An Ontology for Constraints in Custom IC Design

Andreas Krinke, Jens Lienig
Dresden University of Technology
Institute of Electromechanical and Electronic Design
Dresden, Germany
krinke@ifte.de, jens@ieee.org

***Abstract*—The design of integrated circuits involves the consideration of a large number of constraints of various types. In addition to the definition of these constraints in a constraint-driven design flow, the declaration of new, yet unknown constraint types might be necessary.**

We define an *ontology for constraints* as a universal approach for the definition of constraint types and their behavior. This formal constraint representation categorizes the behavior of constraint types that are common in custom IC design. It also gives special attention to the parameters they constrain, the sensitivity to other parameters and the set of valid target design objects of these types.

The formalization of this domain greatly eases the introduction of new constraint types by reusing both, formalization and implementation of existing ones. Furthermore, this ontology enables for the first time consistent constraint type and constraint data transfer between different applications.

I. Introduction

The design of custom integrated circuits (ICs) often requires the integration of both analog and digital modules. With regard to used die area, the development of analog cells generally takes much more time than establishing digital cells. The self-evident explanation for this difference is the comprehensive automation of digital design in contrast to the isolated approaches of the analog counterpart. This difference has various reasons, e.g., the larger parameter space of analog circuits and their greater sensitivity to variations caused by the manufacturing process. As a consequence, various constraints have to be taken into account throughout the design process. Examples are matching of devices, noise immunity of signals and limitation of voltage drops (IR-drop).

In general, constraints are requirements related to design parameters that need to be satisfied by a finished design. Currently, most of these constraints exist as so-called "expert knowledge" and need to be considered and verified manually. All constraints that are unknown to design tools cannot be incorporated into algorithms for automation in the first place. The machine-readable representation of constraints in a so-called constraint-driven design flow is regarded as an important step in the direction of analog design automation [1].

Constraints can be categorized in constraint types based on the corresponding design parameters and the type of their relation. Manual and automatic definition of constraints and their tool-independent handling require a classification of constraint types combined with the description of their properties. Due to the infinite number of possible constraint types, the easy extension of the classification is of particular interest.

A. Current Practice

The EDA industry has already incorporated features of constraint-driven flows into various design tools. As an example, the tools provided by the Cadence custom IC design environment support a predefined set of constraint types and the manual definition of custom variants. The assignment of constraints happens either manually or automatically. In the latter case, common structures, such as current mirrors and differential pairs, are recognized in an automated manner. Then, appropriate constraints, e.g., symmetry and placement constraints, are assigned to these structures. Each tool of the design flow considers a specific subset of all available constraint types in order to automate particular design steps. Newly created constraint types need custom code to be taken into account by these tools.

Despite these achievements, a universal approach for the formulation of constraint types and their behavior in the design flow is still missing. Especially the interoperability of the utilized tools depends on such a formulation for constraint data transfer and manipulation.

B. Our Contribution

We present an ontology to provide a universal formulation of constraint types for the first time. As a result, questions about which design parameters have an influence on a specific constraint can be answered. Additionally, the proper set of design tools to manipulate a constrained parameter is inferable.

The foundation for easy addition of more parameters and constraint types is formed by providing basic sets of design parameters, design elements to which these parameters can be attached to, and constraint propagation types. New types are easily added in a formal way by reusing this information.

This paper is organized as follows. In Section II, we present terms and definitions used in the context of ontologies and constraints in analog IC design. Section III contains the description of the proposed ontology. In Section IV, we evaluate use cases for this ontology. Section V concludes this paper and gives an outlook for future research.

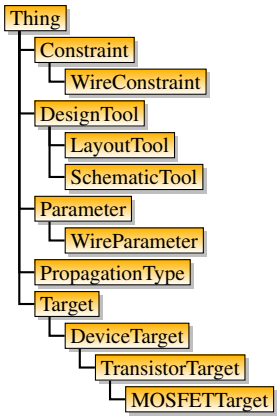

Figure 1. The class hierarchy contains definitions for design parameters, target elements, design tools and constraint types as major building blocks of the ontology.

II. Terms and Definitions

Ontologies are used to formalize knowledge about a specific domain by capturing concepts and their relations to each other. One goal is to make software *understand* these concepts. There are several ontology languages with different features. For our research, we use the Web Ontology Language (OWL) which was specified by the World Wide Web Consortium (W3C). An OWL ontology is formulated with individuals (instances), properties and classes. Individuals are objects in the domain described by the ontology. Two individuals can be linked together by properties such as *isInfluencedBy* (see Section III-A). Classes correspond to concepts and are equivalent to sets of individuals [2]. The domain of analog IC design contains a great number of parameter types. Each of these parameter types is applicable to a certain set of design elements, which we will call *target elements* of that type. Parameters can be restricted to a range of values by constraints.

III. Constraint Ontology

We have prototyped the ontology using Protégé, a "free, open source ontology editor and knowledge-base framework" [3]. Figure 1 shows the class hierarchy of the ontology.

A. Design Parameters

The first step is the definition of design parameters and target elements together with applicable object properties. Based on the root class `Thing`, we create the subclasses `Parameter` and `Target`. If one parameter influences another one, both are connected by the transitive properties *isInfluencedBy* and its inverse, *hasInfluenceOn* (see Figure 2). The property *hasTarget* assigns the appropriate target element to each parameter. Based on these properties, the class `Parameter` is defined as a `Thing` that has exactly one target and is only influenced by other parameters. Using this concept, individuals can automatically be classified as parameter. The classification also verifies the correct definition of newly created instances. As examples we add several parameters of layout net segments (wires) to the ontology (see Section IV).

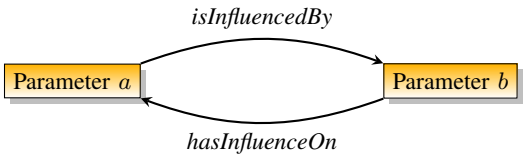

Figure 2. The inverse properties *isInfluencedBy* and *hasInfluenceOn* connect different parameters.

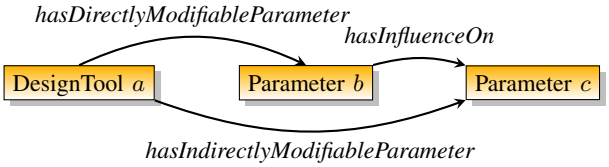

Figure 3. If a design tool can directly modify one parameter that has influence on a second one, then this second parameter is indirectly modifiable by this tool.

B. Design Tool Definition

Different design tools are described by the class `DesignTool`. Each individual tool can change certain design parameters which are associated by using the property *hasDirectlyModifiableParameter*. Again, an inverse property *isDirectlyModifiableBy* expresses which tool can manipulate a specific parameter. Besides these properties for direct modification, there are two equivalents which describe indirect adjustment of parameters. We use a property chain axiom to define the inference of those parameters that are indirectly modifiable by a design tool (see Figure 3). The classification of an individual as `DesignTool` is due to its connection to directly modifiable parameters.

C. Constraint Definition

The third main part of the ontology is the definition of the class for constraint types. Each `Constraint` is associated with the design parameter it constrains. Therefore, we declare the property *hasConstraintParameter* and its inverse *isConstrainedBy*. The propagation type is likewise linked by *hasPropagationType*. A constraint's sensitivity to design parameters is expressed by the property *isSensitiveTo*. Again, a property chain axiom declares when to infer this property: If a constrained parameter is influenced by a second one, then the constraint is sensitive to this second parameter (cf. Figure 3). Finally, we extend the existing property *hasTarget* to get automatically assigned to a constraint according to the target element of the constrained parameter.

The three properties that link parameter, propagation type and target to a constraint define the corresponding class. Subsequently, they are used for classification.

IV. Discussions

An important tool for the work with ontologies is the reasoner. For this, we use FaCT++ [4], which is integrated into Protégé. One task of the reasoner is to check the consistency of an ontology, i.e., if it is possible for all classes to have

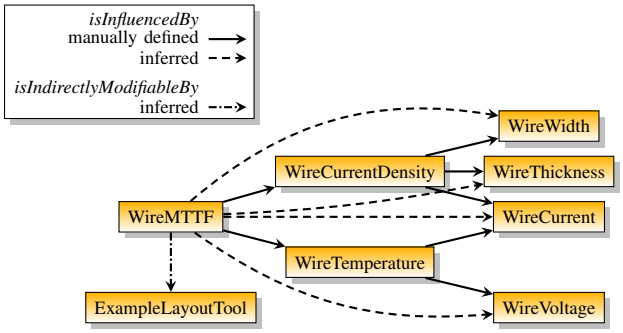

Figure 4. Selection of properties of a net segment's MTTF parameter showing other parameters that have an influence. Additionally, it is inferred that this parameter is indirectly modifiable by the exemplary layout tool.

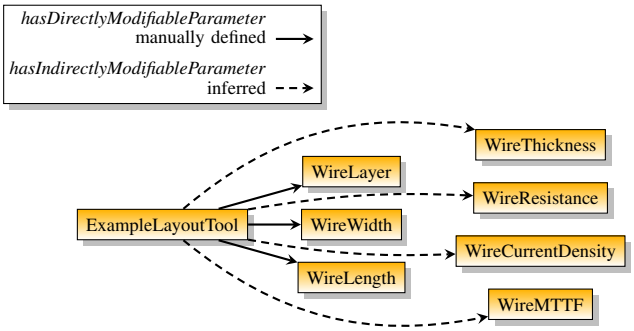

Figure 5. Selection of wire parameters that the reasoner FaCT++ correctly infers as indirectly modifiable by the exemplary layout tool (cf. Figure 3).

instances. Secondly, it computes the inferred class hierarchy containing information about which classes are subclasses of other classes [2]. This step, also known as classification, has the side effect to infer the properties of all individuals. The benefit of this feature is discussed in this section.

After the definition of the ontology, we create exemplary individuals. For the class `Parameter`, various parameters of net segments are defined. Using the properties *isInfluencedBy* and *hasTarget*, we manually connect these instances with other parameters and targets. As an example, the mean time to failure (MTTF) of a segment due to electromigration depends on current density and temperature [5]. In addition, we describe (a) a layout tool that is capable of directly modifying the width, length and layer of a net segment and (b) a constraint on the MTTF of such a segment.

Then the reasoner infers properties of these individuals. Figure 4 shows the result for the MTTF parameter. All parameters with influence on the MTTF are listed together with the design tools that can directly or indirectly modify the MTTF. In Figure 5, the inferred properties of the exemplary layout tool are shown, representing indirectly modifiable parameters. Finally, Figure 6 shows the properties of a sample constraint on the MTTF parameter. The target and sensitivities are properly inferred from the constrained parameter MTTF.

This illustrates the power of our approach: For a set of constraints it is possible to derive their dependency on design parameters combined with the information which design tools

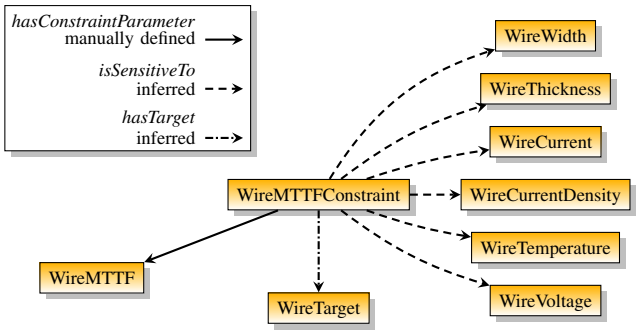

Figure 6. Selection of inferred properties of a constraint on a net segment's MTTF. Its target element and sensitivities are inferred from the constrained parameter.

can influence their satisfaction. Applied to this example it becomes obvious that a wire's MTTF depends among other things on its width, thickness and current. Therefore, the value of the constraint can be modified using the exemplary layout tool. The deduction of more complex relations uses the exact same mechanisms.

The addition of new constraint types, parameters or design tools only requires the manual definition of basic properties for the integration into the ontology (cf. Figure 4). Inference ensures the connection to all other relevant elements of the ontology, i.e., other parameters or design tools.

V. Conclusion and Outlook

We have shown that an ontology for constraints in analog circuit design has the potential to properly formalize this domain. The formalization is the foundation for an easy extension of constraint management systems with new, yet unknown, constraint types. It is based on the availability of a hierarchy of reusable building blocks, e.g., parameter and propagation types. Additionally, our approach can help answering important questions that occur in a constraint-driven design flow. These include, for example, how to manipulate a constrained parameter and which design tool to use.

Constraint data is design data. Thus, every design tool possibly manipulates constraint data to achieve its purpose. Standardization of constraint data is therefore a crucial requirement for tool interoperability. A well thought-out ontology can meet both demands, formalization and standardization. The prototyped ontology is only a first step in this direction. Future research will aim for full-fledged formalization of this domain.

References

[1] 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.

[2] M. Horridge *et al.* (2009, 3) A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition 1.2, The University Of Manchester.

[3] Protégé. [Online]. Available: http://protege.stanford.edu

[4] FaCT++. [Online]. Available: http://owl.man.ac.uk/factplusplus

[5] J. R. Black, "Electromigration—A Brief Survey and Some Recent Results," *IEEE Trans. on Electron Devices*, vol. 16, no. 4, pp. 338–347, 4 1969.