Layout Verification Using Open-Source Software

Andreas Krinke
Dresden University of Technology
Institute of Electromechanical and Electronic Design
Dresden, Germany
andreas.krinke@tu-dresden.de

Robert Fischbach
Dresden University of Technology
Institute of Electromechanical and Electronic Design
Dresden, Germany
robert.fischbach@tu-dresden.de

Jens Lienig
Dresden University of Technology
Institute of Electromechanical and Electronic Design
Dresden, Germany
jens@ieee.org

ABSTRACT

The design and manufacturing of integrated circuits is an expensive endeavor. The use of open-source software can lower the barrier to entry significantly, especially for smaller companies or startups. In this paper, we look at open-source software for layout verification, a crucial step in ensuring the consistency and manufacturability of a design. We show that a comprehensive design rule check (DRC) and layout versus schematic (LVS) check for commercial technologies is possible with open-source software in general and with KLayout in particular. To facilitate the use of these tools, we present our approach to automatically generate the required DRC scripts from a more abstract representation. As a result, we are able to generate nearly 74% of the over 1000 design rules of X-FABs XH018 180 nm technology as a DRC script for the open-source software KLayout. This demonstrates the potential of using open-source software for layout verification and open-source process design kits (PDKs) in general.

CCS CONCEPTS

• **Hardware → Physical design (EDA)**; **Software tools for EDA**; **Design rules**.

KEYWORDS

design rule check, layout versus schematic, KLayout

ACM Reference Format:
Andreas Krinke, Robert Fischbach, and Jens Lienig. 2024. Layout Verification Using Open-Source Software. In *Proceedings of the 2024 International Symposium on Physical Design (ISPD '24), March 12–15, 2024, Taipei, Taiwan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3626184.3635289

1 INTRODUCTION

The cost of designing an integrated circuit (IC) is high, not only because of the labor costs, but also because of expensive software licenses. It is difficult to estimate these software costs, but a comparison with the overall size of the semiconductor market, which also includes manufacturing costs and profits, helps to put them in perspective. The global semiconductor market size was at US$574 billion in 2022 [27], while the electronic design automation (EDA) market was valued at US$12.9 billion, or 2.2%, in the same year [13]. Furthermore, these software costs vary greatly, depending on factors such as the type of chip (analog and/or digital), the complexity of the design, and the technology used. This can easily lead to the cost of software licenses for a single seat exceeding an employee's wage costs.

Especially for smaller companies that want to design an IC, these software costs can be a high barrier to entry. Open-source tools are a means to replace some of the proprietary design tools in a design flow and thus reduce these costs. Even complete open-source design flows are gradually becoming viable. A crucial advance in this direction for *digital designs* has been the development of OpenROAD [1, 28], an open-source technology-independent RTL-to-GDSII flow that performs floorplanning, global/detailed placement and routing, and chip finishing steps. OpenROAD is used both in research and commercial flows. One example is OpenLane [11, 26], a flow developed by Efabless that includes OpenROAD and is tailored to SkyWater's 130 nm open-source process design kit (PDK) [7]. Additionally, OpenROAD/OpenLane also support GlobalFoundries' 180 nm open-source PDK [2] and several proprietary PDKs. These open-source flows have been used in over 600 tapeouts in SKY130 and GF180. However, customized PDKs must first be developed for use with new technologies.

For *analog design*, a schematic editor is used to create the netlist for subsequent simulation and layout design. During layout design, the layouts of individual devices are generated using parametric cells (PCells). More complex generators are also available that can generate entire sub-circuits automatically, e.g. [10, 24]. Nevertheless, the layout design of analog circuits is still dominated by manual tasks due to the large number and variety of constraints [16, 17, 20]. Table 1 shows a selection of common open-source tools for the multiple steps in analog IC design.

Table 1: Selection of analog design steps and related open-source tools

Design Step	Open-Source Tool(s)
Schematic Entry	Xschem [25], Qucs [5], Qucs-S [6, 9]
Analog and RF Simulation	ngspice [21], Xyce [15]
Layout Design	KLayout [18], Magic [4]
Design Rule Check (DRC)	KLayout, Magic
Layout Versus Schematic (LVS)	KLayout, Magic
Parasitic Extraction (PEX)	Magic
Post Layout Simulation	ngspice, Xyce

ISPD '24, March 12–15, 2024, Taipei, Taiwan

ACM ISBN 979-8-4007-0417-8/24/03
https://doi.org/10.1145/3626184.3635289

Once the physical design has been finalized, it is important to verify the layout to confirm correct electrical and logical functionality and to ensure manufacturability. This step, known as *layout verification*, is the focus of this paper and comprises several tasks: (1) design rule check (DRC), (2) layout versus schematic (LVS) check, (3) parasitic extraction, (4) antenna rule check, and (5) electrical rule check (ERC) [14, 19]. Obviously, layout verification is crucial for the final signoff in chip design. As shown in Table 1, Magic [4] and KLayout [18] are currently the most important open-source tools for layout verification.

The aforementioned layout verification steps require rule decks that contain, for example, the design rules for DRC and allow device extraction for LVS. These rule decks are part of a foundry's PDK and are provided as reference documents and in proprietary file formats for one or more supported commercial verification tools. For example, while the Cadence Physical Verification System (PVS) uses rule decks written in the Physical Verification Language (PVL), Siemens Calibre nmDRC uses the Standard Verification Rule Format (SVRF).

The success of open-source flows is tightly coupled with the availability of compatible PDKs, since only then can ICs for real-life semiconductor technologies be designed. To support smaller customers that want to use open-source tools, foundries are encouraged to publish open-source PDKs. The effort required to create such a PDK specifically for open-source tools is, however, very high. A large amount of necessary information, e.g., the layer stack, PCells, standard cells, device models, design rules, and rules for device extraction, has to be translated into open file formats that are supported by the chosen open-source tools.

Our goal is to facilitate the creation of open-source PDKs, which then allows for a comprehensive DRC and LVS check for commercial technologies using open-source software. For this purpose, we present our approach for generating DRC and LVS runsets for KLayout. We introduce our program *Babylon* with which the necessary DRC and LVS scripts can be created efficiently.

This paper is structured as follows. We first discuss the history and capabilities of the two most important open-source layout tools, Magic and KLayout. Section 3 describes the main contribution of our paper, the aforementioned runset generation using *Babylon*. We summarize with a conclusion and outlook in Section 4.

2 STATE OF THE ART

2.1 Magic

Magic is an open-source VLSI layout tool that was originally created by John Ousterhout and his graduate students at UC Berkeley, with work on the project commencing in February 1983. The history of Magic VLSI is intertwined with the evolution of very-large-scale integration (VLSI) technology. By April 1983, a primitive version of Magic was operational, serving the needs of graduate student chip designers working on the SOAR CPU chip, a follow-on to Berkeley RISC. [22]

Magic remained relevant for universities and smaller companies throughout the following decades. A diverse team of contributors from industry and academia worked on Magic during this period. From 2005 to 2020, the development was mainly underwritten by the companies employing Timothy Edwards, basically MultiGiG (acquired by Analog Devices) and Efabless. [4]

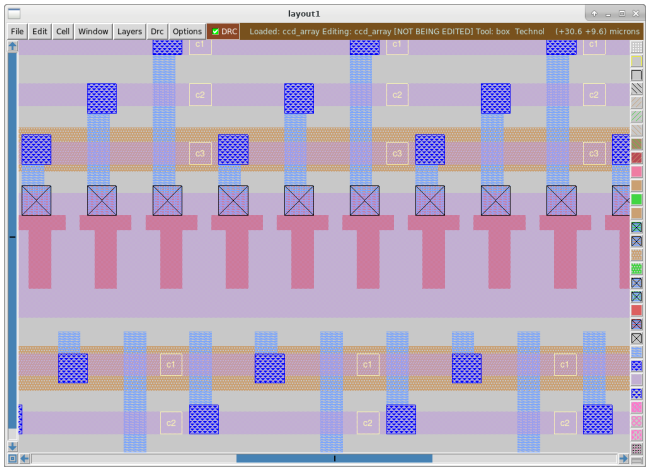

Figure 1: Screenshot of Magic

In 2017, Magic was released on GitHub with Timothy Edwards as lead developer and maintainer of the code repository. With more than 300,000 lines of C code, the estimated development costs amount to US$8 million (calculated using the COCOMO model [8]).

2.1.1 Capabilities. This subsection provides a short review of the main capabilities of Magic based on its documentation [4].

Magic provides a graphical user interface for interactive layout editing of IC cell hierarchies. Figure 1 shows a screenshot of the Magic layout editor, which allows basic polygon entry but also advanced manipulation techniques, such as wiring and plowing. It also integrates automatic routing to connect subcells based on connectivity information provided by a netlist.

The ability to continuously check for design rule violations reduces the turnaround time during layout validation. Magic's DRC can also be globally applied to a loaded layout making it viable as a DRC engine within open-source flows.

Circuit netlists can be extracted from a layout for use with simulation tools, such as IRSIM or SPICE. Furthermore, Magic enables layout vs. schematic comparisons by handing the source and extracted netlists to Netgen. Netgen is an open-source netlist comparison tool, which was incorporated as part of the Tcl-based tool suite.

Repetitive tasks can be automated with the Tcl scripting interface, and the built-in functionalities can be extended and customized. Technology information is stored in a self-contained technology file, which is divided into multiple specific sections (e.g., layer parameters, design rules, router settings). This format is continuously updated as new features are added to Magic.

2.1.2 Utilization in OpenROAD. Magic has been integrated into OpenROAD due to the above capabilities. It is used to perform the sign-off DRC for the layouts generated by the place and route steps in the OpenROAD flow. The files (control script, technology file) required for the target technology are prepared by OpenROAD and

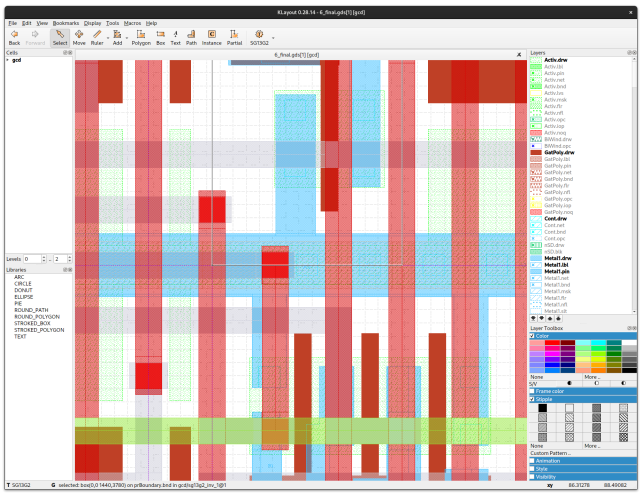

Figure 2: Screenshot of KLayout

then used with Magic to check if the layout fulfills all design rules provided by the foundry.

2.2 KLayout

At its core, KLayout [18] is a fast layout viewer and editor that supports GDS and OASIS files. The main developer of KLayout, Matthias Köfferlein, published the first version in 2006 and continues to work on it to this day. The repository currently contains more than 500,000 lines of C++ code, with an estimated development cost of about US$24 million (calculated using the COCOMO model [8]). KLayout is open-source software and licensed under the GPLv3.

2.2.1 Capabilities. In this section, we will only briefly discuss the features of KLayout. Further details will follow in Section 3.

KLayout can be used to efficiently load and display large layout files. Figure 2 shows a screenshot of KLayout's main window. When loading multiple layouts, these can be displayed in separate (synchronized) tabs or overlayed. How individual layers are displayed can be configured extensively.

KLayout supports many drawing functions, operations on shapes and entire layers; and cells in lower hierarchy levels can be edited in place with it. Parameterized cells (PCells) can be implemented in Ruby or Python for analog design. The same two languages can be used to program macros using the integrated development environment (IDE); these macros can be used for comprehensive automated layout manipulations.

DRC and LVS scripts are special types of macros. They are written in Ruby and executed in a separate environment with access to the commands of the KLayout DRC/LVS application programming interface (API). DRC errors can be analyzed or an LVS netlist comparison can be assessed via two built-in database browsers.

In addition to the functions mentioned, KLayout also includes other useful utilities such as an XOR tool, a tool for hierarchical comparison (Diff), and a 2.5D view of the layout. Technology information and add-ons that extend KLayout's functionality can be managed with two package managers.

2.2.2 Usage. KLayout is used for layout verification in GlobalFoundries' 180 nm open-source PDK [2] and in IHP's SG13G2 OpenPDK [3].

3 GENERATING DRC AND LVS RUNSETS FOR KLAYOUT

In this section, we present our approach for generating scripts for the design rule check (DRC) and the layout versus schematic check (LVS) with KLayout.

3.1 Why KLayout?

Our decision to target KLayout was based on several factors. As described in Section 2.2, KLayout already implements the infrastructure for running DRC and LVS scripts, viewing DRC errors with a marker browser, and writing and viewing report databases (RDBs). KLayout DRC and LVS scripts are written in the Ruby programming language, which supports multiple programming paradigms, e.g., procedural, object-oriented, and functional programming. Theoretically, two types of scripts can be developed: declarative scripts, where the order of the commands is irrelevant, similar to SVRF, and imperative scripts, where the order of the commands is decisive. KLayout supports many typical DRC operations including Boolean layer operations, width and space checks, density checks, and many more. Connectivity extraction is also supported—enabling connectivity-aware layout operations and antenna checks. All these operations are implemented in Ruby as methods of a few classes. These classes can be extended directly in Ruby, e.g., to implement new operations based on low-level commands, or to adapt existing operations. Another important consideration is KLayout's very comprehensive and well-structured documentation. These aspects make it much easier to get started with the program than with Magic. Last but not least, KLayout is available under the GPLv3, a strong copyleft license that ensures changes remain open-source. In theory, there are no limitations as to what we can do—if there are missing functions or speed problems in Ruby, we can change the C++ source code as a last resort.

3.2 Why Generate?

At this point, we could have started writing the KLayout DRC and LVS scripts manually for a specific technology. However, we have decided to generate them automatically instead. This approach has the obvious disadvantage that we have to define a suitable input data format and design a software tool that generates the script. However, there are also a number of advantages, which we outline next.

In general, several hundred to several thousand design rules are checked for a technology. There are a variety of rule types and even seemingly simple rules may require a large amount of code; this means the DRC scripts may be very extensive. These challenges can be mitigated by using custom classes, functions, and methods to modularize the scripts. An example of this approach is shown in Listing 1 and will be discussed in Section 3.3.5. Nevertheless, writing a DRC script for a new technology is still a major undertaking.

By generating the scripts, best practices and improvements in implementing individual rules can be quickly applied to the entire script and also for multiple technologies. The same applies to

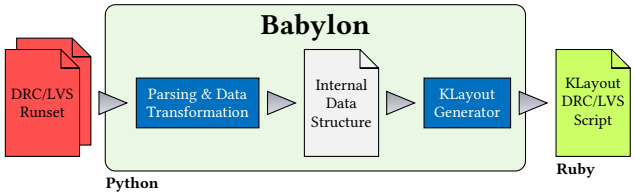

Figure 3: Overview of our approach to automatically generate DRC and LVS commands for KLayout

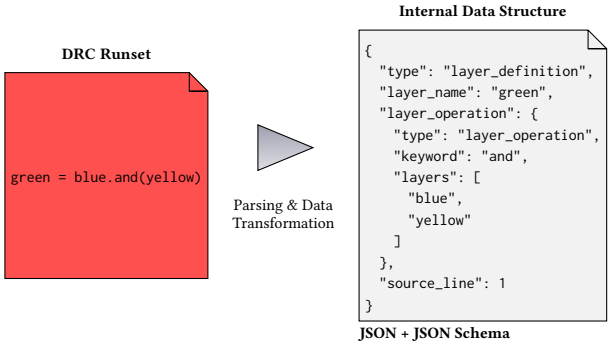

Figure 4: Internal data structure using JSON and JSON schema

changes to the DRC application programming interface (API) in KLayout, regardless of whether new or modified methods or altered program behavior are involved. Ideally, the work involved in coding a DRC script for a new technology should decrease over time.

3.3 Our Approach

Next, we present our methodology to generate DRC and LVS scripts automatically for KLayout; Figure 3 shows an overview of our approach. Our software tool is called *Babylon* and essentially consists of three parts: (1) a parser that reads and translates the input file(s), (2) the internal data structure, and (3) the actual generator that creates the DRC and LVS scripts for KLayout.

3.3.1 Input Format. The *Babylon* input format is based on the KLayout DRC API itself. This means that the input files are syntactically correct Ruby scripts that can use all DRC commands defined by KLayout. However, there are two differences w.r.t. to a valid script: (1) methods can have a different signature, e.g., additional arguments, and (2) the order of the commands is irrelevant.

With these modifications *Babylon* can support completely new or extended DRC commands with additional options. Furthermore, the format allows the rules to be described declaratively without having to consider dependencies between the operations, such as the correct definition of connectivity.

3.3.2 Internal Data Structure. The parser reads our input format and translates it into an internal data structure that can be directly exported as a JSON (JavaScript Object Notation) [12] document. Figure 4 shows an example for this translation. The JSON file encodes layer imports, layer definitions, layer operations, and rule checks. Layer operations, whether stand-alone or as part of a rule check, are translated into an equivalent JSON tree structure. The file

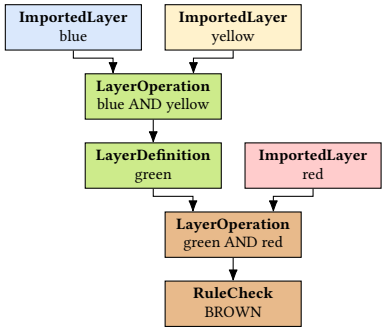

Figure 5: Example of our internal data model

produced is easy to debug as each entry refers to the original line in the input file. Debugging is further assisted by a JSON schema [23] that describes the structure of a valid JSON document.

The generator is separated from the input file by this intermediate file. In future, we will be able to support additional input formats without having to change the generator. Only the parser will need to be modified in this case.

3.3.3 Internal Data Model. The generator loads the internal data structure and transforms its entries into equivalent Python objects. These objects are processed multiple times to establish all dependencies between their respective layer definitions, layer operations, and rule checks. A simple example is shown in Figure 5. Operations that need connectivity information are flagged, as the connectivity and all layers required for it must be defined in advance. Using topological sorting, all layer definitions and operations as well as (partial) connectivity extractions are ordered correctly. Another result is that we know whether each object can be implemented at this stage in KLayout and whether all the required arguments are available. This information helps to generate clean KLayout DRC scripts in the next step.

3.3.4 Generating KLayout DRC Commands. As soon as all entries are in the correct order, they can be translated into the corresponding KLayout DRC commands. In general, the result looks similar to the input file, but with a different command order.

First, we load the required layers from the layout, as shown below.

```
# name = input(GDS layer number)
  blue = input(1)
```

While KLayout supports loading layers directly from a GDSII file, the DRC is normally executed for a layout that has been loaded already. This means that any DRC errors that might occur can be viewed directly in the open layout using KLayout's marker browser. However, it is also possible to execute the DRC directly on the command line (without graphical user interface) and write the result to a report database (RDB).

After loading layers from the layout, new layers can be created by performing operations on a single layer or on multiple layers. Since these operations are implemented as methods of a

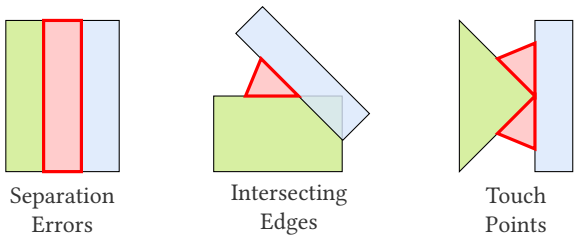

Figure 6: Types of two-layer separation errors

common `DRCLayer` class, multiple operations can be chained together. Besides named layer definitions, as in the following example, anonymous definitions can also be made.

```
# name = layer operation
  green = blue.and(yellow)
```

In order to define a design rule, we simply call the `output` method for the layer to be checked. This layer can be specified by its name or created ad-hoc with an anonymous layer definition, like in the example below. The layer can contain polygons, edges, edge pairs, or points that represent DRC errors; all these geometries can be displayed in the marker browser and exported to an RDB.

```
# (layer operation).output(rule name, comment)
  (green.and(red)).output("BROWN", "Is that
      ↪ chocolate?")
```

Some rule checks require temporary layers that are only used for this rule. A local scope can be created so as not to "pollute" the global namespace. All layer definitions in this scope can only be accessed inside this rule check. The following example shows the code for creating such a local scope.

```
-> (;brown) do
  # separate scope that allows temporary layers
      ↪ to be defined without naming conflicts
  brown = green.and(red)
end.().output("BROWN", "Is that chocolate?")
```

3.3.5 Extensions. So far, we have created a series of KLayout DRC commands for each Python object in our internal data model. More complex layer operations may require extensive coding. To improve the readability of the DRC script, it is worth extending the API.

Ruby allows us to add new methods to a class. In KLayout, all layer operations are methods of the class `DRCLayer`. Therefore, whenever we want to change the behavior or signature of an existing DRC command, we can add a new method with the `ext` prefix. The built-in `separation` command is a case in point: here, the command checks the spacing of polygons on two different layers.

Despite the numerous configuration options, touch points (also called kissing corners) and intersecting edges cannot yet be considered separately by the `separation` command. Figure 6 shows the different types of separation errors.

We have implemented the new method outlined in Listing 1 that enables the different types of errors to be identified. Individual error-type detection can be enabled or disabled with this method (not shown in the listing). For *Babylon* we added 30 methods that extend built-in KLayout layer operations. Functions that are used by more than one layer operation can be added as a method to the `DRCEngine` class.

Listing 1: Extending the layer class with new methods

```
class DRC::DRCLayer
    def ext_separation(other, value [, options])
        separation_errors = ...
        intersecting_edges_errors = ...
        touch_point_errors = ...
        return (separation_errors
            + intersecting_edges_errors
            + touch_point_errors)
    end
end
blue.ext_separation(green, 1.0).output("S1BLGR",
    ↪ "Minimum BLUE spacing to GREEN ... 1.0")
```

3.4 Verification

During the development of *Babylon*, we used X-FAB's XH018 180 nm technology as a reference. Individual layer operations were verified on test layouts. The complete DRC script was verified on a known DRC-clean example layout, to which polygons with deliberate DRC errors were added. As expected, no DRC errors were found in the layout, while the deliberate DRC errors were correctly detected.

The final KLayout DRC script supports 74% of the over 1000 XH018 design rules. *Babylon* ensures that every layout operation in the final DRC script (1) is supported in KLayout, (2) has access to all required layers, and (3) is a direct or indirect dependency of a design rule. The final KLayout DRC script is thus always executable; this facilitates further development of the tool.

3.5 Extension for LVS

So far, we have focused on DRC script generation. However, KLayout also supports layout versus schematic (LVS) checking using the same Ruby environment as for DRC scripts. It therefore makes sense to broaden the scope of *Babylon* so that both DRC and LVS scripts can be generated.

The following steps are necessary for the LVS check in KLayout:

(1) Layer import
(2) Definition of derived layers
(3) Device recognition
(4) Device parameter calculation
(5) Connectivity extraction
(6) Netlist comparison

The first two steps can be carried out in exactly the same way as for the DRC (cf. Section 3.3.4).

Device recognition and device parameter calculation are often summarized under the heading "Device extraction". KLayout supports, for example, the extraction of MOS and bipolar transistors (with three and four terminals), diodes, resistors, and capacitors. For each of these devices, detection and parameter-calculation algorithms are predefined. Only the layers required for recognition have to be specified in the LVS script.

Certain types of devices may not be recognized with these supported device classes. This may occur if the recognition algorithm

is too inflexible or if device parameters are calculated differently or not at all. To solve this problem, new device types can be defined in KLayout. This can be done directly in Ruby by implementing two new classes: (1) a `DeviceExtractor` class that performs recognition and parameter calculation, and (2) a `DeviceParameterCompare` class that compares the parameters of the extracted device with those of the corresponding device from the netlist.

Once the devices have been extracted, connectivity extraction follows. In addition to specifying pairs of electrically connected layers, global connections can also be defined. These can be used for bulk connections, for example; and they ensure that all shapes in a layer belong to the same net.

To summarize, *Babylon* can reuse existing functions from DRC script generation when generating LVS scripts. Only a few commands need to be added for device extraction, connectivity definition, and netlist comparison. Only new component types need to be realized manually.

4 CONCLUSION AND OUTLOOK

The goal of this paper was to show that a comprehensive design rule check (DRC) and layout versus schematic (LVS) check for commercial technologies can be performed with open-source software in general and with KLayout in particular. KLayout supports the fundamental features necessary to run these checks, such as a wide range of layer operations, parallel processing for fast execution, a marker browser for efficient investigation of DRC errors, and the creation of result databases (RDBs).

We have presented our *Babylon* program with which the necessary DRC and LVS scripts can be generated efficiently. While its output are valid KLayout DRC/LVS scripts, the input is an extended script file that allows the use of new complex methods. The plan is to replace this input format by an open rule format; this will enable complete design rule documents to be created, for example. Although *Babylon* is currently not open source, the generated DRC and LVS scripts can become part of an open-source PDK.

Using KLayout with X-FAB's XH018 180 nm technology, we are currently able to check 74% of its over 1000 design rules. We are working to increase the number of rules supported by *Babylon*. In addition to our work on XH018, we are also developing the KLayout DRC script for IHP's SG13G2 OpenPDK [3].

Our current research is focused on extending the DRC script generation to support the verification of assembly rules for packaging. The assembly rules for a specific (advanced) package are informed by (1) the assembly technologies for individual dies (e.g., pick and place, micro transfer printing), and (2) the connection technologies (e.g., wire bonding, bumps, micro bumps). Our goal is to formally describe these different packaging technologies as a foundation for automatic package-level DRC runset generation for KLayout.

ACKNOWLEDGMENTS

We would like to thank R. Timothy Edwards and Matthias Köfferlein for their many years of work on Magic and KLayout respectively. Our thanks also go to all other contributors to these two open-source projects. This work is supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 16ME0484.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu. 2019. Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 4 pages. https://doi.org/10.1145/3316781.3326334

[2] GlobalFoundries PDK Authors. 2022. *GF180 PDK*. https://github.com/google/gf180mcu-pdk

[3] IHP PDK authors. 2023. *IHP Open Source PDK*. https://github.com/IHP-GmbH/IHP-Open-PDK/

[4] Magic authors. 2020. *Magic*. http://opencircuitdesign.com/magic/

[5] Qucs authors. 2017. *Quite Universal Circuit Simulator (Qucs)*. http://qucs.sourceforge.net/

[6] Qucs-S authors. 2023. *Qucs-S: Qucs with SPICE*. https://ra3xdh.github.io/

[7] SkyWater PDK Authors. 2020. *SKY130 PDK*. https://skywater-pdk.readthedocs.io/

[8] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. 2000. *Software Cost Estimation with COCOMO II* (1st ed.). Prentice Hall, NJ, USA.

[9] M. E. Brinson and V. Kuznetsov. 2016. A new approach to compact semiconductor device modelling with Qucs Verilog-A analogue module synthesis. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields* 29, 6 (2016), 1070–1088. https://doi.org/10.1002/jnm.2166

[10] E. Chang, J. Han, W. Bae, Z. Wang, N. Narevsky, B. Nikolić, and E. Alon. 2018. BAG2: A Process-Portable Framework for Generator-Based AMS Circuit Design. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*. 1–8. https://doi.org/10.1109/CICC.2018.8357061

[11] Efabless Corporation. 2023. *OpenLane Website*. https://efabless.com/openlane

[12] ECMA 2017. *The JSON Data Interchange Syntax* (2nd ed.). ECMA. https://ecma-international.org/publications-and-standards/standards/ecma-404/

[13] Global Market Insights. 2023. *Electronic Design Automation (EDA) Market*. Retrieved December 11, 2023 from https://www.gminsights.com/industry-analysis/electronic-design-automation-eda-market

[14] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. 2022. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer, Cham. https://doi.org/10.1007/978-3-030-96415-3

[15] E. R. Keiter, R. L. Schiek, H. K. Thornquist, T. Mei, J. C. Verley, K. V. Aadithya, G. J. Templet, J. D. Schickling, and G. L. Hennigan. 2023. *Xyce Parallel Electronic Simulator*. https://github.com/Xyce/Xyce

[16] A. Krinke. 2020. *Constraint Propagation for Analog and Mixed-Signal Integrated Circuit Design*. Number 474 in Fortschritt-Berichte VDI, Reihe 20. VDI Verlag, Dresden.

[17] A. Krinke, M. Mittag, G. Jerke, and J. Lienig. 2013. Extended Constraint Management for Analog and Mixed-Signal IC Design. In *2013 European Conference on Circuit Theory and Design (ECCTD)*. 1–4. https://doi.org/10.1109/ECCTD.2013.6662319

[18] M. Köfferlein and contributors. 2023. *KLayout*. https://www.klayout.de/

[19] J. Lienig and J. Scheible. 2020. *Fundamentals of Layout Design for Electronic Circuits*. Springer, Cham. https://doi.org/10.1007/978-3-030-39284-0

[20] A. Nassaj, J. Lienig, and G. Jerke. 2009. A New Methodology for Constraint-Driven Layout Design of Analog Circuits. In *2009 16th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. 996–999. https://doi.org/10.1109/ICECS.2009.5410838

[21] ngspice authors. 2023. *ngspice*. http://ngspice.sourceforge.net/

[22] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor. 1983. *A Collection of Papers on Magic*. Technical Report UCB/CSD-83-154. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1983/5295.html

[23] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč. 2016. Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web*. 263–273.

[24] B. Prautsch, U. Eichler, S. Rao, B. Zeugmann, A. Puppala, T. Reich, and J. Lienig. 2016. IIP framework: A tool for reuse-centric analog circuit design. In *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. 1–4. https://doi.org/10.1109/SMACD.2016.7520725

[25] Stefan Schippers and contributors. 2023. *Xschem*. https://xschem.sourceforge.io/

[26] M. Shalan and T. Edwards. 2020. Building OpenLANE: A 130nm OpenROAD-based Tapeout-Proven Flow. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–6.

[27] World Semiconductor Trade Statistics. 2023. *WSTS Semiconductor Market Forecast Fall 2023*. Retrieved December 11, 2023 from https://www.wsts.org/76/Recent-News-Release

[28] OpenRoad Team. 2023. *OpenROAD Website*. https://theopenroadproject.org/