Extended Constraint Management for Analog and Mixed-Signal IC Design

Andreas Krinke* krinke@ifte.de
Maximilian Mittag† maximilian.mittag@de.bosch.com
Göran Jerke† goeran.jerke@ieee.org
Jens Lienig* jens@ieee.org

*Dresden University of Technology, Dresden, Germany
†Robert Bosch GmbH, Reutlingen, Germany

***Abstract*—The consideration of a growing number of design constraints is becoming a bottleneck in the design of analog and mixed-signal integrated circuits and is blocking more, much-needed automation in this area. In this paper, we propose a solution to these issues with a new methodology for constraint propagation and transformation. This technique allows designers and software tools to consider all relevant constraints when modifying a design, regardless of where these constraints were originally created. We integrated our ideas in an industrial design flow. The implementation of an electrical constraint type demonstrates the practical relevance. With constraints of this type the ON resistance of power stages in smart power ICs can be limited for the first time.**

I. Introduction

The design of analog and mixed-signal integrated circuits (AMS ICs) is dominated by manual, non-automated tasks, compared with the extensive automation of digital design. The lack of automation is essentially caused by the great number of constraints that need to be considered during analog design [1]. These constraints define mandatory requirements on design parameters. On the one hand, constraints can capture the engineer's intent, e.g., the relative positions and projected cell sizes in a floorplan. On the other hand, they can express parameter requirements. An example is the maximum permissable voltage drop (IR drop) across a wire for module to work properly. Long-time, analog design constraints used to be implicit, existing only as so-called "expert knowledge" or, at best, as informal notes on a schematic sheet [2]. Recently, the concept of constraints in AMS design has been formalized and increasingly integrated into various design tools (e.g. [3], [4]). Using these tools, one can assign constraints to elements in the currently edited cell. These constraints become part of the design data and can be verified automatically.

The major drawback of this approach is the limited visibility and verifiability of constraints. Currently, constraints are generally only visible and verifiable in the original cell—where they were created in the first place. However, so-called hierarchical constraints (see next section) reference elements in other cells (e.g. nets and instances) and, therefore, are of relevance outside of the original cell. The same applies to constraints that do not use the design hierarchy for information transfer, such as the connectivity-based IR drop constraint mentioned earlier. In both cases, designers and software tools do not have the information at hand as to whether design decisions in the current cell influence constraints declared in some other cell. This problem is solved by *propagating* constraints throughout the design. *Transformation* as an extension of this concept allows the conversion of constraints and therefore, the derivation of additional information.

This work was supported by the German Ministry of Education and Research (BMBF) under grant 01M3195B.

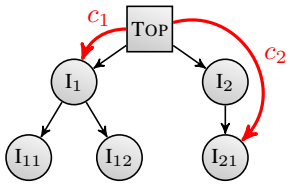

Figure 1. Constraint c_1 is local: The referenced instance I_1 belongs to the same cell as c_1 itself. Constraint c_2 is hierarchical; it belongs to cell TOP and refers to instance I_{21}, which does not belong to TOP. Modifying I_2's cell might influence c_2. Pins, instance terminals and nets are not shown in this graph.

A. State of the Art

Modern software tools for the design of AMS circuits (e.g. Cadence Design Framework II [3]) support the common hierarchical design approach that breaks the circuit into cells. The design may contain multiple *instances* (copies) of each cell. The integrated constraint management systems allow users to create new constraints that belong to a specific cell. Each constraint refers to a set of design elements (e.g. instances and nets), called constraint *members*. Each of these members can be either local or hierarchical. While local members belong to the cell of the constraint itself, hierarchical members are located in cells further down in the design hierarchy (see Fig. 1). In addition, every constraint is of a specific *type* that defines the parameters of all derived constraints and the verification procedure used to check if those constraints are met (see Section II). An example is the *resistance constraint type* used to constrain the resistance of wire segments. All constraints of this type have to provide maximal and minimal resistance values and are verified using a type-specific resistance model.

Current constraint management systems, e.g. the one integrated in [3], support only connectivity-based propagation. Furthermore, it is not possible to step over specific devices that are connected by a net. Constraint transformation is not supported either. Today's verification tools, e.g. Mentor Graphics' Calibre PERC, support topological constraints. They check the existence of circuit structures (e.g. for ESD protection) that are required by design rules [5]. However, this verification step does not provide assistance during the process of designing a circuit.

B. Our Contributions

In this paper, we present two extensions to constraint management systems: (1) *Propagation* distributes constraints based on connectivity across the design hierarchy. This information is thus available in all relevant cells. (2) *Transformation* allows the automatic creation of new constraints from a single original

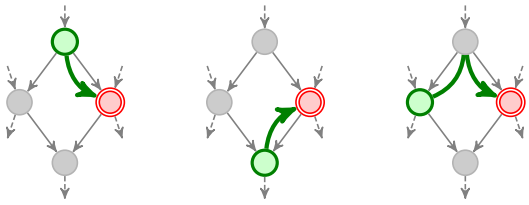

Figure 2. The three types of constraint propagation. Top-down propagation (a), bottom-up propagation (b) and bottom-up top-down propagation (c) [2]. Cells with hierarchical constraints are drawn as ○; the current cell is marked by ◎.

one. In combination, both methods enable the visibility and verifiability of constraints in all cells, where they potentially have influence on design decisions. Thus, all constraints can be considered continuously—regardless of where in the design hierarchy they were originally created. To demonstrate the practical relevance of our approach, we implemented an R_{ON} constraint type for the first time. Constraints of this type enable the ON resistance between package pins to be limited. Such constraints are transformed and propagated based on pin connectivity.

II. Problem Formulation

A constraint $c \in C(cell)$ belongs to the context of a specific *cell*, represented by the set C of all its constraints. They are defined as tuple $c = (t(c), M(c), P(c))$ where $t(c) \in T$ is the constraint type, $M(c) = \{m_1, \ldots, m_j\}$ is a set of constraint members and $P(c)$ is a tuple $(p_1, \ldots, p_k)$ of specific values for all parameters of $t(c)$. Members are design elements to which c has been assigned. Each member can be either *local* or *hierarchical*. Local members belong to the same cell as c itself, while hierarchical members are located in another cell further down in the hierarchy (see Fig. 1). If at least one constraint member is hierarchical, we call the constraint itself *hierarchical* as well. For each constraint type, there exists a Boolean-valued function that verifies all corresponding constraints:

$$\text{verif}_t : \{c \in C : t(c) = t\} \to \{\text{True}, \text{False}\}.$$

Given a constraint of type t, verif_t returns True only if the constraint is fulfilled. The constraint propagation problem is to guarantee visibility and verifiability of hierarchical constraints in *all* cells where they might influence design decisions.

III. Constraint Management

During the design of an integrated circuit, all cell modifications (e.g. adding a device) should obey all relevant constraints (e.g. some resistance limit) at any time. Aggravating this situation, these constraints may exist in all cells that are hierarchically connected to the current cell. Two cells are hierarchically connected, if there exists a path between instances of the cells in the design hierarchy graph (cf. Fig. 1). Our approach propagates these constraints by deriving new ones in all relevant cells. Hence, they become visible and verifiable.

Besides top-down propagation, which distributes constraints further down in the hierarchy, Fig. 2 shows two additional types of propagation. As an example, bottom-up propagation supports constraints on the absolute position of cells in the layout. Cells higher up in the hierarchy influence this position and, therefore, these constraints have to be propagated. Finally, bottom-up top-down propagation allows arbitrary distribution of constraints in the hierarchy, e.g., along connectivity. Our approach implements

```
1:  procedure TopDownPropagation(H, currentCell)
2:      Visited ← ∅
3:      Queue ← H[currentCell]
4:      while Queue ≠ ∅ do
5:          parentalCell ← Dequeue(Queue)
6:          if not Visited[parentalCell] then
7:              Visited[parentalCell] ← True
8:              for all c ∈ C(parentalCell) do
9:                  ▸ Check whether a constraint member
10:                 ▸ is accessible from the currentCell.
11:                 if ∃m ∈ M(c) : m ∈ currentCell then
12:                     ▸ Create new constraint in currentCell.
13:                     c′ ← Move(c, currentCell)
14:                     C(currentCell) ← C(currentCell) ∪ {c′}
15:                 end if
16:             end for
17:             Enqueue(Queue, H[parentalCell])
18:         end if
19:     end while
20: end procedure
```

Figure 3. Top-down propagation of constraints to the current cell requires the examination of all cells higher up in the hierarchy. Every constraint with at least one member being accessible from *currentCell* is propagated. Move (line 13) adjusts the paths of the members and introduces new parameters.

constraint propagation in two stages: (a) top-down propagation and (b) general propagation. Their combination supports, among other things, all three types of constraint propagation.

A. Top-Down Propagation

For top-down propagation, we limit the search for relevant constraints to cells that are higher up in the hierarchy. For example, in Fig. 1 the cells corresponding to Top and I_2 are considered during constraint propagation to I_{21}'s cell. Our algorithm in Fig. 3 stores all cells that might contain relevant constraints in a queue for later examination.

A constraint is propagated, if at least one of its members is reachable from the current cell by going further down in the hierarchy. Newly created constraints have the form $c' = (t', M', P')$. The derived type t' defines additional parameters that help identify the original constraint while retaining the original values $P(c)$. For M', all members' paths are adjusted to start at the current cell.

B. General Propagation

Our approach to bottom-up and bottom-up top-down propagation utilizes a new kind of function, that is specific for a constraint type t:

$$\text{propagate}_t : \{c \in C : t(c) = t\} \to c'$$

The propagate_t function takes a constraint c of type t as argument and returns a single propagated constraint $c' = (t', M', P')$ whose type t' extends the original type t (cf. previous section). These new constraints can be located in all cells higher up in the hierarchy. Therefore, bottom-up propagation is directly supported. Bottom-up top-down propagation is implemented by performing top-down propagation, as described in the previous section, *after* bottom-up propagation via propagate_t. In order to support all three propagation types, we significantly extended the algorithm for top-down propagation presented in Fig. 3 as follows. Now, the search for relevant constraints not only inspects cells higher up in the hierarchy, but all cells that are reachable via hierarchical paths. For each constraint c with a corresponding function $\text{propagate}_{t(c)}$, this function is called resulting in a propagated constraint c'. Afterwards, top-down propagation creates the actual constraints in the currently edited cell.

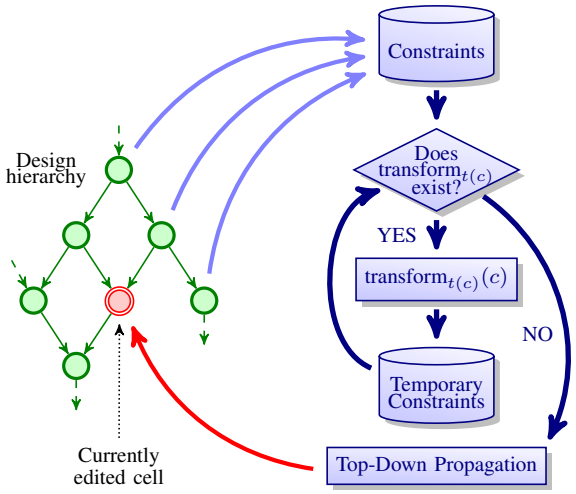

Figure 4. The new methodology for constraint propagation and transformation into a cell can easily be integrated into the design flow.

C. Constraint Transformation

Constraint transformation enables the derivation of constraints from existing ones while manipulating their types, members and parameters. As an example, if the maximum allowed voltage drop and maximum current for a wire are available as constraints, a constraint for the maximum allowed resistance is deducible. We integrated constraint transformation in the design flow by further generalizing the propagate$_t$ function presented in the previous section. Ultimately, we specify a transform$_t$ function for each transformable constraint type t. This function is defined as:

$$\text{transform}_t : \{c \in C : t(c) = t\} \rightarrow \{c'_1, c'_2, \dots\}$$

In contrast to propagate$_t$, multiple constraints of arbitrary type can be returned. As an example, one IR drop constraint can be transformed into multiple resistance constraints. Depending on the returned constraint(s), this function can replace propagate$_t$ and perform both, propagation and transformation. It is important to note that while exact knowledge of the design hierarchy is required for finding all relevant constraints, their subsequent transformation may use arbitrary design information. One example is the transformation of connectivity-based constraints utilizing the netlist of the circuit.

Fig. 4 shows the final process of propagation and transformation. It starts in the currently edited cell by examining the constraints of all hierarchically connected cells. If no transformation function is associated with a constraint's type, only top-down propagation is carried out. Otherwise, if transform$_t$ exists, it is executed, thereby creating one or more new constraints. These constraints are only temporary and thus, invisible. Transformation is repeated for all temporary constraints until top-down propagation is the only remaining choice. Not until then does each influenceable constraint become visible and verifiable in the currently edited cell.

IV. Experimental Results

A. Implementation

The methodology we developed is generic and, therefore, independent of the specific design environment. For demonstration purposes, we implemented the process as an extension to the constraint management system of Cadence Design Framework II, using the built-in scripting language SKILL++ [6].

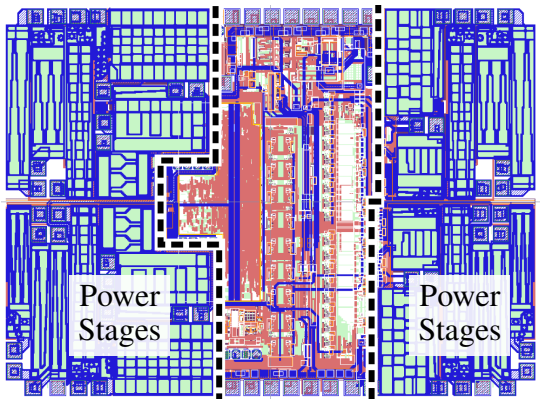

Figure 5. Layout of an industrial smart power IC featuring a large number of power stages left and right of the dashed lines. Each power stage corresponds to a single MOSFET in the circuit's schematics.

Constraint transformation and propagation, as shown in Fig. 4, is activated when a user opens a cell. The implementation is divided into the following steps: (1) Determination of design hierarchy. (2) Identification of all constraints in hierarchically connected cells. (3) Transformation and propagation of all these constraints.

B. Complexity

The run-time complexity of our approach depends on the size of the design hierarchy as well as the number and types of constraints. For hierarchies with n individual cells that contain m given constraints, the search for relevant constraints is $\mathcal{O}(n+m)$. The run-time of subsequent constraint transformation and/or propagation depends on the number of constraints of a certain type and the complexity of the corresponding transformation functions. In addition, these functions can create an arbitrary number of temporary constraints that might be transformable by themselves. Because these properties are specific to a project and the set of available constraint types, we are unable to characterize more comprehensively their impact on run-time. Nevertheless, it is self-evident that the implementation of transformation functions should be optimized for speed. For instance, caching mechanisms can be used to avoid complex recalculations of data. Additionally, such data should be persistent between multiple design tool invocations.

C. Demonstrator

Using our methodology, we created a custom constraint type for the specification of maximum resistances between package pins of smart power ICs [7]. Such integrated circuits contain large MOSFETs (power stages) whose drain and source terminals are connected to IC package pins, either directly or via measure or protection circuits. These transistors switch high powers and, therefore, occupy large areas on a chip (cf. Fig. 5). One of the main design goals is a low ON resistance, R_{ON}, between package pins that belong to power stages. The specification defines an upper limit, $R_{\text{ON,max}}$, which directly translates into a design constraint. The new constraint type transforms $R_{\text{ON,max}}$ into maximum resistance constraints for all contributing circuit elements. For example, $R_{\text{DS,ON,max}}$ constraints are assigned to all transistors with their values being calculated from $R_{\text{ON,max}}$ and the current resistance values of all other elements between the two package pins.

Fig. 6 shows the outline of the transformation function. Because the transformation depends on the electrical connectivity

```
1: procedure TRANSFORM_{R_ON,max}(c)
2:     net_1, net_2 ← GetMembers(c)
3:     limit ← GetValue(c)
4:     netlist ← ExtractConnectivity(c, net_1)
5:     equations ← NodalAnalysis(netlist)
6:     constraints ← Solve(equations, limit)
7:     return constraints
8: end procedure
```

Figure 6. Outline of the transform$_t$ function's implementation for the $R_{\text{ON,max}}$ constraint type.

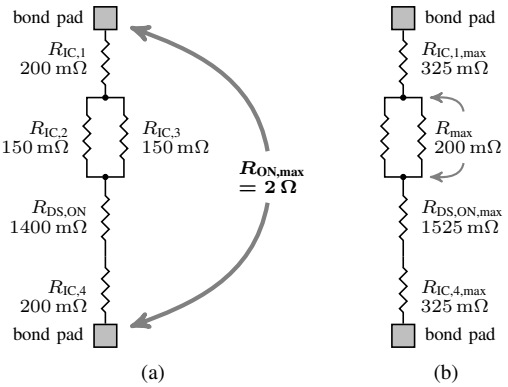

Figure 7. Exemplary extracted netlist with equivalent resistance values and $R_{\text{ON,max}}$ constraint (a). Constraint transformation and propagation automatically generate constraints limiting the resistance of each contributing sub-circuit (b). These limits are updated if resistance values change.

between the package pins, the first step is netlist extraction. The resulting netlist only contains devices from the physical design kit (PDK) and the nets in between. Next, the current resistance values of all PDK devices are determined. For this, we use either the resistance parameter from the device, or, in the case of transistors, a model for $R_{\text{DS,ON}}$ based on geometric transistor parameters [8]. Devices with unknown resistance are ignored and replaced by shorts later on. In order to calculate resistance budgets, we perform a nodal analysis. The resulting system of equations is automatically solved by the open source computer algebra system REDUCE [9], [10]. Its output defines *upper resistance limits* for each PDK device which remain valid as long as device resistance values don't change. Afterwards, the transformation function generates temporary hierarchical R_{max} and $R_{\text{DS,ON,max}}$ constraints for all sub-circuits being processed. These constraints are located in the original constraint's cell and become visible and verifiable in the currently edited cell after top-down propagation.

A simplified example of an extracted netlist for a power stage is shown in Fig. 7. In this case, the power stage (represented by $R_{\text{DS,ON}}$) is connected to bond pads via interconnects (represented by R_{IC}). Transformation and propagation of the $R_{\text{ON,max}}$ constraint generate $R_{\text{DS,ON,max}}$ and R_{max} constraints for all relevant elements. These resistance limits are valid as long as all other contributing elements remain constant.

We applied our methodology and the new type of constraint to three industrial smart power designs with 24, 27 and 67 power stages, respectively. Table I summarizes the results. In spite of the low number of power stages and R_{ON} constraints, the cells that influence these constraints occupy a large chip area. Depending on the cell to be modified, only a small portion of all temporary constraints becomes visible. Using our approach, designers were able to perform local modifications to these cells while simultaneously keeping an eye on the global impact of their changes.

Table I. COMPARISON OF THREE INDUSTRIAL DESIGNS WITH $R_{\text{ON,MAX}}$ CONSTRAINTS.

Circuit	design1	design2	design3
No. of cells	278	335	184
No. of power stages = No. of R_{ON} constraints	24	27	67
No. of cells relevant to R_{ON}	32	23	28
Chip area occupied by relevant cells	42 %	58 %	67 %
No. of temporary constraints	78	93	208

V. CONCLUSION

In this paper, we presented a new methodology for constraint propagation and transformation. Regardless of where in the design hierarchy a constraint was created, these fundamental operations enable the constraint's visibility and verifiability in all cells where it is relevant for design decisions. The methodology supports constraint types whose propagation and transformation depends on hierarchy and/or on connectivity. To the best of our knowledge, it is the first to include the ability to step over specific design elements while traversing nets. Beyond that, the integration of arbitrary application-specific constraint types is possible.

We believe that our approach is an important advance in the automation of analog and mixed-signal IC design. It provides a basis for the automatic consideration of *all* constraints which is a fundamental requirement for analog synthesis as aspired to in the long run. The methodology allows designs to be as correct as possible at all stages of the flow. Cell reuse becomes much more efficient because all constraints assigned to a cell's instances are visible when the cell is being modified. The demonstrated creation of a completely new, and complex, constraint type emphasizes the practical relevance. Initial application to the design of AMS circuits in an industrial design flow has shown its usefulness by significantly easing constraint management, facilitating cell reuse and reducing the number of design iterations.

REFERENCES

[1] R. A. Rutenbar, "Design Automation for Analog: The Next Generation of Tool Challenges," in *Proc. Int'l Conf. on CAD*, ser. ICCAD, 2006, pp. 458–460.

[2] G. Jerke, J. Lienig, and J. B. Freuer, "Constraint-Driven Design Methodology: A Path to Analog Design Automation," in *Analog Layout Synthesis – A Survey of Topological Approaches*, H. E. Graeb, Ed. New York: Springer, 2011, pp. 271–299.

[3] Cadence Design Systems, Inc. (2006) Speeding Design of Custom Silicon – The Virtuoso Custom Platform. [Online]. Available: http://www.cadence.com/rl/Resources/white_papers/Virtuoso_WP.pdf

[4] P. Subasic and G. B. Arsintescu, "Constraint data management for electronic design automation," U.S. Patent 7 003 749, Feb. 21, 2006.

[5] H. Marquardt, H. Wagieh, E. Weidner, K. Domanski, and A. Ille, "Topology-Aware ESD Checking: A New Approach to ESD Protection," in *34th Electrical Overstress/Electrostatic Discharge Symposium*, 2012, pp. 1–6.

[6] T. J. Barnes, "SKILL: A CAD System Extension Language," in *Proc. 27th Design Autom. Conf.*, ser. DAC, 1990, pp. 266–271.

[7] B. Murari, F. Bertotti, and G. A. Vignola, Eds., *Smart Power ICs: Technologies and Applications*, 2nd ed. Berlin, Heidelberg: Springer, 2002.

[8] M. L. Kniffin, R. Thoma, and J. Victory, "Physical Compact Modeling of Layout Dependent Metal Resistance in Integrated LDMOS Power Devices," in *Proc. 12th Int'l Symp. on Power Semiconductor Devices and ICs*, ser. ISPSD, 2000, pp. 173–176.

[9] A. C. Hearn, "REDUCE: A User-Oriented Interactive System for Algebraic Simplification," in *Interactive Systems for Experimental Applied Mathematics*, M. Klerer and J. Reinfelds, Eds. New York: Academic Press, 1968, pp. 79–90.

[10] ——. (2004, Feb.) REDUCE User's Manual Version 3.8. Santa Monica, CA, USA. [Online]. Available: http://www.reduce-algebra.com/docs/reduce.pdf