further.

Software structure and new feature development: MAGICAL is written in Python and C++. It consists of five modules: device generation, constraint generation, placer, router, and hierarchical flow integration. The first four modules are developed as standalone sub-modules, and each of them can function individually. At the same time, they all provide a Python interface for easy integration with other programs. The MAGICAL flow is for integrating the sub-modules into a complete end-to-end framework. It accesses other sub-modules through their Python interfaces.

The ideology of MAGICAL software structure is to expose the Python parts to the users and develop the computation-intense parts in C++. For computational efficiency, most data structures and core optimization schemes are written in C++. We use Pybind11 [20] to implement the Python interfaces for these C++ modules. We encourage the software developers to follow a similar approach. A developer can change the existing or extend the Python interfaces of the sub-modules exposed to the top-level flow scripts. The users can adjust the programming language based on their needs.

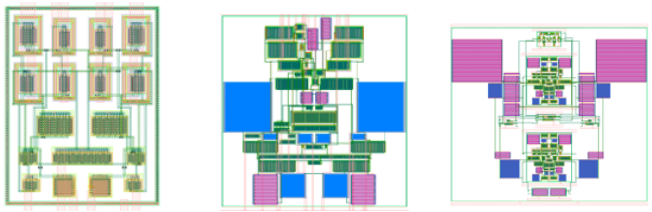
Migration to other technologies: Porting MAGICAL to another technology can be non-trivial. MAGICAL 1.0 is currently developed for planar CMOS technology. The device generation and technology configuration need to be changed to migrate MAGICAL to another planar CMOS technology. The device generation module generates the correct layout of devices, such as transistors and capacitors, from the netlists. The device generation needs to be correctly changed according to the technology. The engineering efforts may vary depending on the technology.

Another challenge is on the design rules. The design rules differ in different technologies. The configuration, such as metal width and spacing, can be edited in the file `flow/python/Params.py`. The additional design rules in placement or routing, on the other hand, might need modification on the placer and router modules.

C. Selected MAGICAL 1.0 Results

The MAGICAL system overall provides a fully-automated solution to AMS layout synthesis. MAGICAL can generate the layouts in seconds to minutes while providing competitive post-layout performance. Table I shows the comparisons between the MAGICAL generated and manual layouts of a comparator and an operational amplifier. The MAGICAL 1.0 reduces signal mismatching in `OP` and obtains better offset and CMRR than the manual layout. The delay and area overheads for the `COMP` mainly come from the individual wells. The shared well islands algorithm is being developed to optimize the placement [21]. The post-layout simulation results demonstrate the capability of the MAGICAL system.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2022.3172869, IEEE Transactions on Circuits and Systems II: Express Briefs



(a) (b) (c)

Fig. 5: Examples of MAGICAL 1.0 Generated layouts. (a) A comparator. (b) An inverter-based feed-forward OTA. (c) A second-order CTDSM.

Furthermore, the 40nm CTDSM ADC tape-out measurement results show that the MAGICAL system can produce similar performance, area, and power compared to manually optimized layouts. The measured SNDR and SFDR of 67.4 dB and 80.8 dB of the MAGICAL layout are close to the manual layout of 65.6 dB and 82.6 dB. The detailed measurement results can be found in [10]. More post-layout simulation results of MAGICAL 1.0 can be found in the papers [16], [17]. Figure 5 shows more examples of MAGICAL 1.0 generated layouts.

TABLE I: Comparisons of Manual and MAGICAL Generated Layouts.

Circuits		Schematic	Manual Layout	MAGICAL 1.0
COMP	Delay (ps)	69	99	122
	Offset (μV)	-	0.122	0.233
	Noise (μV_{rms})	372	380	362
	Area (μm^2)	-	51	156
	Runtime	-	Hours	9.5s
OP	DC Gain (dB)	38.2	37.0	37.9
	Bandwidth (MHz)	110.5	110.0	108.2
	Phase Margin (°)	64.7	67.8	63.3
	Offset (μV)	-	0.20	0.01
	CMRR (dB)	-	103.0	184.7
	Area (μm^2)	-	2550	2400
	Runtime	-	Hours	14.2s

IV. Perspectives On Future Directions

This section gives our perspectives on MAGICAL and the general research in automating AMS layout synthesis. It also discusses the future research and software development directions in the field.

A. Open-source Software Development

MAGICAL focuses on expanding the user base and participating in the open-source hardware community.

MAGICAL has demonstrated silicon-proven success. However, there are still challenges to making it cover a larger user base and provide consistent and high-quality results. A major issue we observe is that the users are using different technologies. It is difficult for us to obtain access to many technology PDKs, and there are large amounts of engineering efforts to migrate MAGICAL to new technologies. Therefore, the users sometimes need to make significant development efforts to use the MAGICAL tool, especially those unfamiliar with software development.

We are currently closely collaborating with the open-source hardware community and the MAGICAL users to expand our capability. The ongoing actions include developing toward the open-source Skywater 130nm PDK [3] and making MAGICAL more technology-agnostic based on the users' feedback.

B. MAGICAL for Advanced FinFET Technologies

Another ongoing development target for MAGICAL is on supporting the advanced FinFET technologies. In FinFET, the layout design methodology differs from the conventional planar CMOS technologies. The FinFET layouts are more regular in a grid structure and have more strict design rules from the layout automation perspective. Instead of directly adapting the current MAGICAL to FinFET technology, our vision predicts AMS layout synthesis for FinFET technology needs a distinctive design methodology.

We are currently actively researching the AMS layout synthesis for FinFET technologies. Our ongoing research for the FinFET AMS layout system has a region-based and gridded style [22]. It includes significant changes are placement, routing, and device generation.

C. Machine Learning in AMS Layout Synthesis

Incorporating more Machine learning (ML) and artificial intelligence (AI) into AMS layout synthesis system is an important direction in the field. ML for AMS layout synthesis is an active research area, from placement to routing, to performance/parasitic predictions [21], [23]–[29]. Adopting ML techniques allows automatic tools to learn from human layout [9], [15], [21], [23], [30] and makes optimization more efficient [14], [26], [28]. In addition to applying ML, an open question to answer is how to make the AI-assisted design automation frameworks reliable and robust.

We are currently actively researching and developing ML-assisted MAGICAL. Our ongoing exploration includes robust ML models and effective ML-assisted layout optimization techniques.

V. Conclusion

The paper presents a comprehensive and updated tutorial/perspective on MAGICAL, a silicon-proven open-source analog IC layout system. The overall flow and algorithms in MAGICAL are described, together with how to use and extend MAGICAL. We also present our perspectives on the future directions of automatic AMS layout synthesis and open-source software development. Although analog layout design automation is still far behind its counterpart of digital IC design automation, we believe that with the advancement of computing power and AI, coupled with the stringent demand of design productivity and turn-around-time, analog design automation shall receive more and more attention and adoption.

[3] https://github.com/google/skywater-pdk

REFERENCES

[1] R. A. Rutenbar, "Analog circuit and layout synthesis revisited," in *ACM International Symposium on Physical Design (ISPD)*, 2015, p. 83.

[2] M. P.-H. Lin, Y.-W. Chang, and C.-M. Hung, "Recent research development and new challenges in analog layout synthesis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2016.

[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. 20070021, p. 111407, 2020.

[4] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and R. L. Carley, *Analog Device-Level Layout Automation*, 1st ed. USA: Springer US, 1994.

[5] T. Ajayi, Y. Cherivirala, K. Kwon, S. Kamineni, M. Saligane, M. Fayazi, S. Gupta, C.-H. Chen, D. Sylvester, D. Blaauw, R. Dreslinski, B. Calhoun, and D. D. Wentzloff, "Fully Autonomous Mixed Signal SoC Design & Layout Generation Platform," in *2020 Hot Chips: A Symposium on High Performance Chips*, 2020.

[6] J. Crossley, A. Puggelli, H. . Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon, "Bag: A designer-oriented integrated framework for the development of ams circuit generators," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 74–81.

[7] J. Crossley, A. Puggelli, H.-P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon, "BAG: A designer-oriented integrated framework for the development of ams circuit generators," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.

[8] 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 *IEEE Custom Integrated Circuits Conference (CICC)*, 2018, pp. 1–8.

[9] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[10] H. Chen, M. Liu, X. Tang, K. Zhu, A. Mukherjee, N. Sun, and D. Z. Pan, "MAGICAL 1.0: An open-source fully-automated ams layout synthesis framework verified with a 40-nm 1 GS/s $\Delta\Sigma$ ADC," in *IEEE Custom Integrated Circuits Conference (CICC)*, 2021.

[11] H. Chen, M. Liu, B. Xu, K. Zhu, X. Tang, S. Li, Y. Lin, N. Sun, and D. Z. Pan, "MAGICAL: An open-source fully automated analog IC layout system from netlist to GDSII," *IEEE Design & Test*, 2020.

[12] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[13] M. Liu, X. Tang, K. Zhu, N. Sun, and D. Z. Pan, "OpenSAR: An open source automated end-to-end SAR ADC compiler," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021.

[14] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[15] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S^3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2020.

[16] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Toward silicon-proven detailed routing for analog and mixed signal circuit," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[17] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[18] K. Zhu, H. Chen, M. Liu, and D. Z. Pan, "Automating analog constraint extraction: From heuristics to learning," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2022.

[19] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[20] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 — seamless operability between c++11 and python," 2016, https://github.com/pybind/pybind11.

[21] K. Zhu, H. Chen, M. Liu, X. Tang, W. Shi, N. Sun, and D. Z. Pan, "Generative-adversarial-network-guided well-aware placement for analog circuits," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2022.

[22] J. Han, W. Bae, E. Chang, Z. Wang, B. Nikolić, and E. Alon, "LAYGO: A template-and-grid-based layout generation engine for advanced cmos technologies," *IEEE Transactions on Circuits and Systems I*, vol. 68, no. 3, pp. 1012–1022, 2021.

[23] K. Zhu, M. Liu, Y. Lin, B. Xu, S. Li, X. Tang, N. Sun, and D. Z. Pan, "GeniusRoute: A new analog routing paradigm using generative neural network guidance," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[24] M. Ahmadi and L. Zhang, "Analog layout placement for FinFET technology using reinforcement learning," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021.

[25] 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 *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2020.

[26] R. Martins, N. Lourenço, R. Póvoa, and N. Horta, "On the exploration of design tradeoffs in analog IC placement with layout-dependent effects," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019.

[27] 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 *IEEE Annual Symposium on VLSI (ISVLSI)*, 2020.

[28] ——, "A customized graph neural network model for guiding analog ic placement," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[29] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, and D. Z. Pan, "WellGAN: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[30] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2021.