Application of Mission Profiles to Enable Cross-Domain Constraint-Driven Design

C. Katzschke*, M.-P. Sohn*, M. Olbrich*, V. Meyer zu Bexten†, M. Tristl†, and E. Barke*

*Institute of Microelectronic Systems
Leibniz Universität Hannover, Germany
Email: {Carolin.Katzschke, Markus.Olbrich, eb}@ims.uni-hannover.de
†Infineon Technologies AG, Munich, Germany
Email: {Volker.MeyerZuBexten, Markus.Tristl}@infineon.com

***Abstract*—Mission Profiles contain top-level stress information for the design of future systems. These profiles are refined and transformed to design constraints. We present methods to propagate the constraints between design domains like package and chip. We also introduce a cross-domain methodology for our corresponding constraint transformation system ConDUCT. The proposed methods are demonstrated on the basis of an automotive analog/mixed-signal application.**

***Index Terms*—Mission Profiles and constraints, constraint transformation, cross-domain constraints, constraints in integrated circuits.**

I. Introduction

Mission Profiles cover specific requirements of systems to be designed. Eventually, design constraints can be derived from Mission Profiles and later be refined for usage at different levels of the design process. These constraints must be spread and applied across the design systems. Mature methods already exist for the exchange of digital circuit constraints (e.g. timing) between different EDA tools. In contrast, for analog circuits such methods do not exist today due to several challenging aspects tied to analog constraints. One aspect is the heterogeneity of the involved constraint types (e.g. electrical current and IR-drop, geometrical spacing and width). Therefore, a higher number of relations between constraints of different types and domains must be modeled and taken into account. Another challenge is the use of incompatible design tools from multiple EDA vendors for different design domains such as board, package and chip. Proprietary constraint systems for single tools exist, but the propagation and mapping of constraints from one domain to another is not satisfactorily solved today. This paper provides an overview about issues that arise in design flows with constraints that are used in more than one design domain (so called cross-domain constraints). Feasible approaches to address these challenges are presented and discussed.

The broadcast of constraints and constraint modifications inside a design system across multiple EDA tools requires well-conceived strategies.

Furthermore, a strategy is needed to manage rules that can map constraints and their constraint types from one design domain, such as board, package and chip, to another.

A system based on rule management is proposed that can be used for this purpose. In the following subsections the generation of design constraints driven by Mission Profiles is described and further the state of the art in constraint transformation and management is pointed out. Challenges in cross-domain flows and solution approaches are introduced in Section II. Our tool ConDUCT and the underlying constraint model are presented in Section III. In Sections IV and V we describe the transformation utility of ConDUCT and outline our concept that uses petri nets. We demonstrate our concept using the example of a window lifter in Section VI.

A. *From Mission Profiles to design constraints*

In the course of the project ResCar 2.0, a Mission Profile Framework has been developed. It provides management and transformation utilities for Mission Profiles. Mission Profiles are essential sources of additional design information to consider stress loads. They define stress profiles which must be covered by the final system [15]. To obtain the necessary constraints a comprehensive method must evaluate the content and summarize the contained information. This results in a set of constraints, each of which represents a critical parameter that is influenced by the corresponding stress factors.

An example application for Mission Profiles is the relation between environmental temperature and the required functional loads as the required currents for a window lifter of a car. Usually small currents are sufficient to move the window when temperatures are above the freezing point. However, if the window freezes and the motor gets blocked, higher currents greater by an order of magnitude are necessary to start the movement of the window. This critical situation is modeled via current stress profiles in the dependency of temperature.

The specification defines the required lifetime for the motor driver IC of the window lifter. The values of the intended lifetime and the current vs. temperature stress profile are parameters to calculate or estimate current densities for pins,

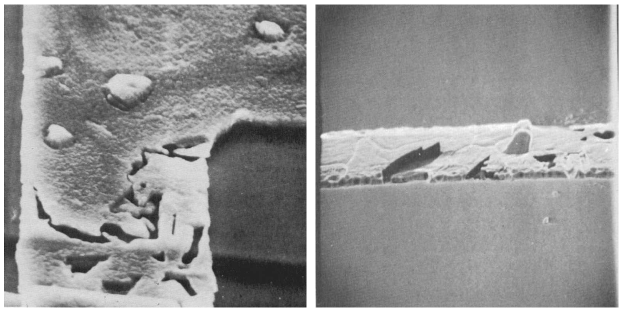
Fig. 1: Aluminum conductor paths damaged by electromigration [3]

transistors or conductor paths. One possibility to obtain a current density constraint is provided by Black's Law [2]. It depicts the relation between lifetime, i.e. mean time to failure, temperature and current density.

A current density that is too high can cause electromigration. During electromigration the conductor paths get thinner by diffusing electrons along the path. Figure 1 shows conductor paths damaged by electromigration. It is one example for possible failure mechanisms and we use it to demonstrate our cross-domain constraint methodology in the following sections. Nevertheless, our approach can be used to take care of other failure mechanisms.

The earlier mentioned Mission Profile Framework allows defining model transformations that appropriately convert profiles to refined models or design constraints. For the following sections of this paper, it can be assumed that constraints exist in several design tools each of which handles a different design domain.

B. State of the Art

Following [5] we define:

Definition 1: Constraint propagation is the derivation of one or more new constraints from a given set of constraints. New constraints are generated from existing constraints that are already part of the constraint set. [5]

We define *constraint transformation* as a process that converts constraints using transformation rules. In the course of constraint transformation design boundaries can be crossed and new constraints can be generated.

In all following sections we use this meaning of constraint transformation in cross-domain methodologies. Constraint management and transformation issues were addressed in different approaches:

In [1] constraint transformation is used only as a pure hierarchical and top-down methodology. Arsintescu et al. [1] divide constraints into two classes: Specification constraints and parameter constraints. Specification constraints $\mathbf{k}$ are input to transformations at the corresponding level. Parameter constraints $\mathbf{p}$ are output of transformations at the same level. In contrast to Jerke [5], Arsintescu [1] defines constraint transformation as application of a transformation function F that generates parameter constraints from specification constraints at the same level:

$$\mathbf{p} = F(\mathbf{k})$$

Further, the generated parameter constraints are partitioned into sets and passed to lower levels. The passing to lower levels is called propagation in [1]. During propagation given parameter constraints become specification constraints on the next lower level and are the basis of further transformation on this lower level.

Dhanwada et al. [4] developed an idea using a genetic algorithm that replaces the usual constraint transformation function. It is proposed as a hierarchical top-down approach. System-level performance values provide target values for the genetic algorithm. Performance values of the next lower level components in design hierarchy are also input to this genetic algorithm. Component performance values are varied during the execution of the algorithm. This allows evaluating system level performance values as a function of component performance values and assures that appropriate values are chosen as design constraints for components. Krinke et al. [6] contributed to the generation of hierarchical constraints by adding a bottom-up and mixed top-down/bottom-up transformation and propagation methodology. Constraints can be propagated throughout the entire hierarchy. Arsintescu [1], Dhanwada [4] and Krinke [6] provide solutions inside the design hierarchy. In contrast, we focus on challenges and methods that allow constraints to cross design boundaries.

We propose to use petri nets to model constraint transformation dependencies. There are some similar approaches in the fields of work flow management and analysis that also base on petri nets. Salimifard et al. [12] and van der Aalst [14] give a general description of work flow management: Work or design steps can be governed in modified petri nets that are named work flow nets (WF-nets). Tasks are mapped to petri net transitions. Pre- and post-conditions are represented as places. The logical work flow is modeled by arcs of the WF-net. WF-nets can be used to verify and analyze underlying workflows using petri net methodologies.

Another possibility to take advantage of petri nets in electronic design is mapping parts or functionalities of the design to a petri net and applying petri net analysis methods to verify or implement the design ([7], [11]) or make design decisions according to calculated metrics [8]. Our approach belongs to the last category. It provides auxiliary functions to support the design process but our methods do not control the process itself. Another main difference is that we map constraint types and the respective transformation rule system to petri nets. To govern complexity in electronic design, a design is divided into smaller design tasks ("divide and conquer strategy"). Multiple tools work on these tasks and in different domains. Issues arising from cross-domain constraint usage have been barely addressed, because existing approaches are limited to hierarchical systems. In Section II we list some issues that must be taken care of in a cross-domain constraint flow.

II. Challenges in cross-domain flows

A. Consistency

Often, several designers work on one project at the same time in different domains. Some develop the die, some work on package integration for one or more of such dies and others create the PCB for the device. Designers can also be located at different geographical places. In a cross-domain design flow these aspects need to be considered. The issue can be summarized as the fact that constraint information needs to be consistent and fulfill data integrity as any other design data in a cross-domain design system. It is important to provide up-to-date information to all designers. There are several different types of consistency. They result from different policies which define if a given order of message receiving for write and read operations is allowed in a distributed system [13]. According to the chosen type of consistency, corresponding broadcast algorithms can be applied for event-driven update messages.

Our approach uses a central database for persistent storage of design constraints that ensures additional consistency in database transactions.

B. Version Control Systems

Beside the issue that data integrity must be ensured it is also necessary to consider version control systems that are often used in each single tool. Constraint or design changes made in the chip design domain should only be propagated if the current revision of the design and its constraints are valid. That means if the local revision has not been updated, it is not guaranteed that design objects of the local revision exist in later revisions. When constraints are defined in the local revision for design objects that do not exist in later revisions anymore, broadcast is not reasonable. Defined constraints would have a deleted design object as constraint member in the later design versions. The constraint would be invalid in later revisions.

On the other hand, incoming update messages should only cause an update if the correct revision is in use. Listings 1 and 2 demonstrate a solution to take care of this issue. This strategy is well-known in software engineering and can be summarized as “get latest [design version] before check in / check out”.

```
startOutgoingUpdate(server, design, constrChanges) {

  localVersion = getCurrentVersion(local, design);
  serverVersion = getCurrentVersion(server, design);

  if (localVersion < serverVersion) {
    message("Please update design first.");
    abort();
  }

  applyAndStoreChanges(server, design, constrChanges
      );
}
```

Listing 1: Steps to broadcast outgoing constraint updates

```
startIncomingUpdate(server, design, constrChanges) {

  chosenConstr = selectConstr(constrChanges);
  localVersion = getCurrentVersion(local, design);
  serverVersion = getCurrentVersion(server, design);

  if (localVersion < serverVersion) {
    message("Please update design first.");
    abort();
  }

  importAndApplyUpdate(server, design,
    chosenConstr);
}
```

Listing 2: Steps to import incoming constraint updates

The following two sections elaborate the structure and ideas of our cross-domain tool (Section III) and highlight challenges and solutions regarding cross-domain transformation (Section IV).

III. Cross-Domain Constraint Tool

ConDUCT stands for Constraint Delivering and Updating Cross-domain Tool. It uses a database that implements the constraint model discussed in this section. A project can be created to store constraints of multiple design domains that are imported from the corresponding design tools. We use a constraint model similar to the model of Krinke[6]. However, the model must be extended to describe domain aspects:

$$c = (t(c), M(c), P(c), C(c)) \quad (1)$$
$$C(c) = \{c_1, .., c_n\} \quad (2)$$

with

$$\begin{aligned} t(c) &= \text{constraint type = (type, originTool)} \\ M(c) &= \text{members; design objects that c is applied to} \\ P(c) &= \text{parameters; set of parameter values} \\ &\quad\ \text{that are used for constraint verification} \\ C(c) &= \text{constraints; set of child constraints} \end{aligned}$$

The constraint type requires the information in which tool the original constraint was defined. Constraints are instances of constraint types. Constraint types define which kind of design objects are covered by the respective constraint and how the verification function evaluates constraint parameters. For cross-domain usage it is necessary to extend the constraint type by the additional information in which design tool it was initially defined. Design tools define and verify constraints with different parameters.

We have to ensure that constraints can be safely synchronized in the corresponding design tool. There are different underlying constraint models that belong to the respective tool of the domain. Hence, we extend our constraint model and define that constraints can also consist of multiple other constraints.

In most cases the supported recursion depth of constraints in design tools is one or zero. That means if a constraint contains child constraints, these child constraints do not have child constraints themselves. Design tools use these composed constraints to define so-called constraint sets. Constraint sets can quickly be replaced by other constraint sets and allow managing and testing different constraint and technology configurations.

IV. Constraint transfer and transformation across domains

ConDUCT enables the transfer of constraint information from one domain to another. In contrast to pure hierarchical transformation, we assume that constraints which are defined for design objects in one domain are transferred to a different domain to other design objects.

We define a constraint with constraint type t as c^t. Constraint type t consists of the tuple

$$t = (typeInTool, originTool).$$

For example, we use

$$currentChipType = (current, chipTool).$$

A constraint transformation rule ϕ that is able to transform from constraint type $c^{t_{source}}$ to constraint types $c^{t_{target1}}, .., c^{t_{targetn}}$ is represented as shown in Equation 3.

$$\phi^{t_{source} \rightarrow t_{target1}, .., t_{targetn}} : c^{t_{source}} \rightarrow \{c^{t_{target1}}, .., c^{t_{targetn}}\} \quad (3)$$

Transformation rules are provided globally in the system and can be reused in every new design project.

The flow of constraint transformations in ConDUCT (see Figure 2) begins when the user, i.e. the designer, starts the transformation interactively via a user interface. The user specifies if

- a source constraint generates target constraints of one specified target constraint type,
- all possible target constraints are generated from given source constraints or
- **all** possible target constraints are generated from all existing constraint sets.

The controller redirects the transformation request to the constraint transformer. This component receives the constraint objects from the ConDUCT constraint model, which was described in Section III. First, the constraint type of a given source constraint is looked up. Each rule with given source constraint type is chosen for transformation. The constraint transformer receives all chosen transformation rules from the rule manager. Finally, when the constraint target type is found in a transformation rule or no further matching transformation rules exist, the rule manager returns all transformation rules that have been found so far. Then, the constraint transformer forwards the constraint objects from ConDUCT and the detected transformation rules. The rule interpreter executes the transformations with the forwarded set of transformation rules and constraint objects.

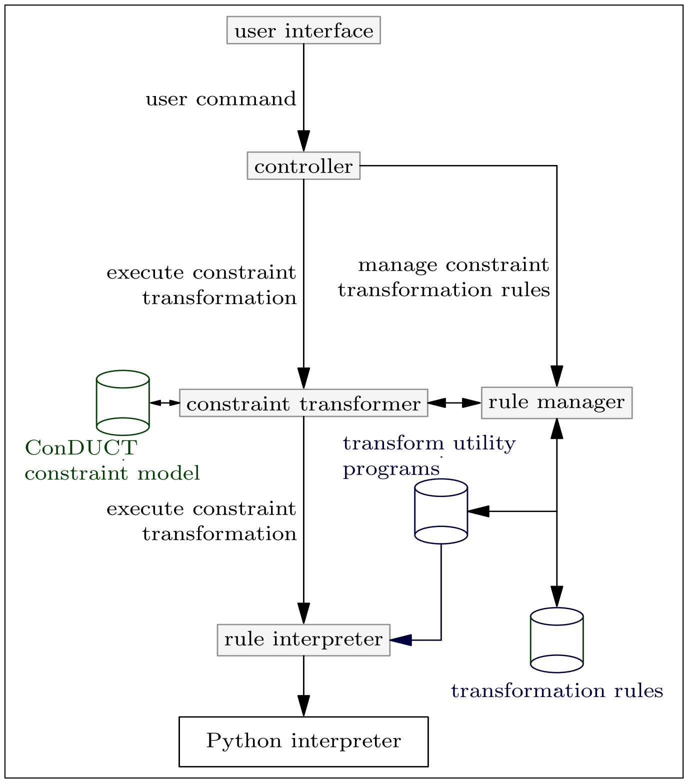

Fig. 2: Transformation flow in ConDUCT

V. Petri Net analysis in Constraint transformation system

Transformation rules provide the possibility to analyze the given sets of constraints in distributed domains. Statements can be derived, e.g. which new constraints can be generated in a given design project and which can not be generated. In this section we show that reachability analysis in petri nets can be used to undertake this task. Furthermore, when constraints can not be generated, it is possible to determine the constraints and transformation rules that are missing to generate specific target constraints. This can be done by using backtracking algorithms. Arsintescu [1], Dhanwada [4] and Krinke [6] use graphs to visualize transformation flows in hierarchical systems. To depict relations of constraint types and transformations appropriately, we decided to use petri nets considering cross-domain transformations. The advantage of our approach is that we have a bijective mapping from the transformation rule set to the corresponding petri net. The relation between constraint types and transformation rules can be extracted from the petri net and vice versa. Simple graphs (used in [1], [4], [6]) need to be modified to store the assignment of transformation rules to graph edges and identify constraint type nodes that are generated by the same transformation. Hence, petri nets can be adapted more easily.

Petri nets are bipartite graphs that are defined as a tuple (P, T, F, W, M_0) [9]:

- P is the set of places
- T stands for the set of transitions

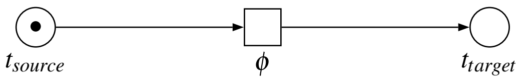

Fig. 3: Constraint transformation visualized as petri net

- F provides the set of flow relations
- W is the weight function that maps a weight to each arc
- M_0 describes the initial marking

We customize the petri net definition to illustrate the constraint transformation rule set and the constraints of the design project:

- P is the set of constraint types
- T stands for the set of transformation rules
- F provides the set of flow relations from source constraint type to respective transformation rules and from transformation rule to target constraint types
- W is the weight function that maps each arc to value 1
- M_0 describes the initial existence of constraints of the corresponding constraint type

We use our set of transformation rules and the set of constraints in an active project to build a petri net. Constraint types are modeled as places P. Note that, e.g., the constraint type *symmetry* in a system-in-package tool is interpreted as a different type than the constraint type *symmetry* in a chip design tool. We make this distinction, because constraints might be verified in different ways and need also different parameters. The constraint transformation rules are represented as transitions T. F contains the information which places are connected to which transitions and vice versa. If there is at least one constraint of a type, the corresponding place is marked with a token. The weight function W assigns the value 1 to all arcs of the flow relation F, because we model the existence of constraints of the respective type and not the respective number of constraint instances. Hence, a place can contain zero or one token. M_0 shows which constraint types are present in the project, before a constraint transformation starts.

Figure 3 illustrates a constraint transformation from constraint type t_{source} to constraint type t_{target} using the transformation transition ϕ. We added the token to the t_{source} place, because one or more instances of constraints with constraint type t_{source} exist.

When a set of constraints is given, we propose to use topological sort [10] to determine in which order the constraints are transformed. The constraint type possessing the lowest number of dependencies to other constraints is processed first. That means we first transform constraints of constraint types that depend on a small number or no other constraint types. After transformation the place and corresponding transitions in the petri net are deleted and we repeat the steps until no places are left.

Petri nets enable an efficient search for parallel paths in transformation processing. If there is no dependency between paths of transformations they can safely be parallelized. This is an advantage that can reduce the execution time of constraint transformation significantly.

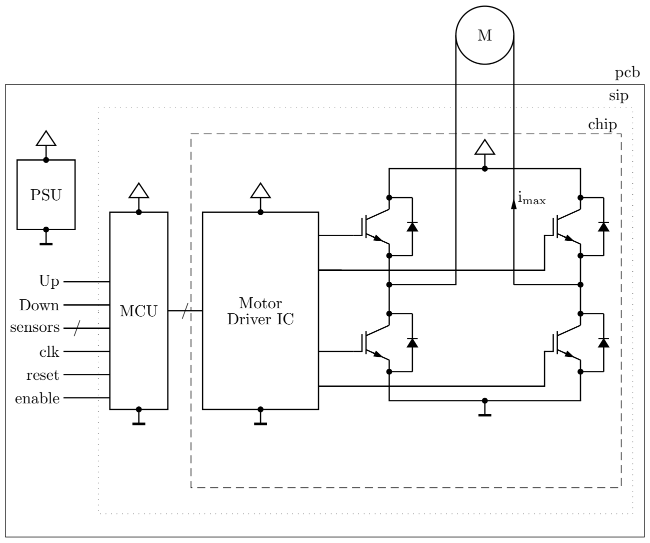

Fig. 4: Window lifter example

VI. Results

Figure 4 illustrates an example application, the window lifter of a car. It consists of a microcontroller unit, a motor driver IC, an H-Bridge circuit and the motor itself. In Figure 4 domain boundaries are shown as dashed (chip domain), dotted (system-in-package domain) and thin line (pcb domain). The motor driver IC and the H-Bridge circuit can be implemented on one die that is designed using a chip design tool. Our system-in-package is composed of the die and the microcontroller unit that processes the incoming signals from sensors and window switches. Additionally to the system-in-package, the PCB contains a power supply unit. The motor is not located on the PCB. The system as a whole is designed using three design tools, one for each domain.

A lot of design tools provide the possibility to manage constraints in their own constraint manager. Unfortunately, there is no support to use constraints that are defined in a chip design tool across the domain boundaries to system-in-package or PCB design. We assume that in the chip design tool a current constraint i_{max} on a conductor path was set. Further, the following set of transformation rules Φ is stored by the rule manager of ConDUCT:

$$\begin{aligned} \phi_1 &: c^{currentChipType} \rightarrow \{c^{currentSipType}\} \\ \phi_2 &: c^{currentSipType} \rightarrow \{c^{widthSipType}\} \\ \phi_3 &: c^{currentSipType} \rightarrow \{c^{impedanceSipType}\} \end{aligned} \quad (4)$$

Transformation rules Φ are declared once and can be reused for multiple transformations. It has to be considered that knowledge about connections and equivalence of design objects in different domains must be obtained from the design tools. One method to solve this mapping problem could be to use the same name of the conductor path in chip and system-in-package domain.

In Section V we have described the generation of petri nets using all transformation rules in the design project. Given the

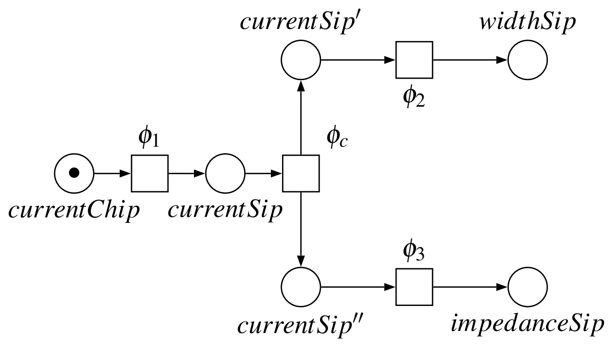

Fig. 5: Example for constraint transformation

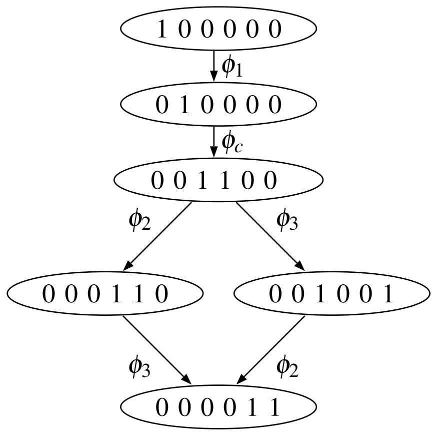

Fig. 6: Reachability graph for the petri net in Figure 5

set of constraint transformation rules Φ (Equations 4), we are able to generate a petri net that visualizes all possible constraint transformations (see Figure 5).

Additionally, one transition ϕ_c is introduced to copy the token and enable both constraint transformations ϕ_2 and ϕ_3 at the same time. We have outlined in Section V that we use the generated petri nets to perform a reachability analysis. With the given current constraint i_{max} we obtain our initial marking for the current constraint type in the chip domain (Figure 5). A reachability graph can be created (Figure 6) with petri net and initial marking of the window lifter example as shown in Figure 6. Every place that contains a token during transformation means that constraints of that type are generated and added to the constraint set of the corresponding domain. We execute our constraint transformations and generate the system-in-package constraints of types *currentSipType*, *widthSipType* and *impedanceSipType*.

VII. Conclusion

Cross-domain constraint transformation offers the generation of new constraints. Hence, designers can propagate constraints to ensure a correct design in all design domains. Issues like data consistency, usage of version control systems in design tools and mapping of constraint members from one domain to another need to be considered in cross-domain constraint methodologies.

As a solution we introduced our tool ConDUCT that is based on a unified constraint model and uses a rule-based transformation method. Furthermore, the analysis of a given transformation rule set and the design constraints was described. We propose to use customized petri nets to obtain information about transformations that can be parallelized or missing constraints that are required for the generation of other design constraints.

Further applications are possible, e.g. the usage of constraints in common design steps like floorplanning, routing or placement. Another application is to contribute to the generation of simulator stimuli and assertions from constraints.

Our cross-domain methodology is a useful instrument to propagate stress profiles, trace design constraints and complete the flow of Mission Profile information down to IC design.

References

[1] B. Arsintescu, "Constraint management and transformation," Ph.D. dissertation, Technical University Delft, 1998.

[2] J. R. Black, "Mass transport of aluminum by momentum exchange with conducting electrons," in *Reliability Physics Symposium, 1967. Sixth Annual*, 1967, pp. 148–159.

[3] ——, "Electromigration failure modes in aluminum metallization for semiconductor devices," *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587–1594, 1969.

[4] N. Dhanwada, A. Doboli, A. Nunez-Aldana, and R. Vemuri, "Hierarchical constraint transformation based on genetic optimization for analog system synthesis," *Integr. VLSI J.*, vol. 39, no. 3, pp. 267–290, Jun. 2006.

[5] G. Jerke, J. Lienig, and J. Freuer, "Constraint-driven design methodology: A path to analog design automation," in *Analog Layout Synthesis*, H. E. Graeb, Ed. Springer US, 2011, pp. 269–297.

[6] A. Krinke, M. Mittag, G. Jerke, and J. Lienig, "Extended constraint management for analog and mixed-signal ic design," in *Circuit Theory and Design (ECCTD), 2013 European Conference on*, 2013, pp. 1–4.

[7] S. Little, D. Walter, C. Myers, R. Thacker, S. Batchu, and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid petri nets," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 617–630, 2011.

[8] P. Maciel, E. Barros, and W. Rosenstiel, "A petri net model for hardware/software codesign," *Design Automation for Embedded Systems*, vol. 4, no. 4, pp. 243–310, 1999.

[9] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[10] D. J. Pearce and P. H. J. Kelly, "A dynamic topological sort algorithm for directed acyclic graphs," *J. Exp. Algorithmics*, vol. 11, Feb. 2007.

[11] P. Rokyta, W. Fengler, and T. Hummel, "Electronic system design automation using high level petri nets," in *Hardware Design and Petri Nets*, A. Yakovlev, L. Gomes, and L. Lavagno, Eds. Springer US, 2000, pp. 193–204.

[12] K. Salimifard and M. Wright, "Petri net-based modelling of workflow systems: An overview," *European Journal of Operational Research*, vol. 134, no. 3, pp. 664 – 676, 2001.

[13] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[14] W. M. van der Aalst, "The application of petri nets to workflow management," *Journal of circuits, systems, and computers*, vol. 8, no. 01, pp. 21–66, 1998.

[15] ZVEI, "Handbook for robustness validation of semiconductor devices in automotive applications," ZVEI - Zentralverband Elektrotechnik- und Elektronikindustrie e.V., Tech. Rep., 2013.

This contribution has been funded as part of the ResCar 2.0 project (project label 01 M 3195) within the research program ICT 2020 by the German Federal Ministry of Education and Research (BMBF).