# Achieving Language Independence with Paragon

Pinki Mallick, Matt Francis, Vemulapally Chandrasekhar, Anthony Austin, H. Alan Mantooth
University of Arkansas, 3217 Bell Engineering Center, Fayetteville, AR 72701

**Abstract**

This paper describes Paragon[†], a novel modeling tool that promotes the creation of HDL-based models at a more abstract, language-independent level. This modeling environment encapsulates the semantic elements necessary to create models at various hierarchical levels from mixed-signal and mixed-technology behavioral models down to semiconductor device models. Paragon is designed to support the generation of multiple hardware description languages including VHDL-AMS, MAST, and Verilog-A(MS) as well as C and C++-based codes. This paper focuses on the facets of Paragon that enable language independence.

## 1. Introduction

Whether the modeling task at hand for an engineer is of a semiconductor device, an analog building-block circuit, a digital circuit, or a system level block, the one thing for certain is that the current model being constructed won't be the last model ever constructed. Device, circuit and system modeling are an integral part of the design process. The need for models appears during design exploration, design verification, test definition and even design debugging when unfortunate circumstances have led to refabrication.

As our notion of the system extends to non-electrical components more and more models are required to describe and analyze that system. Multidisciplinary systems are now involving people from a variety of scientific backgrounds including physicists, and mechanical, chemical and civil engineers in addition to electrical and computer engineers. It is unlikely that each of these disciplines will have taught the use of the same analysis tools. Even within electrical and computer engineering we use a variety of tools and languages to describe and analyze our circuits and systems.

Paragon [1] was created with the motivation of providing a modeling environment or framework that was language-based, but also language independent. The term *language-based* refers to the fact that Paragon possesses the semantics and constructs required to represent electrical (analog and digital) and non-electrical (mixed-technology) behavior in hardware description languages (HDLs) such as MAST [2], VHDL-AMS [3-6], and Verilog-A(MS) [7, 8]. Conversely, *language-independent* refers to the fact that using Paragon requires no underlying knowledge of any of these languages with respect to their syntax or peculiarities. Code generation technology does that work for the user from an HDL-neutral format in XML and MathML [9, 10]. Therefore, Paragon is language-aware in its internal codes, but does not impose this foundation to the user level.

Tools of an analogous nature to Paragon are commercially available from a number of vendors for pure digital design [11, 12]. This is not true for analog, mixed-signal or mixed-technology. The Paragon research has evolved from the first such effort that began in an industrial setting originally [13, 14].

More recently, modeling tools for translating models from a description in one language into a form for use in circuit simulation (i.e., typically C code) have been described [15-17]. In [15] and [16], the specifications of the model were entered in languages specifically designed for the purpose of translation to compact form. In [17], the user employs Verilog-AMS to describe the model, which is then translated into an XML format for subsequent C code generation. Shi's group [18] has recently achieved similar functionality using VHDL-AMS as the input language, employing their own internal format. In all of these instances [15-18], the ultimate goal is the C code realization for compilation and linkage to the simulator for fast execution. In contrast to Paragon, each of these approaches begins with a language input – some use HDLs, others do not. In each case, the issue is that of being faced with a language as opposed to a higher-level description that more clearly conveys the essence of a model and makes it far easier to create, modify, debug, support and, very importantly, *transfer* to others. Further, the user can generate any number of language outputs (e.g., VHDL-AMS, Verilog-AMS, MAST) from one description in Paragon that could subsequently be used as input into these C code generators for SPICE-like simulator engines.

## 2. Overview of Model Creation in Paragon

Paragon enables the designer to easily create new models in multiple HDLs, without knowing or learning any of them. Models can be reused as components of existing models (i.e., macromodeling) for hierarchical modeling. As designers are typically model users, it is beneficial for them to have modeling tools that are easy to use for creating or adjusting models to meet the requirements of the system. Also, as Paragon produces readable standardized code and removes common errors, it is easy to use and modify a model.

Paragon consists of a single interface containing multiple windows much like a multiple document interface (MDI) in a Microsoft tool (e.g., Microsoft Word). However, unlike the MDI found in many applications where a single type of interface exists to create/modify the document, the MDI interface concept is extended in Paragon to multiple types of

[†] This work was supported by NSF Grant EEC-0088011, ONR/USC 01-636, and Texas Instruments.

interfaces all contributing to or operating on the *same* document (that is, the model being created). Each of these user interfaces (UIs) contributes some portion of behavior or structure to a model. In general, all *may* be used if the model is mixed-signal (i.e., mixed continuous-time/discrete-time).

The recommended (but not mandated) modeling process is that of "outside-in", whereby the modeler focuses on the model interface first, then proceeds to describing the internal information of the model, and finally generates code.

A Model Interface Editor within Paragon is designed to allow the user to succinctly specify the model name, connection points, arguments, regions of validity of arguments, and default values of arguments of a model. The connection points can be analog or digital for electrical. Further, connection points can be signal flow, mechanical, optical, thermal, magnetic, etc. [19].

Models generally possess both structure and behavior internally. Structure in this context can simply refer to how equations are related to one another. A simple way of illustrating this point is to take the example of the equations of a resistor and capacitor being used to form a low-pass filter. For the resistor we have V = R*I as the governing equation. For the capacitor we have I = C*dV/dt. If these equations are "connected" such that the voltage for each is the same, a parallel tank is produced. However, if they are connected such that the currents are the same, then a series configuration is realized, which is a different model. Such topological relationships are best described with the large-signal model of the device.

Paragon has a Topology Editor, which resembles a schematic editor, that is used to define the internal structure of the model being created. It is used in conjunction with an Equation Editor interface to define the relationships that exist in the model. Paragon also consists of a variety of other tools for digital modeling, symbol editing, circuit-to-behavioral model generation and model checking [1, 20].

## 3. Achieving Language Independence

Paragon was designed from the outset as a modeling environment for analog, mixed-signal, and mixed-technology entities. The goal has always been to alleviate the need to master the nuances of HDLs for model creation. The initial design involved a proprietary schema for the internal, language-independent model representation [13]. The aim was to enable multilingual code generation from this single representation. The current design utilizes XML and MathML as the internal format. The reasons for the choice of XML are:

1. The use of XML in the database enables the description of model information in a simple and flexible structured text format.
2. XML lends itself to standardization and open sourcing for expressing model descriptions, which will lead to model developers sharing models among different modeling environments in a rich and structured format, that is independent of any HDL.
3. The technologies and applications built on XML provide a powerful and efficient way of expressing and manipulating almost all types of data, including complex mathematics and images. For example, Extensible Stylesheet Language Transformations (XSLT) [25] is used for transforming XML from one form into another.

The model expressions and equations are expressed in MathML, which is an XML application for describing mathematical notation and capturing both its structure and content. MathML is fast becoming the *de facto* standard for encoding the structure of mathematical expressions so that they can be displayed and shared over the World Wide Web. Several mathematical software vendors use MathML for expressing mathematical expressions. Thus, importing complex mathematical expressions from these packages into Paragon (and vice versa) is enabled.

The model symbols in Paragon are saved in the database in Scalable Vector Graphics (SVG), which is a language for describing two-dimensional graphics in XML. Analogous to expressions, more photo editor vendors are adding support to their tools and applications for viewing and editing images in SVG.

### *3.1 XML Schema and Analysis Methods*

The XML database can best be explained using a Document Type Definition (DTD), which is a formal description in XML Declaration Syntax. It defines the different legal building blocks (elements) of the database, describing where they may occur and how they all fit together. The DTD is given below as Fig. 1.

```
<!DOCTYPE PARAGON DATABASE DTD [
<!ENTITY math dtd SYSTEM
"http://www.w3.org/TR/MathML2/appendixa.html">
<!ELEMENT MODEL (INTERFACE, BODY)>
<!ATTLIST MODEL NAME CDATA #REQUIRED>
<!ATTLIST MODEL COMMENTS CDATA #IMPLIED>
<!ELEMENT  INTERFACE (PARAMETER, PORT)>
<!ATTLIST PARAMETER.name CDATA #REQUIRED>
<!ATTLIST PARAMETER.value CDATA #IMPLIED>
<!ATTLIST PARAMETER.type CDATA #REQUIRED
(real|integer|time)>
<!ATTLIST PARAMETER.unit CDATA #IMPLIED>
<!ATTLIST PARAMETER.validity range CDATA #IMPLIED>
<!ATTLIST PORT.name CDATA #REQUIRED>
<!ATTLIST PORT.mode CDATA #REQUIRED (in|out|inout)>
<!ATTLIST PORT.nature CDATA #REQUIRED
(electrical|mechanical|thermal|optical|magnetic)>
<!ATTLIST PORT.type CDATA #REQUIRED (terminal|quantity|signal)>
<!ELEMENT BODY (MODEL EXPRESSIONS,
BRANCH,MACROMODEL)>
<!ELEMENT MODEL EXPRESSIONS (math dtd)>
<!ELEMENT BRANCH (QUANTITY, EQUATION)>
<!ATTLIST BRANCH.name CDATA #REQUIRED>
<!ATTLIST BRANCH.from CDATA #REQUIRED>
<!ATTLIST BRANCH.to CDATA #REQUIRED>
<!ATTLIST QUANTITY.name CDATA #REQUIRED>
<!ATTLIST QUANTITY.nature CDATA (through|across)>
<!ATTLIST QUANTITY.type CDATA #IMPLIED>
<!ATTLIST QUANTITY.unit CDATA #IMPLIED>
<!ATTLIST EQUATION.type CDATA (simulataneous|conditional)>
<!ATTLIST EQUATION.value CDATA (math dtd)>
```

```
<!ELEMENT MACROMODEL (MACROMODEL PARAMETER)>
<!ATTLIST MACROMODEL.name CDATA #REQUIRED>
<!ATTLIST MACROMODEL.entity CDATA #REQUIRED>
<!ATTLIST MACROMODEL.architecture CDATA #REQUIRED>
<!ATTLIST MACROMODEL PARAMETER.name CDATA
#REQUIRED>
<!ATTLIST MACROMODEL PARAMETER.value CDATA
#REQUIRED> ]>
```

Fig. 1. DTD showing the model description syntax of XML database.

Each model document has an interface and a body. The model interface consists of the model name, connection points and parameters. The body contains the model topology and equations. The topology consists of branches and instances of other models. The branches are in turn defined by their 'through' and 'across' variables and mathematical expressions involving these variables. The topology and these mathematical expressions collectively define the model behavior.

The code generation module generates code in multiple HDLs by analyzing the XML database. Some of the important analysis methods that accompany the Paragon XML schema are:

1. Creation of an Abstract Syntax Tree (AST), which is an internal data representation obtained by parsing the MathML expression trees. The AST represents the inter-relationships among the variables and constants in the model equations and expressions.
2. Analysis of the AST for determining the functional dependency and time dependency characteristics. This enables the generation of efficient code by distinguishing constants, time-varying variables, and "post-iterative" calculations (e.g., power).
3. Checking for discontinuities in the model expressions and accordingly generating proper code (e.g., issuing `break` statements as appropriate for discontinuities in VHDL-AMS) ***and*** indicating these to the modeler.

In addition to HDL code generation, Paragon produces other outputs useful in the design process. Paragon can generate symbols from its SVG format for a number of design environments (e.g., Cadence, Mentor and Synopsys). It can also generate model descriptions in an HTML document. This includes the model symbol, the large-signal topology of the model, the model equations and the model parameters including default values, units and ranges of validity.

### *3.2 Illustration of Language Independence*

The language independence of Paragon is illustrated by the use of an example. The example presented here is the behavioral model of a self-heating resistor. It also shows the ability of Paragon to generate mixed technology models. Space does not permit us to show the entire XML database and screenshots of the UIs of Paragon, so the main information will be conveyed through a portion of the XML database and the generated VHDL-AMS code and Verilog-A code. The portion of the XML database given below in Fig. 2 shows one of the branches of the topology of the resistor along with its variables and equations. VHDL-AMS, Verilog-A and MAST codes of the self-heating resistor were generated in Paragon from the HDL-neutral XML database. All were successfully simulated. The VHDL-AMS code is shown in Fig. 3 and was verified in SystemVision [23]. The Verilog-A code is shown in Fig. 4 and was verified in ADMS [26]. These exhibit the powerful language independent nature of the XML schema used for model description in the Paragon database, which in turn shows the language independent nature of Paragon.

```
<branch from="elec1" name="elec_branch" to="elec2">
<quantity name="i" nature="through" type="current" unit="ampere" />
<quantity name="v" nature="across" type="voltage"
unit="volt" />
<equation  type="simultaneous">
<mrow>
<mi>v</mi>
<mo>=</mo>
<mi>i</mi>
<mo>*</mo>
<mi>resistance</mi>
</mrow>
</equation>
</branch>
```

Fig. 2. Portion of the XML model database of the self-heating resistor

```
-- VHDL-AMS Model of thermal_resistor generated by
Paragon
-- This is a machine generated code.
-- Generated on Tue, 01 Apr 2003 12:28:48
library IEEE;
library IEEE_proposed;
use IEEE.math_real.all;
use IEEE_proposed.electrical_systems.all;
use IEEE_proposed.thermal_systems.all;
entity thermal_resistor is
generic
(alpha:real:=0.0068;initial_resistance:real:=5.0;
initial_temp:real:=27.0);
port (terminal elec1 : electrical;terminal elec2 :
electrical;terminal therm : thermal);
end entity thermal_resistor;
architecture Arch1 of thermal_resistor is
quantity resistance : real := 1.0;
quantity i through elec1 to elec2;
quantity v across elec1 to elec2;
quantity heat_flow through therm to thermal_ref;
quantity temperature across therm to thermal_ref;
begin
resistance==(initial_resistance*(1.0+
(alpha*(temperature-(initial_temp+273.0)))));
v==(i*resistance);
heat_flow==-((i**2.0)*resistance);
end architecture Arch1;
```

Fig. 3. Paragon generated VHDL-AMS code of self-heating resistor.

```
-- Verilog-A Model of thermal_resistor generated
by Paragon
-- This is a machine generated code.
-- Generated on Mon, 11 Aug 2003 09:16:29 AM

`include "discipline.h"
```

```
`include "constants.h"
module thermal_resistor(elec1, elec2, therm);
inout elec1, elec2, therm;
electrical elec1, elec2;
thermal therm;
parameter real alpha=0.0068;
parameter real initial_resistance=5.0;
parameter real initial_temp=27.0;
real resistance;
analog begin
resistance = (initial_resistance*(1.0+(alpha*
(Temp(therm)-(initial_temp+273.0)))));
V(elec1,elec2) <+ resistance*I(elec1,elec2);
Pwr(therm) <+ -(pow(I(elec1,elec2),2.0))
*resistance;
end
endmodule
```

Fig. 4. Paragon generated Verilog-A code of self-heating resistor.

## 4. Semiconductor Device Model Example

To further illustrate language independent model creation, the EKV MOSFET [21] model was implemented. This example speaks directly to the ability to produce the HDL descriptions used as inputs for subsequent C code generation in [17] and [18]. Following the procedure outlined in Section 2, the model name, connection points and model parameters are first created. For each model parameter it is possible to specify default values, units, and range of validity information that gets translated into model parameter checking in the HDL code. Next, the user begins to enter the large-signal topology of the MOSFET as shown in Fig. 5. The user is able to create internal nodes in this graphical editor, which are indicated as internal source and drain nodes. Also, the user can either specify branches for items such as resistances, capacitances and body diodes, or instantiate a Paragon model as shown in Fig. 5 for illustration. The branches whose behaviors are defined with expressions are seen as the boxes in Fig. 5. The branch relationships are not repeated here, but can be viewed in the Technical Report on the EKV MOSFET model [22]. The instantiated models appear as a schematic symbol as one would expect (e.g., resistor, diode).

The final step of the model creation process is to generate the HDL code for the model. VHDL-AMS, MAST and Verilog-A(MS) codes were all generated by Paragon. All were verified through simulation to give equivalent results. Further, the generated MAST was compared to the model available in Saber [24] with no discrepancies.

For semiconductor device modeling, the aim of Paragon is to provide the ability to instantiate common effects that are kept in a library analogous to regular models. This *semiconductor device toolbox* will contain behaviors such as junction capacitance relationships, diode relationships, breakdown effects, drain-source current expressions, etc. Once instantiated, Paragon provides the option to flatten the hierarchy upon code generation, so that all code is flattened in the HDL implementation for simulation efficiency as opposed to maintaining a hierarchical netlist of these effects.

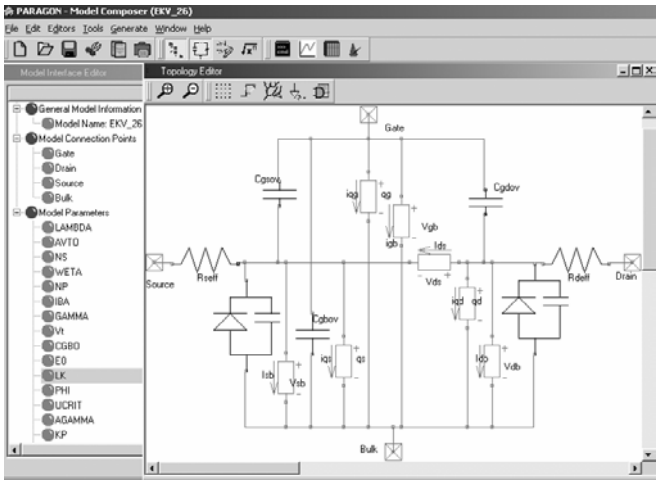


Fig.5. Screenshot of Paragon showing the model interface and topology of the EKV MOSFET model.

## 5. Conclusion

The modeling tool described in this paper enables the user to quickly and correctly create new models and reuse parts of existing models. The environment raises model creation, support, and dissemination a level above programming software, preventing the developer from having to master the HDLs to generate code in them. Knowledge of basic modeling concepts and mathematics is the only requirement for creating models using Paragon.

## References


[1] V. Chaudhary, M. Francis, X. Huang, H. A. Mantooth, "Paragon - A mixed-signal behavioral modeling environment," *IEEE Int. Conf. on Communications, Circuits, & Syst. (ICCCAS)*, pp. 1315-1321, Chengdu, China, June 30, 2002.

[2] H. A. Mantooth, M. Fiegenbaum, *Modeling with an Analog Hardware Description Language*, Kluwer Academic Publishers, Norwell, MA, 1995.

[3] *1076.1-1999 IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual*, IEEE Press, ISBN 0-7381-1640-8.

[4] P. Ashenden, G. D. Peterson, D. A. Teegarden, *The Systems Designer's Guide to VHDL-AMS*, Morgan-Kaufmann, San Francisco, CA, 2003.

[5] R. S. Cooper, *The Designer's Guide to Analog and Mixed-Signal Modeling*, Avant! Press, 2001.

[6] E. Christen, K. Bakalar, "VHDL-AMS-a hardware description language for analog and mixed-signal applications," *IEEE Trans. on Circuits and Systems, part II*, Vol. 46 Issue: 10, pp. 1263–1272, Oct. 1999.

[7] D. Fitzpatrick, I. Miller, *Analog Behavioral Modeling with Verilog-A*, Kluwer Academic Publishers, Norwell, MA, 1997.

[8] P. Frey, D. O'Riordan," Verilog-AMS: Mixed-signal simulation and cross domain connect modules", *Proc. IEEE BMAS*, pp. 103 –108, Oct. 2000.

[9] Extensible Markup Language (XML), http://www.w3.org/XML

[10] Mathematical Markup Language (MathML), http://www.w3.org/Math

[11] *HDL Designer Series*, Mentor Graphics Corporation, http://www.mentorg.com/hdldesigner/

[12] *StateMate User's Manual*, I-logix, http://www.ilogix.com/fs_prod.htm.
[13] H. A. Mantooth, J. P. Skudlarek, J. R. Carlson, D. K. Cooper, I. E. Getreu, G. Graham, S. Pothier, R. Vedam, C. M. Wolff, "Model Architect - An environment for HDL-based mixed-signal model development," *Proc. IEEE BMAS*, 11 pgs, 1999.
[14] H. A. Mantooth, C. M. Wolff, "Component-based analog and mixed-signal simulation model development," U.S. Patent No. 5963724, Filed Feb. 16, 1996, Issued Oct. 5, 1999.
[15] R. V. H. Booth, "An extensible compact model description language and compiler," *Proc. IEEE BMAS*, pp. 39-44, Oct. 2001.
[16] M. Zorzi, N. Speciale, G. Masetti, "Automatic embedding of a ferroelectric capacitor model in Eldo," *Proc. IEEE BMAS*, pp. 97-101, Oct. 2001.
[17] L. Lemaitre, C. McAndrew, S. Hamm, "ADMS – Automatic Device Model Sythesizer," *Proc. IEEE Custom Int. Circ. Conf.*, pp. 27-30, 2002.
[18] C.-J. Shi, "BMAC – Behavioral Model Analyzer and Compiler", private communication, University of Washington, Seattle, WA, Dec. 2002.
[19] Standard VHDL 1076.1 packages for multiple energy domain support, http://www.vhdl.org/vi/analog.
[20] X. Huang, C. Gathercole and H. A. Mantooth, "Modeling nonlinear dynamics in analog circuits via root localization," *IEEE Trans. Computer-Aided Design*, to appear.
[21] C. C. Enz, F. Krummenacher, E. A. Vittoz, "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *J. Analog Integrated Circuits and Signal Processing*, vol. 8, no. 1, pp. 83-114, July 1995.
[22] M. Bucher, C. Lallement, C. Enz, F. Théodoloz, F. Krummenacher, "The EPFL-EKV MOSFET Model Equations for Simulation", Technical Report, Model Version 2.6, June 1997. Revision I, September, 1997, Revision II, July, 1998.
[23] *SystemVision*, Mentor Graphics Corporation, http://www.mentor.com/systemvision/
[24] H. A. Mantooth, M. Vlach, "Beyond SPICE with Saber and MAST," *IEEE Proc. of Int. Symposium on Circuits Syst.*, vol. 1, pp. 77-80, May 1992.
[25] Extensible Stylesheet Language, http://www.w3.org/TR/xslt
[26] *ADMS*, Mentor Graphics Corporation, http://www.mentorg.com/ams/adms.html.
[27] D. Fitzpatrick, Ira Miller, *Analog Behavioral Modeling with the Verilog-A Language*, Kluwer Academic Publishers, San Francisco, Massachusetts, 1998.