Constraint-driven Design — The Next Step Towards Analog Design Automation

Göran Jerke
Robert Bosch GmbH
Reutlingen, Germany
www.bosch.com
goeran.jerke@ieee.org

Jens Lienig
Dresden University of Technology
Dresden, Germany
www.ifte.de
jens@ieee.org

ABSTRACT
The design of analog circuits has historically been a time consuming, manual task. The stringent constraints that must be considered simultaneously make the task particularly difficult, and are a major reason analog design has often not been automated. We believe that constraint-driven design is a prerequisite to analog design automation as it enables expert knowledge to be included in the design flow. This paper provides an introduction to the concept of constraint-driven physical design. First, we identify the major challenges in analog physical design, which we show are mostly constrained-related. We then provide an overview of the essential components of a constraint-driven design methodology. Finally, we discuss the impact this approach has on the analog design flow and design algorithms.

Categories and Subject Descriptors
B7.2[**Integrated Circuits**]: Design Aids

General Terms
Algorithms, Design, Verification.

Keywords
Analog design, physical design, layout, constraints, constraint-driven design

1. INTRODUCTION
While physical design automation of analog IC design has seen significant improvement in the past decade, it has not advanced at nearly the rate of its digital counterpart. This shortfall is primarily rooted in the analog IC design problem itself, which is significantly more complex even for small problem sizes, and which lacks a sufficiently comprehensive and exact descriptiveness with conventional approaches [1][6][10].

The quality of a design result is generally determined by the degree to which compliance constraints have been met and pre-defined optimization goals achieved. Due to the lack of identical expression and interpretation of design constraints in the analog-design flow context, most of the constraints in analog designs are specified and considered manually by expert designers (expert knowledge). Furthermore, analog constraints are often used implicitly (i.e., based on a designer's experience) rather than being explicitly defined, which prevents their effective use in design automation. Progress in physical design automation for analog ICs is urgently needed due to increasing design sizes and aggravating challenges such as more stringent reliability and robustness requirements, as well as a rapidly widening verification gap.

At present, analog circuits are typically designed using the schematic-driven layout (SDL) methodology, which consists of an interactive design style and a subsequent verification step. It is widely believed that this design style will be replaced one day by full-scale "analog design automation" similar to that of today's digital circuits. Rather than announcing this long-awaited solution, we present an approach we believe represents not only a realistic "in-between step" but also a necessary precondition (Fig. 1).

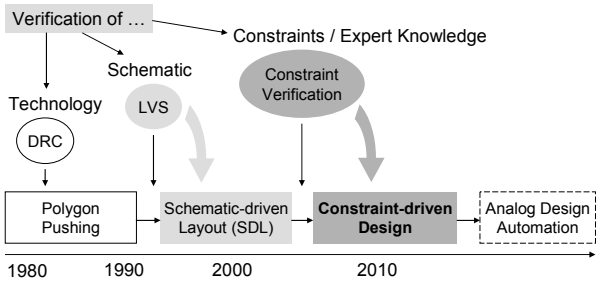

Figure 1. The evolution of analog physical design methodologies towards the goal of a fully automated analog design flow.

We believe the ultimate goal of fully automated analog design (analog design automation) can only be achieved if the current schematic-driven design paradigm evolves into a constraint-driven design. Our approach is based on the belief that we first need a methodology that allows for automatic inclusion of expert knowledge in the form of constraints, which too must be verified automatically. Only then we will be able to tackle the task of analog layout synthesis. In other words, we think that the abilities of "analyzing" and "verifying" are a precondition for "synthesizing" [11].

This paper provides an introduction to the concept of a holistic constraint-driven physical design approach for arbitrary ICs in general, and for analog ICs in particular. Thereby, we identify key similarities and differences between the physical design of analog

ISPD'09, March 29–April 1, 2009, San Diego, California, USA.

and digital IC design, and the corresponding challenges, which we show are primarily constraint-related.

We then give an overview of the constraint-driven design flow and its essential components. We discuss the fundamental components required in a constraint-driven analog design methodology, such as constraint representation, management, transformation, and verification. We then present the impact this methodology has on the overall IC design flow, the core design of EDA algorithms, and the required paradigm adjustments needed for analog physical design approaches. Our paper concludes with an outlook of open problems towards the final goal of analog design automation.

2. PROBLEM DESCRIPTION

2.1 The Design Problem

In general, any (IC) design problem represents a complex and constrained optimization problem. The degrees of design freedom linked to the optimization problem span a multi-dimensional solution space which is (at least partially) constrained by the given global design constraints. A feasible solution for a specific design problem is obtained by sequentially removing all degrees of design freedom while traversing and reducing the solution space and considering all context-relevant constraints and application profiles.

This reduction is done by sequentially transforming functional representations with many degrees of design freedom into equivalent ones with fewer degrees of design freedom. For example, using suitable methods one may transform a given functional specification into a netlist (netlist = functional representation of the given specification), which is then subsequently transformed into a floorplan, a placement order, a wired layout and finally a physical mask layout which contains no further degree of design freedom (physical layout = functional representation of the given netlist).

Several transformations (design steps) can be active at the same time, especially for analog IC designs (Fig. 2). The strategy of how and when to remove a degree of design freedom during the design phase depends on several context-specific factors. Among others, factors may be static or dynamic in nature and may include the type of IC application, its usage profiles, reliability and robustness requirements, as well as the current problem situation in a design phase with its linked constraints.

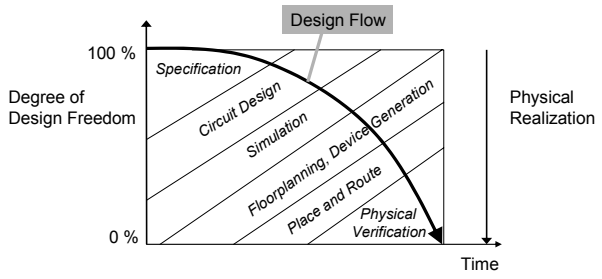

Figure 2. Simplified design flow for analog IC design where steps are typically overlapping and tightly linked. Multiple design steps can be active at the same point of time [11].

2.2 Constraints vs. Optimization Goals

In general, design constraints *must* be fulfilled whereas design goals *may* be fulfilled. An optimization goal that must be fulfilled hence represents a constraint, and must be treated as such. On the other hand, if we have a given design constraint that may be fulfilled, it should then be considered an optimization goal. The design goal is to achieve design results that fulfill all given constraints and which offer the highest level of achievement toward pre-defined optimization goals.

2.3 Constraint Classification

Each single design constraint (hereafter, constraint) must belong to at least one corresponding design object. For example, an IR-drop constraint belongs to at least two net terminals of the same net; or the chip area belongs to the ICs top cell, and so forth.

A constraint can be given in either an *implicit* or *explicit* form. Implicit constraints may be given as plain textual notes or they may arise from assumptions intrinsically built into circuit descriptions or design algorithms. Examples of such constraints are the placement requirements of differential pair transistors – they must be placed symmetrically in order to maximize device matching. While this is obvious to any layout designer, the inclusion of such complex rules into both layout *and* verification tools is often not possible using current methodologies. Hence, due to its often non-formal nature, implicit constraints cannot be utilized for any type of controlled constraint-driven design. On the other hand, explicitly given constraints are accessible to design algorithms and thus are a primary requirement for any constraint-driven design flow.

Each constraint is assigned a specific *constraint type* that represents a classification property for the same class of constraints. Constraint types have a clearly defined unit that belongs to the physical, electrical, mechanical, mathematical or geometrical domain (domain), or a combination of domains (e.g., the constraint type "IR-drop" has the unit Volt, the type "Delay" the unit Seconds, etc.). The relevance and impact of a constraint type strongly depend on the specific design context.

Constraint types are generally assigned to one of the following four categories: (1) technological constraints necessary for manufacturing, (2) functional constraints that guarantee the intended IC functionality, (3) design-methodical constraints that arise from the attempt to reduce design complexity and to guide transformations, and (4) commercial constraints that, among others, arise from chip area or packaging requirements.

Furthermore, constraint types are divided into so called "simple" and "complex" constraint types. Constraints that belong to a *complex* constraint type are conditional and linked with other constraints of any type. Constraints belonging to a *simple* constraint type are not linked at all. For example, the constraint `(IR-drop(pin_A,pin_B)<0.1V)` represents a simple constraint whereas a constraint `(IR-drop(pin_A,pin_B)<0.1V AND net topology must be star-shaped)` represents a complex constraint because the second term "net topology" is linked to the first term "IR-drop". This distinct consideration of simple and complex constraint types is a prerequisite for a constraint-driven design flow implementation.

2.4 The Gap between Analog and Digital Design Automation

Analog IC designs often contain only a small number of devices as compared to digital IC designs. Nevertheless, the effort required to design analog function modules often matches or even exceeds the

effort for the digital modules. This is mainly due to a much richer set of constraints, typically complexly and tightly linked, that must be considered simultaneously and which may span several domains (Section 2.3).

On average, each design object (instance, net, path, etc.) in an analog IC design must comply with a larger and more comprehensive set of constraints to fulfill its intended function, as compared to digital design. The primary reason for this observation is the higher level of functional abstraction offered in digital designs. This allows more robust operation requiring fewer constraints to guarantee the intended function, as compared to the quasi low-level mode of operation in analog designs.

Furthermore, many (low-level) constraints and even constraint types may yet be unknown when the analog design process begins, due the overwhelmingly complex nature of the corresponding design problem. This renders automatic top-level design planning for analog IC designs nearly impossible. It is one of the reasons that at present highly skilled design engineers are required to perform top-level design planning manually.

This constraint-related problem also makes algorithm and tool development for analog IC design much more difficult, because the number of specific design algorithms may increase with each new simple constraint type. Considering today's conventional design-algorithm development approach (one constraint type and one algorithm to handle it), this approach falls short when it comes to new complex constraint types which vastly outnumber simple constraint types. This represents one of the primary reasons why analog design automation is lagging behind its digital counterpart and why this gap is growing.

Another important reason for the design gap is rooted in the level of completeness and consistency that can be applied theoretically to the consideration of constraints during IC design. Due to the above-mentioned possibility to use a comparably small set of well-defined and well-treated constraint types for digital designs, related present-generation design algorithms and design tools can offer consistent and seamless design solutions in the digital domain.

Since comprehensive and unified constraint description model are not used in today's analog design algorithms and design tools, many analog-centric constraint types must currently be considered manually. Their consideration during design is, hence, mostly inconsistent and non-comprehensive.

Any inconsistent or non-comprehensive consideration of constraints and constraint types widens the existing verification gap. This gap exists since DRC and LVS cannot fully include the verification of all (possible) constraints and constraint types. A tremendous amount of research effort has already been expended for the tailored consideration and verification of mostly simple constraint types (e.g., timing, IR-drop). Nevertheless, a complete and unified approach is still missing that is capable of dealing with all simple *and* complex constraints during the entire design and verification phase.

Another difference between analog and digital IC designs can be found in the way the functional transformations, i.e., the design steps, are linked and carried out. While most design steps in digital IC designs are separated from one other, the design steps for analog IC designs are typically overlapping and tightly linked (see Fig. 2). For example, device generation, floorplanning, placement and global routing usually occur simultaneously. Thus, any analog-oriented physical design tool must consider all requirements at the same time. This greatly reduces the impact of EDA point tools, such as pure analog placement and custom routing tools, in favor of a constraint-driven design flow.

To address the current shortcomings discussed in this section, a holistic design approach is required. Its cornerstones will be discussed in detail in Sections 3 and 4.

3. COMPONENTS OF A CONSTRAINT-DRIVEN DESIGN FLOW

A design flow that considers all relevant constraints in a consistent and comprehensive manner is subsequently denoted as a constraint-driven design flow. To achieve this, several distinctive design flow components must be available and special requirements must be met.

We provide an overview of essential components of an analog constraint-driven design flow in this chapter. Specifically, the fundamental concepts related to a constraint-driven design methodology, including constraint representation and management, generation of new constraints (constraint derivation) and constraint verification are discussed.

3.1 Constraint Representation

As mentioned earlier in Section 2.3, constraints must be given in an explicit form to be applied in an automated constraint-driven design flow. From a formal point of view, constraints define relations between instances of a set of either free or fixed design variables. A relation between independent variables represents a simple constraint. Relations between dependent variables can be mapped into combinations of simple constraints. This allows the definition of high-order constraints which represent complex constraints [3].

Additionally, all constraints *and* all related design parameter data must be uniformly represented in an abstract form while preserving constraint type information. The conversion of constraints into a uniform representation must be complete and unambiguous. Uniform representations, such as CLP form (CLP: Constraint Logic Programming) [2][4], enforce a common understanding of constraints and constraint types among all involved design and verification tools. This is a primary requirement to enable the construction of multilateral design and verification algorithms necessary to address the analog constraint-driven design problem (see Section 4.2).

The relation between design variables, representing a constraint, is either linear or non-linear. For instance, the electrical point-to-point resistance between two net terminals in a layout decreases linearly with an increasing wire height, whereas wire capacitance increases non-linearly due to fringe field effects.

Design variables are given as either nominal values, worst-case values or as statistical values depending on the applied design style (nominal design, safety-critical worst-case design, statistical design, respectively). Hence, the representation of constraints must necessarily consider both the design style and the nature of a design variable.

3.2 Constraint Management

The task of constraint management is to administer the storage for all constraint data while synchronizing the link between constraints and the design data [7]. It also enables context-specific access to constraints for design and verification algorithms.

The management system is responsible for keeping constraints up-to-date, which requires close interaction with design data management systems and the design algorithms that manipulate constraints. The system must guarantee the consistency of constraint information while corresponding design data is manipulated.

Constraint-driven design algorithms require fast access to relevant constraint information. Access must be context-sensitive, where the context for example may represent a local cell in the design hierarchy, a temporary result database of a filter or search operation, etc.

Constraint management also incorporates the propagation of constraints (1) in the existing design hierarchy, (2) across the borders of design objects and design steps, as well as (3) within a virtual design hierarchy. All three propagation types strongly depend on the constraint type *and* on the specific single constraint, and may be applied simultaneously. Furthermore, constraint propagation can be performed as either static or dynamic propagation. Static propagation assigns properties directly and permanently to design objects whereas dynamic propagation assigns constraints to design objects "on the fly". In terms of constraint consideration, the static and the dynamic propagation methods are equivalent.

The propagation within the existing design hierarchy can be either performed top-down or bottom-up. For instance, a net shielding constraint may be assigned from the chips IO pad down to a specific instance terminal in a sub-cell. The shielding constraint is then propagated to all nets in the hierarchy until the final terminal is reached. A cell spacing constraint assigned to a cell object is to be propagated as a bottom-up constraint if EMC requirements prevent critical cells to be placed too close to each other.

Cross-border propagation assigns constraints across a set of design objects or across several design steps. For instance considering the first case, top-down propagation of a net-shielding constraint may continue after it has passed a resistor element during the net traversal. The second case may be viewed as a global scope in which a specific constraint is active.

Top-down and bottom-up propagation are required to be performed simultaneously while traversing a propagation tree representing a virtual design hierarchy. For instance, if a chip IO pad is located in a sub-hierarchy cell, then net-shielding constraints must be propagated to the relevant top-level cells and then down-propagated to the final instance terminals. Clearly, this propagation type strongly depends on the constraint type and on the specific design objects linked to a single constraint of this type. Constraint propagations are often applied in virtual design hierarchies.

3.3 Constraint Derivation

The process of generating new constraints is denoted as constraint derivation when performed using either a functional specification or a set of existing design data while performing a design step. In principle, the derivation process can create new constraints belonging to the technological, functional, design methodical, or commercial constraint type category (see Section 2.3).

The derivation process is based on (1) direct derivation rules, (2) deduction processes by using a logic calculus or (3) the designer's knowledge. Constraint derivation based on circuit simulation and design verification results can be seen as a special type of the rule-based derivation method. Strictly speaking, the constraint transformation discussed in the next section is a form of *indirect* constraint derivation due to its transformation-based creation of new constraints during the design process (see Section 3.4).

In general, constraint derivation processes can be performed on either *constant* (e.g., a netlist) or *non-constant* (e.g., flexible wire width) design data. A derivation based on constant design data does not change the design data it uses, whereas a derivation based on non-constant design data can modify or add design data during the derivation process. This differentiation is important because in the case of non-constant design data, dynamic feedback loops are likely to be created by the design algorithms that need to be considered. This feedback-loop problem and relevant applications are discussed in more detail in Section 4.2.

The derivation of constraints using a functional specification is based on constant design data, since the specification itself is normally fixed. Similarly, a design step can fully or partially make use of constant design data. For instance, if the derivation rule "if (condition) then assign constraint C to design object X" will not modify design object X, then the derivation is based on constant design data.

Deduction-based constraint derivation can be seen as a high-level extension of rule-based derivation methods. Here, a so-called reasoning system draws logical conclusions from the given set of design data and constraints and then applies a set of constraint derivation rules to relevant design objects. For example, based on a logical conclusion that CMOS and bipolar transistors belong to the same category of devices, a specific constraint rule may be applied to both CMOS and bipolar transistors, even in the case where the derivation rule was only defined for one transistor type. This functionality permits the development of higher-level constraint derivation methods and offers a necessary and important level of abstraction required for multi-technology analog reuse.

3.4 Constraint Transformation

Constraint transformation is an essential component of the design flow since it translates higher-level constraints into a set of equivalent lower-level constraints (top-down constraint transformation) and vice versa (bottom-up constraint transformation) [5]. Lower-level constraints created by top-down constraint transformations will further constrain the available solution space, thus reducing the number of global degrees of design freedom. Additionally, more than one transformation might be possible from a given higher-level constraint resulting in different sets of lower-level constraints.

Any transformation process must guarantee a complete and unambiguous transformation result, otherwise the related constraint problem cannot be solved. The transformation depends directly on the underlying physical, electrical, design methodical or commercial problem, and hence, it is specific to each constraint type. The same applies to bottom-up constraint transformation, i.e. inverse transformation, which must be defined for constraint verification purposes (see Section 3.5).

A transformation relation can only be defined for simple constraints and their specific context. Complex constraints represent

combinations of simple and other complex constraints. The relation of sub-constraints specific to each complex constraint type is not affected by the transformation since the transformation of simple constraints only focuses on their specific context. This statement is made here since it is assumed that any acceptable top-down transformation will only produce lower-level constraints that do not affect higher-level constraints. In the case where lower-level constraints affect higher-level constraints, design iterations are very likely to occur (i.e., the design steps must be reversed and then guided in a different direction).

The decision as to which transformations to use is context-dependent. For example, suppose the functional specification results in a specific maximum IR-drop between a chip IO pad and a specific instance terminal in a sub-cell. Assuming that the current flow in the respective wire connection is known and constant, application of a forward transformation to this top-level constraint may result in pad and sub-cell placement constraints *and* a corresponding set of routing constraints. While considering all other connected constraint problems as well, a constraint-driven design algorithm can then decide whether the placement in this context is more critical to deal with than the routing and act accordingly. For instance, in case the placement is fixed, the final transformation of the top-level IR-drop constraint would then yield a set of routing constraints and local degrees of design freedom (i.e., routing design parameters such as wire length, layer, wire width). These can then be used by a routing algorithm to find a suitable interconnect layout.

3.5 Constraint Verification

Constraint verification comprises the verification (1) whether a set of existing constraints is fulfilled for a design and (2) if a given set of constraints raises mutual conflicts. Constraint verification is an essential component of the constraint-driven design flow. With the assumption that the IC application functionality is fully specified and *all* required top-level constraints are defined, constraint verification fills the existing verification gap (see Section 2.4). Thus, it ensures correct application functionality as well as design quality, reliability and robustness. It must be noted here that the term "constraint verification" as used in this paper does not include checks for the usefulness and applicability of a given top-level constraint.

As mentioned earlier, a rich set of constraint types must be considered during the design of analog ICs. The majority of these constraints are complex constraints whose fulfillment cannot be verified with conventional verification approaches. This is due to the fact that all of today's verification approaches require one specific verification tool or one embedded verification algorithm for each constraint type to be verified. Clearly, conventional constraint one-to-one verification approaches (one verification algorithm for one constraint type) are in general not feasible for the *complete* verification of analog IC designs. Making matters worse, many complex constraint types and corresponding constraints are unknown at the beginning of the design.

The verification of simple constraint types (e.g., delay, IR-drop, placement orders) requires specialized verification algorithms that are embedded into verification tools and frameworks. In general, complex constraints *should not* be verified by specialized one-to-one verification algorithms due to the inflexibility and in-extensibility of these approaches.

The first approach to address the verification problem for complex constraints, the meta-verification approach, was introduced in [3]. The core idea in meta-verification is that each complex verification problem can be divided into smaller and usually independent verification problems for simple constraints, which in turn can be verified using existing verification algorithms. The definition of meta-verification tasks can reference design data and access functionality, as well as verification functionality that is provided by external design and verification tools (Fig. 3). The meta-verification framework creates an abstraction layer around multiple design and verification tools, and it manages correct execution of the defined meta-verification tasks.

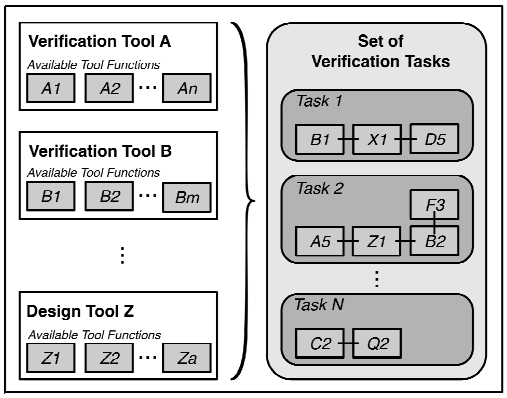

Figure 3. Meta-Verification of complex constraints. A combination of specialized tool functions (*A1…An, B1…Bm*, etc.) are offered by multiple design and verification tools (A…Z) to define high-level verification tasks for each complex constraint [3].

The CLP-based verification approach in [3] can handle independent as well as coupled, i.e. dependent verification problems. It also allows the detection of mutual constraint conflicts by drawing logical conclusions from the given constraint and design data information.

The definition of verification tasks for a meta-verification system is usually done as follows. First, the verification task must be defined and formalized by the designer of the constraint rule set. Second, the formalized verification task is then coded in the language of the meta-verification system. The constraint rules are subsequently used by circuit and layout designers to perform the verification tasks, whereas the application of these rules may depend on the context-specific verification problem.

Finally, it has to be noted that constraint verification is divided into *static* and *dynamic* constraint verification, based on the constancy of the constraint and design data. For example, any sign-off verification of an IC design must be based on constant design and constraint data, hence, static constraint verification is applied in this case. Nevertheless, constraint-driven design algorithms can also use constraint verification for specific "what-if" analyses. Since these algorithms can change design and constraint data during their analyses and during the design step, the related constraint verification is based on dynamic data. Hence, the latter case represents dynamic constraint verification. Both, static and dynamic constraint verification can be applied to either the full set of constraint and design data, or to a context-specific subset.

3.6 Constraint Sensitivity Analysis

Constraint sensitivity analysis (CSA) [9] is very helpful in allowing designers to understand the impact of local design decisions and to find root causes in case compliance requirements cannot be met with the given set of constraints. CSA determines the context-specific sensitivity of design parameters in relation to their assigned constraints. The sensitivity information can be either visualized or it can be used to drive and support design decisions. Sensitivity analysis is the key to the power of decision analysis in situations where the influence of design parameters is not known precisely, since it considers the (dynamic) context in which constraints apply.

4. IMPACT ANALYSIS

Next we discuss the impact of a constraint-driven approach on the overall IC design flow, the core design of EDA algorithms, and the required paradigm adjustments for analog physical design approaches.

4.1 Impact on Design Flow

A holistic approach to analog design automation requires several new design flow components (hereafter referred to as "components") that enable a constraint-driven IC design. Essential components of this holistic flow (constraint management, derivation, transformation and verification) have been discussed in detail in Sections 3.2 - 3.5. Other components, such as constraint sensitivity analysis (see Section 3.6) and constraint visualization, are not essential but can improve constraint-driven design. The impact of essential and non-essential components on the analog IC design flow will be discussed in this section.

First, all essential design flow components must be available at any stage during IC design. Essential components complement existing standard design flow steps (e.g., circuit simulation, schematic entry, floorplanning, placement, etc.), and hence, they must run concurrently to each of the design steps.

Next, the derivation, application and usage of constraints must be done comprehensively throughout the process. Any breach in the constraint application flow will most likely lead to inconsistent design and constraint data. Conclusive constraint verification would then be impossible.

Significant additional effort must be expended to develop and verify the initial set of *efficient* constraint derivation, constraint deduction and constraint verification rules. The practical application of meta-verification has revealed that the sequence in which simple sub-constraints of a complex constraint are executed is likely to have tremendous impact on the required verification time. Thus, verification rule optimization requires a deep understanding of the underlying verification task. For example, suppose there are short-running and long-running sub-verification tasks defined in a specific meta-verification rule. If feasible for the specific verification task, it may be beneficial to shift all long-running sub-verification tasks at the end of the rule so they are executed after the short-running sub-verification tasks. Late sub-verification tasks in a meta-verification rule may not be executed if an earlier call to a possibly short-running sub-verification task already revealed constraint violations, thus preventing unnecessary and potentially long-running sub-verification tasks from being executed.

Nevertheless, practical experiences with meta-verification rule development [3] has revealed that the required initial effort is very similar to the effort needed for the development of DRC and LVS rule sets. The reuse of constraint derivation as well as meta-verification rules is very simple and efficient since, in general, data and rule abstraction can be used for technology, design and constraint data.

The constraint management component must offer flexible, context-sensitive, and fast access to constraint data for any affected design and verification algorithm. Contrary to conventional constraint management solutions, it is essential now to consider real and virtual design hierarchies and to be capable of propagating constraints within any real or virtual design hierarchy tree (see Section 3.2).

The constraint management system must be capable of managing simple and complex constraints. Presently, most commercially available constraint management systems keep design and constraint data separate from each other. The databases for design data and constraint data must be unified since constraint data management is an integral part of the constraint-driven design flows backbone, and hence, is as important as the management of design data.

It is needless to say that any suitable constraint-management component must guarantee constraint consistency at low-level (keeping the constraint and the constraint owning design object in sync). It must also consider the various high-level requirements of design data management systems. The latter requirement especially can lead to significant data consistency issues if design data and constraint data cannot be bound into one data set that can be treated as a single data entity.

As mentioned earlier, DRC and LVS are only capable of covering technological constraints required to ensure manufacturability. Constraint verification must thus complement the LVS check and other specialized checks (e.g., timing checks) required for sign-off verification in order to guarantee the intended circuit functionality. Automatic constraint verification offers much greater verification coverage and reproducibility than manual verification. The overhead for meta-verification is often negligible when compared to the required run-time of the referenced verification tools for sub-verification tasks [3]. In our experience, practical application of automatic constraint verification for automotive ICs has shown that the benefits of achievable design verification quality clearly exceed the required effort for the development and maintenance of meta-verification rules.

The required overhead for static constraint verification is typically significantly smaller than for dynamic constraint verification. The additional overhead in the latter case is primarily caused by the cumulative data latency that occurs if design and constraint data are frequently accessed by the verification framework. Hence, low-latency data access can significantly speed-up dynamic constraint verification. For static constraint verification, design and constraint databases are usually accessed only once during initialization, thus avoiding data-access latency issues.

4.2 Impact on Design Algorithms

While firsthand experience with constraint verification algorithms already exists, the application of the new design flow components for constraint-driven design generation is rather new and hence experience is still limited. Nevertheless, in this section we will discuss the impact constraint-driven design has on design algorithms, including design planning, and we introduce several new and powerful concepts for constraint-driven IC design. While

some of these design approaches are new, others, such as the application of the constraint sensitivity analysis or the introduction of standardized algorithm interfaces, have already matured and thus have led to new insights into the analog design problem.

Present design algorithms are generally built for a special purpose (e.g., specialized design algorithms that focus on floorplanning, placement, etc.). While this may result in several benefits, such as fast execution time, this also brings several significant limitations that prevent further advances in analog design automation. Throughout this section, present design algorithms are denoted as "conventional algorithms" and the required new class of constraint-driven design algorithms as "constraint-driven algorithms".

A primary limitation in conventional algorithm design is the narrow focus on fast, but low-level execution *without* an implementation of standard data interfaces. These interfaces create a layer around the core algorithm. This layer is required to connect a design algorithm to the design and constraint databases as well as to other concurrently executed design algorithms. Thus, all design algorithms share a common understanding of design and constraint data. Standard data and communication interfaces are, hence, an integral part of any constraint-driven design and constraint verification algorithm.

Standard algorithm interfaces enable the modularization and abstraction of constraint-driven design algorithms. The abstraction of their algorithmic work greatly improves algorithm reuse and flexibility because a single algorithm can be used to solve similar design tasks (this concept is similar to algorithm abstraction available in various programming languages). In turn, this flexibility enables the construction of *high level* constraint-driven algorithms that utilize modularized low-level design algorithms in order to perform specific design tasks on a higher level of abstraction.

High-level design planning algorithms greatly benefit from the constraint sensitivity analysis (CSA, see Section 3.6). First, this due to the determination of relevant design parameters and constraints in a specific design context which provides a design-step-specific limitation of constraints. Second, CSA can be used as a method to identify design task parallelism by searching for temporary groups of design variables and constraints that are either not or only very weakly coupled with each other. For these groups, the next design step can then be performed independently of each other. Note that the independency of design variable and constraints in these groups may only be temporary, and hence, may not exist anymore after a design step is completed.

A dynamic hierarchy of concurrent design tasks can thus be established in which all affected design algorithms perform functional transformations (instead of conventional distinct design steps). These transformations are governed by either a fixed execution regime or by more flexible approaches such as high-level design planning algorithms.

Another major advantage in the construction of higher-level design algorithms is the possible dynamic consideration of new constraint types without the need to introduce major low-level algorithm and tool changes. High-level design strategies can now be used to solve low-level design problems by eliminating degrees of design freedom in a top-down methodology. This approach typically leads to better design results because low-level constraints are now less likely to break high-level constraints (see Sections 3.3, 3.4).

4.3 Paradigm Adjustments for Analog Design Approaches

The consideration of a rich set of constraint types and a corresponding large number of constraints is tightly linked to the analog IC design problem. As mentioned earlier, all constraints must be defined and used consistently. Constraint derivation, transformation and verification are now mandatory design flow components that must be used throughout IC design. A new core component of the design flow is the constraint management system that must be capable of handling simple and complex constraints across multiple design steps, as well as across multiple design and verification tools.

As can be seen in Fig. 2, the analog IC design flow exhibits overlapping design steps to account for concurrent design problems. In order to address the tight interaction between these design steps and to consider the concurrent nature of the analog design problem, all artificially introduced boundaries between existing design steps must be gradually dissolved. The removal of degrees of design freedoms must occur gradually rather than abruptly in order to keep them available for design optimization as long as possible.

Comprehensive constraint verification is often more important than the application of analog design generation algorithms [8]. Since analog IC designs have a rather small number of devices, their layout generation can still be done semi-automatically. In contrast, the constraint verification requires automatic approaches due to the size and complexity of the related analog verification problem in modern IC designs.

The reuse of analog layout often fails because small differences between designs may prevent a direct reuse. This is because all degrees of design freedom were already removed from the layout. However, the consistent definition of constraints between design objects (e.g., sub-circuits) allows design reuse of structural information that includes constraints. The structural information represents the most valuable part of the design knowledge, and hence enables more flexible reuse since relevant degrees of design freedom are not fixed yet. In that respect, analog design automation should address low-level layout generation and high-level design planning as discussed in Section 4.2.

5. OPEN PROBLEMS AND OUTLOOK

Despite the recent advances in constraint-driven design for analog IC design, there are several problems that need to be addressed in the near future to further broaden the applicability of analog design automation approaches.

Methods to check the completeness of a set of constraints and constraint (meta-)verification rules, as well as the achieved verification coverage, must be developed to guarantee IC functionality, reliability, robustness, etc. The set of meta-verification rules must be optimized to allow time-efficient constraint verification. Today, such optimization is done manually but automatic rule-optimization methods should be developed to reduce that burden.

As mentioned earlier, constraint sensitivity analysis is a powerful tool to drive and support high- and low-level design decisions, and to develop high-level design algorithms that allow more gradual IC design. The scalability of existing constraint-sensitivity analysis approaches is still limited to a few thousand design variables, which is already sufficient for mid-sized analog blocks with typically in

the range of several hundreds of analog devices. Application to top-level design problems requires development of new complexity reduction methods, as well as fast constraint sensitivity calculation methods to improve scalability.

Key factors for next generation analog design automation are design techniques that reduce the degree of design freedom gradually rather than abruptly while performing several conventional design steps concurrently. This will require that the current artificial boundaries between conventional design steps be (gradually) dissolved in the future. While breaking with conventional design approaches, this paradigm change could lead to a new class of (higher-level) design algorithms that bring us one step nearer to the goal of full-scale analog design automation.

ACKNOWLEDGEMENTS

We would like to thank Jan Freuer for his scientific work related to constraint engineering and his input to this work. We also thank Jürgen Scheible for the numerous fruitful discussions related to the topic of this paper.

REFERENCES

[1] Chang, H., Charbon, E., Choudhury, U., Demir, A., Felt, E., Liu, E., Malavasi, E., Sangiovanni-Vincentelli, A. and Vassiliou, I. 1999. A top-down, constraint-driven design methodology for analog integrated circuits. Springer Verlag, Norwell, MA.

[2] Cohen, J. 1990. Constraint logic programming languages. Commun. ACM, 33(7):52–68.

[3] Freuer, J., Jerke, G. , Gerlach J. and Nebel, W. 2008. On the verification of high-order constraint compliance in IC design. In Proc. Design, Automation and Test in Europe (DATE), 26–31.

[4] Jaffar, J., Michaylov, S., Stuckey, P. and Yap, R. 1992. The CLP(R) language and system. ACM Trans. on Programming Languages and Systems, (July 1992) 14(3):339–395.

[5] Malavasi, E. and Charbon, E. 1999. Constraint transformation for IC physical design. In IEEE Trans. on Semiconductor Manufacturing, vol. 12(4) (Nov. 1999), 386–395.

[6] Malavasi, E., Charbon, E., Felt, E. and Sangiovanni-Vincentelli, A. 1996. Automation of IC layout with analog constraints. IEEE Trans. CAD of Integr. Circuits and Systems, 15(8):923–941.

[7] Malavasi, E., Charbon, E., Arsintescu, B. and Kao, W. 1998. A constraint management system for IC physical design. In Proc. XI. Brazilian Symp. on Integr. Circuit Design, 240–243.

[8] Nassaj, A., Lienig, J., Jerke, G. 2008. A constraint-driven methodology for placement of analog and mixed-signal integrated circuits. Proc. of the 14th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS), 770-773.

[9] Ríos I. D. 1990. Sensitivity analysis in multi-objective decision making. In Lecture notes in economics and mathematical systems. 1990, vol. 347, ISBN 3-540-52692-7.

[10] Rutenbar R. and Cohn J. 2000. Layout tools for analog ICs and mixed-signal SoCs: a survey. In Proc. Int. Symp. on Physical Design (ISPD), 76–83.

[11] Scheible, J. Personal communication.