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**

```
// 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_Q * 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

$$f(x(t)) + ddt(q(x(t))) = u(t)$$

f => resistive
q => reactive (inductors, capacitors)
u => 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) <+ …;
```

use existing state variables. No increase in matrix size

Voltage Sources in Verilog-A

However, if we have a voltage source

```
V(a, b) <+ K;
```

this necessitates adding an extra state variable "I" (the *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) <+ g(I(a, b));
```

To avoid the extra state variable, use

```
I(a, b) <+ f(V(a, b));
```

Parasitic Resistors

- **However, due to convergence considerations, voltage contributions are better when very small resistances are possible:**

```
V(a,b) <+ I(a,b) * Rab;
```

- **Alternatively, consider using a range on the parameter**

```
parameter real Rab = 50.0 from [100m:inf);
```

Voltage-controlled Elements

Inductances in Verilog-A will add an additional state variable:

```
V(a, b) <+ L * ddt(I(a,b));
```

Since this translates to

$$-x_a + x_b + \mathrm{ddt}(L * I_{ab}) == 0$$

where I_{ab} is the flow through the inductor

- **Truly voltage controlled elements should be implemented with voltage contributions.**

Branches from Conditionals

- **When variables that depend on `ddt()` are used in conditionals, the compiler must create extra branch equations**

```
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**

```
if (Mode == 1) begin
     t0 = TYPE * Ibd;
     arg0 =  Qbd;
     t1 = TYPE * Ibs;
     arg1 =  Qbs;
end
else begin
     t1 = TYPE * Ibd;
     arg1 = Qbd;
     t0 = TYPE * Ibs;
     arg0 = Qbs;
end

I(b,di) <+ t0 + ddt(arg0);
I(b,si) <+ t1 + ddt(arg1);
```

Improved MOSFET code

Branch-ddt Equations

- **Branch-ddt equations are state variables related to implementing `ddt()` equations with product terms:**

```
x = V(a, b);
I(a, b) <+ x * ddt(x);
```

- **How they arise:**

  From the basic nodal KCL

$$f(x(t)) + ddt(q(x(t))) == u(t)$$

  Note that it does not support terms of the form

$$g(x(t))*ddt(h(x(t)))$$

Branch-ddt (cont.)

The Verilog-A code

```
x = V(a, b);
I(a, b) <+ x * ddt(x);
```

introduces extra state variable phi**,**

$$\text{phi} - \text{ddt}(x) == 0$$

to effectively contribute

x*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:**

```
if (Rcin > 0.0)
    I(bi,si) <+ V(bi,si) / Rcin;
else
    V(bi,si) <+ 0.0;

if (Rgd > 0.0)
    I(gi,gdi) <+ V(gi,gdi) / Rgd;
else
    V(gi,gdi) <+ 0.0;

if (Ri > 0.0)
    I(gi,gsi) <+ V(gi,gsi) / Ri;
else
    V(gi,gsi) <+ 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

Probing Mistakes

Common mistake when probing port current:

```
$strobe(I(port_name));
```

- **Introduces unnamed branch**
  - **Effectively shorts `port_name` to `ground`**
  - **Adds additional state variable**
- **Instead use**

```
$strobe(I(<port_name>));
```

- **Watch for compile-time warnings**

Superfluous Assignments

Consider:

```
(10)     x = V(a, b)/R;
(11)     if(type == 1)
(12)          x = V(a, b)/R1;
(13)     else
(14)          x = V(a, b)/R2;
```

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.**

```
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 `$abstime` – usually returns 0**
- **Analog Operators**
  - **Allowed:**
    - Differentiation **`ddt()`**, **`ddx()`**
    - Delay **`absdelay()`**
    - Laplace **`laplace()`**
    - Integration **`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
287         Rc1 = Rcmin + Rc_T / tanh_psi;
288
289         case(Capmod)
290         0: begin  // Linear capacitance
```

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:
  <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>(gdi, di) : Current Branch (implicit)
  <unnamed>(gi, di) : Current Branch (implicit)
  <unnamed>(gi, gdi) : Switch Branch
  <unnamed>(gi, gsi) : Switch Branch
  <unnamed>(gi, si) : Current Branch (implicit)
  <unnamed>(gsi, si) : Current Branch (implicit)
  <unnamed>(rf, si) : Voltage Branch
  <unnamed>(si, s) : Switch Branch
  <unnamed>(t, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, xt2) : Voltage Branch
  <unnamed>(xt2, <gnd>) : Current Branch (implicit)

Branch ddt operators:
  [ line 358, col 40 ]
  [ line 362, col 25 ]
  [ line 367, col 39 ]
  [ line 371, col 24 ]
  [ line 376, col 40 ]
  [ line 380, col 25 ]
  [ line 388, col 21 ]
  [ line 326, col 24 ]
  [ line 327, col 24 ]
  [ line 330, col 38 ]
  [ line 331, col 38 ]

Potential memory states:
  'Q_gd'
  'Q_gs'

=== End of summary information for module 'angelov_va':
```

Tiburon compiler option to output diagnostics

Diagnostic output for angelov.va

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:
  <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>(gdi, di) : Current Branch (implicit)
  <unnamed>(gi, si) : Current Branch (implicit)
  <unnamed>(gsi, si) : Current Branch (implicit)
  <unnamed>(s, si) : Switch Branch
  <unnamed>(t, bi) : Current Branch (implicit)
  <unnamed>(xt1, rf) : Current Branch (implicit)
  <unnamed>(xt2, t) : Current Branch (implicit)

Branch ddt operators:
  [ line 358, col 40 ]
  [ line 362, col 25 ]
  [ line 367, col 39 ]
  [ line 371, col 24 ]
  [ line 376, col 40 ]
  [ line 380, col 25 ]
  [ line 388, col 21 ]
  [ line 326, col 24 ]
  [ line 327, col 24 ]
  [ line 330, col 38 ]
  [ line 331, col 38 ]

Potential memory states:
  'Q_gd'
  'Q_gs'

=== End of summary information for module 'angelov_va':
```

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

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

real alpha, P_avg;
real Vgs, Vgd, Vds, Vdg;
```

Unused variable

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:
  <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>(gdi, di) : Current Branch (implicit)
  <unnamed>(gi, di) : Current Branch (implicit)
  <unnamed>(gi, gdi) : Switch Branch
  <unnamed>(gi, gsi) : Switch Branch
  <unnamed>(gi, si) : Current Branch (implicit)
  <unnamed>(gsi, si) : Current Branch (implicit)
  <unnamed>(rf, si) : Voltage Branch
  <unnamed>(si, s) : Switch Branch
  <unnamed>(t, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, xt2) : Voltage Branch
  <unnamed>(xt2, <gnd>) : Current Branch (implicit)

Branch ddt operators:
  [ line 358, col 40 ]
  [ line 362, col 25 ]
  [ line 367, col 39 ]
  [ line 371, col 24 ]
  [ line 376, col 40 ]
  [ line 380, col 25 ]
  [ line 388, col 21 ]
  [ line 326, col 24 ]
  [ line 327, col 24 ]
  [ line 330, col 38 ]
  [ line 331, col 38 ]

Potential memory states:
  'Q_gd'
  'Q_gs'

=== End of summary information for module 'angelov_va':
```

Are branches and types what you thought?

```
V(rf,si)  <+ I(rf,si) * Rc1;
```

```
I(xt1)      <+ -Ids0 + ddt(Tau * V(xt1));
I(xt2)      <+ V(xt2);
V(xt1,xt2)  <+ ddt((Tau/3) * I(xt1,xt2));

I(di,si)  <+ Ids;
```

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:
  <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>(gdi, di) : Current Branch (implicit)
  <unnamed>(gi, di) : Current Branch (implicit)
  <unnamed>(gi, gdi) : Switch Branch
  <unnamed>(gi, gsi) : Switch Branch
  <unnamed>(gi, si) : Current Branch (implicit)
  <unnamed>(gsi, si) : Current Branch (implicit)
  <unnamed>(rf, si) : Voltage Branch
  <unnamed>(si, s) : Switch Branch
  <unnamed>(t, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, <gnd>) : Current Branch (implicit)
  <unnamed>(xt1, xt2) : Voltage Branch
  <unnamed>(xt2, <gnd>) : Current Branch (implicit)

Branch ddt operators:
  [ line 358, col 40 ]
  [ line 362, col 25 ]
  [ line 367, col 39 ]
  [ line 371, col 24 ]
  [ line 376, col 40 ]
  [ line 380, col 25 ]
  [ line 388, col 21 ]
  [ line 326, col 24 ]
  [ line 327, col 24 ]
  [ line 330, col 38 ]
  [ line 331, col 38 ]

Potential memory states:
  'Q_gd'
  'Q_gs'

=== End of summary information for module 'angelov_va':
```

Can branch **ddt**s be replaced?

Compiler Diagnostics

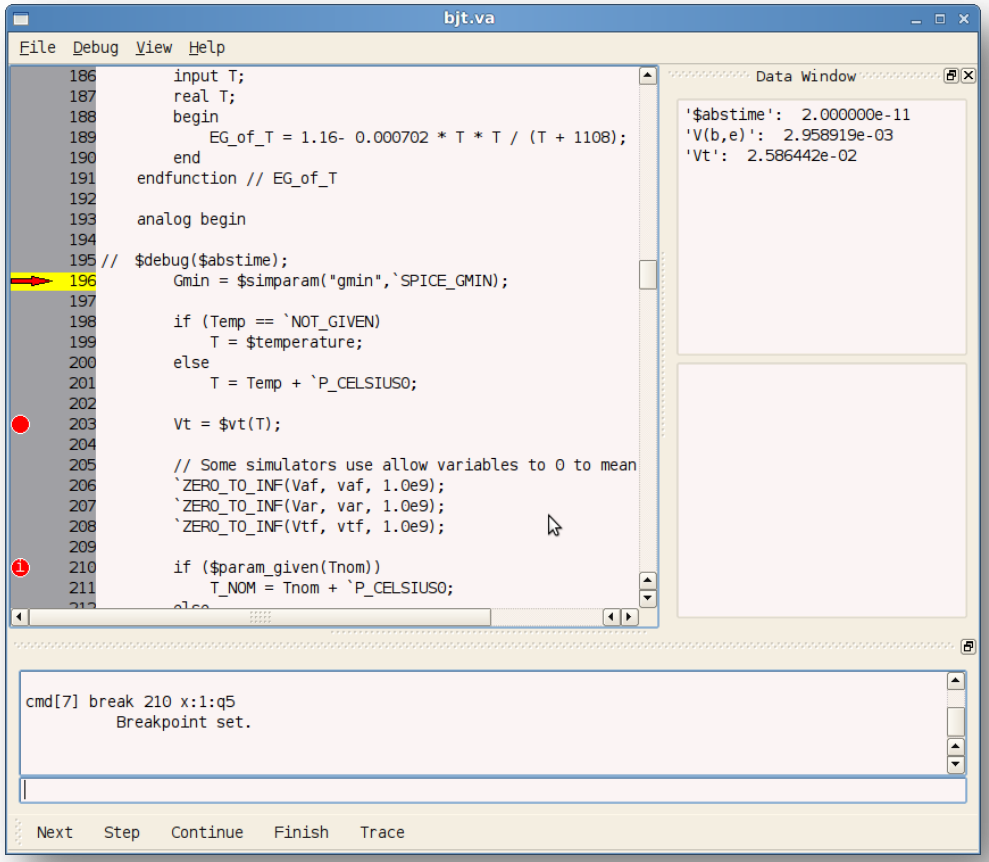
```
$ vacomp -Xinfo angelov.va

Warning: Variable 'P_avg' in module

=== Summary information for module

Branch information:
  <unnamed>(bi, gi) : Current Branc
  <unnamed>(bi, si) : Switch Branch
  <unnamed>(di, d) : Switch Branch
  <unnamed>(di, rf) : Current Branc
  <unnamed>(di, si) : Current Branc
  <unnamed>(g, di) : Current Branch
  <unnamed>(g, gi) : Switch Branch
  <unnamed>(gdi, di) : Current Bran
  <unnamed>(gi, di) : Current Branc
  <unnamed>(gi, gdi) : Switch Branc
  <unnamed>(gi, gsi) : Switch Branc
  <unnamed>(gi, si) : Current Branc
  <unnamed>(gsi, si) : Current Bran
  <unnamed>(rf, si) : Voltage Branc
  <unnamed>(si, s) : Switch Branch
  <unnamed>(t, <gnd>) : Current Bra
  <unnamed>(xt1, <gnd>) : Current B
  <unnamed>(xt1, xt2) : Voltage Bra
  <unnamed>(xt2, <gnd>) : Current B

Branch ddt operators:
  [ line 358, col 40 ]
  [ line 362, col 25 ]
  [ line 367, col 39 ]
  [ line 371, col 24 ]
  [ line 376, col 40 ]
  [ line 380, col 25 ]
  [ line 388, col 21 ]
  [ line 326, col 24 ]
  [ line 327, col 24 ]
  [ line 330, col 38 ]
  [ line 331, col 38 ]

Potential memory states:
  'Q_gd'
  'Q_gs'

=== End of summary information for module 'angelov_va':
```

```
case(Capmod)
0: begin  // Linear capacitance
    Q_gs= Cgspi ;
    Q_gd= Cgdpi;
end
1: begin  // Bias dependent capacitance
    Q_gs  = (Cgspi + Cgs0_T * tanh1 * tanh2);
    Q_gd  = (Cgdpi + Cgd0_T * (tanh3 * tanh4 + 2 * P111));
end
2:begin  // Charge-based (best convergence)
    tanh2 = tanh2 - P111;
    cosh0 = cosh(P10 + P111 * Vds);
    lc10 = ln(cosh0);
    cosh1 = cosh(psi_1);
    lc1 = ln(cosh1);
    Qgs0 = P10 + P111 * Vds + lc10;
    Q_gs = Cgs0_T * ((psi_1 + lc1 - Qgs0) * tanh2 /
        P11 + 2 * P111 * Vgsc) + Cgspi * Vgsc;

    cosh0 = cosh(P40 - P111 * Vds);
    lc40 = ln(cosh0);
    cosh1 = cosh(psi_4);
    lc4 = ln(cosh1);
    Qgd0 = P40 - P111 * Vds + lc40;
    Q_gd = Cgd0_T * ((psi_4 + lc4 - Qgd0) * tanh3 /
        P41 + 2 * P111 * Vgdc) + Cgdpi * Vgdc;
end
endcase

I(xt1)      <+ -Ids0 + ddt(Tau * V(xt1));
I(xt2)      <+ V(xt2);
V(xt1,xt2) <+ ddt((Tau/3) * I(xt1,xt2));

I(di,si)  <+ Ids;
I(gsi,si) <+ Igs;
I(gdi,di) <+ Igd;

if (Capmod == 2) begin
  I(gdi,di) <+ ddt(Q_gd);
  I(gsi,si) <+ ddt(Q_gs);
end
```

Check that memory states are necessary

Interactive Debugging

- **Allows quick iterative investigation of module**



Tiburon Design Automation
www.tiburon-da.com

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

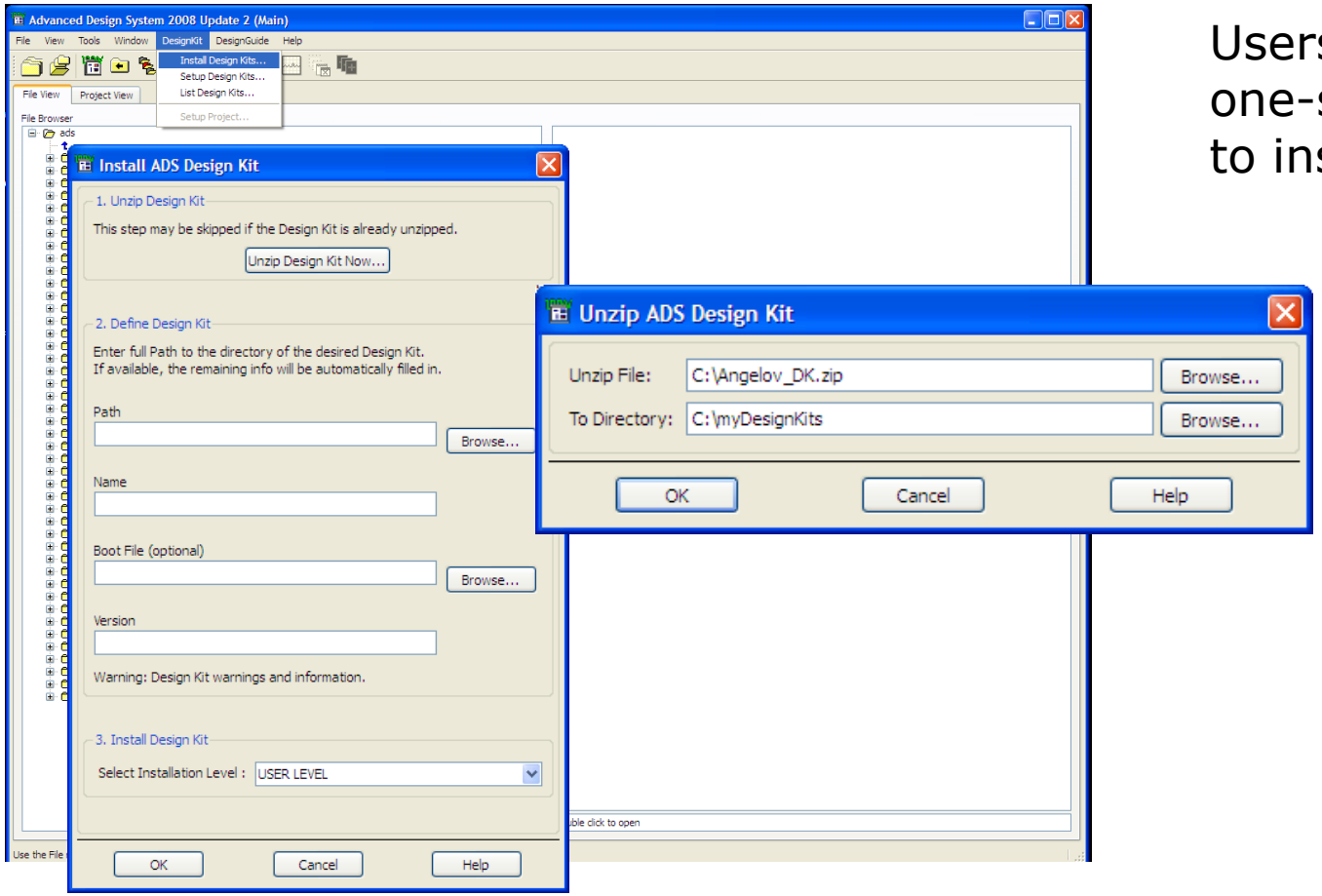
IP Protection

- **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

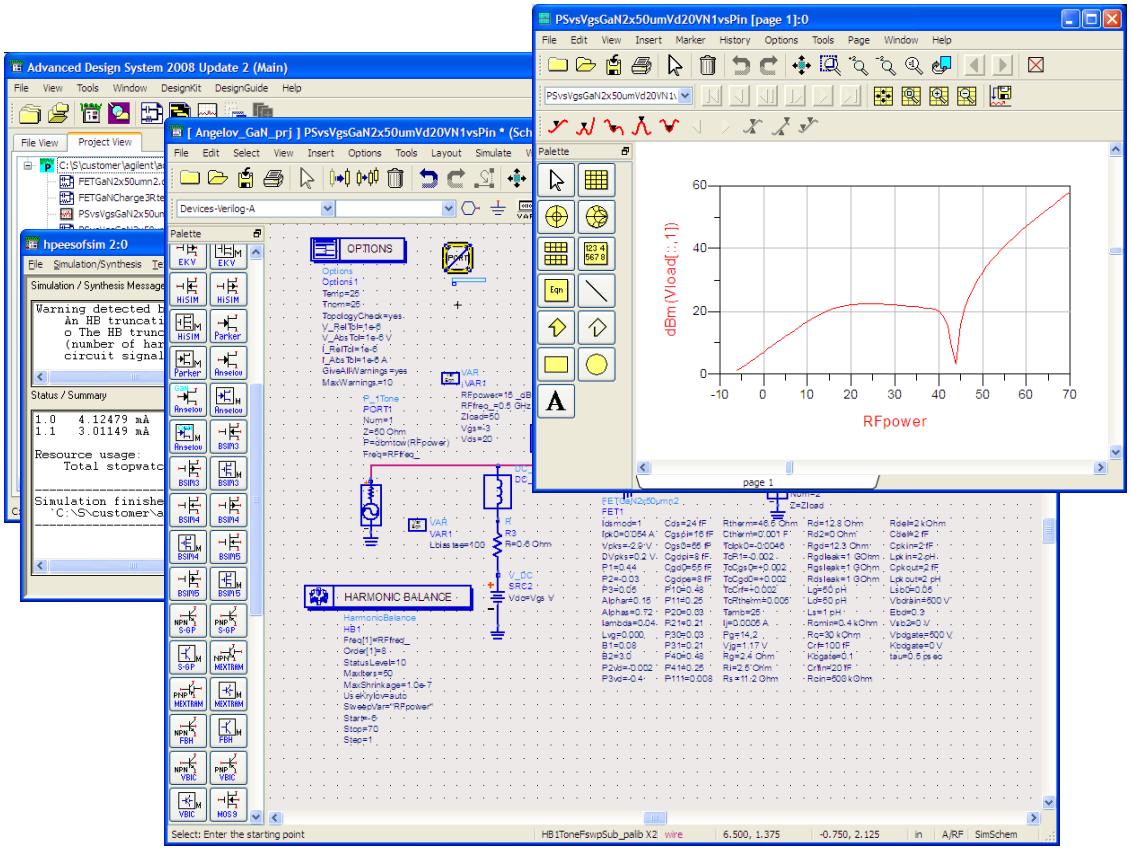
- **Design Kits provide a convenient mechanism for distributing complete model package**
- **End user opens a zipped file**

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**