Writing your first Verilog-A compact model

Geoffrey Coram
Analog Devices
NEEDS External Advisory Group
Assumptions

You are a device engineer / researcher
You need a compact model (not TCAD)
You understand the physics of the device
You have a set of equations that describe the terminal characteristics.

You’ve never written a compact model before
Question

How can you write a model that is good enough for early stage circuit simulation...

... and one that can serve as a starting point for an industrial-strength compact model that designers can use?
Answers

You need to understand:

1) What a compact model must do
2) How a circuit simulator uses a compact model
3) How to translate device equations into Verilog-A
4) Some common mistakes to be aware of
5) How to test your model
6) What a model deployment package consists of.
Outline

1. What circuit simulators and compact models do
2. Verilog-A (vs. Matlab or C)
3. Simple examples
4. Basic language features
5. Coding guidelines
6. Common mistakes
7. Testing your model
8. The model release package
9. Summary
1. Circuit simulators and compact models

Circuit simulators (almost?) all use Modified Nodal Analysis

- Unknowns are node voltages (and certain branch currents)
- Each row in the matrix specifies KCL for a particular node (or KVL for certain elements)

Solutions are obtained by Newton’s method

- Derivatives must be smooth
- Crazy voltages are almost guaranteed

Your model should:

- Be formulated to give \(I\) and \(Q\) as functions of \(V\) \(\text{through}\) quantities as functions of \(\text{across}\) variables
- Be smooth
- Handle unexpected voltages gracefully
2. Verilog-A vs. Matlab or C

Verilog-A is a “hardware description language”

- Intended for high-level behavioral modeling
- Less focused on the math, more on the behavior (physics)
- Much of the simulator interface handled by the compiler
3. Simple examples

module simpleres(a, b);
    inout a, b;
electrical a, b;

    analog I(a,b) <+ V(a,b) / 1000;
endmodule
3. Simple examples

```
`include "disciplines.vams"

module simpleres(a, b);
  inout a, b;
  electrical a, b;
  parameter real r = 1000 from (0:inf);

  analog begin
    I(a,b) <+ V(a,b) / r;
  end
endmodule
```
3. Simple examples

HSpice:
  .hdl "simpleres.va"
  x1 top 0 simpleres r=2k
  v1 top 0 5

Spectre:
  ahdl_include "simpleres.va"
  r1 (top 0) simpleres r=2k
  v1 (top 0) vsource dc=5
4. Basic language features

Header files contain standard information

```
#include "disciplines.vams"
```

Defines "electrical" discipline and access functions $V$ and $I$
(also thermal, kinematic, ...)

```
#include "constants.vams"
```

Physical (`$P_Q$) and mathematical (`$M_PI$) constants

Please! don't `define M_PI 3.14
4. Basic language features

Module is the standard building block:

```plaintext
module mymodule(list_of_ports);
endmodule
```

Ports or terminals are the connections to the circuit:

```plaintext
inout port1, port2;
electrical port1, port2;
```

Always use `inout` for your compact models.
4. Basic language features

Powerful syntax for declaring parameters with default and range:

\[
\text{parameter real } r = 1000 \text{ from } (0:inf);
\]

Chained defaults:

\[
\begin{align*}
\text{parameter real } l &= 1u \text{ from } (0:inf); \\
\text{parameter real } w &= 1u \text{ from } (0:inf); \\
\text{parameter real } \rho &= 1 \text{ from } (0:inf); \\
\text{parameter real } r &= \rho \times l/w \text{ from } (0:inf);
\end{align*}
\]
4. Basic language features

All behavior in the \texttt{analog} block

\begin{verbatim}
analog begin
  I(a,b) <+ V(a,b) / r;
end
\end{verbatim}

Use \texttt{begin / end} for multi-line blocks – like you would use braces \{ \} in C

*** Emacs and VIM have "Verilog-mode" plug-ins that can highlight keywords and keep your indentation correct.
4. Basic language features

The contribution operator:

```
analog begin
  I(a,b) <+ V(a,b) / r;
  I(a,b) <+ white_noise(4*P_K *temperature, "thermal");
end
```

Not quite an assignment – all contributions to $I$ are summed.
3. Simple examples

```vhdl
`include "disciplines.vams"

module simplecap(a, b);
    inout a, b;
    electrical a, b;
    parameter real c = 1000 from [0:inf);
    (* desc = "charge" *) real q;

    analog begin
        q = c * V(a,b);
        I(a,b) <+ ddt(q);
    end
endmodule
```

Always use ddt( charge )
not C*ddt(V)
4. Basic language features

Where to go next?


• Designer's Guide to Verilog-AMS by Ken Kundert

• Existing models (mostly open-source) Mextram, Hicum, BSIMSOI, PSP, MOS20, ...
5. Coding guidelines

• Write legibly
  – indent blocks consistently
  – align equations vertically on = or <+
  – use spaces, not TAB (ts=4 or 8?)
  – use comments to document equations
  – use meaningful variable names
    (not T0, T1, ... and Vd, Vs not Vx, Vy)
  ➢ can’t hide behind compiled code
5. Coding guidelines

Can anyone besides you read your code?

\[ Rd = \frac{Rc}{W} \left(1 + rv \times \max((V_{gg} - V_{t0}), 0)\right) + \frac{R_{sh} \times L_{d}}{W} \frac{1 + \lambda \times (d_{vd} - d_{vg})}{L_{d}} \times \left(1 + zeta \times I_{(b,a)} \times \frac{V_{d}}{300}\right)^{2.3} \times \left(1 + \left(\frac{(d_{vd} - d_{vg} + 1e-10)}{L_{d} \times E_{cd}}\right)^{\theta}\right) \times \left(1.0 / \theta\right); \]

Self-heating
5. Coding guidelines

// MOS Model 11 Verilog-A implementation
// Level 11010 "physical scaling"
// Based on the report "NL-UR2002/802: MOS Model 11 Level 1101"
// by R. van Langevelde, A.J. Scholten, and D.B.M Klaassen
// Copyright Koninklijke Philips Electronics N.V. 2003/2004

... Ref to docs

// Calculation of Conductance Parameters
// (5.17) .. (5.19)
theth = thethr * (1.0 + We_inv * swtheth) * exp(lnLe*thethe
ssf = ssfr * (1.0 + We_inv * swssf) * (1.0 + Le_inv * slssf
alp = alpr * (1.0 + We_inv * swalp)
     * (1.0 + slalp * (exp(lnLe*alpexp) - 1.0));
5. Coding guidelines

Mix of TABs and spaces

No range on phib
5. Coding guidelines

```java
analog begin
  if (SHAPE==1)
    begin
      surface=a*b;  //SQUARE
    end
  else if (SHAPE==2)
    begin
      surface=`M_PI*a*b/4.0;  //ELLIPSE
    end
  else
    begin
      surface=`M_PI*r*r;  //ROUND
    end
Vc=V(T2,T1);
Vb=V(T1,T2);
```

Inconsistent indentation
5. Coding guidelines

```verbatim
analog begin
    if (SHAPE==1) begin
        surface=a*b;     //SQUARE
    end else if (SHAPE==2) begin
        surface=`M_PI*a*b/4.0;  //ELLIPSE
    end else begin
        surface=`M_PI*r*r;    //ROUND
    end

Vc=V(T2,T1);
Vb=V(T1,T2);

//initial conditions
@(initial_step) begin
   El 8800.53 (P1)

Better!
```

Better!
5. Coding guidelines

Mix of TABs and spaces

No range on phib
5. Coding guidelines

V(b) – V(si) = V(b,si)

Best to use branch voltages!

No range on phib
5. Coding guidelines

• Use SI units: meters, Amps, etc.
  – One NEEDS model used cm
  – Design environment (schematic capture) uses m

• Don't mix Case of Parameters
  – Some tools are case-sensitive, some are not; can cause difficulties moving from extraction to simulation tools
5. Coding guidelines

• Many references:
  – Coram BMAS 2004
  – Coram/McAndrew CMRF 2005
  – Coram MOS-AK 2006
  – Mierzwinski et al. MOS-AK 2008
  – Mierzwinski et al. MOS-AK 2009
  – NEEDS talk on nanoHUB
6. Common mistakes

- Verilog-AMS uses $\ln()$ instead of $\log()$
- $\frac{1}{2} = 0$ (integer division)
- Use `begin/end` to delimit all if blocks

```verilog
if (rdeff < 0) begin
    $debug("fixing rdeff");
    rdeff = 0;
end  // OOPS!
```
6. Common mistakes

A word about tolerances:

- SPICE-like simulators try to satisfy KCL: $\sum i = 0$
- Double-precision arithmetic: you never get 0
  $((1A - 1pA) - 1A) + 1pA = 1pA$
- But that's OK: $\text{abstol} = 1pA$, $\text{reltol} = 0.001$

$| \Sigma i | < \text{abstol} + \text{reltol} \times | \text{max } i |$
6. Common mistakes

A word about tolerances:

• HOWEVER, if your "flow" variables are much smaller or much larger than "typical" currents, convergence will be bad
7. Testing your model

• Of course you will test your model over the measurement range
  – To show fit to measurements

• You should also test OUTSIDE the measurements:
  – How does the model behave at high voltages?
  – Extreme temperatures?
  – What happens for Vds < 0 or other symmetries?
7. Testing your model

• Consider unexpected parameter values
  – Some extraction tools may pick random values
  – Set ranges to prevent mathematical errors
  – Consider warnings for unexpected values

• N and P type devices (if applicable)
7. Testing your model

• Run DC, AC, TRAN analyses
  – AC currents can expose derivatives

• Run in multiple simulators
  – Each has its own quirks
  – Spectre's PSS particularly good at catching "hidden state"
8. The model release package

- Not just the source code!
- Documentation – of the model equations and parameter extraction, if possible
- Sample parameter set and netlist

- BE SURE TO INCLUDE LICENSE TERMS! and copyright notice
8. The model release package

Abstract

The MIT Virtual Source (MVS) model is a semi-empirical compact model for nanoscale transistors that accurately describes the physics of quasi-ballistic transistors with only a few physical parameters.

Model Release Components:

- MVS Model 1.0.1 in Verilog-A
- MVS Model Circuit Simulation Benchmarks
- MVS Model Exerciser in MATLAB
- MVS Model Parameter Extractor in MATLAB
- MVS Model Manual
- Experimental Data from Intel 32 nm and 45 nm N-type devices
- Details of changes in this version
  
  The model release components are licensed under a modified CMC license.

- Licensing
9. Summary

• Verilog-A is a useful language
  – Parameter extraction as well as circuit simulation

• Verilog-A is easy to learn
  – Much easier than C/C++ interfaces to simulators

• Many models available
  – Examples to follow
  – Don't reinvent the junction diode
9. Summary

• Your code should be able to stand alone
  – New students picking up your research
  – Industrial users who might not read your paper

• Be proud of your code!
Further Work

• Establish requirements for posting on nanoHUB
• Web-based model exerciser
• Definition of "NEEDS-Certified" compact models
NEEDS-Basic

• Basic level of quality for posting to nanoHUB
  – No syntax errors
  – Clean formatting: no TABs, block indentation
  – No "improper" functions:
    • absdelay, transition, analysis, initial_step
    • $system_functions (except $temperature, $limit)
Bronze, Silver, Gold

• Additional levels of code checking
  – Proper variable initialization (no hidden state)
  – No unused variables / orphan code
  – Documentation in code

• Netlists for commercial simulators
  – With reference results

• Parameters and operating-point variables declared with units and description

• What else?