Reinforcement Learning-Enhanced Cloud-Based Open Source Analog Circuit Generator for Standard and Cryogenic Temperatures in 130-nm and 180-nm OpenPDKs

Ali Hammoud[1], Anhang Li[1], Wen Tian[1], Ayushman Tripathi[1], Karthik Lakshmanan[1]
Harsh Khandeparkar[1], Ryan Wans[1], Gregory Kielian[2], Boris Murmann[3], Dennis Sylvester[1], Mehdi Saligane[1]
[1] University of Michigan, Ann Arbor, MI, USA
[2] Google AI, San Francisco, CA, USA
[3] University of Hawai'i, Honolulu, Hawaii, USA

***Abstract*—This work introduces an open-source, Process Technology-agnostic framework for hierarchical circuit netlist, layout, and Reinforcement Learning (RL) optimization. The layout, netlist, and optimization python API is fully modular and publicly installable. It features a bottom-up hierarchical construction, which allows for complete design reuse across provided PDKs. The modular hierarchy also facilitates parallel circuit design iterations on cloud platforms. To illustrate its capabilities, a two-stage OpAmp with a 5T first-stage, common-source second-stage, and miller compensation is implemented. We instantiate the OpAmp in two different open-source process design kits (OpenPDKs) using both room-temperature models and cryogenic (4K) models. With a human designed version as the baseline, we leveraged the parameterization capabilities of the framework and applied the RL optimizer to adapt to the power consumption limits suitable for cryogenic applications while maintaining gain and bandwidth performance. Using the modular RL optimization framework we achieve a 6x reduction in power consumption compared to manually designed circuits while maintaining gain to within 2%.**

***Index Terms*—Open-Source, Generator, AMS, PDK agnostic, Cloud, Cryogenic, Reinforcement learning.**

I. Introduction

A. Background and Motivation

The surging demand for Analog and Mixed Signal (AMS) design automation and open-source reproducible chip design, underscored in recent works [1], [2], reflects the growing need for automated generators to help build a common infrastructure for the open-source chip design ecosystem. Template generators have been applied to AMS circuits with acceptable upfront engineering effort and high-performance design, such as [3], [4]. However, these approaches are not suitable for some analog cells where a higher degree of complex, non-pattern layout optimization is required. Programmatic approaches can be applied where template approaches fall short.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
http://dx.doi.org/10.1145/3676536.3676823

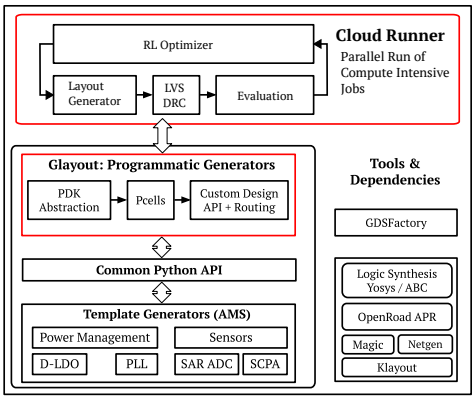


Fig. 1: Overview of this work with key contributions highlighted in Red.

Tools such as [5], [6] enable the automated construction of such analog blocks and demonstrate high-performance, but they require a large design effort to port to new technologies and are not demonstrated with open-source technologies [7]. Tools like [8] attempt to fulfill the open-source ethos of reproducibility in addition to reported silicon results with template based generators. This work seeks to build upon the aforementioned foundation, addressing the short-comings of template approaches, by providing a programmatic framework with the following objectives:

- **Open Development & Common Infrastructure:** Generators are constructed with open-source tools and community support to build a design library. The modular python interface available publicly on PyPI makes it easy to interface with other tools, develop add-ons and contribute.
- **Cloud-based deployment:** This project utilizes existing multi-platform parallelization libraries, Python tools, and package managers, allowing for easy installation and cloud deployment.
- **Easy to port between PDKs:** The project aims to provide a layer of abstraction for planar technologies, making it easy to add support for additional technologies. Once new technology layers, rules, and models are provided, the existing generators can run fully automatically end-to-end with 100% code reuse.

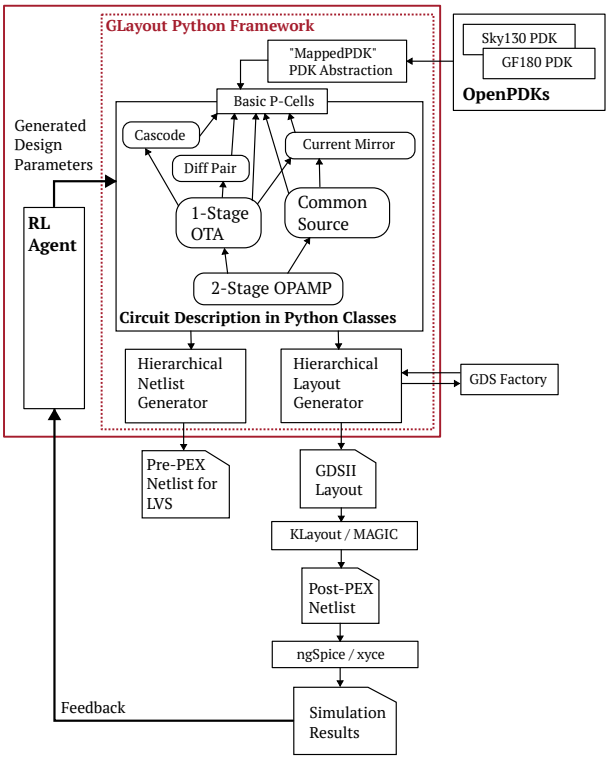


Fig. 2: Detailed View of the Framework and Workflow

Glayout, the open-source Python framework included in the OpenFASOC GitHub repository [8] as summarized in Fig. 1, is detailed in this work. OpenFASOC includes both template and programmatic analog generator tools (as described here) for a holistic approach. The key elements of this work, highlighted in Fig. 1 includes the Glayout Programmatic Generator and the Reinforcement Learning Optimizer. These tools are all modular, allowing for easy importing and deployment on cloud infrastructure.

B. Proposed Framework

This work is comprised of two components:

1) GLayout: A Programmatic, PDK Agnostic Analog Layout & Netlist Generator
2) RLOpt: A Reinforcement-Learning Circuit Design Optimizer

The workflow for using our framework is illustrated in Fig. 2. With GLayout, the user is able to create hierarchical parameterized layouts. The circuit that is generated can then be extracted and simulated automatically within the Python environment. The RLOpt module uses an RL algorithm to generate a new set of parameters using the simulation results. This process of parameter generation, layout, extraction, and simulation is repeated until target specifications are met.

Our infrastructure is demonstrated with the complete design process of an OpAmp and integration into nanofabrication test tiles targeting cryogenic and room temperature applications. For comparison, our baseline, manually designed OpAmp has a 3mW power consumption which is not suitable for cryo applications. The RLOpt module produced designs which met this spec while trading off bandwidth and noise performance. This demonstration is reproducible from the code available on GitHub [8] and the 130nm tile has been submitted for tapeout.

TABLE I: Generic Layers abstracts technology-specific layers.

	Generic Layers	SKY130	GF180
FEOL	**Polysilicon**	(66, 20)	(30, 0)
	n-select	(93, 44)	(32, 0)
	...	...	...
BEOL	**Metal 1**	(67, 20)	(34, 0)
	Via 1	(67, 44)	(35, 0)
	...	...	...

II. GLayout: A Generic Layer for Abstracted Layout & Netlist Generation

Glayout is installable on Linux platforms from the PyPI package manager. GLayout offers tools for generating the layout file (GDSII) and the pre-parasitic extraction (pre-PEX) netlist.

A. MappedPDK: Technology Abstraction

As depicted in Fig. 2, GLayout interfaces with a Process Design Kit (PDK) through the "MappedPDK" class. MappedPDK additionally calls the DRC-rule checker, which can verify rule compliance at any level of hierarchy. Adding support for a new technology only requires implementing a MappedPDK object. This involves providing layer and rule mappings (described below), and simulation model files. To aid in reproducibility, Glayout comes pre-packaged with open-source 130nm and 180nm PDK support.

1) Generic Layers: CMOS technologies generally require similar layers across PDKs, some of which are: "active/diffusion", "p-select", "n-select", "metal contact", "metal 1", "via1", ... etc. Table I illustrates that even though different Process Design Kits (PDKs) have similar layers, they are named and represented differently. For instance, the Front-End-of-Line (FEOL) layers that define transistors may have similar functions with different representations. MappedPDK offers a unified representation of layer functionality, which other software components utilize as an abstraction.

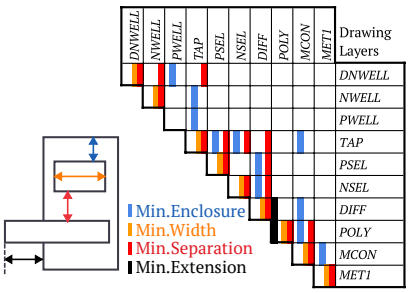


Fig. 3: Generic Rules are stored as an undirected graph for easy lookup. Edges store rules, while vertices store generic layers.

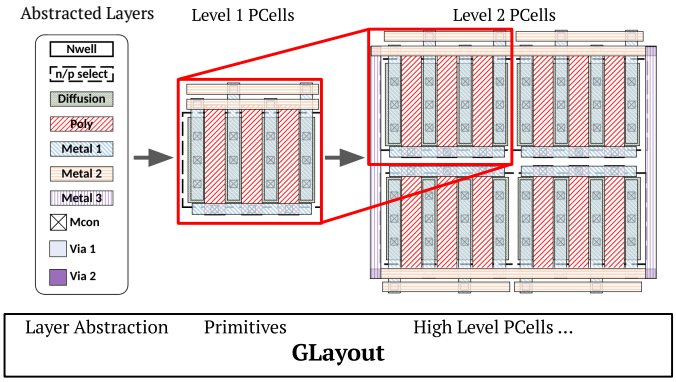


Fig. 4: Hierarchical Design Flow with Strong Parameterization Capability

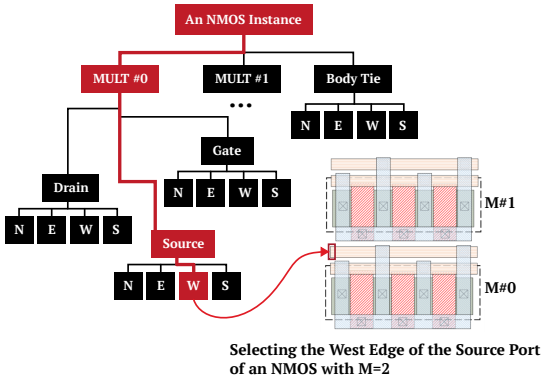


Fig. 5: Port tree of a Multiplier = 2 Transistor, The Selected Port is Highlighted. Port trees organize port names which allows the programmer to specify point to point routing in Python code

2) Generic Rules: DRC rules are mapped as a graph structure. Rule decks typically consist of three similar rules: "*min_separation*", "*min_enclosure*", and "*min_width*" (or "*width*" for via layers). By prescribing one of these three rules between combinations of layers, hundreds of rules can be created. Additionally, there can be "self rules" that describe a requirement between a layer and itself; most "*min_width*" rules are self-rules. Fig. 3 shows the rule matrix implemented in MappedPDK. Note that the context-dependent rules (i.e. lambda rules) are omitted from this simplified checker. By taking the worst-case value for each rule, MappedPDK enables rule lookup with limited required context of surrounding layer geometry.

B. A Hierarchical Design Approach

1) High-Level Parameterized-Cells (PCells) : All circuit designs in this software framework are created using a bottom-up approach, combining hierarchy and context-conscious routing at each level to produce clean layouts and increase complexity. The generic framework provides a library of basic cells (MOSFETs, capacitors, etc) using the MappedPDK API. Those generic implementations ensure standardized layouts and features so that high-quality parameterized primitives can be built on top layer-by-layer. Fig. 4 illustrates how complex structures are constructed by combining simpler primitives in a layered manner.

2) Indexing of Terminals and Routing: Ports are polygon edges. Glayout routing functions create paths between ports. These routing functions are context-insensitive so are referred to as "dumb routes". The DRC checker is used to ensure their accuracy. Larger designs with complicated hierarchies tend to include thousands of ports, so port organization is key. This framework uses a syntax that describes a "PortTree", as shown in 5. These trees are implemented as a Python class and store port names for easy lookup. To ensure legal construction, routing over blocks which are lower in the hierarchy is avoided if possible.

3) Layout & Netlists Generation: The framework includes functions that enable automatic traversal in the hierarchy and creation of the GDSII layout file. Additionally, it is necessary to have a pre-PEX netlist for layout-versus-schematic (LVS) checks. The framework also provides APIs to generate this netlist automatically.

III. RLOpt: A Reinforcement Learning Optimizer

TABLE II: Comparison of optimizers tools. Optimized specifications are soft constraints and only optimized after hard (capped) constraints are met.

Tool	Specification	PEX Sim	Example	Method
[9]	Capped+Optimized	no	TIA	DDPG
[10]	Optimized	no	LDO, TIA	DDPG
[11]	Capped	yes	OpAmp	PPO
[12]	Capped	yes	OpAmp	DNN
RLOpt	Capped+Optimized	yes	OpAmp	PPO

Machine Learning (ML) approaches to circuit optimization, such as [13] and [14], conceptualize automatic analog sizing as an optimization problem and employ Bayesian optimization (BO) techniques to efficiently search for sizing solutions. Works like [9], [11], [15] use RL, where the circuit simulator serves as an environment that the RL agent interacts with. These approaches train specialized RL agents using algorithms like deep deterministic policy gradient (DDPG) [16]. RL-inspired methods [17], [18] use an RL actor-critic method [19] to strategically combine the strengths of RL, BO, and population-based techniques [17]. As a result, they can meet specified performance constraints while minimizing the number of required simulations (and compute time).

The overview of our framework is shown in Fig. 6, and Algo. 1. In our framework, the RL agent observes the specifications returned by the post-layout simulation environment and decides an action for each layout parameter based on its learned policies. After receiving the new set of parameters, the simulation environment provides a new set of specifications for the RL agent to calculate the reward associated with the specific action taken.

In this process, we have two types of specifications: capped specifications (cS) and optimized specifications (oS). The

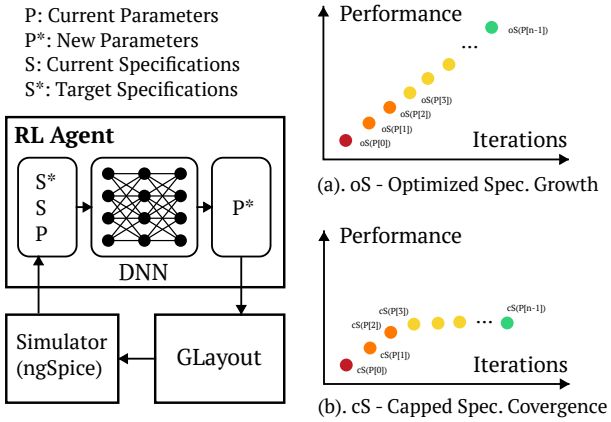


Fig. 6: The RL Agent's position in the optimization loop is shown, along with the agent-generated sequence of actions/parameters for both capped and optimized specifications.

Algorithm 1 Reinforcement Learning Training Algorithm

Initialize $S*$ //create targets for training
$env \leftarrow$ Layout generator and simulation
while $\sum \frac{Reward}{L} < 6$ **do**
  $Dataset \leftarrow []$
  $Reward \leftarrow []$
  for $len(batchsize)$ **do**
    $s \leftarrow S*$ //pick one set of target specs
    $Trajectory \leftarrow []$
    for $horizon$ **do**
      Initialize s, p
      if $s >= s*$ & $reward > last\ reward$ **then**
        $t \leftarrow nextStep(env, s, s*, p)$
        $Trajectory \leftarrow append(t)$
        $s, p \leftarrow env(s, s*, p)$
      else
        break
      end if
    end for
    $Dataset \leftarrow append(Trajectory)$
    $Reward \leftarrow reward(Trajectory)$
    $PPO\_Update(Dataset, Reward)$
  end for
end while

goal for cS is to reach our target specification and then stop growing, while for oS, we aim to maximize their growth. As illustrated in Fig. 6, the RL agent generates a trajectory with n steps. Once the cS reach the target specification, the oS continues to be optimized as much as possible. This dynamic reflects the RL agent's ability to adapt and prioritize different types of specifications during the trajectory generation process

To provide the RL agent with a comprehensive understanding of various regions within the design space, we generate 100 different target specifications for the training process. During the training process, we generate L trajectories that comprise multiple sequential environment steps, each with targets selected from the set of M specifications. At each step, rewards are accumulated until the predefined predetermined maximum number of steps is reached or the capped specifications are reached and optimized specifications are not getting higher. This accumulation is computed as the unweighted sum of the bounded normalized differences between each specification, denoted as 'oS, cS' and their corresponding target/global value, represented as 'oS^*, cS^*'.

$$r_{cS} = \min(\frac{cS - cS*}{cS + cS*}, 0), \qquad r_{oS} = (\frac{oS - oS*}{oS + oS*}) \tag{1}$$

$$r = \sum r_{cS} + \sum r_{oS} \tag{2}$$

In the agent's reward scheme, it doesn't receive any reward for excessively surpassing any cS target. If the r_{cS} exceeds the target, it is effectively clipped to zero in such instances, ensuring that the agent is not rewarded for excessive performance in a single metric while neglecting others. However, for oS, the agent will receive rewards for achieving excessive performance, as this aligns with the optimization objectives for these metrics.

The accumulation of rewards usually serves as a measure of the agent's performance throughout the trajectory. After executing multiple trajectories with varying specifications, the neural network undergoes updates using policy gradient techniques. This updating process is aimed at maximizing the expected cumulative reward, which enhances the agent's ability to make optimal decisions in future interactions with the simulation environment.

Training sessions are executed repeatedly to ensure the robustness of the RL agent. The training process is completed when the mean reward reaches or surpasses a value of 6. This threshold indicates that the agent has effectively achieved the capped specifications and has also made progress in optimizing the optimized specifications for several additional steps. A summary of this training procedure is in Algo. 1.

IV. Experimental Validation: A Two-Stage OpAmp

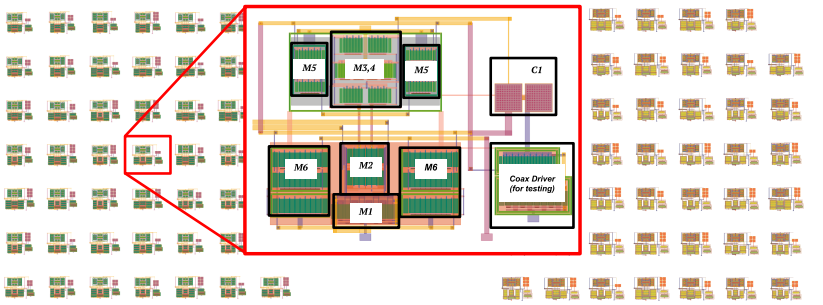
A. Design Specification & Baseline

In this section, a two-stage OpAmp targeting a specific tapeout is presented. The goal is to fabricate an array of around 50 OpAmps with many different flavors (Absolute highest UGB / Absolute lowest Power / High FoM / Different Temperature, etc.) to make this wafer suitable for various applications by post-fabrication metallization. This task is challenging for traditional approaches due to the vast number of simulations and human hours required to finalize all the designs.

As shown in Fig. 8. The OpAmps are paired with probe pads for the nano-fabrication (NanoFab) experiment which target both cryogenic and room temperature use. The cryogenic models used are open-source Cryogenic transistor model set in 130nm targeting 4K [20]. The human designer produced a

TABLE III: Human-designed OpAmp vs machine-designed OpAmp runs aimed at finding low power and high performance tradeoff

	Human	Machine Shotgun	Itr. 1	Itr. 20
PM (deg)	54	66	83	82
Power (W)	3 m	0.5 m	0.35 m	0.26 m
Noise (V/sqrt(Hz))	5.6 n	8.3 n	15 n	22 n
dcGain (dB20)	92	92	97	90
UGB (Hz)	300 M	115 M	52 M	135 M



(a). SKY130 Layouts

(b). SKY130 - Zoomed-in View of One Instance

(c). GF180 Layouts

Component		Desig-nators	Tunable Parameters		
Stage 1	Input Pair	M2	W	L	N
	Current Source	M1			
	PMOS Load	M3, 4			
Stage 2	Input PMOS	M5			
	Current Source	M6			
Coax Driver	Input NMOS	-			
	Current Source	-			
MIM Cap		C1			
Routing		-	Width Multipler		
Antenna Diode		-	N		
			26 Params.		

(d). Table of All Tunable Parameters

Fig. 7: Selected GDS layouts for SKY130 on the left and GF180 on the right. One high UGB OpAmp is enlarged and labeled

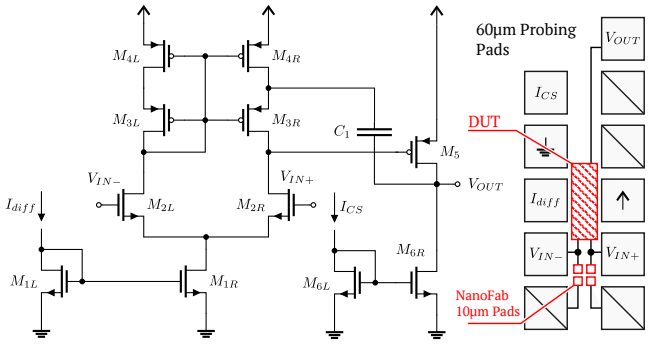


Fig. 8: Schematic of two-stage OpAmp design used in this project

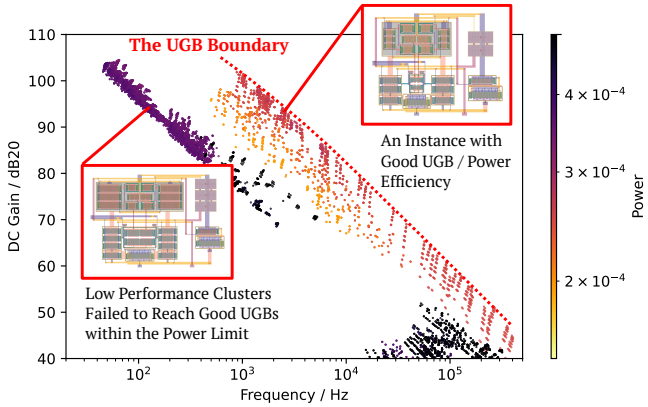


Fig. 9: The Results Scatter Plot from the Shotgun Approach

high-quality design over a few weeks which was at a power of 3mW. However, the cryogenic experiment environment has a significant power budget due to its limited thermal capabilities, demanding further optimization of the power consumption. To achieve this, we programmed the design in GLayout and did an automated optimization targeting lower power. We focus on a bandwidth and noise to power tradeoff. The result is an OpAmp with similar gain but with a 6 times reduction in power. The results are shown in Table III.

Due to device threshold changes, the same design will perform differently when put in room temperature (RT). Thus, it's beneficial to simulate the same designs using both 4K and RT models to select optimal designs for different temperature ranges.

B. The Implementation Process in GLayout

The design is translated to Python classes as described in the previous sections. To demonstrate the functionality of our generator and to seed the RL run, we generated an array of 16000 different OpAmps in 130nm technology and 180nm technologies. The software framework is deployed across multiple cloud servers, utilizing more than 256 CPU threads to generate & simulate in a massively parallel way.

In Fig. 7(a) are 48 selected OpAmps from the 130nm array with the highest OpAmp by unity gain bandwidth highlighted to show a specific layout. This specific layout also shows how the schematic in Fig. 8 with transistors labeled is mapped to the layout in Fig. 7(c). In Fig. 7(b) are 48 selected OpAmps from the 180nm array of generated OpAmps. The table in Fig. 9 shows all the parameters being swept during the shotgun approach.

C. GLayout with Reinforcement Framework

Using the Glayout framework with layout parametrization allows the optimizer to consider layout parasitics within the design loop. As shown in Fig. 10, ignoring post-layout parasitics results in large error in simulated metrics, especially impacting bandwidth, noise, and power metrics. Furthermore, the best OpAmp when considering parasitics is not the best pre-layout.

1) Approach: The RL loop is enabled with a number of seed designs from the shotgun approach as the starting point, and the cryogenic power target is added. Target specifications include unity gain bandwidth (UGB) within the range of $[1 \times 10^7, 1 \times 10^8]$ Hz and a figure of merit (FoM) of "UGB efficiency", which is calculated as $FoM = UGB/power$. During the RL process, UGB serves as the capped specifications, while FoM represents the optimized specification. The RL loop will try to reach the target range of UGB, and try to make FoM as high as possible. This will result in tradeoffs between UGB and power.

As seen in 11, our RL agent followed an average trajectory length of 10 simulation steps to converge efficiently. We employed 200 target design specifications for training the RL agent and an additional 100 for testing. Importantly, all of

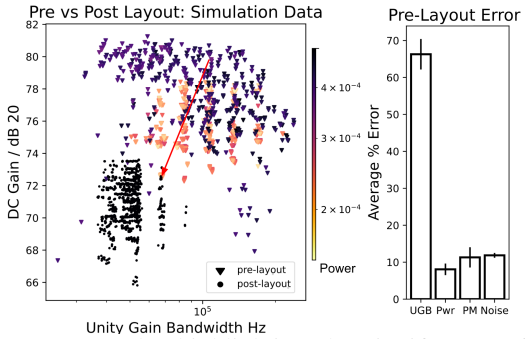


Fig. 10: Scatter plot highlighting the significance of layout parasitics, with percent error for key results pre-layout.

these test specifications were entirely new and fell within the predefined range established during training.

2) Results: Fig. 11 illustrates a graphical comparison between data generated with and without our RL framework. Notably, we exceed the boundary of the shotgun approach. Additionally, for the design points to be successfully achieved, our framework requires an average of approximately ten simulation steps, a significant efficiency improvement compared to traditional genetic algorithms.

D. Accelerating nanofabrication research of emerging technologies

To address the challenge of parasitic de-embedding during device characterization, the generator is deployed in a project aimed at constructing tiles with densely packed active, supportive circuit elements and bare pads. As illustrated in Fig. 12 The goal is to use the active elements to characterize novel CMOS+X nano-devices fabricated directly on top of the tiles. Because the nano-devices under test are not deterministic, the tile must provide a wide variety of such active elements to satisfy the wide range of applications and operation environments (from room temperature to cryogenic). From a library of OpAmp cells generated using the tool presented in this paper, we selected OpAmps optimal for different - 3dB bandwidth, UGB, DC gain, noise, power, and optimal operating temperature goals to construct the tile. Two tiles, as shown in Fig. 13 of test devices were generated using the software framework. In addition to a Phase-Lock-Loop (PLL) design done using the traditional cell-based approach [21], the tile contains 50 OpAmps, with different optimization goals, target temperature range, and pad designs. The tiles have

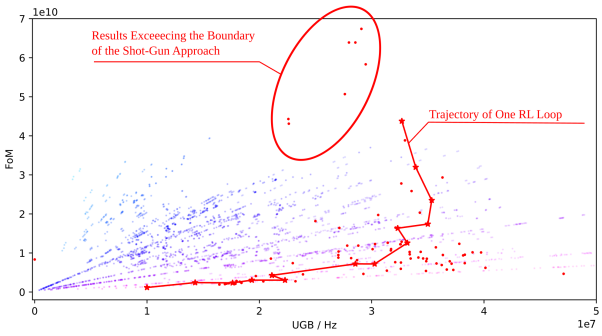


Fig. 11: 2-D distribution of final results (4K)

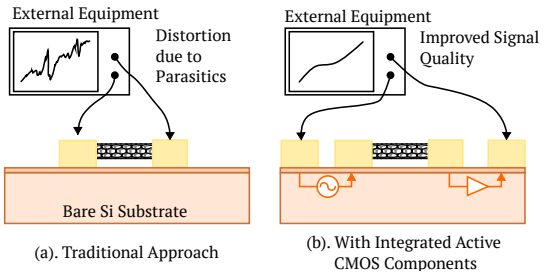


Fig. 12: Illustration of the Novel Device Characterization Method

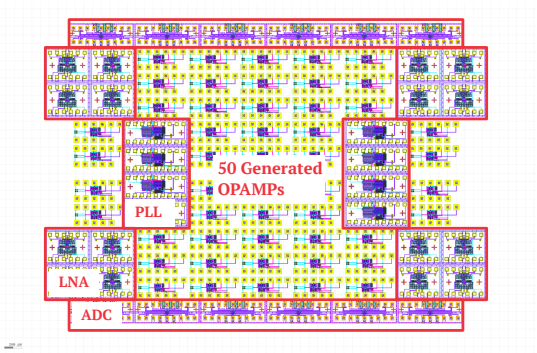


Fig. 13: GDSII Layout Submitted for Tapeout

been submitted for a tapeout in the Skywater 130nm process. The layout-vs-schematic (LVS) check is done in commercial tools using the pre-PEX netlist generated by GLayout to ensure correctness.

V. Conclusions

We introduced a software framework that automates from Python code to layout. The approach is validated through a comprehensive demonstration involving a two-stage OpAmp integrated into a Nano-Fabrication test tile, subsequently delivered to tapeout. We showcase the use case of our RL framework and accounting for parasitics using the Glayout framework to produce layout within the optimization loop.

Compared to human designs, the final design achieved a power improvement of more than 10 times while maintaining the gain specification and optimally trading off bandwidth. Our tool achieves high circuit performance and significantly reduces the design cycle duration. Incorporating open-source principles, cloud deployment capabilities, seamless integration, and PDK portability collectively position Glayout as a valuable asset in advancing the efficiency and accessibility of analog and mixed-signal open-source design automation.

Acknowledgment

The authors would like to thank the open-source community, the GDSfactory [22] Team, Dr. Brian Hoskins, and the National Institute of Standards (NIST) for their support.

References

[1] T. Ansell and M. Saligane, "The missing pieces of open design enablement: A recent history of google efforts : Invited paper," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8, 2020.

[2] "Ieee solid-state circuits society launches a technical committee to support open source ecosystem ramp-up [society news]," *IEEE Solid-State Circuits Magazine*, vol. 14, no. 2, pp. 75–76, 2022.

[3] Q. Zhang, W. Duan, T. Edwards, T. Ansell, D. Blaauw, D. Sylvester, and M. Saligane, "An open-source and autonomous temperature sensor generator verified with 64 instances in skywater 130 nm for comprehensive design space exploration," *IEEE Solid-State Circuits Letters*, vol. 5, pp. 174–177, 2022.

[4] Y. K. Cherivirala, M. Saligane, and D. D. Wentzloff, "An open source compatible framework to fully autonomous digital ldo generation," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2023.

[5] 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 *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–8, 2018.

[6] K. Zhu, H. Chen, M. Liu, and D. Z. Pan, "Tutorial and perspectives on magical: A silicon-proven open-source analog ic layout system," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 2, pp. 715–720, 2023.

[7] "Openpdks." https://github.com/RTimothyEdwards/open_pdks, 2023.

[8] "Openfasoc." https://github.com/idea-fasoc/OpenFASOC, 2023.

[9] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," 2020.

[10] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, July 2020.

[11] K. Settaluri, Z. Liu, R. Khurana, A. Mirhaj, R. Jain, and B. Nikolic, "Automated design of analog circuits using reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, pp. 2794–2807, sep 2022.

[12] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović, "Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[13] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *International Conference on Machine Learning*, 2018.

[14] S. Zhang, W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Bayesian optimization approach for analog circuit synthesis using neural network," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1463–1468, 2019.

[15] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20, IEEE Press, 2020.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[17] A. F. Budak, P. Bhansali, B. Liu, N. Sun, D. Z. Pan, and C. V. Kashyap, "Dnn-opt: An rl inspired optimization for analog circuit sizing using deep neural networks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1219–1224, 2021.

[18] Y. Choi, M. Choi, K. Lee, and S. Kang, "Ma-opt: Reinforcement learning-based analog circuit optimization using multi-actors," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–5, 2023.

[19] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Society for Industrial and Applied Mathematics*, vol. 42, 04 2001.

[20] A. Akturk, B. D. Hoskins, and P. Shrestha, "Cmos sky130 primitives measured at cryogenic temperatures, national institute of standards and technology." https://doi.org/10.18434/mds2-2997, 2023.

[21] A. Hammoud, V. Shankar, R. Mains, T. Ansell, J. Matres, and M. Saligane, "Openfasoc: An open platform towards analog and mixed-signal automation and acceleration of chip design," in *2023 International Symposium on Devices, Circuits and Systems (ISDCS)*, vol. 1, pp. 01–04, 2023.

[22] "Gdsfactory." https://github.com/gdsfactory, 2023.