Ensuring Consistency during Front-end Design using an Object-oriented Interfacing Tool called NETLISP

Michael Goffioul, Gerd Vandersteen†, Joris Van Driessche,
Bjorn Debaillie, and Boris Come
IMEC vzw. - Wireless Research, Kapeldreef 75, B-3001 Leuven, Belgium
† also Vrije Universiteit Brussel, Dept. ELEC, Pleinlaan 2, B-1050 Brussels, Belgium
Michael.Goffioul@imec.be, Gerd.Vandersteen@imec.be,
Joris.VanDriessche@imec.be, Bjorn.Debaillie@imec.be, Boris.Come@imec.be

ABSTRACT

The design of complex analog front-ends demands the exploration of a huge design space at different levels of abstraction and using a multitude of simulators. One of the main issues is to guarantee the consistency of the models and the model parameters between the different abstraction levels.

A methodology and a tool, called NETLISP, has been developed for the purpose of guaranteeing the consistency between different levels of abstraction. The tool was successfully applied to the design of a multi-mode multi-band analog front-end, showing its applicability on real large-scale designs.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Object-oriented design methods*

General Terms

Design

Keywords

Design flow, front-end, object-oriented design, model consistency

1. INTRODUCTION

The design of multi-mode multi-band analog front-ends demands the exploration of a huge design space. This complex problem is conquered using design flows which study the system at various levels of abstraction.

The use of different abstraction levels implies the use of different representations and different simulation techniques. This will be illustrated in the design example in section 3. It includes the following levels of abstraction.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

An exploration phase determines implementation losses of the different non-idealities of a behavioral front-end model using dataflow simulations. All non-idealities are centralized in this model (i.e. one noise source, one transfer function...).

A cascade analysis phase distributes the global specifications for each non-ideality over the building blocks of a specified front-end architecture.

A design phase designs the circuits as well as compensation techniques for front-end non-idealities using respectively a Verilog-A simulation model and a dataflow model.

A verification phase verifies the performance of the designed front-end for each abstraction level.

A weak point in a design flow with various abstraction levels is the consistency of both the models and the parameters in between these levels. Several design errors can be expected:

- **copy-and-paste errors** by changing the parameters manually from one level to another;
- **out-dated designs** since parameter changes are not propagated automatically;
- **oversimplification errors** if extracted (bottom-up) models from a lower abstraction levels can't be plugged into the higher level automatically.

This illustrates the need for automating the transition between abstraction levels to ensure consistency between the different levels of abstraction. This paper presents a methodology, supported by a tool called NETLISP, which guarantees the consistency between the different levels of abstraction. First the tool and its general concepts will be briefly described, then the application of the tool on a real-life example will be illustrated on the design of a multi-mode multi-band front-end RFIC.

2. TOOL DESCRIPTION

The primary goal of the NETLISP framework – which stands for NETLISt Processing – is to describe and manipulate netlists at various abstraction levels during a complete design flow. It has been designed to be the glue between the different simulation tools that are commonly used through

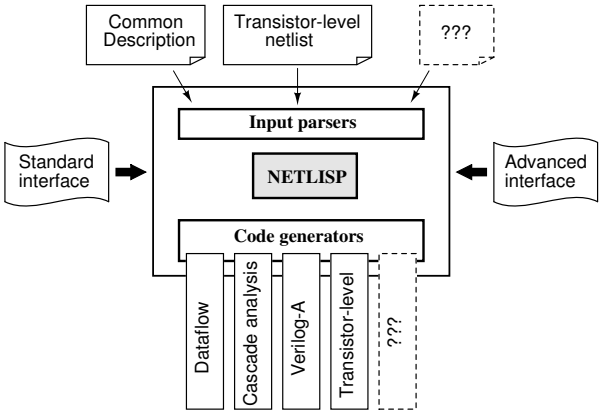

Figure 1: NETLISPs modular architecture

a design flow, but it is not *per se* another simulation environment. In addition to this, the framework also provides a powerful development platform for more advanced operations, like automatic bottom-up model extraction and shared test-bench definitions. Finally, NETLISP includes an intuitive user-interface based on netlists and parsers that allow a designer to start with the tool without much effort.

2.1 NETLISP architecture

The global NETLISP structure is presented in Figure 1. It consists of a programmable core that defines the data structure used to represent systems, and a set of input and output modules (parsers and code generators). As stated above, NETLISP is intended to be a glue between commonly used simulation tools. By generating simulation files for different tools from a single shared data structure, model consistency can be guaranteed at all abstraction levels of the design flow. Moreover, by allowing back-propagation of parameter changes from one level to the NETLISP core, the model consistency and updating can be guaranteed automatically at all iterations of the whole design process. This drastically reduces potential errors as well as the overall design time.

The standard interface consists of netlist parsers that allow the designer to describe a system intuitively, even at system-level. This netlist-based interface reveals especially useful for instance during architecture exploration phase, as various architectures can be analyzed quickly through the use of some common test-benches and a varying netlist. By providing also input modules for common netlist format like SPECTRE, NETLISP also promotes efficient design re-use. In addition, the framework has been embedded into a MATLAB module; on one hand, this provides partial access to the manipulation capabilities of the tool using an environment and language that many designers already use (for instance, this allows to re-use existing MATLAB commercial toolboxes); on the other hand, this provides access to all pre- and post-processing capabilities of MATLAB.

The advanced interface is tightly bound to the development platform used to build the framework. NETLISP is developed as an embedded language on top of LISP [2] using the ECL implementation [3]. While not being a widely used development language, LISP – which stands for LISt Processing – is particularly well suited to the intrinsic nature of netlist representation, which involves mainly lists of components, nodes... Using the rich set of primitives available in Common Lisp, complex operations can be performed with only a few lines of code. Moreover, as code can be interpreted or compiled in most modern LISP implementations, LISP provides a rapid prototyping platform while still having the possibility to distribute efficient compiled code. Being built on top of LISP, the NETLISP framework automatically provides an advanced programmable and extensible platform that can be used to implement more advanced operations or extends NETLISP's capabilities.

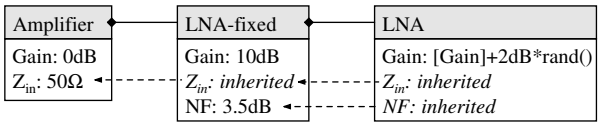

Figure 2: Prototype-based inheritance. `Amplifier` is a prototype for `LNA-fixed`, which itself is a prototype for `LNA`.

2.2 Object description

NETLISP has been developed using an object-oriented (OO) paradigm. Unlike the classical class-based inheritance scheme (Smalltalk, C++), the framework uses a prototype-based inheritance scheme (Self [6]) where any block can serve as prototype for another one [1,4,5]. The derived block then inherits all its parent's properties through delegation (based on a *message passing mechanism*) and is able to overload some of them or add new ones, as illustrated on Figure 2. This inheritance scheme provides the required flexibility for netlist description and manipulation; for instance, block specialization, as in the amplifier example, is a basic single step operation and does not require writing a new class, like in the class-based inheritance approach. Moreover, this OO approach is easily implemented in the targeted LISP development platform.

3. DESIGN EXAMPLE

The NETLISP tool was used for the design of a multi-mode multi-band transceiver front-end (FE) and corresponding compensation techniques. NETLISP ensures the consistency between the different steps of the methodology, introduced in section 1, by using a common description, common test-benches as well as automatic back-propagation, as illustrated in Figure 3.

3.1 Exploration phase

The exploration phase derives specifications for the different considered FE non-idealities. It starts with the generation of a generic behavioral model (Figure 4) from the NETLISP description. This model is both system and architecture independent. This means that it can be applied for different front-end architectures (super-heterodyne, direct-conversion...) and that it can be used with different link models (WLAN, UMTS...). Plugged into the link model, end-to-end system simulations are performed to determine the implementation losses corresponding to a target Bit Error Rate (BER). The resulting implementation loss curves allow the system designer to derive specifications for the different FE impairments based on a budget for the complete

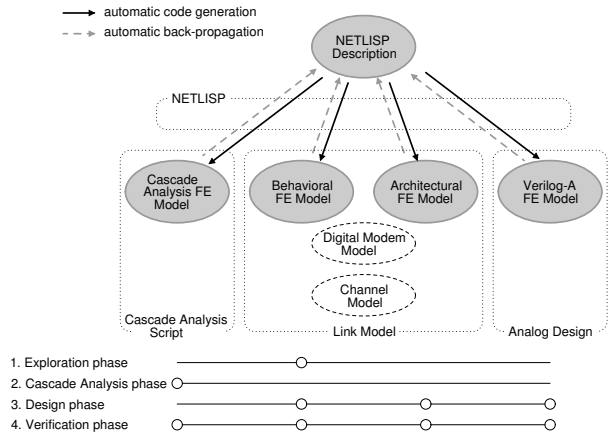

Figure 3: Design methodology using NETLISP.

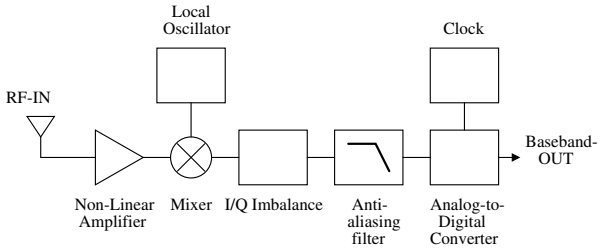

Figure 4: Block diagram of the generic receiver FE behavioral model.

link. These specifications are used as an input for the cascade analysis and the design of compensation techniques.

3.2 Cascade analysis phase

The cascade analysis distributes the specifications for the different non-idealities derived in the exploration phase over the different FE blocks. It starts from an initial FE architecture (Figure 5) described in the NETLISP framework using an in-house developed library of relevant high-level FE blocks. From this description the corresponding MATLAB models are generated. These models are equation-based and allow fast budgeting of the impairments over the different blocks in the transmitter or receiver. The models are applied in a generic MATLAB cascade analysis script. This script together with the NETLISP framework allow the system designer to easily evaluate different architectures and Automatic Gain Control (AGC) or power control algorithms. This in contrast to a cascade analysis in classical spreadsheet-format. In case of a receiver FE, the script evaluates the Signal-to-Noise-and-Distortion Ratio (SNDR) for a required range of receiver powers to ensure the purity of the signal at the input of the digital modem (Figure 6). The SNDR is the accumulation of the thermal noise of the circuitry (SNR), the distortion caused by the desired signal (SiDR) and the distortion caused by interferers in the adjacent and alternate adjacent channels (SDR). When the specification is reached, the NETLISP description is automatically updated with the detailed building block specifications and the AGC-settings.

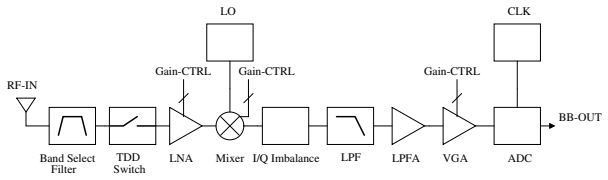

Figure 5: One instantiation of an architectural model.

3.3 Design phase

This phase includes both the design of the analog circuits and the development of algorithms for compensation of the FE non-idealities.

Based on the detailed building bock specifications derived in the cascade analysis, the actual analog design can start and this with the support of Verilog-A models. These models are generated from the NETLISP description and can be simulated in analog design environments e.g. cadence toolsuite. They support the analog design in various ways. First, the Verilog-A model clearly defines the specifications that are requested from the analog blocks. Hence, they can be seen as runnable specifications. Second, the Verilog-A models allow the circuit designer to have high-level blocks of the circuits surrounding his design. In this way he can simulate the impact of the other circuits in his design. Examples are the input and output impedances which load the circuit under design. Through NETLISP the common description is kept up-to-date.

The design of techniques for estimation and compensation is assisted by two dataflow MATLAB FE models: a simple behavioral model and a more complex architectural model, both generated from the NETLISP description. The first model is an update of the model used in the exploration phase (Figure 4). The model parameters are refined with the global receiver/transmitter behavior. This simplified model allows fast simulations. The second model is an architectural dataflow model (Figure 5), modeling the actual implementation of the FE more accurately. This model is system and architecture dependent. Which model is used depends on what is required by the designer. For example, when interferers are relevant, the architectural model should be used since it is the distribution of the gain and the filtering that ensures that no FE block saturates. Both models can be plugged into a link model, enabling a mixed analog/digital co-simulation and hence the design of estimation and compensation techniques. Again, an up-to-date common description is guaranteed by NETLISP.

3.4 Verification phase

In this phase a verification of the designed FE is done for each abstraction level. As an example, the evaluation of the designed FE together with compensation techniques in end-to-end system simulations is presented in Figure 7. It shows the BER as a function of the received power of the desired signal.

In a typical design scenario, there are different iterations between step 2 to 4: after a careful analog design of a subcircuit, the designer may notice that he cannot reach the demanded performance specification. He extracts the obtained specifications from his design and updates the NETLISP de-

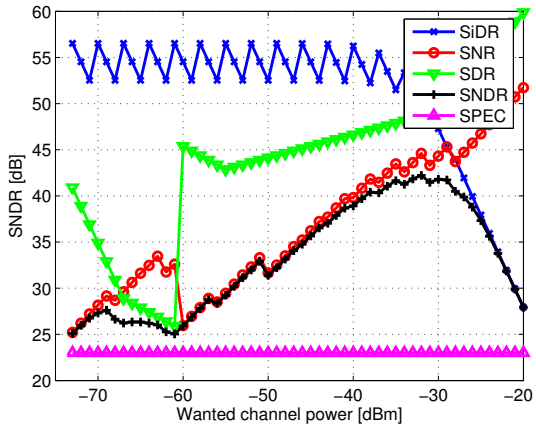

Figure 6: Performance metrics to be evaluated in the cascade analysis for the receiver FE.

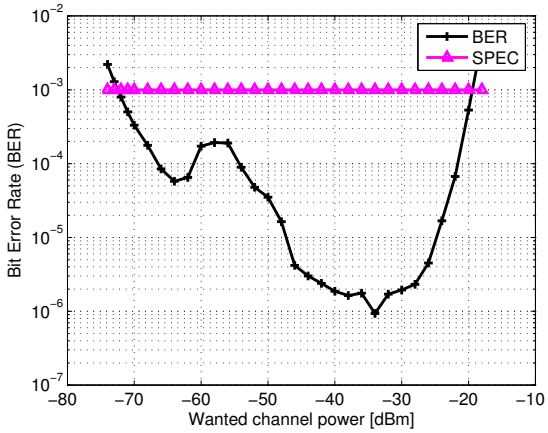

Figure 7: Evaluation of the designed receiver FE w.r.t. the BER.

scription. The new architectural model is generated from the updated description and plugged-in into the link model. If the impact on the BER is limited and the required BER is still reached one can stop here. If the required BER is not reached anymore, a new partitioning of the impairments should be done using the cascade analysis. The description is updated afterwards. These iterations continue until all blocks are within the required specifications.

In conventional design flows, precious time is lost during such iterations: discrepancy between the different models results into considerable debug-time; FE blocks do not work properly in the integration phase due to impedance mismatches, common mode levels... leading to the ad-hoc design of buffers, level-shifters... In contrast, the presented NETLISP framework and methodology enables the design-team to keep track of the system performance during every step in the design flow, reducing the design time. It is applied for the design and measurement of a multi-mode multi-band transceiver FE (Figure 8) and corresponding compensation techniques.

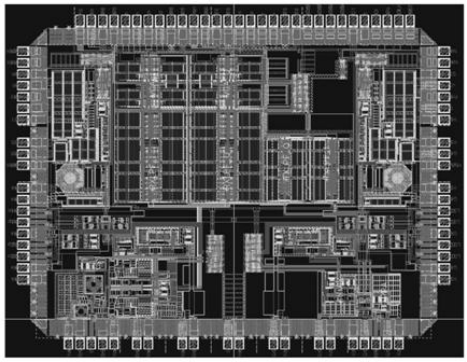

Figure 8: Overview of the tape-out layout.

4. CONCLUSIONS

This paper presents a tool-based front-end design methodology that helps to maintain the consistency of all models and model parameters during the complete design-flow. This methodology has been successfully applied to a real multi-mode front-end design at system-level. Current development will extend the methodology towards the circuit-level .

The tool – called NETLISP – behind the methodology is implemented as an embedded language built on top of LISP. It provides a basic netlist-based interface to describe a system together with a graphical interface (MATLAB) to manipulate it. It also provides an advanced powerful interface, where the complete LISP language is available. The implementation uses prototype-based object-oriented techniques to store and manipulate the parameters and the connectivity of a system. This approach allows to build a flexible object-oriented framework for netlist description and manipulation, at both system-level and circuit-level, helping designers to maintain consistency during the various steps of a design flow.

5. ACKNOWLEDGMENTS

This works was carried out in the context of IMEC's multimode multimedia program, which is partly sponsored by Samsung inc.

6. REFERENCES

[1] C. Dony, J. Malenfant, and P. Cointe. Prototype-based languages: From a new taxonomy to constructive proposals and their validation. In *Proc. OOPSLA'92 Conf.* ACM Press, 1992.

[2] P. Graham. *On Lisp: Advanced Techniques for Common Lisp.* Prentice Hall, 1993.

[3] http://ecls.sourceforge.net/.

[4] H. Lieberman. Using prototypical objects to implement shared behaviour in an object-oriented system. In *Proc. OOPSLA'86 Conf.*, pages 214–223. ACM Press, 1986.

[5] D. Ungar, C. Chambers, B.-W. Chang, and U. Hlzle. Organising programs without classes. *Lisp and Symbolic Computation*, 4(3), 1991.

[6] D. Ungar and R. B. Smith. Self: The power of simplicity. In *Proc. OOPSLA'87 Conf.*, pages 227–242. ACM Press, 1987.