Practical Considerations for Developing, Debugging, and Releasing Verilog-A Models

Marek Mierzwinski, Patrick O'Halloran, and Boris Troyanovsky

Tiburon Design Automation
Santa Rosa, CA

MOS-AK /GSA Workshop
December, 2009 Baltimore
Outline

- Motivation
- Implementation issues
  - Primer on circuit simulation
  - Performance
  - Coding
- Debugging
- Model distribution
- Future directions / Conclusions
Why Verilog-A?

- Natural language for analog model development
- Succinct
  - derivatives, loads all handled by compiler
  - simple parameter support
- Active standard
Verilog-A: Analog Example

- **Basic diode**
  - Spice c-code would be ~10k lines

```verilog
// Very simple diode
#include "disciplines.vams"
#include "constants.vams"
module diode_simple(a,c);
    inout a,c;
    electrical a,c,m;
    parameter real IS = 1e-14 from [0:inf);
    parameter real RS = 1.0 from [0:inf);
    parameter real AF = 1.0 from [0:inf);
    parameter real KF = 0.0 from [0:inf);
    branch {a,m} diode;
    branch {m,c} resistor;
    real Id;
    analog begin
        Id = IS*(limexp(V(diode)/$vt)-1);
        I(resistor) <+ V(resistor) / RS;
        I(diode) <+ Id +
            white_noise(2 * 'P_0 * Id, "thermal") +
            flicker_noise(KF * pow(Id, AF), 1.0, "flicker");
    end
endmodule
```

- Compiler directives
- Module and port definitions
- Parameter definitions
- Analog behavior
Overcoming the Barriers

• Performance
  – No theoretical reason for Verilog-A to be inferior in performance to built-ins

• Availability
  – Verilog-A now supported by virtually all major commercial vendors
  – Support for all analysis types, e.g. transient, harmonic balance, shooting, nonlinear noise

• End user
  – Their experience must be as good as or better than using existing model distribution method
University/Industry Models

• Most new compact transistor models are being developed/released in Verilog-A:
Outline

• Motivation

• Implementation issues
  – Primer on circuit simulation
  – Performance
  – Coding

• Debugging

• Model distribution

• Future directions / Conclusions
Performance

- Model coding can have a big influence
  - execution speed
  - memory use
- Choice of particular constructs can result in performance degradation
- Models can be unintentionally analysis-dependent
Nodal Analysis

\[ f(x(t)) + \frac{d}{dt}(q(x(t))) = u(t) \]

- \( f \Rightarrow \) resistive
- \( q \Rightarrow \) reactive (inductors, capacitors)
- \( u \Rightarrow \) current sources

For a hypothetical circuit with current sources, resistors, capacitors:

\( x(t) \) is a vector of voltages, all the equations are standard KCL, and so PURE NODAL ANALYSIS.
Contributions

Current contributions

\[ I(a, b) <+ \ldots; \]

use existing state variables. No increase in matrix size
However, if we have a voltage source

\[ V(a, b) \leftrightarrow K; \]

this necessitates adding an extra state variable "I" (the \textit{flow} through the source) into the \( x \) vector, with the corresponding extra equation branch:

\[ x_a - x_b = K \]

( or in reality \(-x_a + x_b + K = 0\) )
Contributions

• Try to formulate contributions as current

• A nonlinear capacitance, \( f(V) \), \( g(I) \) could be implemented as:

\[
V(a, b) \leftarrow g(I(a, b));
\]

To avoid the extra state variable, use

\[
I(a, b) \leftarrow f(V(a, b));
\]
Parasitic Resistors

- However, due to convergence considerations, voltage contributions are better when very small resistances are possible:
  \[ V(a,b) \leq I(a,b) \times R_{ab}; \]

- Alternatively, consider using a range on the parameter
  parameter real \( R_{ab} = 50.0 \) from \([100m:\text{inf})\);
Inductances in Verilog-A will add an additional state variable:

\[ V(a, b) \leftarrow L \times ddt(I(a,b)) \; ; \]

Since this translates to

\[ -x_a + x_b + ddt(L \times I_{ab}) == 0 \]

where \(I_{ab}\) is the flow through the inductor

- Truly voltage controlled elements should be implemented with voltage contributions.
• When variables that depend on \texttt{ddt()} are used in conditionals, the compiler must create extra branch equations.

```plaintext
if (Vds < 0.0) 
  Mode = -1; // Inverse mode 
else 
  Mode = 1;

Qbd_ddt = ddt(Qbd); 
Qbs_ddt = ddt(Qbs);

if (Mode == 1) begin 
  t0 = TYPE * Ibd + Qbd_ddt; 
  t1 = TYPE * Ibs + Qbs_ddt; 
end 
else begin 
  t1 = TYPE * Ibd + Qbd_ddt; 
  t0 = TYPE * Ibs + Qbs_ddt; 
end 
I(b,di) <+ t0;
I(b,si) <+ t1;
```

Typical MOSFET code
Branches from Conditionals

• Avoiding unwanted branches
  – Place the arguments to `ddt()` in the conditionals

Improved MOSFET code

```plaintext
if (Mode == 1) begin
  t0 = TYPE * Ib0;
  arg0 = Qbd;
  t1 = TYPE * Ibs;
  arg1 = Qbs;
end
else begin
  t1 = TYPE * Ib0;
  arg1 = Qbd;
  t0 = TYPE * Ibs;
  arg0 = Qbs;
end
I(b,di) <+ t0 + ddt(arg0);
I(b,si) <+ t1 + ddt(arg1);
```
Branch-ddt Equations

• Branch-ddt equations are state variables related to implementing $\text{ddt}(\cdot)$ equations with product terms:
  \[
  x = V(a, b) ; \\
  I(a, b) \leftarrow x \ast \text{ddt}(x) ;
  \]

• How they arise:
  From the basic nodal KCL
  \[
  f(x(t)) + \text{ddt}(q(x(t))) == u(t)
  \]
  Note that it does not support terms of the form
  \[
  g(x(t)) \ast \text{ddt}(h(x(t)))
  \]
The Verilog-A code

\[ x = V(a, b); \]
\[ I(a, b) \leftarrow x \ast \text{ddt}(x); \]

introduces extra state variable \( \phi \),
\[ \phi - \text{ddt}(x) = 0 \]
to effectively contribute
\[ x\ast\phi \]
which fits into the \( f(x(t)) + \text{ddt}(q(x(t))) \) form

- Avoid if not really needed
Collapsible Nodes

- Native models can remove or collapse unneeded nodes
- Common “idiom” for collapsible nodes:

```c
if (Rein > 0.0)
  I(bi,si) += V(bi,si) / Rein;
else
  V(bi,si) += 0.0;
if (Rgd > 0.0)
  I(gi,di) += V(gi,di) / Rgd;
else
  V(gi,di) += 0.0;
if (Ri > 0.0)
  I(gi,si) += V(gi,si) / Ri;
else
  V(gi,si) += 0.0;
```

- Implementations may treat as
  - Collapsible node, or
  - Switch branch
Performance Impact

• Be aware of what causes extra equations
  – Voltage contributions on the left-hand-side, or
  – Current access on the right-hand side
  – Consider alternate formulations
  – Consider simplified, approximate expressions

• Collapse nodes when possible
Outline

• Motivation

• Implementation issues
  – Primer on circuit simulation
  – Performance
  – Coding

• Debugging

• Model distribution

• Future directions / Conclusions
Common mistake when probing port current:

\[ \texttt{$\text{strobe(I(port\_name))}$}; \]

- **Introduces unnamed branch**
  - Effectively shorts `port_name` to ground
  - Adds additional state variable
- **Instead use**
  \[ \texttt{$\text{strobe(I(<port\_name>))}$}; \]
- **Watch for compile-time warnings**
Consider:

\[(10) \quad x = \frac{V(a, b)}{R};\]
\[(11) \quad \text{if}(\text{type} == 1)\]
\[(12) \quad \quad \quad x = \frac{V(a, b)}{R_1};\]
\[(13) \quad \text{else}\]
\[(14) \quad \quad \quad x = \frac{V(a, b)}{R_2};\]

**Diagnostic message from compiler:**

Warning: Assignment to ‘x’ may be superfluous.
[ filename.va, line 10 ]
Noise

- Verilog-A has a very simple way to contribute noise power to a branch
- Either voltage or current contribution selects whether noise is implement as voltage or current source
- Be sure to be consistent with other contributions, otherwise you will inadvertently have a switch branch and the last contribution will win.

```verilog
I(di,si) <+ ddt(Cds * Vds);
...
I(di,si) <+ flicker_noise(Kf * pow(Ids, Af), 1.0, "flicker");
```
Parameter Case

- Verilog-A is case sensitive
- Some simulators are case sensitive, others are not
  - Many issue warnings but these are often ignored
- Provide aliases

  aliasparam AREA=Area;
Memory States

• Variables are initialized to zero on first call to module

• The simulator retains the value between calls to module
  – If used in assignment before it is assigned, it will have the value of the previous iteration

• Also known as hidden states

• Compact models should not use them
  – could cause unexpected behavior
Multiplicity

- This feature is automatically managed by the simulator
- Early versions of some simulators used proprietary methods
  - Avoid this complication
- Verilog-A does provide a mechanism to access the multiplicity value
  - $mfactor
  - Strongly encouraged to avoid this
Limiting

• As in c-coded models, the model developer has to provide a means to limit the size of changes between iterations

• Verilog-A has several ways
  – Simple limiting exponential function
  – Links in to simulator’s limiting code
  – Links in to user’s custom limiting code
Portability Restrictions

- Certain language constructs are not supported by RF analyses (e.g., Harmonic Balance, Shooting, Envelope)

- Explicit use of $\text{abstime} – usually returns 0

- Analog Operators
  - Allowed:
    - Differentiation $\text{ddt}()$, $\text{ddx}()$
    - Delay $\text{absdelay}()$
    - Laplace $\text{laplace}()$
    - Integration $\text{idt}()$ without initial conditions
  - Others are:
    - Not safe for RF analysis
    - Not (typically) useful for compact modeling
Outline

• Motivation

• Implementation issues
  – Primer on circuit simulation
  – Performance
  – Coding

• Debugging

• Model distribution

• Future directions / Conclusions
Debugging

• Basic
  – `$strobe` – outputs every converged iteration
  – `$debug` – outputs every call to module
  – Use macros to disable in general use
    `ifdef DEBUG

• Compiler flags for runtime
  – Too expensive for production code
  – Very useful during development phase

• Compile time diagnostics

• Interactive debugging
Numerical Issues

- The developer must prevent floating point exceptions, overflows, underflows, etc.
  - Simple parameter range controls make it easy to prevent garbage in:

```
parameter real Length = 1.0u from (0:100u);
```

Define a parameter ‘Length’ whose default value is 1.0e-6 and can be infinitely small, but not zero, and can be as large as 100e-6.

- Typically, if simulator has to monitor for these, it can slow down execution.
Compiler Flag Example

1. Compile with flag

$ vacomp -f angelov.cml

2. Simulate

$ spice3 -r raw -b dc.spice

3. Simulator runs until floating point exception occurs

A division by zero has occurred  
[ instance 'x1' of module 'angelov_va', line 287 ]. The Verilog-A Device 'x1' has encountered an error. The simulation must exit.

4. Debug

```
284 psi_4 = P40 + P41 * Vgdc - P111 * Vds;
285 tanh4 = 1 + tanh(psi_4);
286 Rc1 = Rcmin + Rc_T / tanh_psi;
288 case(Capmod)  
289 0: begin // Linear capacitance
290     etc...  
```
Compiler Diagnostics

```
$ vacomp -Xinfo angelov.va
Warning: Variable 'P_avg' in module 'angelov_va' is never set.

=== Summary information for module 'angelov_va':

Branch information:
- ['b1, g1'] : Current Branch (implicit)
- ['b1, s1'] : Switch Branch
- ['d1, d1'] : Current Branch (implicit)
- ['d1, d1'] : Switch Branch
- ['d1, r0'] : Current Branch (implicit)
- ['d0, s0'] : Current Branch (implicit)
- ['g0, d0'] : Current Branch (implicit)
- ['g1, g1'] : Switch Branch
- ['g1, d1'] : Current Branch (implicit)
- ['g1, g1'] : Switch Branch
- ['s1, g0'] : Current Branch (implicit)
- ['s1, g1'] : Current Branch (implicit)
- ['s1, v0'] : Voltage Branch
- ['s1, v0'] : Switch Branch
- ['s1, <gnd>'] : Current Branch (implicit)
- ['xt1, <gnd>'] : Current Branch (implicit)
- ['xt1, xt2'] : Voltage Branch
- ['xt2, <gnd>'] : Current Branch (implicit)

Branch ddt operators:
- Line 0:50, col 109
- Line 0:62, col 119
- Line 0:71, col 128
- Line 0:80, col 137
- Line 0:90, col 147
- Line 1:20, col 18
- Line 1:27, col 40
- Line 1:30, col 60
- Line 1:31, col 80

Potential memory states:
- 'Q_gd'
- 'Q_sg'

=== End of summary information for module 'angelov_va':
```
$ vacomp -Xinfo angelov.va

Warning: Variable 'P_avg' in module 'angelov_va' is never set.

=== Summary information for module 'angelov_va':

Branch information:

- unnamed(bi, gi): Current Branch (implicit)
- unnamed(bi, si): Switch Branch
- unnamed(di, d): Switch Branch
- unnamed(di, rf): Current Branch (implicit)
- unnamed(di, si): Current Branch (implicit)
- unnamed(g, di): Current Branch (implicit)
- unnamed(g, gi): Switch Branch
- unnamed(g, rf): Current Branch (implicit)

Unused variable

parameter real Tnom = 'NOT_GIVEN' from ('P_CELSIUS0':inf); // parameter

electrical d, g, s, di, gi, si, gd, sdi, gdi, gsi, bi, rf, t, xt1, xt2;

real alpha, P_avg;

real Vgs, Vgd, Vds, Vdg;

real Igs, Igd;

Branch ddt operators:

Potential memory states:

-Q_dg
-Q_gs

=== End of summary information for module 'angelov_va':
Compiler Diagnostics

Are branches and types what you thought?

```
$ vacomp -Xinfo angelov.va
Warning: Variable 'P_avg' in module 'angelov_va' is never set.

=== Summary information for module 'angelov_va':

Branch information:
- unnamed(b1, g1): Current Branch (implicit)
- unnamed(b1, s1): Switch Branch
- unnamed(d1, d): Switch Branch
- unnamed(d1, rf): Current Branch (implicit)
- unnamed(d1, si): Current Branch (implicit)
- unnamed(g1, di): Current Branch (implicit)
- unnamed(g1, g1): Switch Branch
- unnamed(g1, di, d1): Current Branch (implicit)
- unnamed(g1, d1): Current Branch (implicit)
- unnamed(g1, gdi): Switch Branch
- unnamed(g1, si): Current Branch (implicit)
- unnamed(gsi, si): Current Branch (implicit)
- unnamed(rf, si): Voltage Branch
- unnamed(s1, s1): Switch Branch
- unnamed(t, <gnd>): Current Branch (implicit)
- unnamed(x1, <gnd>): Current Branch (implicit)
- unnamed(x1, xt2): Voltage Branch
- unnamed(xt2, <gnd>): Current Branch (implicit)

Branch ddt operators:
- line 0056, col 056
- line 0067, col 067
- line 0071, col 071
- line 0046, col 046
- line 0088, col 088
- line 0096, col 096
- line 0124, col 124
- line 0127, col 127
- line 001, col 01

Potential memory states:
- 'q_gd'
- 'q_gsi'

=== End of summary information for module 'angelov_va':
```
Can branch ddt's be replaced?
Compiler Diagnostics

Check that memory states are necessary
Interactive Debugging

- Allows quick iterative investigation of module
Outline

- Motivation
- Implementation issues
  - Primer on circuit simulation
  - Performance
  - Coding
- Debugging
- Model distribution
- Future directions / Conclusions
Model Distribution

• Complete model support requires
  – model version control
  – schematic capture information
  – simulator dependent

• End-user experience
  – easy installation
  – look and feel of native device
    • instance/modelcard
    • multiplicity
IP Protection

• Compiled libraries effectively hides source code as well as built-in models

• Model parameters can be ‘hidden’ in source code by assigning them as default values
Example ADS

- Design Kits provide a convenient mechanism for distributing complete model package
- End user opens a zipped file
Example

Users have a one-step process to install model
Example

Users see no difference when using Verilog-A implemented models.
Outline

• Motivation

• Implementation issues
  – Primer on circuit simulation
  – Performance
  – Coding

• Debugging

• Model distribution

• Future directions / Conclusions
Future Directions

• Tools for improved model development
  – Automatic checking of smoothness, continuity, etc.
  – Automated checks for passivity / stability / etc. where appropriate
Conclusions

• Continued adoption of Verilog-A presents numerous benefits for
  – Compact model developers
  – Circuit designers
  – Tool vendors

• Benefits include
  – Portable, robust compact models
  – Fast model distribution and modification

• But developers must be aware of performance and distribution issues