ADMS – Automatic Device Model Synthesizer

Laurent Lemaitre[1], Colin McAndrew[2], Steve Hamm[3]
SPS – Motorola
laurent.lemaitre@motorola.com, colin.mcandrew@motorola.com; steve.hamm@motorola.com
[1]Geneva - Switzerland, [2]Tempe - Arizona, [3]Austin - Texas

Abstract

This paper presents ADMS, a new open-source tool that supports automatic synthesis of compact models into circuit simulators. ADMS takes as input Verilog-AMS compact model descriptions and generates C code that conforms to circuit simulator interfaces. ADMS supports the simulators Mica, Spectre, and HSIM, and has been used to implement the SP and SSIM MOSFET models, the VBIC BJT model, and the R3 resistor model, the last two including self-heating.

Introduction

Implementing compact device models (e.g. BSIM [1] and VBIC [2]) into circuit simulators requires a large amount of work. Much of this is tedious, error-prone, manual implementation of low-level code, and (apart from making a model available) is a non-value-added task as far as a model developer is concerned. The latest BSIM code consists of 21 different files, consisting of almost 20,000 lines of C code tied to implementation in one specific simulator.

Understanding the details of, and dependencies between sections of, code this large is a daunting task that has a very serious ramification. It erects a huge barrier to developing, implementing, testing, and sharing model improvements. Also, the model code cannot be easily used just by itself, but must be compiled and linked into a simulator.

ADMS addresses these issues. It has been designed to make implementation of compact models simple, efficient, and robust. It integrates a built-in symbolic derivative calculator. Today it supports C code generation for the Abstract Programming Interface (API) of three different simulators, Spectre [3], Mica [4], and HSIM [5], and support for more is under development. The specification of the code generator is written in XML, the Extensible Markup Language [6]. It can be modified without the need to recompile ADMS source code. This simplifies adding API specification of other simulators to ADMS.

Other approaches similar to ADMS have been considered recently [7][8]. In both approaches the specification of a compact model is in a language specifically designed for this purpose. Using one of the approaches requires learning a new modeling language. One has to install specific tools to be able to check the validity of a model, so these are not generic approaches, and inhibit widespread model dissemination and sharing.

ADMS uses Verilog-AMS [9]. This language was designed for analog and mixed-signal systems. It can be judiciously re-used for the description of compact models. The constructs used in the Verilog-AMS language are intuitive and easy to learn, and the language is standard and supported by many simulators, e.g. [3] [4].

Verilog-AMS allows models to be defined hierarchically. This allows model developers to increase the readability of their code, and simplifies the sharing of blocks of code between different device models (e.g. for junction diodes).

A major advantage of the ADMS approach is that it allows Verilog-AMS simulators to be used to check the validity of a compact model prior to implementation into simulators. This significantly helps model development, as all simulation capabilities, including DC, AC, noise, transient, etc., are directly available for the Verilog-AMS code.

The format used by ADMS to describe a compact model can be used as a *de facto* standard to share new ideas and model code among device modeling engineers. No particular programming language skills are needed to develop or understand the syntax of the model definition.

Overview of ADMS code

A. *Overview*

Fig. 1 gives an overview of the interactions between ADMS, the Verilog-AMS source code description, and target circuit simulators.

The model is specified in Verilog-AMS. ADMS parses the source code and creates an XML internal data tree that mirrors the Verilog-AMS model description. Plug-in applications process the XML internal data representation. Different outputs can then be obtained from the initial Verilog-AMS description. Formatting of the output can be specified in XML. The APIs of circuit simulators have been partially translated into XML. This way the update of simulator-specific API versions can be done without the need to recompile the ADMS source code.

Since many simulators embed Verilog-AMS parsers this configuration allows testing of model descriptions "off-line," and enables verification of model implementation in the same simulation environment.

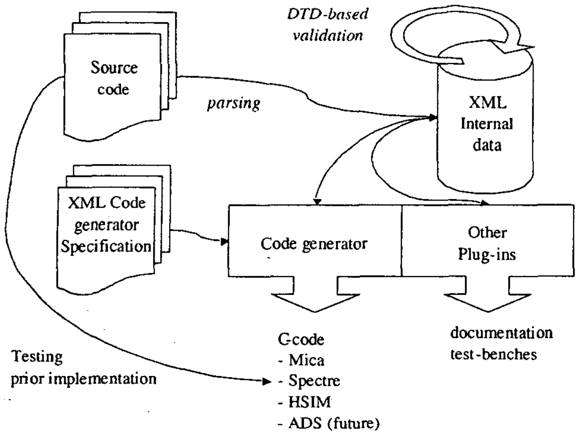

Fig. 1 Interactions between ADMS components

B. ADMS code implementation

ADMS has been implemented in the C language. It is based on open-source C library Glib [10]. Glib is a utility library that provides many useful data types such as *n*-ary trees, string manipulation, and a simple XML subset parser. The ADMS code has been written in an object-oriented style. The coding design work has been made easier by the use of Gobject, the native object system of Glib.

The choice of using an open-source library as the building framework of ADMS was driven by the desire to make ADMS source code freely available to the device modeling community.

C. Internal Data Representation

The main data representations used in ADMS are based on XML, which is the universal format for structured documents and data on the worldwide web [6]. XML provides a large set of technologies that simplify the design of robust, re-usable code.

XML offers a simple way to validate internal data. Rules that shape valid data are written in a subset of XML. The set of rules forms the so-called Document Type Definition (DTD). Fig. 2 shows the complete DTD that defines the internal data format implemented by ADMS. Embedding of an external DTD is possible. This feature favors the re-use of well-established formats. For instance, MathML [11] has been adopted for the formatting of mathematical expressions.

Technologies built around XML give a powerful means to manipulate XML data.

```
<!-- DTD ADMS DATA STRUCTURE -->
<!-- RE_USE OF MATHML2 DTD -->
  <!ENTITY % math SYSTEM ". /mathml2.dtd" >
  %math;
<!-- LOCAL ENTITIES -->
  <!ENTITY % atomic
"(assignment|contribution|conditional|while|for)" >
  <!ENTITY % code "(%atomic;|block)" >
  <!ENTITY % class "(node|branch|variable|parameter|function)" >
<!-- START OF THE DTD -->
  <!ELEMENT XVeriloga (discipline|nature|device)+ >
<!-- DISCIPLINE/NATURE SECTION -->
  <!ELEMENT discipline EMPTY >
  <!ELEMENT nature EMPTY >
<!-- DEVICE DECLARATION SECTION -->
  <!ELEMENT device (name,declaration,analog) >
  <!ELEMENT declaration
(l.terminal|l.node|l.variable|l.branch|l.parameter|adms)* >
  <!ELEMENT l.terminal (s.terminal)* >
  <!ELEMENT s.terminal (name) >
  <!ELEMENT l.node (s.node+) >
  <!ATTLIST l.node flow (input|output|inout) 'inout'>
  <!ATTLIST l.node discipline CDATA 'electrical'>
  <!ELEMENT s.node (name) >
  <!ELEMENT l.parameter (s.parameter)* >
  <!ATTLIST l.parameter type (real|integer) 'real'>
  <!ELEMENT s.parameter (name,default?,l.range?) >
  <!ELEMENT l.variable (s.variable)* >
  <!ATTLIST l.variable type (real|integer) #REQUIRED >
  <!ELEMENT s.variable (name) >
  <!ELEMENT l.branch (s.branch)* >
  <!ELEMENT s.branch (branch.p,branch.n?,name) >
  <!ELEMENT l.range (s.range+) >
  <!ELEMENT s.range (math) >
  <!ATTLIST s.range type (from|exclude) #REQUIRED >
<!-- DEVICE ADMS SECTION -->
  <!ELEMENT adms (attribute+,class,name) >
  <!ELEMENT class (name) >
  <!ELEMENT attribute (key,value) >
  <!ELEMENT key (name) >
  <!ELEMENT value (math|%code;) >
<!-- DEVICE ANALOG SECTION -->
  <!ELEMENT analog (block) >
  <!ELEMENT block (name?,(%code;)+) >
  <!ELEMENT conditional
(conditional.if,conditional.then,conditional.else?) >
  <!ELEMENT while (while.test,while.code) >
  <!ELEMENT while.test (math) >
  <!ELEMENT while.code %code; >
  <!ELEMENT for (for.init,for.test,for.post,for.code) >
  <!ELEMENT for.init (assignment) >
  <!ELEMENT for.test (math) >
  <!ELEMENT for.post (assignment) >
  <!ELEMENT for.code %code; >
  <!ELEMENT conditional.if (math) >
  <!ELEMENT conditional.then %code; >
  <!ELEMENT conditional.else %code; >
  <!ELEMENT default (math) >
  <!ELEMENT assignment (name,math) >
  <!ELEMENT contribution (branch,math) >
  <!ELEMENT branch (name,branch.p,branch.n?) >
  <!ELEMENT branch.p (name) >
  <!ELEMENT branch.n (name) >
<!—general name -->
  <!ELEMENT name (#PCDATA) >
  <!ELEMENT discipline_name (name) >
```

Fig. 2 Complete DTD used by ADMS

For instance Xpath [12] is a subset of XML that defines the way XML data are traversed. Transformation scripts can be written in XSLT [13], a subset of XSL. XSLT scripts are part of the implementation of the circuit simulator code generators.

Device Model Description

A. *Example*

Fig. 3 gives a complete example of a Verilog-AMS description of a simple resistor model.

Verilog-AMS has some restrictions that prevent it from completely describe compact models, and details of how they should be implemented as built-in models in circuit simulators. For instance, Verilog-AMS does not distinguish between model parameter and instance parameters. Section A shows how it is still possible to "pass" this concept to ADMS while not breaking the Verilog-AMS validity of the model description. The macros *model* and *instance* are set to void in Verilog-AMS mode. Both macros are redefined in Section B. When ADMS reads the model description, definitions of Section B override previous definitions, and extra information is passed to ADMS. Verilog-AMS simulators do not see the extra information and parse the model description correctly. The macros `info, `spicename, `node_spicename and `flag work in a similar way.

The `INITIAL_MODEL and `INITIAL_INSTANCE keywords mark code executed during non-bias dependent (e.g. geometry and/or temperature dependent) updates of model and instance parameters. The keyword `LOAD marks code executed for the evaluation of current flows and voltage potentials. The keyword `FINAL marks code executed after convergence of a circuit. This helps speed up the code execution of built-in models. For example, computation of power dissipation does not need to be done after each bias dependent Newton iteration during convergence.

B. *Hierarchy*

Verilog-AMS offers the possibility to define models hierarchically. Fig. 4 gives an excerpt of the code used for the description of the new MOSFET model SP [14]. A hierarchical definition of the model improves the readability of the code, and allows easy substitution or interchange of sub-blocks of models. In SP the extrinsic diodes are just standard models. These elements can easily be replaced by similar model elements from different sources. Verilog-AMS allows passing parameter value by name between different block levels, facilitating the plug-in of sub-model descriptions to a top-level model description. In Fig. 4 values of the top-level parameters JCTareaBD and JCTareaBS are passed to the internal diode parameter JCTarea.

```
// Example of Compact Model Specification
  'include "discipline.h"
  'define KELVIN 273.15
// SECT. A - Usefull declarations
  'define INIT_MODEL @initial_step
  'define INIT_INSTANCE @initial_step
  'define LOAD /* not supported */
  'define FINAL_STEP @final_step
  'define node_spicename(node,text)
  'define model(param)
  'define instance(param)
  'define info(param,text)
  'define spicename(param,text)
  'define flag(param,value)
module R(p, n);
// SECT. B - ADMS support
  'include "R.adms"
// SECT. C - Node declaration */
  inout p, n;
  electrical p, n;
  'node_spicename(p,"anode")
  'node_spicename(n,"cathode")
// SECT. D - Model Parameter Declaration
  parameter real rsh=1.0 from (0.0:inf);
  'model(rsh)  'info(rsh,"sheet res")
  'flag(rsh,"DEFAULT"|"ASK")
  parameter real tc1=1.0 from (0.0:inf);
  'model(tc1) 'info(tc1,"temp. coeff.")
  'flag(tc1,"DEFAULT"|"ASK")
  parameter real tnom=25.0;
  'model(tnom) 'info(tnom,"nom. Temp.")
  'flag(tnom,"DEFAULT"|"ASK")
// SECT. E - Instance Parameter Declaration
  parameter real w=1.0 from (0:inf);
  'instance(w) 'info(w,"width")  'flag(w,"DEFAULT"|"ASK")
  parameter real l=1.0 from (0:inf);
  'instance(l) 'info(l,"length") 'flag(l,"DEFAULT"|"ASK")
// SECT. F - Variable Declaration
  real tdiff;  'flag(tdiff,"ASK")
  real rnom, r_t; 'flag(rnom,"ASK")  'flag(r_t,"ASK")
  real i, power;  'flag(i,"ASK") 'flag(power,"ASK")
  analog begin
// SECT. G - Model Update
    'INITIAL_MODEL begin
      tdiff = $temperature - ('KELVIN+tnom);
     end
// SECT. H - Instance Update
    'INITIAL_INSTANCE
     begin
      rnom = rsh * l/w;
      r_t = rnom * (1.0 + tc1*tdiff);
     end
// SECT. I - Evaluation and Load
    'LOAD
     begin
      i = V(p,n)/r_t;
      I(p,n) <+ i;
     end
// SECT. J - at final step
    'FINAL_STEP
     begin
      power = V(p,n)*i;
     end
  end
endmodule
```

Fig. 3 Verilog-AMS description of simple resistor model

```
'include "StandardExtrinsicJunction.va"
'include "SPintrinsic.va"
module SP(dnode,gnode,snode,bnode);
 ...
 SPintrinsic
 #(
   // model parameter mapping
   .TOX     (TOX),
   ...
   // instance parameter mapping
   .w (w),
   .l (l),
   ...
 ) SP   (dpnode,gnode,spnode,bnode);
StandardExtrinsicJunction
 #(
   // model parameter mapping
   .JCTis     (JCTis),
   ...
   // instance parameter mapping
   .JCTarea    (JCTareaBD),
   ...
 ) JCTbd(dpnode,bnode);
StandardExtrinsicJunction
 #(
   // model parameter mapping
   .JCTis     (JCTis),
   ...
   // instance parameter mapping
   .JCTarea    (JCTareaBS),
   ...
 ) JCTbs(spnode,bnode);
 ...
endmodule
```

Fig. 4 Hierarchical model description

C. Recent Results

ADMS has been used to implement several models in one or more simulators. The SSIM MOSFET model [15] has been implemented in Spectre and HSIM. An accurate non-linear 3-terminal resistor model R3, based on [16] with some extensions including self-heating, has been implemented in Mica. The VBIC BJT model, with self-heating, has also been implemented in Mica. The new MOSFET model SP [14] is currently being implemented into Spectre.

D. Speed Efficiency

In order to measure the speed efficiency of ADMS-implemented models versus manually implemented models, SSIM was re-implemented into MICA using ADMS (it already existed as a hand-coded built-in model). The description of the model is about 3000 lines of code. The automatically generated ADMS implementation is about 20% slower than the hand coded version.

Future Work

A primary goal of ADMS is to propose a standard for the description of compact models. Proposals to the Verilog-AMS Language Reference Committee will be made to include features useful for the complete description of a compact model in Verilog-AMS. XML specifications for the interfaces for additional circuit simulators are being developed.

Conclusion

This paper presented a new tool ADMS for automatic implementation of compact models in circuit simulators. ADMS has been successfully used for the integration of new device models into Mica, Spectre and HSIM.

Compact model standardization is of great benefit to the semiconductor industry. However, a continued reliance on hand coding in low level languages is a huge barrier to model improvement and dissemination. Defining models in a standard high-level language like Verilog-AMS, and automatically generating efficient, robust, correct-by-construction code, is a big step forward in modeling.

Acknowledgments

The authors acknowledge the help and suggestions of Surya Veeraraghavan, Chip Workman, Kiran Gullapalli, Steve Beckerich, Jim Victory, Nathan Finkelstein, Kristin Beggs and Dan Feng.

References

[1] Y. Cheng and C. Hu, "*MOSFET Modeling & BSIM3 User's Guide*," Kluwer Academic Publisher, 1999.
[2] C. C. McAndrew *et al.*, "VBIC95, The Vertical Bipolar Inter-Company Model," *IEEE J. Solid-State Circuits*, vol. 13, no. 10, pp. 1476-1482, Oct. 1996.
[3] *SPECTRE CMI Reference Manual,* Cadence Design System, 1995.
[4] *MICA Device Programming Interface, Documentation and Programmer's Guide*, Motorola Internal Document, 1998
[5] *HSIM User Guide*, NASSDA Corporation, 2001.
[6] *Extensible Markup Language (XML)*, http://www.w3.org/XML
[7] R. V. H. Booth, "An Extensible Compact Model Description Language and Compiler," *Proc. IEEE BMAS*, pp. 39-44, Oct. 2001.
[8] M. Zorzi, N. Speciale, G. Masetti, "Automatic embedding of a ferroelectric capacitor model in Eldo," *Proc. IEEE BMAS*, pp. 97-101, Oct. 2001.
[9] *Verilog-AMS Language Reference Manual*, Open Verilog International, 1999.
[10] *Glib Reference Manual*, http://developer.gnome.org/doc/API/glib/index.html
[11] *Mathematical Markup Language*, http://www.w3.org/Math
[12] *XML Path Language (XPath)*, http://www.w3.org/TR/xpath
[13] *XSL transformations (XSLT)*, http://www.w3.org/TR/xslt
[14] G. Gildenblat, N. Arora, R. Sung, and P. Bendix, "Scalable Surface Potential Based Compact MOSFET Model," *Proc. International Semiconductor Device Research Symposium,* p. 333, 1997
[15] S. Veeraraghavan, "SSIM: A New Charge Based MOSFET Model," presented at the *MCNC Circuit Simulation Workshop*, Nov. 1990.
[16] R. V. H. Booth and C. C. McAndrew, "A 3-terminal model for diffused and ion-implanted resistor," *IEEE Trans. Electron Devices*, vol. 44, no 5, pp. 809-814, May 1997.