Architecture Refinements by Code Refactoring of Behavioral VHDL-AMS Models

Kaiping Zeng, Sorin A. Huss
Integrated Circuits and Systems Laboratory, Department of Computer Science
Darmstadt University of Technology
Hochschulstrasse 10, D-64289 Darmstadt, Tel: (+49 6151) 16 6694, Fax: (+49 6151) 16 4810
Email: zeng@vlsi.informatik.tu-darmstadt.de

Abstract—This paper presents an automatic structure synthesis technique by partitioning the system representation, which consists of interconnected analog and mixed-signal behavioral models coded in VHDL-AMS. The proposed model code refining methodology restructures, refines, and simplifies behavioral models by means of code transformations of given abstract models. The fundamental approach for these transforms is code refactoring—an approach taken from software engineering and adjusted to the requirements of analog circuit synthesis. Through code refactoring one improves the comprehensibility, expandability, and reusability of a behavioral block model and restructures the model such that subsequent circuit synthesis steps may produce adequate structural representations of the intended behavior. Application examples demonstrate the feasibility of this novel approach to architecture synthesis.

I. INTRODUCTION

The advance of new process technologies has enabled designers to integrate more complex and more functionality for IC devices, as evidenced by system-on-chip (SoC) designs which incorporate whole systems on a single chip, containing processors, fundamental analog, radio frequency (RF), and mixed analog/digital signal components of communications [1]. While electronic design automation (EDA) tools are well developed and widely available for designs of digital circuits, designs of analog and mixed-signal circuits remain largely a heuristic and a labor-intensive task due to the short of mature analog EDA tools, especially at levels higher than the opamp level, resulting in unacceptably long times for getting SoC products to market [2], [3], [4].

In this paper we focus on the high-level synthesis of analog models and introduce a methodology for automatic generation of structure-level analog designs from behavioral-level ones which are written in VHDL-AMS. The application of this methodology to the design of a fourfold integrated clocked current controller is illustrated.

II. METHODOLOGY OF CODE REFACTORIZATION

The concept of code refactoring was introduced by Kent Beck in 1997 in the context of software engineering [5]. Code refactoring is a behavior-preserving source-to-source transformation. It is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Although refactoring techniques were firstly applied to software design, they can be extended to the high-level synthesis of analog models due to obvious similarities between the software engineering and the hierarchial top-down design flow. In this context, refactoring of analog model means to modify a model description synthesis-oriented without changing the envisaged behavior of the system model, so that an implementable description for the next-level synthesis is reached by iterative application of a sequence of refactoring methods. That is, part of the model is rewritten in order to improve certain qualities, such as readability, extensibility, reusability, or synthesability, making the system easier to change, less error-prone and most important more synthesis-oriented.

Based on our practical experience related to code refactoring, the classification of these techniques is grouped into five distinct catalogs [6]:

- **dataflow-oriented**, dedicated to simplification and optimization of dataflow within a component description,
- **controflow-oriented**, aimed to an improvement of code quality for models featuring a large amount of controflow, which is quite common in complex sequential processes coded in VHDL-AMS,
- **structure-oriented**, aimed to a reorganization of an architectural description, which exploits structural views of an entity,
- **relocating**, providing means for shifting parts of the model description,
- **concretizing**, in order to yield refined model descriptions.

The code refactoring based design flow is illustrated in the Fig. 1. In the following, some examples of code refactorings are detailed.

A. Interface Transformation

Interface transformation is a refactoring method which belongs to the dataflow-oriented catalog. It tries to transform signal flow models to conservative ones and bridges the hierarchical top-down design-flow gap between mathematical functional models and physical behavioral models [7] (see Fig. 2).

The signal flow model simply maps input quantities to output quantities according to the detailed conversion algorithms, which are described in generally by ODAEs (Ordinary Differential Algebraic Equations). It has only flow or potential...
Dataflow-Oriented Methods

Original Specification in VHDL-AMS

Refactored Specification in VHDL-AMS

Controlflow-Oriented Methods

Dislocating Methods

Concretizing Methods

Structure-Oriented Methods

Generic Module Library

Design Rules, etc.

Fig. 1. Design flow supporting high-level analog synthesis by means of code refactoring.

quantities associated with a node. A signal flow diagram is used to describe the signal flow model (see Fig. 3).

The outputs of conservative model consists of a voltage and current waveform, respectively, according to the generic definition of behavioral models. This is reflected in both the conversion algorithm and in the required \(v/i\) characteristic at the output pin of the block. Conservation laws have to be satisfied on the pins connecting different blocks. The potentials

[Diagram showing the pin compatibility of functional and behavioral models.]

B. ODAE Partition

ODAE partition is a refactoring method in the structure-oriented catalog. It brings the model from algorithmic functional level closer to the circuit level. It is thus a special form of high-level analog synthesis called structure synthesis, which transforms a behavioral specifications of a design model to a primitive structural representation. Structure synthesis is one of the most difficult tasks in the top-down hierarchical design flow for analog application, due to the inherent complexity and the diversity of structures of analog circuits. There exist two considerations, which have to be observed for achieving this goal. First of all, it is important to provide the flexibility in the choice of the desired degree of accuracy depending on the application. In addition to that, this techniques should assist to create an analog library in the next hierarchical level due to the fact that analog primitives in general have more than one function depending on the used technology.

One of the abstract implementation of the ODAEs model is described by the following equation [7]

\[
\begin{align*}
\dot{x} &= F(t, x, u),  \\
y &= G(t, x, u),
\end{align*}
\]

where \(F, G\) are vectors of expressions which are assumed to be sufficiently smooth, is illustrated by the functional block diagram in Fig. 5. Therefore, the main task of the structure

[Diagram showing the abstract implementation of the behavioral model described by equation (1).]

synthesis of the corresponding behavioral model is how to implement the functions described by \(F\) and \(G\), i.e., how to partition \(F\) and \(G\) into small functional analog blocks.
In order to simplify the task of the partition of the functions \( F \) and \( G \) in (1), functions \( F \) and \( G \) are supposed to be simple enough, satisfying the following conditions:

- there are no high-order differential terms,
- there are no transcendental functions such as \( \sin x, e^x \), etc.,
- there are only five basic operations: add, substract, multiply, differentiate, and integrate.

With the above assumptions, the ODAEs model of analog circuits can be partitioned and be electrically implemented by so-called analog primitives, such as integrators, summers, scalers, and multipliers (see Fig. 6).

\[
\begin{align*}
&u_1 \\
&u_2 \\
&u_3
\end{align*}
\]

Fig. 6. Definitions of analog functional primitives and their symbols: (a) Scaler: \( f = K \cdot u_2 \); (b) Summer: \( f = \sum_{i=1}^{n} u_i \); (c) Multiplier: \( f = u_1 \cdot u_2 \); (d) Integrator: \( f = \int u(t) dt \).

III. IMPLEMENTATION OF THE METHODOLOGY

Since code refactoring operations are frequent during the overall design life-cycle, automatization is an indispensable step towards productivity gains in analog design. The methodology we discussed above is implemented as a tool called RAMS (Refactoring Analog and Mixed-signal System), which is done with Java as the programming language and Eclipse [8] as the integrated developing environment [9].

The roadmap of implementation of our methodology is summarized in Fig. 7.

Fig. 7. Framework of implementation with ANTLR [10].

The process starts with building a parser that constructs a tree intermediate representation of the input VHDL-AMS source.

A parse tree is a representation of the information obtained by parsing a particular input. It contains a node for each token identified during the parse. ANTLR [10] (Another Tool for Language Recognition), developed by Terence Parr, helps one to build this intermediate form trees—abstract syntax tree (AST) by augmenting a grammar with tree operators, rewrite rules, and actions. An abstract syntax tree consists of a root node and subnodes. Each subnode may, in turn have its own subnodes. Each node in the abstract syntax tree has a type. Examples of node type include operators, such as assignment and plus. A node may also have a leaf type, which would be a constant or an identifier. The advantage of creating a syntax tree is that once the whole tree is available, one can traverse the tree as many times as necessary to collect various kinds of data or examine the entire tree.

With the help of ANTLR, a VHDL-AMS parser is generated. Through the syntactically and semantically analysis of the input source code, the corresponding AST is produced [9].

Although ANTLR tree parsers with their basic navigation methods can traverse the AST and impose a child-sibling like structure to any tree data-structure, it is in general difficult to make more sophisticated transformation on the AST. Therefore, AST is converted to a work format, the so-called object tree in Java. On this object tree, every node is an AST node, which is equipped with possible available methods.

Now the algorithms of our proposed methodology are applied on the object tree and synthesis-oriented structure-level designs may be generated.

IV. CASE STUDY: A FOURFOLD INTEGRATED CLOCKED CURRENT CONTROLLER

The high level synthesis of a fourfold integrated clocked current controller is present as an example using our methodology. This controller will transfer through the pulse width modulated (PWM) CMOS input signal as the set value in the relative control current. An integrated clock oscillator controls the output port and should be adjusted by the external capacitor. The transient response of the set value and the slew rate of the control current should be adjusted through an external capacitor. The operation of all four current channels is activated through a central enable input. The operation of the single current control channels should be activated by their input control PWM signals. This part is implemented by the controller, and its main function is to compare the real value and the set value to generate a control signal (see Fig. 8).

The design starts with a signal flow model of a PI controller. The VHDL-AMS specification of it is listed in Fig. 9.

In order to transform the design to a real circuit, the first necessary step of refinement is through the refactoring—interface transformation. Fig. 10 gives the refined model, i.e., the conservative model.

The next step of refinement is ODAE partition, which is demonstrated in Fig. 11. This resulting analog design is now partitioned into subtractor, integrators, summers, scalers and get ready to be passed to the next-level synthesis.

V. CONCLUSION

The prototype of implementation of automatic code refactoring for analog model codes in VHDL-AMS has been setup. This contribution is our effort to develop an analog and mixed-signal circuits synthesis supporting system. It demonstrated the feasibility to automatically synthesize an analog design coded...
in VHDL at behavioral level into structure level, which separates the creative activities of a designer from the unavoidable routine work.

ACKNOWLEDGMENT

This work was supported in part by German Federal Ministry of Education and Research (BMBF) edacentrum under Grant 01M3070E.

REFERENCES


