Automatic Generation of Compact Semiconductor Device Models using Paragon and ADMS

Vivek Chaudhary, University of Arkansas, vchaudh@uark.edu
Matt Francis, University of Arkansas, afranci@uark.edu
Wei Zheng, University of Arkansas, wzheng@uark.edu
Alan Mantooth, University of Arkansas, mantooth@uark.edu
Laurent Lemaitre, Freescale, Laurent.Lemaitre@freescale.com

ABSTRACT

This paper illustrates the automatic generation of a complex semiconductor device model (BSIMSOI [1]) into the Spectre simulator [2] from a higher-level representation using Paragon[‡]. Paragon [3] is used to capture the conceptual level description of the model and generate Verilog-AMS code. This code is used for quick turn-around model validation and subsequently used by the ADMS model compiler [6] to generate C code for implementation through Spectre's compact model interface (CMI).

1. INTRODUCTION

The most significant building block to successful analog and mixed-signal integrated circuit (IC) design is the availability of high quality semiconductor device models that have been characterized to the IC process. At present, BSIM3 remains the standard device model available from the majority of semiconductor foundries, while BSIM4 is now beginning to take hold [4]. Semiconductor device modeling takes an enormous amount of time and resource at present. This is primarily due to a lack of modeling tools to facilitate the research, implementation and characterization of these complex models, in addition to the cumbersome and error-prone nature of implementation of such models directly in SPICE-like simulators. Without advanced modeling tools the ability to compile other types of models, such as behavioral models of circuits, into many advanced simulators is effectively disabled.

The objective of this paper is to illustrate the usefulness of an advanced modeling scheme for compact models, based upon a well defined, scalable, comprehensive and extensible metafile for representing compact models [5], and an advanced model compiler (ADMS) and modeling tool (Paragon). The methodology presented is used to generate an industry standard model, the BSIM3-SOI model (version 2.2) [1]. The format utilized provides a neutral interface between hardware description languages (HDLs), compilers and modeling tools, and simulators within the complete modeling methodology.

The concept and benefit of a compact model compiler based on standard high-level behavioral languages has been described by a number of investigators [7-9]. However, a major obstacle for model compilers is the generated code efficiency, which has been reported to be 100 to 1000 times slower than hand-developed code. Recent work demonstrated by researchers at the University of Washington indicates that this deficiency can be overcome [10]. With some compilation optimization technologies developed specifically for compact model compilers, it has been shown that even BSIM models, implemented in VHDL-AMS, can be created with performance comparable to human optimized SPICE simulator code. Such results serve to validate the promise of maintainable and efficient compact models from high-level languages.

This paper illustrates the usefulness and extensibility of the high-level representation by using Paragon to generate Verilog-AMS [11-12] code for the ADMS model compiler for the BSIMSOI model. Utilizing Verilog-AMS and ADMS allows validation not only of the compiled compact model with original SPICE code (as implemented in Spectre), but also allows validation of the intermediate Verilog-AMS code, as both formats are supported by the Spectre simulator.

2. MODEL COMPILATION

Model compilation is the process of automatically generating compact semiconductor device models in C/C++ for SPICE-like simulators from a higher-level abstract representation of the model. The overall model creation process is illustrated in Fig 1. The higher-level representation of the model is captured by Paragon and various model compilers like ADMS and MCAST work on this higher-level representation of the model to generate SPICE code for the target simulator. The biggest challenge has been developing model compilers capable of generating compact low level C/C++ code comparable in speeds to the hand-generated models. MCAST has developed industry-

[‡]This work is sponsored by DARPA/MTO NeoCAD Program under Grant No. N66001-01-1-8919, by Office of Naval Research under Subaward No. USC 01-636, by an NSF CRCD award (EEC-0088011) and by the Semiconductor Research Corporation (SRC)

grade device models comparable in simulation speeds to the hand-written models. Paragon is capable of generating low-level C/C++ code for the fREEDA [17] simulator. ADMS is a freely available model compiler available through the open-source community and it supports popular simulators such as Spectre, ADS, McSPICE, and NanoSim. ADMS uses a Verilog-AMS description of the model as input and generates C/C++ code for these target simulators. Paragon can automatically generate Verilog-AMS code from a higher-level description of the model, which can then be fed into ADMS to generate C/C++ models for all the simulators supported by ADMS. The advantages of using a model compiler to develop these models over hand-written models are the following:

1. The model development time is dramatically reduced as the model developer does not need to manually write low level C/C++ code.

2. The generated model is easier to maintain and reuse as the modeler does not have to read and modify the low level C/C++ code.

3. The same abstract representation of the model can be used by a model compiler to generate low level C/C++ code for different simulators.

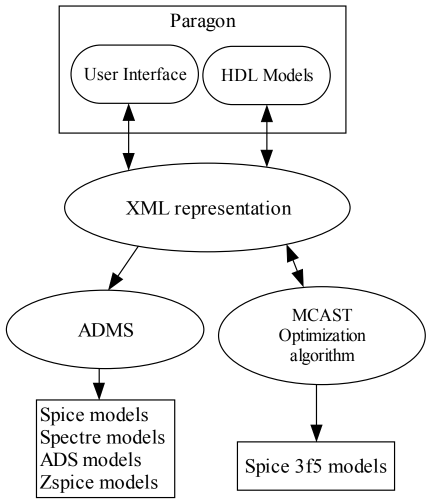


Fig. 1. Methods of model creation utilizing XML schema and advanced compilation tools.

3. MODELING METHODLOGY

The overall modeling methodology used in this paper is best summarized by Fig. 2. Starting with BSIMSOI model documentation and source code [1], the model was implemented utilizing the graphical editors of the Paragon modeling tool. The result of this model editing is the XML Abstract Model Representation. From this model description, Verilog-AMS code, suitable as an input to the ADMS model compiler was generated. This intermediate Verilog-AMS code was validated in the Spectre simulator before CMI code for Spectre was created utilizing the ADMS tool.

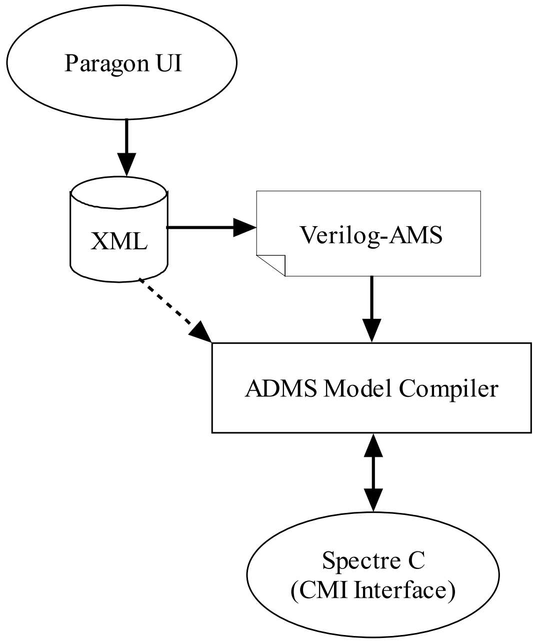
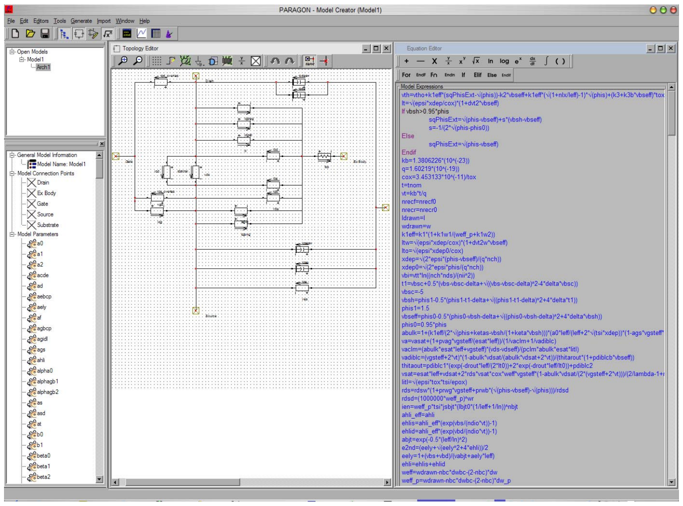


Fig. 2. Automatic generation of native Spectre models using Paragon and ADMS.

3.1 Abstract Model Representation

The abstract model format is designed to capture all of the information necessary to create a semiconductor device model, with the emphasis being on the data specific to the model and not a specific simulator. The schema removes the simulator dependence from device models, facilitates rapid adoption, and supports enhancements through the use of XML [18]. The use of XML enables the description of model information in a simple and flexible structured text format, which lends itself to standardization and open sourcing. It is easily interchanged and adopted, as many standard technologies exist for its manipulation and creation, including XSLT transformations, which are used by Paragon for such tasks as code generation [19].

The model expressions and equations are expressed in MathML [20], which is an XML standard for describing mathematical notation. 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 MathML mathematical expressions involving these variables. The topology and these mathematical expressions collectively define the model behavior. The specifics of the format, including the Document Type Definition (DTD) are published in [5].


PARAGON - Model Creator (Model1)
Topology Editor
Equation Editor
For EndF Fn Endn If Elif Else Endif
Model Expressions

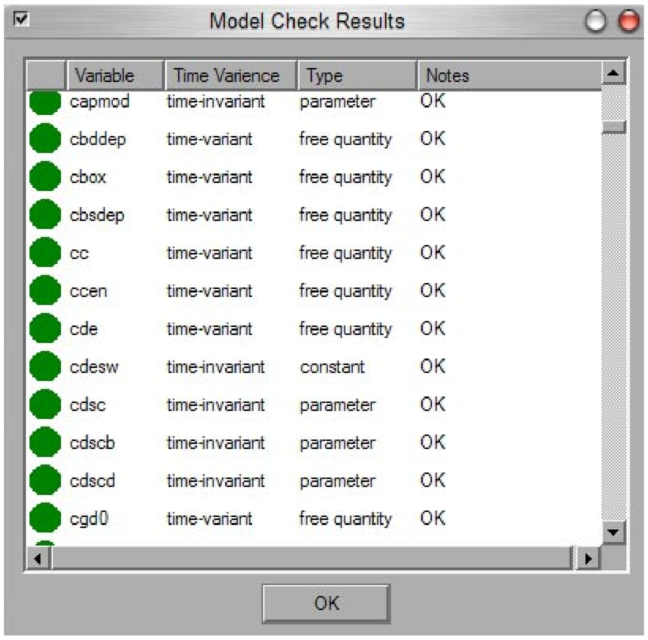
```
vth=vtho+k1eff*(sqPhisExt-√(phis))-k2*vbseff+k1eff*(√(1+nlx/leff)-1)*√(phis)+(k3+k3b*vbseff)*tox
lt=√(epsi*xdep/cox)*(1+dvt2*vbseff)
If vbsh>0.95*phis
        sqPhisExt=√(phis-vbseff)+s*(vbsh-vbseff)
        s=-1/(2*√(phis-phis0))
Else
        sqPhisExt=√(phis-vbseff)
Endif
kb=1.3806226*(10^(-23))
q=1.60219*(10^(-19))
cox=3.453133*10^(-11)/tox
t=tnom
vt=kb*t/q
nrecf=nrecf0
nrecr=nrecr0
ldrawn=l
wdrawn=w
k1eff=k1*(1+k1w1/(weff_p+k1w2))
ltw=√(epsi*xdep/cox)*(1+dvt2w*vbseff)
lto=√(epsi*xdep0/cox)
xdep=√(2*epsi*(phis-vbseff)/(q*nch))
xdep0=√(2*epsi*phis/(q*nch))
vbi=vtt*ln((nch*nds)/(nii^2))
t1=vbsc+0.5*(vbs-vbsc-delta+√((vbs-vbsc-delta)^2-4*delta*vbsc))
vbsc=-5
vbsh=phis1-0.5*(phis1-t1-delta+√((phis1-t1-delta)^2+4*delta*t1))
phis1=1.5
vbseff=phis0-0.5*(phis0-vbsh-delta+√((phis0-vbsh-delta)^2+4*delta*vbsh))
phis0=0.95*phis
abulk=1+(k1eff/(2*√(phis+ketas-vbsh/(1+keta*vbsh)))*(a0*leff/(leff+2*√(tsi*xdep))*(1-ags*vgsteff*
va=vasat+(1+pvag*vgsteff/(esat*leff))/(1/vaclm+1/vadiblc)
vaclm=(abulk*esat*leff+vgsteff)*(vds-vdseff)/(pclm*abulk*esat*litl)
vadiblc=(vgsteff+2*vt)*(1-abulk*vdsat/(abulk*vdsat+2*vt))/(thitarout*(1+pdiblcb*vbseff))
thitaout=pdiblc1*(exp(-drout*leff/(2*lt0))+2*exp(-drout*leff/lt0))+pdiblc2
vsat=esat*leff+vdsat+2*rds*vsat*cox*weff*vgsteff*(1-abulk*vdsat/(2*(vgsteff+2*vt)))/(2/lambda-1+
litl=√(epsi*tox*tsi/epox)
rds=rdsw*(1+prwg*vgsteff+prwb*(√(phis-vbseff)-√(phis)))/rdsd
rdsd=(1000000*weff_p)^wr
ien=weff_p*tsi*jsbjt*(lbjt0*(1/leff+1/ln))^nbjt
ahli_eff=ahli
ehlis=ahli_eff*(exp(vbs/(ndio*vt))-1)
ehlid=ahli_eff*(exp(vbd/(ndio*vt))-1)
abjt=exp(-0.5*(leff/ln)^2)
e2nd=(eely+√(eely^2+4*ehli))/2
eely=1+(vbs+vbd)/(vabjt+aely*leff)
ehli=ehlis+ehlid
weff=wdrawn-nbc*dwbc-(2-nbc)*dw
weff_p=wdrawn-nbc*dwbc-(2-nbc)*dw_p
```

Fig. 3. Screenshot of Paragon showing the creation of the BSIMSOI MOSFET model.

3.2 Paragon Modeling Tool: BSIMSOI Model

A tool that can directly operate upon the XML-format model, Paragon, was utilized to create the BSIMSOI model from documentation. Using Paragon, a user has the ability to enter the large-signal model topology and graphically define branch-based behavior. *Through* and *across* quantities and the associated expressions are easily defined for each branch. Generic model expressions are defined in an expression editing tool, which uses the general language of math, rather than any specific HDL, to define the model equations. Model interface objects, such as parameters and connections points, are defined in a simple form which also permits the user to add their default values and appropriate ranges of validity and comments, all of which are propagated to the abstract format and any generated code. In addition, Paragon provides methods not only for the generation of multiple languages (Verilog-A, VHDL-AMS [21-22], MAST [15], fREEDA [17] and VTB [23]) but also for model debugging and analysis, such as time-variance determination and other sanity checks (Fig. 4).

The large-signal topology of the BSIMSOI model that was implemented in Paragon is shown in Fig. 3. Creating the model interface was the first step in the process of creating the model in Paragon. The model interface consisting of model name (BSIMSOI), external connection points (drain, gate, source, substrate, and body) and all the process and instance parameters were captured in Paragon. The BSIMSOI model consists of 4 fixed and 3 optional nodes. This makes the total number of connection points vary from four to seven. Paragon does not currently support modeling of dynamically varying topologies or optional connection points; hence BSIMSOI was modeled with 6 fixed nodes. Along with the five external connection points, an internal body node was also modeled as shown in the model topology in Fig. 3. The only node that was neglected was the thermal junction node required for a self-heating version of the BSIMSOI model. Finally, the behavior of the model was entered by defining all the branch relationships [1].



	Variable	Time Varience	Type	Notes
●	capmod	time-invariant	parameter	OK
●	cbddep	time-variant	free quantity	OK
●	cbox	time-variant	free quantity	OK
●	cbsdep	time-variant	free quantity	OK
●	cc	time-variant	free quantity	OK
●	ccen	time-variant	free quantity	OK
●	cde	time-variant	free quantity	OK
●	cdesw	time-invariant	constant	OK
●	cdsc	time-invariant	parameter	OK
●	cdscb	time-invariant	parameter	OK
●	cdscd	time-invariant	parameter	OK
●	cgd0	time-variant	free quantity	OK

Fig. 4. A screen-shot of model-profiler window inside Paragon environment showing dependencies of all the BSIMSOI model variables.

The usefulness of a high-level tool such as Paragon is illustrated through the example of this BSIMSOI model, as illustrated in Fig. 3. The model's interface, topology and expressions are easily browsed and modified, allowing another developer to quickly identify and modify specific facets of the model. The validity and sanity of all the model parameters and expressions can be done before actually generating the Verilog-AMS code. Fig. 4 shows a screen-shot of the model-profiler tool in Paragon. All the model parameters, constants and variables are analyzed and listed in tabular form. The modeler can verify their validity before proceeding ahead with automatic code-generation.

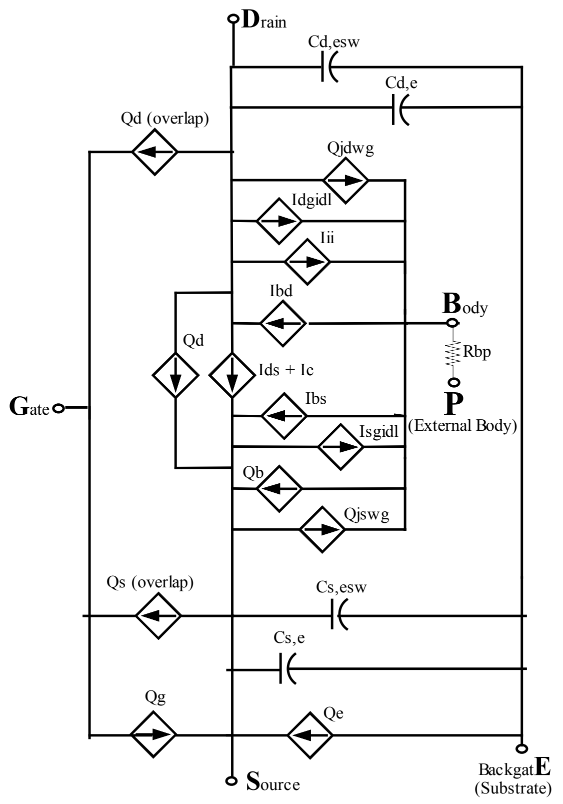


Fig. 5. Large-signal topology of the BSIMSOI, version 2.2 MOSFET model.

3.3 ADMS

The final step in the model creation process is to generate the low level C code for the Spectre circuit simulator. Paragon was used to first generate the Verilog-AMS code for BSIMSOI model. The generated Verilog-AMS code was validated in Spectre for correctness. ADMS was then used to implement this model directly in Spectre using the CMI toolkit. ADMS takes Verilog-AMS as input and generates compact C/C++ model for the target simulator. 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 Verilog-AMS language has some restrictions that prevent it from creating a model which can provide all the information needed by a tool like ADMS to generate C code for SPICE-like circuit simulator [7]. For instance, Verilog-AMS does not distinguish between model parameter and instance parameters. It is possible to "pass" all this information to ADMS without breaking the Verilog-AMS validity of the model description [7]. This is achieved by defining macros in the Verilog-AMS code that are set to *void* in Verilog-AMS mode. Verilog-AMS simulators do not see the extra information and parse the model description correctly. On the other hand ADMS parses the extra-information correctly and is able to generate a compact C model for the target simulator. Paragon-generated Verilog-AMS code of BSIMSOI model was manually appended with these macros before feeding it to ADMS.

ADMS has been designed to make the implementation of compact models simple, efficient and robust. It supports C code generation for the Application Programming Interface (API) of various simulators including Spectre [2], Mica [24], HSIM [25] and zSpice [26]. The specification for code-generation for various simulators is written in XML, which can be developed and supported by simulator vendors without recompiling ADMS source code. This also simplifies adding the API specification of other new simulators to ADMS in the future.

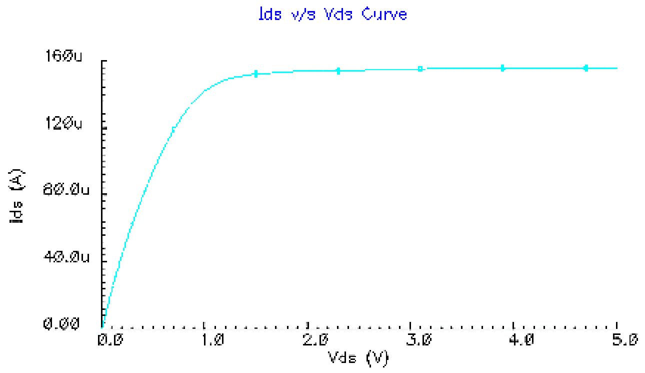


Fig. 6. Ids-Vds curve for generated and compiled BSIMSOI, version 2.2 MOSFET model.

ADMS was used to generate a compact C model of BSIMSOI in Spectre from modified Verilog-AMS code generated by Paragon. An XML file for the Spectre interface and CMI toolkit was used in generating and compiling this model into Spectre. This XML file and the CMI toolkit are developed and maintained by Cadence [27]. This XML file is used by ADMS to build the CMI source code of the BSIMSOI model from its Verilog-AMS description.

The generated BSIMSOI model was simulated in Spectre and its results were compared with the built-in BSIMSOI model in Spectre for speed and accuracy. The automatically generated C code performed accurately and simulated in about the same time as the native Spectre model. Implementation and validation using Paragon took less than two weeks, which is significantly less than the time it takes to implement a new model of BSIM3 complexity in a SPICE-like simulator such as Spectre.

4. FUTURE WORK

Future work involves developing more synergy between Paragon and ADMS to further facilitate the ease of generating complex SPICE models for various circuit simulators. Work is in progress to make ADMS directly generate SPICE models from the XML schema instead of using a Verilog-AMS description of the model. Also, in future work code-generation capabilities of Paragon will be enhanced to generate Verilog-AMS descriptions of models that will have all the additional information that AMDS needs to generate compact models without manually defining any macros in the Verilog-AMS code.

5. CONCLUSIONS

The modeling methodology described in this paper enables the user to quickly and correctly create complex new models with relative ease for various SPICE-like circuit simulators. Once the model has been entered, tested and validated in Paragon, ADMS can be used in conjunction with Paragon to generate compact C code for any new circuit simulator in the future. The generated C code from ADMS is only slightly slower than hand-written code, but with further optimization this will be overcome. Also, there is an enormous decrease in the model development time. The model development time is significantly reduced because the model developer does not have to deal with the low level C code for each circuit simulator. The time required to validate and test a new model is also significantly reduced because the model developer can carry out the model debugging process at a higher level. The model can be analyzed and debugged in the Paragon environment and the generated Verilog-AMS code can be tested thoroughly before the creation of the C code for implementing the model in a target simulator. By debugging the model at a level higher than C code, more bugs are filtered out before they reach the low level C code. This greatly increases the efficiency of the model debugging process because the model developer does not have to carry out C code debugging of his model for each circuit simulator in which he intends to implement the model. The higher-level XML description of the model is easy to maintain, reuse and update and this leads to an increased efficiency in the overall model creation process.

6. REFERENCES

[1] BSIM3SOI Source code and Documentation, http://www-device.eecs.berkeley.edu/~bsimsoi/

[2] Spectre Circuit Simulator, http://www.cadence.com/products/custom_ic/spectre/

[3] 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.

[4] BSIM3 Homepage, http://www-device.eecs.berkeley.edu/~bsim3/intro.html

[5] Francis, M.; Chaudhary, V.; Mantooth, H.A.; Compact modeling of semiconductor devices using higher level methods
IEEE 2004 International Symposium on Circuits and Systems, 29 May-1 June 4 2004

[6] ADMS Model Compiler, http://sourceforge.net/projects/mot-adms

[7] L. Lemaitre, C. McAndrew, S. Hamm, ADMS – Automatic Device Model Sythesizer, Proc. IEEE Custom Int. Circ. Conf., pp. 27-30, 2002.

[8] R. V. H. Booth, An extensible compact model description language and compiler, Proc. IEEE BMAS, pp. 39-44, Oct. 2001.

[9] M. Zorzi, N. Speciale, G. Masetti, Automatic embedding of a ferroelectric capacitor model in Eldo, Proc. IEEE BMAS, pp. 97-101, Oct. 2001.

[10] B. Wan, B. P. Hu, L. Zhou, C.-J. Shi, MCAST – An abstract-syntax-tree based model compiler for circuit simulation, IEEE Custom Integrated Circuits Conf. (CICC), pp. 249-252, Sept. 2003.

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

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

[13] S. Liu, K.C. Hsu, P. Subramaniam, ADMIT-ADVICE Modeling Interface Tool, IEEE Custom Integrated Circuits Conference, 1988.

[14] A.T. Yang, and S.M. Kang, iSMILE: A Novel Circuit Simulation Program with emphasis on New Device Model Development, 26th Design Automation Conference, 1989.

[15] MAST/Saber User Manual, Synopsys, Inc.

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

[17] fREEDA Circuit Simulator, http://guppie.egrc.ncsu.edu/freeda

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

[19] Extensible Stylesheet Language, http://www.w3.org/TR/xslt

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

[21] 1076.1-1999 IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual, IEEE Press, ISBN 0-7381-1640-8.

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

[23] Santi E., Dougal, R.A., Monti, A. The VTB Environment for Virtual Prototyping of Dynamic Ship Systems. American Society of Naval Engineers Annual Meeting, Arlington, VA, March 24, 2003

[24] Mica Device Programming Interface, Documentation and Programmer's guide, Motorola Internal Document, 1998

[25] HSIM User's Guide, NASSDA Corportaion, 2001

[26] Zspice Circuit Simulator, http://mot-zspice.sourceforge.net/

[27] Cadence Design System, http://www.cadence.com/