

# Standardization of Compact Device Modeling in High Level Description Language

L. Lemaitre<sup>\*</sup>, C. McAndrew<sup>\*\*</sup> and W. Grabinski<sup>\*</sup>

<sup>\*</sup>Motorola – Geneva – Switzerland

<sup>\*\*</sup>Motorola, Tempe, AZ, USA

[laurent.lemaitre@motorola.com](mailto:laurent.lemaitre@motorola.com)

[colin.mcandrew@motorola.com](mailto:colin.mcandrew@motorola.com)

[w.grabinski@motorola.com](mailto:w.grabinski@motorola.com)

## ABSTRACT

This paper proposes a methodology based on hardware description languages (HDL) to efficiently develop compact device models. After an introduction to compact device modeling we describe Verilog-AMS, a popular HDL. Then we show how Verilog-AMS syntax can be used to fully encode compact device models. We conclude with some results obtained by following the presented methodology and by using a model builder called ADMS.

**Keywords:** spice, HDL, compact, device, modeling, adms.

## 1 COMPACT DEVICE MODELING

Compact device modeling is the art of building physics based equations of micro-electronic circuit devices. Many independent compact modeling teams are developing, improving and maintaining different models: mosfets, bjts, diodes, varactors, RAM memory cells, and power devices just to list a few.

Compact models are developed, encoded and implemented in very different ways. Each modeling groups uses its own tools and techniques for these tasks, which are in many cases strongly dependent on a selected target simulation tool. Generally speaking compact device modeling suffers from a lack of well-established, standardized methodology. Main reasons of this lack of methodology result from the way people are doing device modeling.

Tasks involved in compact device modeling can be summarized as follows:

- 1- build physics based constitutive equations
- 2- encode constitutive equations in computer language
- 3- implement the code into electrical simulators
- 4- validate compact device model implementation

There is no standardized language to formulate the model equations at step 1. Most developers write first drafts of their device models in interpreted languages. Mostly used interpreted languages are those offered by commercial tools such as Matlab<sup>TM</sup>, Mathematica<sup>TM</sup> and Mapple<sup>TM</sup>. Sometimes developers rely on the use of open-source interpreters like Perl. Model developers have easy access to numerical evaluators that give them a good guess on how well the model behaves numerically. However this approach provides no easy way to share the code between

different models. In this paper we propose to base the development of device modeling on the use of hardware description languages (HDL). More specifically we demonstrate how Verilog-AMS [1][2] can help to do device modeling. Most electrical circuit simulators support Verilog-AMS. Model developers can evaluate the validity and robustness of their code by running any electrical circuit simulator.

Note that the same work could be repeated by using other HDL such as VHDL-AMS [3][4].

Step 2 requires more elaborated work. Model equations formulated in step 1 have to be converted in computer language. Partial derivative of equations must be calculated. This is the prerequisite to insert compact device model into electrical simulators. Electrical circuit simulator vendors provide device model interfaces through which new device models can be imported as simulator built-in component. The way constitutive equations of the model are encoded into computer language strongly depends on the way the device model interface of the targeted electrical circuit simulator is implemented.

Step 3 involves the contribution of electrical model programmers. Programmers read the definition of a device model and try to adapt it to the targeted simulator, according to the rules defined by the interface of the simulator. Simulator device model interfaces have different flavors from one vendor to another. This makes difficult the elaboration of standardized template at this level of programming. BSIM3 [5] device developers chose the device model interface of open-source simulator Spice3 as standardized template to disseminate the code of their model. Spice3 model interface facilitates model distribution only to simulation tools based on Spice3 interface. Even for this type of simulators hand-coded operations are necessary to adapt Spice3-based coding to other interfaces. There is no guaranty that the same model will have the same implementation in different simulators. HDL-based definition of device model can help here. If the semantics of the HDL is well defined it is then easy to build a model compiler or model builder that converts any set of HDL equations into ready-to-compile code for specific simulator interface. Such a model builder has been presented in [6].

Step 4 is the procedure to follow in order to qualify model implementation. No well-established procedure has been defined so far. This paper introduces a new paradigm for validating and benchmarking device model implementation.

In the following sections we propose a guideline that makes easier and more efficient the building of new compact device models. All the discussion is based on the use of Verilog-AMS. But all statements remain valid for other HDL.

## 2 DEVICE MODELING TEMPLATE

This section discusses in detail a typical template that can be used to implement new device models in Verilog-AMS.

### 2.1 Few words on Verilog-AMS

The Verilog-AMS Hardware Description Language defines a behavioral language for analog circuit simulators. It gives analog designers a means to encapsulate behavioral description of analog systems into modules. The language has been derived from the IEEE 1364 Verilog HDL specification. Only a subset of Verilog-AMS is used to build the compact device-modeling template presented below. Few extensions of the language are needed. We will introduce the extensions by using the (\*...\*) construct.

### 2.2 Device Modeling Template

Figure 1 gives the proposed compact device module template. Below we detail the contents of each item of the template. We give small examples that implement each items using Verilog-AMS. Verilog-AMS keywords are stressed in bold letters.

### 2.3 Preprocessor Declaration - Comments

This part helps to make easier the code readability. It consists of lines like:

```
/* this is a multi-line comment */
`ifdef USE_PARASITIC_EFFECTS
... code ...
`endif
`define ECHARGE (1.6021918e-19) // this is an in-line comment
`define CtoK (273.15)
`define BOLTZMAN (1.3806226e-23)
`define ALIAS `BOLTZMAN
```

Note that preprocessing declarations and comments can occur at any place in the template.

### 2.4 Module Declaration

The declaration of a device module follows very simple rules. Two main items have to be provided - the name of the module and the list its terminals. For instance:

```
module myC ( p, n) (* author="unknown" proprietary="Motorola"
info="test" version="0.0.0" spice:id="c" spice:level="1" *);
```

### 2.5 Topology Declaration

This sub-section illustrates how the topology of a two-terminals compact device is declared. In this example two

```
Preprocessor Declaration - Comments
Module Declaration
Topology Declaration
  Node topology
  Branch topology
Parameter/Variable Declaration
  Model Parameters
  Instance Parameters
  Global Variables
Constitutive Equations
  Local Variable Declaration
  Model Temperature Setting
  Instance Temperature Setting
  Evaluation (current, voltage, charges)
    Linear Static Evaluation
    Non-Linear Static Evaluation
    Linear Dynamic Evaluation
    Non-Linear Dynamic Evaluation
    Noise Evaluation
  Loading (residuals + Jacobian terms)
    Linear Static Loading
    Non-Linear Static Loading
    Linear Dynamic Loading
    Non-Linear Dynamic Loading
    Noise Loading
Finalization
User-defined Validation Modules
```

Figure 1: Compact device modeling template

internal nodes are declared. Some properties are attached to each terminal. Three branches are declared. The only purpose of branch declaration is to improve the coding style.

```
inout p, n;
electrical p (* info="P terminal" flag="ASK" *);
electrical n (* info="N terminal" flag="ASK" *);
electrical rp (* info="internal P node" *);
electrical rn (* info="internal N node" *);
branch (p,rp) Rp;
branch (rp,n) C;
branch (rn,n) Rn;
```

### 2.6 Parameter/Variable Declaration

In this sub-section parameter and global variable declarations are illustrated. Below two model parameters and two instance parameters are defined:

```
parameter real rmod=1.0 (* type="model" info="resist/m2n *);
parameter real cmod=1u (* type="model" info="capa/ m2n *);
parameter real w=1u
(* type="instance" info="width " unit="m" *) from (0.1u:+inf);
parameter real l=1u
(* type="instance" info="length" unit="m" *) from (0.1u:+inf);
parameter real rc=1u
(* type=" instance" info="contact geometry" unit="m*m" *);
```

Extra properties are attached to parameter l by using the (\* ... \*) construct. Each **from ...** constructs defines a range of allowed values. Verilog-AMS syntax offers a good means to provide compact easy-to-read parameter data.

Global variables are declared as follows:

```
real Ceff (* info="effective capacitance" spectre:unit="f" *);
real powerR (* info="power dissipation - resistances" *);
```

Global variable values as opposed to local variable values are saved after each simulation. Global variables values can be printed/plotted by circuit simulators.

## 2.7 Constitutive Equations

This section contains the physic based or constitutive equations of the device. This section is made of series of “@” constructs:

```
@{"property1","property2",...} begin
... C language like assignments ...
OR
... C language like conditionals ...
OR
... other begin...end blocks
end
```

Special syntax @{...} assigns properties to **begin..end** blocks of code. Note that in all what follows the use of @{...} is recommended by not mandatory. The use of this syntax may help a device builder to generate more efficient simulator code.

Local variable declarations can occur at any **begin...end** blocks.

Model temperature setting and instance temperature setting are made of blocks with property values @{"model","init"} and @{"instance","init"}. Code that initializes/checks/sets device parameters should occur in these blocks. Here is an example that combines model temperature setting and instance temperature setting:

```
@{"initial","model"} begin
rmod_t = rmod * ($temperature)/TNOM;
end
@{"initial","instance"} begin
Ceff = w*I*cmod;
end
```

After parameter setting electrical quantities that depend on current flows or node voltages are evaluated and loaded into the different branches of the device model. We propose below an extension to Verilog-AMS that gives a means to better modularized the way quantities are evaluated and loaded. More specifically evaluation code and loading code of a pure linear model look like as follows:

```
@{"evaluate","static","linear"} begin
Irp = rmod_t * (rc) * V(Rp);
Irn = rmod_t * (rc) * V(Rn);
end
@{"evaluate","dynamic","linear"} begin
Q = Ceff * V(C);
end
@{"load","linear"} begin
I(Rp) <+ Irp;
I(Rn) <+ rn;
end
@{"load","dynamic","linear"} begin
I(Rp) <+ ddt(Q);
end
```

Special symbol “<+” is used to store its right-hand side term to the branch specified at its left-hand side.

Note that all @{...} constructs are used by model builders to improve C code synthesis. They add extra data to **begin...end** blocks. Verilog-AMS source code remains valid even if all @{} constructs are omitted.

## 2.8 Finalization

Verilog-AMS code stored at the finalization section is executed after each simulation step. The goal of this section is to avoid useless computation of pieces of code at each Newton-Raphson iterations. Finalization code is executed after each successful simulation steps. For instance the code that calculates power dissipation can be moved to this section:

```
@{"final"} begin
powerR =V(Rn) * Irn+V(Rp) * Irp;
end
```

## 2.9 Full code

Figure 2 shows full code of a simplified capacitance model. It illustrates how modularized compact device modeling can be done in HDL. Note that current Verilog-AMS simulators do not support the (\*...\*) extensions.

## 2.10 User-defined Validation Modules

Netlists can be defined using Verilog-AMS. We propose here to describe some basic validation modules that can be used to validate the implementation of a compact device into a given simulator.

The model builder translates validation modules into simulator-specific netlists.

## 3 SOME RESULTS

This section gives some results that have been achieved by applying above methodology.

Results are summarized in the snap shot of figure 4. First table of the snap shot gives a list of binary files created from MOSCAP [7] Verilog-AMS source code for different operating systems and different simulators.

Each binary file results from the compilation of c code derived from the same Verilog-AMS source code. ADMS model builder generated the C code. This flow guarantees consistency of the implementation of MOSCAP between different simulators.

One change at the model level can be immediately propagated to simulator level in few minutes.

Validation circuit netlists are proposed in a second table. ADMS model builder has created the net-lists. Any modification made at the Verilog-AMS level will be propagated to these files. These files are good starting point for new users of the compact model.

## 4 CONCLUSION

This paper presented a methodology that can greatly help the work of compact device modeling. This methodology has been successfully applied to the development of compact devices at Motorola. It can be a starting point for standardizing compact device modeling.

```

// simplified capacitor model (NOT PHYSICAL! only show-case)
// p o --- [ ] --- o --- | --- o --- [ ] --- o n
//          Rp   rp   C   rn   Rn
`define TNOM (272.15+27.0)
module myC ( p, n ) (* author="unknown" proprietary="Motorola"
  info="test" version="0.0.0" spice.id="c" spice:level="1" *);
  inout p, n;
  electrical p (* info="P terminal" ask="ASK" *);
  electrical n (* info="N terminal" ask="ASK" *);
  electrical rp (* info="internal P node" *);
  electrical rn (* info="internal N node" *);
  branch (p,rp) Rp;
  branch (rp,m) C;
  branch (rn,n) Rn;
  parameter real rmod=1.0 (* type="model" info="resist/m2" *);
  parameter real cmod=1u (* type="model" info="capa/ m2" *);
  parameter real w=1u
    (* type="instance" info="width " unit="m" *) from (0.1u:+inf);
  parameter real l=1u
    (* type="instance" info="length" unit="m" *) from (0.1u:+inf);
  parameter real rc=1u
    (* type="instance" info="contact geometry" unit="m*m" *);
  real Ceff (* info="effective capacitance" spectre:unit="F" *);
  real powerR (* info="power dissipation - resistances" *);
  analog
  begin
    real rmod_t, Q; // parameters not be saved per simulation
    @{"initial","model"} begin
      rmod_t = rmod * ($temperature)/ TNOM;
    end
    @{"initial","instance"} begin
      Ceff = w*l*cmod;
    end
    @{"evaluate", "static", "linear"} begin
      Irp = rmod_t * (rc) * V(Rp);
      Irn = rmod_t * (rc) * V(Rn);
    end
    @{"evaluate", "dynamic", "linear"} begin
      Q = Ceff * V(C);
    end
    @{"load","linear"} begin
      I(Rp) <+ Irp;
      I(Rn) <+ Irn;
    end
    @{"load", "dynamic", "linear"} begin
      I(Rp) <+ ddt(Q);
    end
    @{"final"} begin
      powerR =V(Rn) * Irn+V(Rp) * Irp;
    end
  end
endmodule

```

Figure 2: Verilog AMS code of simplified capacitor model

```

// simple qualification test – test implementation of w and l
module test_dc_instance_parameters();
  real wgiven, lgiven;
  // netlist
  myC #( .w (wgiven), .l (lgiven) ) m1 (1, 0);
  VoltageSource #( .dc (1.0) ) v1 (1, 0);
  for (wgiven=1u;wgiven<10u;wgiven=wgiven+1u) begin
    for (lgiven=1u;lgiven<10u;wgiven=lgiven+1u) begin
      @{"dc"} begin
        $assert(m1#w==wgiven);
        $assert(m1#l==lgiven);
      end
    end
  end
endmodule

```

Figure 3: User-defined qualification test in Verilog-AMS

The screenshot shows a Netscape browser window with the address bar displaying 'http://gex.sps.mot.com/~Lemaitre/MOSCAP3/'. The page content includes:

- Motorola Internal Use Only**
- MOSCAP3 model for evaluation**
- A paragraph stating: "This repository contains basic data related to the implementation of MOSCAP3 into SPICE simulators. MOSCAP3 is being developed by Colin McAndrew - Jim Victory - Laurent Lemaitre (Implementation). Contact: laurent.lemaitre. Automatically Generated: Wed Nov 20 16:46:24 2002"
- NOTE**
  - MOSCAP3 is under development
  - Current MICA implementation NOT supported by MICA team
- Binaries**
  - Tar Ball: Binary.tar.gz
  - Note: platform names are guessed by running script "config.guess" You can get a copy of "config.guess" by searching key-word "config.guess Per Botner" with search-engine google
- Platform** table:

Platform	mica120	mica130	spectre446	ads2002
hppa1.1-hp-hpux10.20	MOSCAP3.sl	MOSCAP3.sl	MOSCAP3.sp	MOSCAP3.sl
hppa2.0-hp-hpux10.20	MOSCAP3.sl	MOSCAP3.sl	MOSCAP3.sp	MOSCAP3.sl
hppa2.0w-hp-hpux11.00-64b	MOSCAP3.sl	MOSCAP3.sl	MOSCAP3.sp	MOSCAP3.sl
i686-pc-linux-gnu	MOSCAP3.sp	MOSCAP3.sp	on request	on request
sparc-sun-solaris2.6	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp
sparc-sun-solaris2.6-32b	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp
sparc-sun-solaris2.6-64b	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp	MOSCAP3.sp
- Netlists** table:

Netlist	Description	mica120	mica130	spectre446	ads2002	Contact
MOSCAP3_dc	basic dc test	on request	on request	MOSCAP3_dc.ckt	MOSCAP3_dc.ckt	L.Lemaitre
MOSCAP3_ac	basic dc test	on request	on request	on request	MOSCAP3_ac.ckt	J.Slaughter
- Installation**
  - MICA

Figure 4: Snap shot of automatically generated compact device model described in Verilog-AMS language

## REFERENCES

- [1] Open Verilog International, "Verilog-AMS, Language Reference Manual," Version 1.9, Dec. 15, 1999
- [2] <http://www.accelera.org/>
- [3] IEEE VHDL 1076.1 Language Reference Manual
- [4] <http://www.vhdl.org/analog/>
- [5] <http://www-device.eecs.berkeley.edu/~bsim3/>
- [6] L. Lemaitre, C. McAndrew, S. Hamm, "ADMS - Automatic Device Model Synthesizer," CICC2002, Orlando, May 2002
- [7] Victory, J.; McAndrew, C.C.; Gullapalli, K., "A time-dependent, surface potential based compact model for MOS capacitors," IEEE Electron Device Letters, pp. 245-247, Vol. 22, Issue: 5, May 2001