INVITED: ALIGN – Open-Source Analog Layout Automation from the Ground Up

Kishor Kunal
University of Minnesota
Minneapolis, MN
kunal001@umn.edu

Meghna Madhusudan
University of Minnesota
Minneapolis, MN
madhu028@umn.edu

Arvind K. Sharma
University of Minnesota
Minneapolis, MN
aksharma@umn.edu

Wenbin Xu
Texas A&M University
College Station, TX
wbxu@tamu.edu

Steven M. Burns
Intel Corporation
Hillsboro, OR
steven.m.burns@intel.com

Ramesh Harjani
University of Minnesota
Minneapolis, MN
harjani@umn.edu

Jiang Hu
Texas A&M University
College Station, TX
jianghu@tamu.edu

Desmond A. Kirkpatrick
Intel Corporation
Hillsboro, OR
desmond.a.kirkpatrick@intel.com

Sachin S. Sapatnekar
University of Minnesota
Minneapolis, MN
sachin@umn.edu

ABSTRACT

This paper presents analog layout automation efforts under the ALIGN ("Analog Layout, Intelligently Generated from Netlists") project for fast layout generation using a modular approach based on a mix of algorithmic and machine learning-based tools. The road to rapid turnaround is based on an approach that detects structure and hierarchy in the input netlist and uses a grid based philosophy for layout. The paper provides a view of the current status of the project, challenges in developing open-source code with an academic/industry team, and nuts-and-bolts issues such as working with abstracted PDKs, navigating the "wall" between secured IP and open-source software, and securing access to example designs.

KEYWORDS

Analog circuits, physical design, hierarchy, machine learning.

ACM Reference Format:
Kishor Kunal, Meghna Madhusudan, Arvind K. Sharma, Wenbin Xu, Steven M. Burns, Ramesh Harjani, Jiang Hu, Desmond A. Kirkpatrick, and Sachin S. Sapatnekar. 2019. INVITED: ALIGN – Open-Source Analog Layout Automation from the Ground Up. In *Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 4 pages.

1 MOTIVATION AND GOALS

The problem of analog layout synthesis has attracted considerable interest for several decades [1–8, 10], but these efforts have not seen very widespread adoption by circuit designers. The traditional perception has been that the results of these tools are unable to match the expert designer, both in terms of the ability to comprehend and implement specialized layout tricks, and the number and variety of topologies with circuit-specific constraints. Consequently, automated layouts were unable to match the performance of handcrafted layouts. The first generation of approaches for solving the problem used rule-based methods. However, distilling designer intent into a limited set of rules can be challenging.

In recent years, the landscape has shifted in several ways, making automated layout solutions attractive. *First*, in nanometer-scale technologies, restricted design rules with fixed pitches and unidirectional routing limit the full freedom for layout that was available in older technologies, thus reducing the design space to be explored during layout, reducing the advantage to the human expert. *Second*, today more analog blocks are required in integrated systems than before, and several of these require correct functionality and modest performance. The combination of increasing analog content with the relaxation in specifications creates a sweet spot for analog automation. Even for high-performance blocks, an automated layout generator could considerably reduce the iterations between circuit optimization and layout, where layout generation is the primary bottleneck. *Third*, the advent of machine learning (ML) provides the promise for attacking the analog layout problem in a manner that was not previously possible.

The ALIGN (Analog Layout, Intelligently Generated from Netlists) project engages a joint academic/industry team to develop open-source software for analog/mixed-signal circuit layout to translate a netlist into a physical layout, with 24-hour turnaround and no human in the loop. The ALIGN flow inputs a netlist whose topology and transistor sizes have already been chosen, specifications, and a process design kit (PDK), and outputs GDSII.

The philosophy of ALIGN is to use a mix of algorithmic techniques, template-driven design, and ML to create layouts that are at the level of sophistication of the expert designer. The solution proceeds through a compositional approach that builds designs by assembling structures multiple levels of hierarchy. Thus, ALIGN

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN XXX-X-XXXX-XXXX-X/XX/XX…$15.00
DOI: 10.1145/3316781.3323471

identifies hierarchies to recognize the building blocks of the design so that they may be appropriately optimized, in much the way that an expert analog designer builds a circuit. At the lowest level of this hierarchy is an individual transistor; these transistors are then combined into larger fundamental primitives (e.g., differential pairs, current mirrors), then modules (e.g., opamps), up through several levels of hierarchy to the system level (e.g., an RF transceiver).

In fact, the key to making ALIGN generally applicable to circuits lies in its use of hierarchy. By defining an appropriate set of primitives at the lowest level of hierarchy, and by using ML capabilities to handle ambiguity in the way these primitives are assembled, we believe that it is possible to mimic the expert designer.

The sets of circuits targeted by ALIGN fall into four broad classes:

- Low-frequency components that include analog-to-digital converters (ADCs), amplifiers, and filters.
- Wireline components that include clock/data recovery, equalizers, and phase interpolators.
- RF/Wireless components that implement transmitters, receivers, etc.
- Power delivery components include capacitor and inductor based DC-to-DC converters.

Each class is characterized by similar building blocks that have a similar set of specifications, although it should be mentioned that there is considerable diversity even within each class.

2 OVERVIEW

The ALIGN flow consists of five modules, illustrated in Fig. 1:[1]

- *Design Rule Capture* abstracts the proprietary PDK into a set of constraints that must be obeyed by the layout generator.
- *Netlist Auto-annotation* groups transistors and passives in the input netlist into building blocks and identifies geometric constraints on the layout of each block.
- *Electrical Constraint Generation* identifies the performance constraints to be obeyed, and transforms them into layout constraints, such as the maximum allowable route length.
- *Parameterized Layout Generation of Primitives* automatically builds layouts for primitives, the lowest-level blocks in the ALIGN hierarchy, parameterized by variables that characterize the size of a transistor, the capacitance of a MOM capacitor, the resistance of a serpentine, etc.
- *Block Assembly* takes all blocks and places/routes the design hierarchy to build the overall layout.

The first three derive the netlist structure and the constraints that guide the last two modules that perform constraint-driven layout generation. The flow creates a separation between open-source code from proprietary data. Proprietary PDK models must be translated into an abstraction that is used by the layout generators. Various parts of the flow are driven by ML models: the flow provides infrastructure for training these models on proprietary data. The models are trained on public data, and ALIGN is also assembling a set of public-domain benchmark circuits from all available sources.

[1]It is important to point out that ALIGN is a multiyear project that is not yet a year old, and therefore, not all of these modules are fully populated at this time.

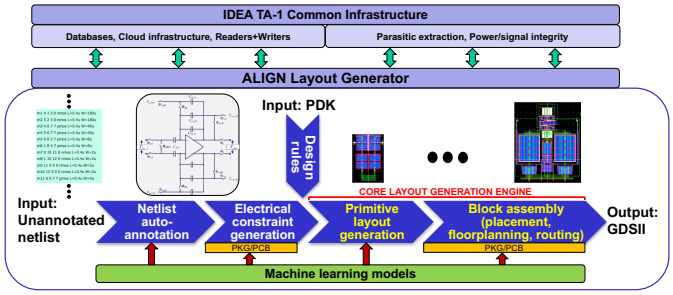

Figure 1: Overview of the ALIGN flow.

2.1 Design rule abstraction

A key step in closing the interface between a proprietary PDK and the layout generation engine is an appropriate abstraction of the PDK that can be comprehended by the layout generation engines. Several efforts in this direction have been made (e.g., [9])), and ALIGN attempts to abstract the design rules using a simplified grid, for both the FEOL and BEOL layers.

Major features of advanced process nodes (22nm, 10nm, 7nm, beyond) have been abstracted into a simplified form. The abstraction enables layout tools to comprehend PDK features such as regular and irregular width and spacing grids (for each layer), minimum end to end spacing design rules (between metals in the same track), minimum length design rules, and enforced stopping point grids.

Our simplest uniform grid, for a specific metal layer, is illustrated in Fig. 2. The grid consists of major grid (dark) lines on which features are centered, and minor grid (light) lines that act as stopping points. Here, metal2 (horizontal, blue) and metal3 (vertical, pink) grids are shown, along with a grid-based minimum length and minimum end-to-end rules. Each center line (routing track) is associated with a specific wire width. The diagram also shows via enclosures (metal surrounding the via cut) and how their dimensions correspond to the stopping point grid lines. These abstract grids for base layers and metals are described in simple JSON files.

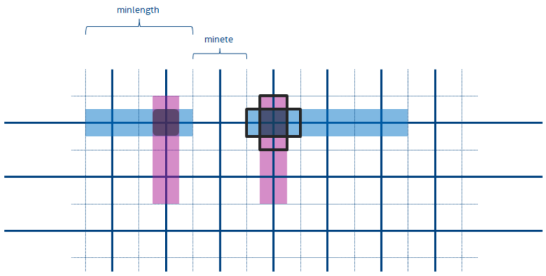

Figure 2: A uniform grid to simplify more complex design rules.

2.2 Auto-annotation of input netlists

The input to ALIGN is an unannotated input netlist. The netlist is first represented by a graph, and then features in the graph are recognized at various levels of hierarchy. If the input netlist is partitioned into subcircuits, such information is used during recognition, but ALIGN does not count on netlist hierarchy. Instead, hierarchies are automatically identified and annotated.

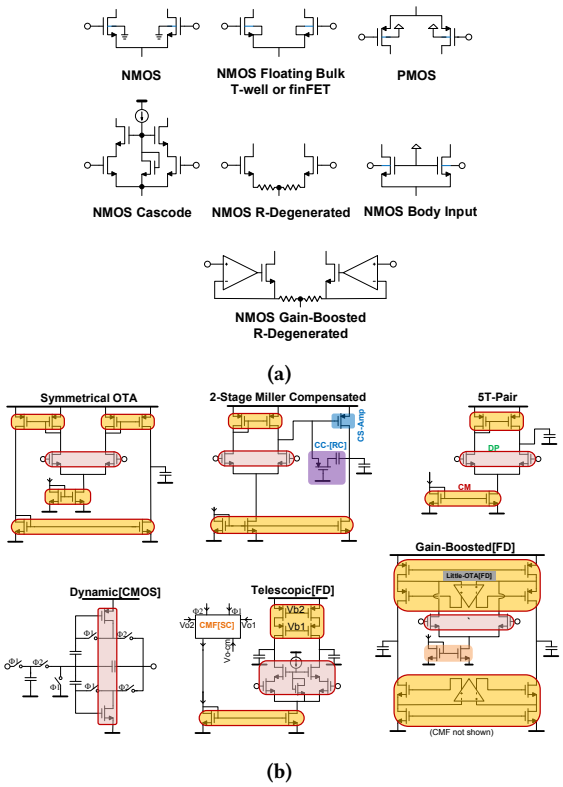

Figure 3: Example variants of (a) the differential pair (DP) primitive (b) the opamp/OTA, showing fundamental primitives.

While graph-based methods can be very efficient at recognizing fixed structures through subgraph isomorphism operations, a problem in analog design is that there may be a large number of variations in how each circuit functionality can be implemented. For example, Fig. 3a shows various implementations of a differential pair. Many of these are purely transistor-level structures, but the last configuration in the figure uses a mix of transistors and amplifier building blocks. At a higher level of design hierarchy, Fig. 3b demonstrates different ways of building an operational amplifier: here, the sub-blocks include groups of transistors recognized as differential pairs, current mirrors, differential loads, or OTA blocks.

Enumerating graph patterns for recognizing these structures is feasible at lower levels of design hierarchy, but the number of permutations becomes impractically large at higher levels. An expert human designer who examines a schematic instinctively performs such recognition based on patterns learned from prior experience. The ALIGN approach matches this through the use of ML methods that recognized standard structures based on their features.

2.3 Constraint generation

Based on the recognized hierarchical blocks, further annotations are added at each level to identify geometric constraints such as symmetry, matching, or common centroid. Electrical performance metrics for the system are percolated down to individual sub-blocks and translated into layout rules (e.g., maximum routing lengths or parasitic requirements). All such constraints are passed on to the layout generation engine to guide layout at all levels of hierarchy.

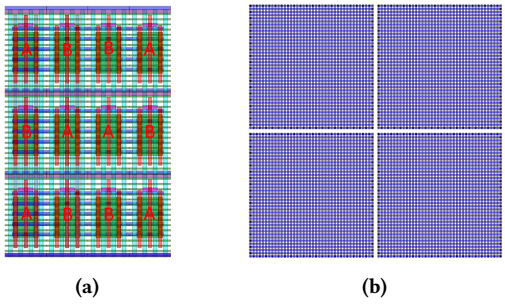

Figure 4: Parameterized primitive layouts for (a) a differential pair with transistors placed using common centroid, and (b) a 2×2 MOM capacitor array.

2.4 Parameterized primitive layout generation

The first level of structures above the elemental (transistor or wire) level are referred to as primitives. Predefined parameterized templates for the layouts of these blocks are stored in a library. A gridded layout style is used, following the grids defined by PDK abstraction. The template parameters define specific implementational details, e.g., a current mirror is parameterized by the number of outputs and the sizes (or number of fins) of each transistor; a MOM capacitor array is parameterized by the desired capacitance value. The templates interact with the PDK abstraction and are constructed to guarantee design-rule-correct layouts, including optimal transistor placement and within-primitive routing. Example primitive layouts are shown in Fig. 4.

2.5 Block assembly, placement, and routing

During block assembly, layouts for all blocks are progressively created from their subblocks, with the leaf-cell layouts corresponding to the parameterized primitive layouts described above. Primitives have rigid shapes and use placement-like algorithms, and multiple layout options with different shapes are be generated for each module. At higher levels of hierarchy, flexible shapes drive floorplanning-like placement algorithms that deliver compact layouts under the electrical and geometric constraints passed on to them by the constraint generation step. Routing is integrated into each hierarchical level, accounting for net length/parasitic constraints, net density/dummy fill constraints, and design rules, again obeying electrical and geometric constraints.

3 OPEN SOURCE CHALLENGES

Several significant challenges are being surmounted in building open-source EDA software for analog layout, as outlined below:
Working with common PDKs: The process of obtaining legal access to a commercial PDK requires considerable patience and involves signoffs on nondisclosure agreements (NDAs). Even PDKs that are freely available to academia are restricted for circulation to non-academic institutions, such as industry or government laboratories. This limits our ability to exchange information across institutions within the ALIGN team, and with external designers.

To circumvent this issue, we have developed realistic "mock PDKs" representing typical bulk and FinFET technology nodes, based on published data. While they do not represent a real technology, validation of the design tools on these PDKs, which can

be freely shared, helps the software development process. Once the software has been developed and proven on mock PDKs, a developer with full PDK access can run ALIGN on real PDKs.

Access to design examples: Sharing designs based on a commercial PDK over multiple institutions requires a multiway NDA involving the institutions, the foundry, and the foundry access provider. Within the ALIGN team, this issue was complicated by the need for such an agreement to cover both academia and industry.

We have chosen a multipronged approach to solving this problem by (a) mining prior academic designs from the Harjani group at Minnesota (b) collaborating with other design teams through multiway NDAs (c) building new designs, or retargeting old designs to new technologies. To carry the designs through the entire design flow, the input must be an optimized netlist with appropriately chosen device sizes, and each of these methods provides an avenue to access such designs. We choose representative designs in the space of low-frequency analog, wireline, wireless, and power delivery circuits to build and exercise the ALIGN flow.

Superficially, it may seem that there is a wealth of available designs in prominent conferences and journals that cover analog circuit design, but these sources typically do not specify the details of a design, and may at best present a coarse schematic, with numerous details hidden within black boxes. This limited information is also being exercised by ALIGN, primarily by providing training examples for the auto-annotation block.

4 SOFTWARE INFRASTRUCTURE

The software flow is maintained on a github repository, and is aided by the use of tools that are vital to a open-source infrastructure with continuous integration (CI). These include:

- lightweight Docker containers that perform operating system virtualization and enable portability and ease of maintenance, and enabling the use of other open-source tools such as the KLayout layout viewer;
- CI build flows, using CircleCI, for automated build of new components as they are added to the repository;
- unit testing, using pytest, to verify the correctness of individual units of source code that is added to the repository;
- code coverage to measure how much of the code is executed by the automated tests, using coverage.py with Codecov for tracking; and
- automated code review for code quality checks using Codacy.

It is worth pointing out that many of these cloud-based software development tools are free of charge for open-source code.

5 EARLY RESULTS

We show an early application of the ALIGN flow to the layout of a switched capacitor filter, whose schematic is shown in Fig. 5a. The auto-annotation component recognizes various blocks of the netlist, as shown in the figure, and the lines of symmetry are marked out.

Individual primtives are identified and laid out: for example, the capacitor layouts correspond to Fig. 4b. The primitives are assembled into blocks: Fig. 5b shows how various primitives (differential pair, differential load, current mirror) are assembled into a layout. Finally, the top level layout, consisting of all components of the filter, satisfying all constraints, is shown in Fig. 5c.

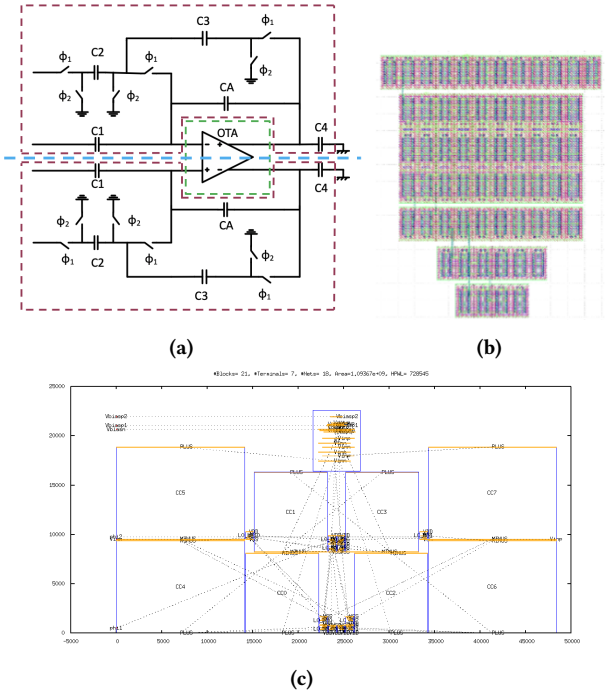

Figure 5: (a) A switched capacitor (SC) filter, and the ALIGN-generated layout of (b) the OTA within the SC filter and (c) the entire SC filter, where "CC" refers to a capacitor array. The opamp is placed on top; other transistors correspond to switches.

6 CONCLUSION

This paper summarizes early efforts in putting together ALIGN, an open-source layout generation flow for analog circuits for rapid turnaround with no human in the loop. The solution is architected to enable users to incorporate proprietary process and design information into the flow. The project works with a PDK abstraction and leverages hierarchy, machine learning, and gridded layouts to control the complexity of the design space, with hierarchy being a critical feature that can enable future scalability of this project to handle a large variety of analog designs.

ACKNOWLEDGMENTS

This work was supported by the DARPA IDEA program under SPAWAR contract N660011824048.

REFERENCES

[1] J. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley. 1991. KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing. *IEEE Journal of Solid-State Circuits* 26, 3 (March 1991), 330–342.
[2] C. R. C. De Ranter, G. Van der Plas, M. S. J. Steyaert, G. G. E. Gielen, and W. M. C. Sansen. 2002. CYCLONE: Automated Design and Layout of RF LC-Oscillators. *IEEE T. Comput. Aid D.* 21, 11 (Oct. 2002), 1161–1170.
[3] M. Eick, M. Strasser, K. Lu, U. Schlichtmann, and H. E. Graeb. 2011. Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits. *IEEE T. Comput. Aid D.* 30, 2 (Feb. 2011), 180–193.
[4] H. E. Graeb (Ed.). 2010. *Analog Layout Synthesis: A Survey of Topological Approaches.* Springer, New York, NY.
[5] R. Harjani, R. A. Rutenbar, and L. R. Carley. 1989. OASYS: A Framework for Analog Circuit Synthesis. *IEEE T. Comput. Aid D.* 8, 12 (Dec. 1989), 1247–1266.
[6] Q. Ma, L. Xiao, Y.-C. Tam, and E. F. Y. Young. 2011. Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints. *IEEE T. Comput. Aid D.* 30, 1 (Jan. 2011), 85–95.
[7] E. Ochotta, R. A. Rutenbar, and L. R. Carley. 1994. ASTRX/OBLX: Tools for Rapid Synthesis of High-Performance Analog Circuits. In *Proc. DAC.* ACM, New York, NY, 24–30.
[8] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang. 2013. Simultaneous Analog Placement and Routing with Current Flow and Current Density Considerations. In *Proc. DAC.* ACM, New York, NY, 6 pages.
[9] G. Soto. 2017. Discover the Power Of OPAL, A New High-Level Design Rule Modeling Language. http://www.si2.org/events/opal/
[10] C.-Y. Wu, H. Graeb, and J. Hu. 2015. A Pre-search Assisted ILP Approach to Analog Integrated Circuit Routing. In *Proc. ICCD.* IEEE, Piscataway, NJ, 244–250.