Abstract Technology Handling for Generator-Based Analog Circuit Design

Benjamin Prautsch*, Uwe Eichler*, Torsten Reich*, Ajith Puppala*, Jens Lienig†
*Fraunhofer Institute for Integrated Circuits IIS,
Design Automation Division EAS, Dresden, Germany
{Benjamin.Prautsch, Uwe.Eichler, Torsten.Reich, Ajith.Puppala}@eas.iis.fraunhofer.de
†Dresden University of Technology, Dresden, Germany
jens.lienig@tu-dresden.de

Abstract—**Designing analog and mixed-signal integrated circuits is still a matter of comprehensive manual tasks. Although a variety of optimization-based and procedural generator-based analog design automation approaches have been presented, they still lack a proper handling of so-called expert knowledge in an abstract way. We present a new method to capture expert knowledge by an abstract, generator-based analog circuit description. This approach moves detailed procedural circuit descriptions further towards a high-level description. Using the presented method, the circuit is defined by generic code which is converted to an abstract graph representation. The graph is subsequently used to apply technology-specific design rules and further constraints to ensure DRC-clean and robust layouts. As a result, a much wider set of advanced technology nodes can be targeted by the same parameterizable, procedural circuit description compared to previous approaches. Therefore, re-use of dedicated circuit blocks is improved which both eases utilization by designers and supports circuit optimization.**

Keywords— Layout; Analog Automation; Generator; Technology Independence; Reuse

I. INTRODUCTION

The design of analog and mixed-signal integrated circuits demands a significant quantity of manual work done by expert analog designers. Therefore, both design time and quality strongly depend on the designer's experience, thus making analog design an uncertain process. As it is frequently mentioned in the literature, automation is rarely used in the analog domain, although many approaches already exist in academia and even commercial tools have appeared. One reason is that analog design constraints are usually considered in an implicit way, instead of an explicit fashion, since the explicit definition of such constraints is still challenging [1, 2].

Proposed methodologies to automate analog design can mainly be divided into optimization-based and procedural generator-based approaches [3, 4] (see Fig. 1), and the combination of both named template-based optimization [5, 6]. The main advantage of optimization-based approaches is their high flexibility. The user has to define the performance targets together with a very detailed set of analog constraints [7] to reduce the solution space. Complex constraint handling techniques such as propagation, transformation, and verification are necessary to consider constraints throughout complex design hierarchies [8]. Subsequently, an optimization is used to find the optimal solution of this constrained problem. For procedural generator-based approaches, on the other hand, the sequential manual design tasks are defined as a sequence of program commands, which create, parameterize, move, place, and align circuit elements. Such approaches enable fast, parameterizable generation of schematic and layout realizations to enable efficient reuse of approved modules while ensuring design safety [9, 10, 11, 12]. Due to the fixed procedure, they are typically less flexible regarding placement and routing but they enable distinct definition of detailed layout tasks by means of expert knowledge. This makes them especially suited for smaller circuits with strong requirements on quality or basic building blocks with regular structures. The definition of such expert knowledge is still a main challenge for optimization-based approaches [4]. Therefore, experts agree that each approach may not separately solve the analog design problem in its entirety, but a combination of both approaches is promising [4, 10].

In [4] the vision of a "bottom-up meets top-down design flow" is presented and "context-aware" module generators are found to be required. The proposed generators provide an extensive parameter interface to parameterize details of the concrete circuit to be aware of influences by the layout environment. Varying these parameters, one is able to consider LDE (layout dependent effects) such as WPE (well proximity effect). In advanced technology nodes, dependent design rules

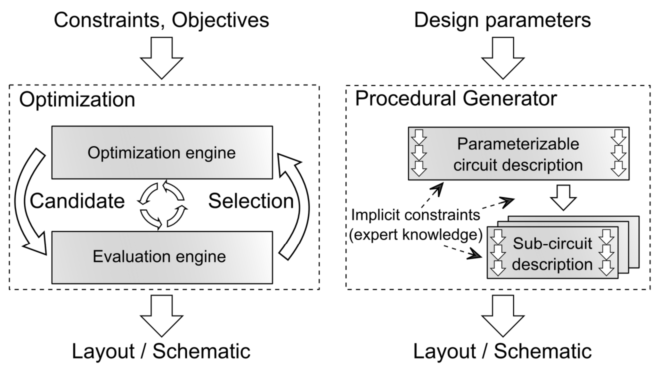

Fig. 1 Comparison of optimization-based (left) and procedural generator-based approaches (right), adapted from [2, 4]. Whereas a suitable solution is searched using an iterative method in optimization-based approaches, a sequence of commands defines a circuit solution in a deterministic way in procedural generator-based design. The latter re-use expert knowledge with the result of solutions previously conceived and captured in a procedural description, such as in Cadence's PCells concept [13].

ISBN 978-3-8007-4071-0

could also be considered this way. However, if such generators are in the inner loop of an optimization, the increasing count of parameters would tend to cause long iteration cycles. Moreover, in this phase the parameters would be decoupled from the particular layout task, although rules (e.g. dependent spacing) are known and could be automatically derived within the module generator in a deterministic manner. Therefore, an increased level of abstraction within procedural generators is needed to encapsulate the variety of parameters by means of a powerful generator programming interface.

A. State of the Art

Previous procedural analog automation approaches provide technology-handling especially for "neighboring" technology nodes with minor changes regarding the complexity of design rules. The solution of [12] provides an interface to encapsulate technological parameters by a large set of variables which are used within the code description of e.g. placement steps of layout components. It was applied in [10]. This approach works well as long as the technology variables are not dependent on the particular layout situation, which unfortunately is the case in advanced technology nodes. The framework in [11] also provides an interface to the technology. The authors claim that the technology independence of their generators is eased by the tool, but, since the programmer has to ensure it for each module explicitly, technology independence is not intrinsically ensured. Both approaches are well-suited to realize re-use for similar technologies. They create the full set of design data, namely schematic, symbol, and layout which is important for data consistency and an effective method to improve the productivity by new opportunities for analog design flows [9]. However, strongly different technologies cannot be handled by these generator description approaches in an appropriate way.

Template approaches often combine generators with optimization [5, 6]. The relative position of layout elements is defined using a generic template language such as presented in [14] resulting in a much more technology-independent representation. The template is then used as a set of constraints for the automatic placement. Auto routing is usually done in a separate step, but can also be part of the template [15]. The disadvantage of template-based optimization is that strongly technology-dependent module generators are still required to realize the particular layout, as mentioned in [5]. Advanced technology handling is necessary to improve both re-use of procedural generators and template-based optimization.

B. Our Contribution

We propose a new generator programming interface that contains a separate command layer for technology-dependent parts of the circuit description (see Fig. 2). This *technology abstraction layer* (TAL) separates detailed, technology-dependent placement, routing, and device-level parameterization commands from the topological schematic and layout description. This means that each elementary design step (generator command) is not directly executed by the framework. Instead, each command is used to successively create an abstract graph representation which allows the consideration of other layout elements during a design step, if necessary. Therefore, e.g. automatic consideration of the layout

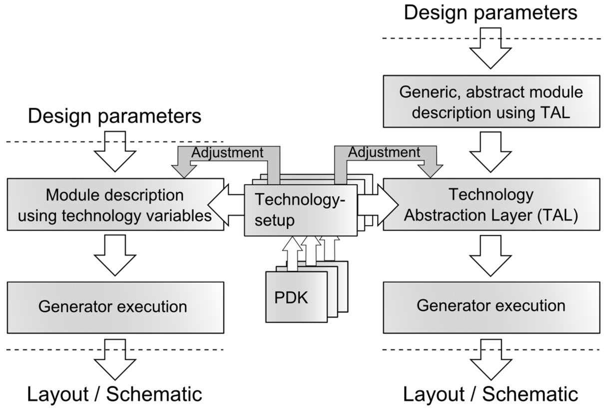

Fig. 2 Comparison of conventional technology abstraction (left) and our TAL-based generator approach (right). New technology nodes are adjusted according to the gray arrows. Dashed lines represent the "borders" within the design flow.

context is possible during the placement of a new element in relation to another one. This way, TAL saves design know-how even at this low level of complexity and covers a much larger variety of technology nodes which is essential for managing the complex design rules of advanced technology nodes.

This paper is organized as follows: Section II provides an overview of the main technology differences from a designer's point of view; Section III contains our new abstraction layer approach; in Section IV an example and its results are presented. The conclusion and outlook are given in Section V.

II. DIVERSITY OF TECHNOLOGIES

One of the first steps of analog design is the selection of a technology to be used. The choice depends on performance and commercial constraints and addresses a large variety of available technologies. Furthermore, updates of technologies can also behave like a change of the technology and force designers to re-design older circuits at least partly. To be aware of this diversity and to enable new module parameterization a generic technology handling by reusable generator implementations (i.e. parameterizable circuit descriptions) is desirable. From a designer's point of view, technologies are represented by process design kits (PDKs) which generally differ in number and type of layers, devices and their models, layer interconnects (vias), and rules, as well as their identifiers and parameters. For a true generic technology handling a detailed analysis has been performed and a software structure has been created to handle the particular technologies by a generic, procedural, object-oriented design description. This software structure can be easily extended for further technologies with other design rules.

A. Technology Parameters

Technology parameters can basically be divided into identifiers and technological data. Identifiers can be names of layers and devices and, thus, do not represent physical quantities. Technological data relate to physics, as e.g. the mobility of a particular MOS-transistor or the sheet resistance of a particular layer. These parameters are mapped by simple functions F_{sm} which map generic parameters p_g out of a set of generic parameters P_g to a set P_p of particular parameters p_p:

$$F_{sm}: P_g \to P_p,\ p_g \mapsto p_p$$

This mapping is implemented in a class *TechMap()* which is frequently specialized for a variety of elements in each PDK.

B. Library Elements

Each technology provides a library of elementary cells such as MOS-transistor or metal capacitor devices, vias, or substrate contacts. Devices are usually provided as parameterized cells, for example PCells [13], implemented in a design-tool-specific programming language. They create custom layout representations of the device each time an instance of such cell is to be accessed. Although the basic functionality of same devices is comparable for different technologies, the particular realization is diverse. These differences for cells include the cell identifier, parameters (and again their identifiers), size and pins (and their position) of both symbol and layout of a device, and the effect of device parameters on the resulting layout configuration (geometry).

A cell identifier and a part of the device parameters can be mapped using the aforementioned class *TechMap()*. However, in order to realize an overall generic description, an abstract class needs to be defined which encapsulates the remaining particular parameters in a generic manner. For example, we have defined a MOS-transistor class *TechMosDevice()* which inherits from an again abstract class *TechPdkDevice()*. A generic device may define a set of n generic device parameters $P_g = \{p_{g\,1}, p_{g\,2}, \dots, p_{g\,n}\}$, with each ith generic parameter having a set of generic values $V_{g\,i}$ of which one single value can be chosen. Each generic parameter value results in a variable number $m_i = n_{pg}(p_{g\,i})$ of particular parameters $p_p \in P_p$ of the device, resulting in a more complex mapping function $F_{cm\,i}$ for each ith generic parameter:

$$F_{cm\ i}: P_g \to P_p, p_{g\,i} \mapsto \{p_{p\,1}, p_{p\,2}, \dots, p_{p\,m_i}\}, i \in \{1, \dots n\}$$

For example, in our implementation a generic parameter to vary the connection style of the gate of a MOS-transistor is called "$gConnect$". Allowed generic values are {"$none$", "$poly$", "$met1$", "$met2$", ...} which, as in this case, can itself depend on technology (here: number of metal layers). A particular value results in a dedicated set of technology-dependent parameters which are applied to the concrete PCell of the PDK. This complex mapping is realized by a separate class hierarchy which eases the transformation between generic and particular parameters.

Unfortunately PDKs show a large variety in terms of PCell parameters. Therefore, the previously mentioned simple example could already fail, since the concrete PCell could miss parameters to realize the desired gate connection (e.g. certain metal layer). One possibility to overcome this problem is to utilize completely self-made device generators as in [12] or [16] which are defined using the generic generator interface. However, this would result in problems of acceptance by designers, since such generators are not validated by the PDK provider. Therefore, we have implemented device generators

Table 1: Classification of design rules and their applications

	Fixed Rule	Dependent Rule
Rule can be evaluated	Fixed Constraint (e.g. minimal width, fixed spacing)	Dependent Constraint (e.g. dependent spacing in a fixed, known environment)
Further information necessary	Fixed Constraint in design hierarchy context (e.g. density rules)	Dependent Constraint in design hierarchy context (e.g. spacing between different cells)

such that concrete PCells are always instantiated from the PDK. Only if necessary, missing parts are added to the PCell instance to obtain structurally generic devices including generic pin identifiers. This is done by a generic device wrapper description that was implemented once per device type.

C. Design Rules

Design rules can basically be separated into those which have a fixed value and those which can have multiple values (dependent rule). The latter dependent design rules can only be evaluated if the concerned layout elements and their dimensions (and sometimes also their connectivity) are known, while the evaluation of fixed rules does not require such considerations. One could transform a dependent rule to a fixed rule, if e.g. the maximum value of a dependent spacing rule is used. However, this would result in huge waste of chip area, since these spacing values can, depending on the PDK, easily vary by a factor of up to ten for a single layer. Therefore, such simple solution has not been applied. The consideration of dependent rules necessitates the definition of dependent rule types in a generic manner to handle them within the TAL system. Furthermore, it is important to consider that some rules can be immediately evaluated during a design step, while others need further information from other design steps or the design hierarchy to be evaluated. Accordingly, these possibilities can be summarized as shown in Table 1.

For example, a dependent spacing rule can be calculated as soon as layers, shapes, and sizes of the concerned layout elements are defined. A density rule (percentage of a layer area in the whole design or a defined window of the design),

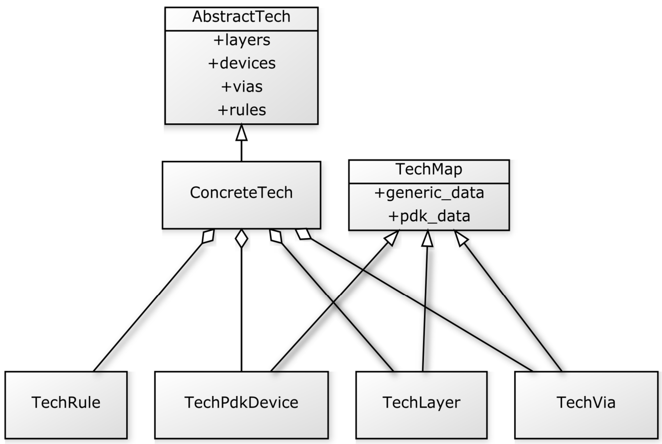

Fig. 3 Sketch of the implementation for the generic technology data handling. Data of a PDK and their mapping functions are stored in objects for design rules, devices, layers, and vias. Each of the latter objects is further refined.


```
// define NMOS-, and PMOS-transistor (Nodes)
nmos= generate_device(Mos,nmos_params)
pmos= generate_device(Mos,pmos_params)
// place PMOS-transistor (Edge)
pmos.plc_above(nmos, ul, ll, nmosLay, pmosLay,
                         spc_default)
```

Fig. 4 Pseudo-code of an example placement of a MOS-transistor in relation to another one. Both transistors are represented by nodes within TAL. The placement task will generate an edge, where "ul" and "ll" are reference points of the boundary box identifying upper left and lower left, respectively. For a more accurate placement, definition layers of layout elements can be defined, as well as an additional constraint.

on the other hand, can only be considered if such layout blocks are finished. Since the layout must be completed to evaluate the density rule, this evaluation cannot take place during a particular design step. Therefore, each step of the procedural description of an integrated circuit needs to be transformed into a placement relation within an abstract graph representation (e.g. relative placement with dependent spacing rules). The graph is subsequently analyzed to evaluate the particular rule necessary to obtain a DRC-clean layout. As a result, relations not yet evaluated during the current step must be stored to be considered later. In addition, a method must be available to enable proper consideration of such rules by means of overriding previously calculated rule values.

III. ABSTRACTION OF TECHNOLOGIES

In the previous chapter a variety of technological differences, seen by the designer, was presented which should all be part of TAL. Therefore, TAL must be divided into the following parts to consider these differences:

(1) Technology data mapping: All data and parameters to be mapped are collected in a generic manner and stored in object representations of design rules, devices, layers, and vias (see Fig. 3). A particular version of the class *ConcreteTech()* is defined for each applied PDK. This class contains all necessary data for the proper circuit generation in the particular technology. All instantiated classes are further refined by inheritance and provides a variety of information (e.g. maximum current density or sheet resistance of a layer or interconnects). The simple mapping is realized as discussed in Section II.A. Complex mapping, as discussed in Section II.B, is done for each element of a concrete technology within class *ConcreteTech()*. All parameters can be accessed through the methods defined in the abstract class *AbstractTech()* which serve as a generic interface for each particular technology.

(2) Design task support: Each layout command of TAL is encapsulated such that particular technology information is not directly used. This way the generator description is as generic and simple as possible. Effectively, each design task adds nodes, edges, or information to the abstract graph representation (see the code in Fig. 4, which creates the graph in Fig. 5a). Currently, tasks are implemented which realize instantiation, placement, and alignment of arbitrary layout elements. Required data are automatically evaluated

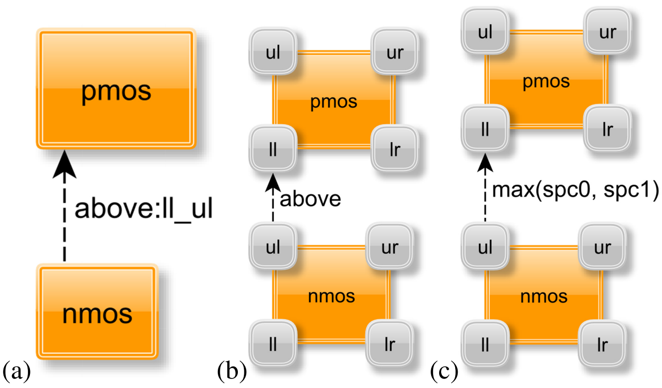

Fig. 5 Graph generation through the generator API by the "above" placement task with reference points "ll" and "ul". Instance-based representation (a), detailed placement graph with logical reference points (b), and detailed placement graph with resolved additional rule where "spc" denotes spacing (c).

depending on the particular design task the generator programmer defines. The relation between class objects for layout information and class objects, which handle design tasks, are shown in Fig. 6. For example, detailed placement, which supports dependent design rules, would need to access the structure and relation of involved layout elements (i.e. reference points and relative placement direction of layout elements) as shown in Figs. 5a and 5b. The required spacing rule including its dependency on the layout situation is stored in the *technology data mapping* and is used by the framework to automatically calculate the minimum spacing for the current task dependent on the abstract graph representation. In addition, a user-defined minimum spacing (here: "spc_default", see Fig. 4) can still be defined, which is comparable, but not equal, to the technology variables of former approaches (see Fig. 2). The difference is that this user-defined minimum spacing can still be overridden by another rule, if necessary. In the example of Fig. 5a, the layout situation is first defined by the shapes of both the "pmos" and "nmos" MOS-transistor instances and their placement relation ("above"). This layout situation is then used by the generic

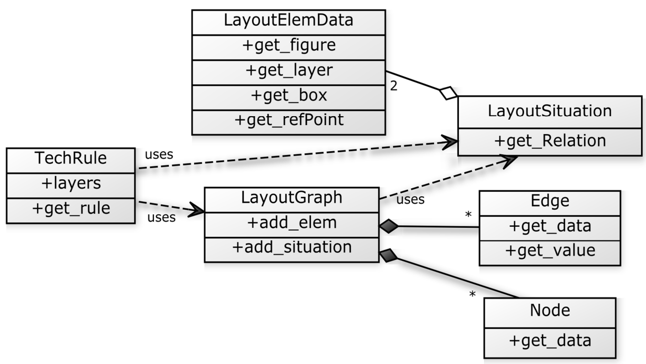

Fig. 6 Implementation of the abstract graph representation. Each layout situation is stored in an object of the class *LayoutSituation()*, which contains data *LayoutElemData()* of two layout elements and their relation. The overall graph *LayoutGraph()* stores this information in edges and nodes, which is used by the particular technology rule *TechRule()*.

spacing rule in form of a class object representation, which stores all information about both involved layout elements and their relation. Using this class object, the generic rule applies its strategies to evaluate the related rule value.

(3) Consideration of all rules: Since multiple rules can affect the same elementary design step, as discussed in Section II.C, a previous decision can be overridden. For example, during a design task an instance is placed dependent on the concrete minimum spacing design rules of a PDK (e.g. edge "above" in Fig. 5). Another rule (e.g. spacing rules for other layers of a placed instance or minimum layer density), forces the evaluated spacing to a larger value resulting in a actually larger spacing constraint for this particular placement step (e.g. edge in Fig. 5c).

IV. IMPLEMENTATION

A simple implementation of two transistors including routing has been executed in two technology nodes (180 nm and 40 nm) in order to show the capability of the approach (see Fig. 7). The technologies are very different regarding the properties discussed in Section II (i.e. device identifiers and parameters, dependent vs. independent design rules), thus enabling an efficient verification of our approach with reasonable effort. Using the new TAL, the generator can handle these differences by means of a graph construction step including mapping followed by the particular layout generation.

A. Graph Construction

An abstract graph representation is derived from the generic, abstract generator description to address the aforementioned challenges. This description contains each circuit element (devices, routing, etc.) and their relative positions to other circuit elements and, thus, represents a very dedicated template comparable to [14] and [15]. However, the difference with TAL is that e.g. spacing is not defined by a variable. It is directly calculated considering the current design task and the related advanced node design rule, making the code much more technology independent. The procedure first maps all parameters (see Section III.1) and then generates the generic devices to derive geometric information. The geometric details of these instances can still depend on the particular technology. Nevertheless, all pins and their positions are defined in a generic manner using the device generators mentioned in Section II.B. The spacing values (see dashed edges in Fig. 7a) are not explicitly defined yet, but are evaluated depending on both the particular layout environment and the concrete technology (see Section III.2). The graph contains all information necessary to enable both realizing the expert designer's aim and fulfilling concrete rules from the technology. Moreover, with TAL it can be checked if a generator description contains logical errors, as for example unwanted overlaps of layout instances. The simplified graph of the example including hierarchy is shown in Fig. 7a.

B. Generator Execution

Once the graph has been created, the particular rule of the concrete technology has to be evaluated for each layout task. Since the graph contains all relations of layout elements and

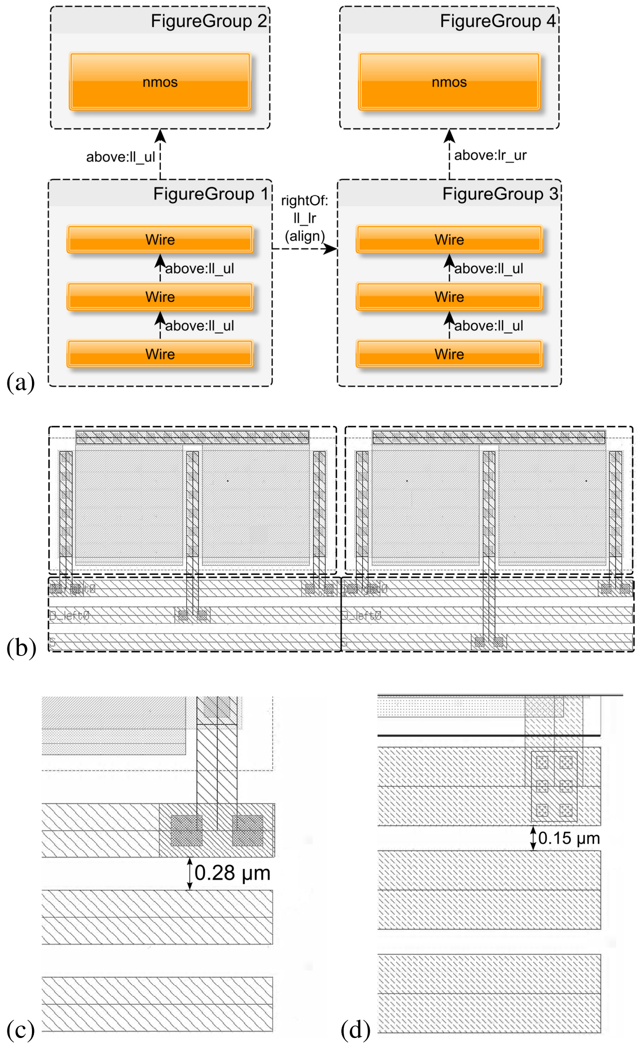

Fig. 7 Layout example created through TAL for detailed technology-aware layout description considering dependent design rules. Figure (a) shows the abstract graph representation (rectangle boxes represent layout elements, which can be organized in "FigureGroups"), (b) shows the overall layout example in 180 nm, (c) and (d) depict parts of this example in 180 nm and 40 nm, respectively.

their parameters, the particular object of a rule class can compute this information to return a resolved rule value (see Fig. 6). This result is then applied to the dashed edges shown in Fig. 7a. If another rule affects the same edge, a particular solution for each layout task is defined. In our previous example (see Fig. 5), the maximum of both rules is used since the layout task is a placement using the minimum allowed spacing value.

Figure 7 shows a part of the resulting layout for both 180 nm and 40 nm technologies generated by the same procedural generator description. All generic parameters of the generator are mapped to the particular ones. It is important to mention that for the wires (denoted as "Wire" in the graph) the minimum possible spacing of the 180 nm technology is static (fixed rule) while the 40 nm technology encompasses five different values, which are to be evaluated from thirty different cases of values for this layout situation. The lowest

ISBN 978-3-8007-4071-0

possible value within this set of dependent minimum spacing values is less than 150 nm, but since the rule is dependent on this particular layout situation, the actual value was set to the according spacing of 150 nm. It is obvious that a consideration of such dependent rules within the procedural generator description code itself, as it is done in former approaches, would be strongly technology dependent, as well as error-prone.

V. CONCLUSION AND FUTURE WORK

The presented approach is a step towards the conflation of both implicit expert knowledge and constraint-based analog circuit design automation. Our technology abstraction layer (TAL) enables a new explicit, well-readable and robust description of layout blocks while considering dedicated design rules for a wide range of technology nodes.

TAL is implemented such that new design rules from technologies not considered yet, as well as new design tasks, can easily be added into the software structure. Thus, the variety of target technologies will increase with reduced migration effort. This is supported by an automated import of technology data as well as a generic interface to add particular methods. These methods define how to apply new rules to particular design tasks and layout situations. Moreover, in future work the system will be extended such that further electrical constraints like layout dependent effects, current densities [17], or IR-drop are considered.

This work is part of the development of a complex Intelligent IP library of parameterizable and robust circuits for a variety of target technologies including advanced nodes. The library is planned to contain highly technology-independent elements (e.g. operational amplifiers, comparators, ADCs etc.) to ease the design of analog circuits. Moreover, the presented approach can be part of a technology-independent template-based optimization approach to support the further automation of analog design.

ACKNOWLEDGEMENTS

The presented work was partly supported by the European Union and the German Ministry for Education and Research BMBF within the project THINGS2DO (Ref. No. 16ES0240).

We would like to thank Mohamed Sabra for his extensive analysis of design rules of diverse technology nodes.

REFERENCES

[1] G. Jerke and J. Lienig, "Constraint-driven Design — The Next Step Towards Analog Design Automation," *Proc. of the 2009 International Symposium on Physical Design,* pp. 75–82, 2009.

[2] R. Rutenbar, "Analog Synthesis (and Verification) Revisited: What's Missing? Invited Talk," *Computer Aided Verification (CAV) Frontiers in Analog Circuit (FAC) Synthesis and Verification Workshop,* July 2011, http://www.async.ece.utah.edu/FAC2011/abstracts/RutenbarSlides.pdf.

[3] G. G. E. Gielen and R. A. Rutenbar, "Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits," *Proc. of the IEEE 88.12,* pp. 1825–1854, Dec 2000.

[4] J. Scheible and J. Lienig, "Automation of Analog IC Layout — Challenges and Solutions," *Proc. of the 2015 Symposium on International Symposium on Physical Design,* pp. 33–40, 2015.

[5] R. Martins, N. Lourenco and N. Horta, "LAYGEN II—Automatic Layout Generation of Analog Integrated Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions,* pp. 1641–1654, 2013.

[6] H. Graeb, F. Balasa, R. Castro-Lopez, Y.-W. Chang, F. V. Fernandez, P.-H. Lin and M. Strasser, "Analog Layout Synthesis – Recent Advances in Topological Approaches," *Proc. of the Conference on Design, Automation and Test in Europe,* pp. 274–279, 2009.

[7] A. Krinke, G. Jerke and J. Lienig, "Adaptive Data Model for Efficient Constraint Handling in AMS IC Design," *20th International Conference on Electronics, Circuits, and Systems,* pp. 285–288, 8–11 Dec 2013.

[8] A. Krinke, M. Mittag, G. Jerke and J. Lienig, "Extended Constraint Management for Analog and Mixed-Signal IC Design," *Circuit Theory and Design (ECCTD), 2013 European Conference,* pp. 1–4, 2013.

[9] T. Reich, H. D. B. Prautsch, U. Eichler and R. Buhl, "Silicon Proof of The Intelligent Analog IP Design Flow for Flexible Automotive Components," *Proc. of the 2015 Design, Automation & Test in Europe Conference & Exhibition,* pp. 403–404, 2015.

[10] A. Graupner, R. Jancke and R. Wittmann, "Generator Based Approach for Analog Circuit and Layout Design and Optimization," *Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 1–6, 2011.

[11] J. Crossley, A. Puggelli, H.-P. Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. An, A. Sangiovanni-Vincentelli and E. Alon, "BAG: A Designer-Oriented Integrated Framework for the Development of AMS Circuit Generators," *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference,* pp. 74–81, 2013.

[12] IPGen 1Stone Developer, [Online]. Available: http://www.ipgenme.de/eda-and-ip-products/1stone-developer.html. [Accessed 06 2015].

[13] Cadence, "Virtuoso Parameterized Cell Reference Product Version 6.1.6," 2015.

[14] A. Unutulmaz, G. Dündar and F. Fernández, "LDS based Tools to Ease Template Construction," *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012 International Conference,* pp. 61–64, 2012.

[15] A. Unutulmaz, G. Dündar and F. V. Fernández, "A Template Router," *Circuit Theory and Design (ECCTD), 2011 20th European Conference,* pp. 334–337, 2011.

[16] X. Jingnan, J. Vital and N. Horta, "A SKILL-based Library for Retargetable Embedded Analog Cores," *Conference on Design, Automation and Test in Europe,* pp. 768–769, 2001.

[17] J. Lienig, "Electromigration and Its Impact on Physical Design in Future Technologies," *Proc. of the ACM 2013 International Symposium on Physical Design,* pp. 33–40.

