Guidelines for Writing NEEDS-compatible Verilog-A Compact Models

Tianshi Wang and Jaijeet Roychowdhury

EECS Department
University of California at Berkeley
Outline

• What is a NEEDS-compatible Verilog-A model
  • flow of compact modelling with Verilog-A today
    ➔ improvements made by NEEDS-compatible Verilog-A

• A brief introduction to Verilog-A
  • by example

• How to write NEEDS-compatible Verilog-A
  • recommendations
  • identifying and working around potential problems in Verilog-A
How to put a compact model into a simulator?

- Few models
- Diode
- BJT
- MOS

So many models!
- SPICE
- VBIC
- EKV
- BSIM
- ACM
- SP
- USIM
- HiSIM
- HSPICE
- AMS
- Spectre
- Eldo
- ADM

So many simulators!
- Verilog-A
- Not easy

Different results in different simulators

Have to wait for simulator release of models

Adapted from L. Lemaitre, G. Coram, and C. McAndrew et al [1].
NEEDS-compatible Verilog-A

- confusing constructs
  - initialization
  - limiting

Verilog-A

Commercial Simulators

Open-source Simulators

ModSpec models
(MATLAB)

Certain constructs confusing/irrelevant even dangerous

Too much “freedom”

Numerical hazards

Lack of support for initialization/Limiting

Fas reliable controllable
NEEDS-compatible Verilog-A

Open-source NSF requirement

NEEDS-compatible Verilog-A

provide guidelines

commercial

translator

automatic

open/free

ModSpec Model (C++ API)

compile standalone

.so libraries (dynamically loadable)

Test immediately (standalone)

compact model is certified simulation-ready

confirm DC/AC/TRAN in C++ MDE

Open-source Simulators

Commercial Simulators

V-A Interpreter

ModSpec C++ API support

much easier than V-A interpreter to implement

NEEDS-compatible Verilog-A

 transparency and standard initialization/limiting support

fast execution

binary release becomes possible (model IP protection)
Verilog-A Code:

```verilog
1 `include "disciplines.vams"
2 `include "constants.vams"
3 4 module Horizontal_Cap(t, b, s);
5
6     inout t, b, s;
7     electrical t, b, s;
8
9     branch (t ,b )              cap;
10     branch (t ,s )              ts;
11     branch (b ,s )              bs;
12
13     parameter real L = 4.0e-5   from (0:inf);
14     parameter real W = 4.0e-5   from (0:inf);
15     parameter real D = 0.6e-6   from (0:inf);
16     parameter real EPSR = 3.9   from (0:inf);
17     parameter real CTS =1.0e-15 from (0:inf);
18     parameter real CBS =1.0e-14 from (0:inf);
19
20     real C;
21
22     analog begin
23         C = `P_EPS0 * EPSR * L * W / D;
24         I(cap) <+ ddt(C * V(cap));
25         I(ts)  <+ ddt(CTS * V(ts));
26         I(bs)  <+ ddt(CBS * V(bs));
27     end
28 endmodule
```

Compact Model Example: Capacitor

**Verilog-A Code:**

```
1 `include "disciplines.vams"
2 `include "constants.vams"
3
4 module Horizontal_Cap(t, b, s);
5
6     inout t, b, s;
7     electrical t, b, s;
8
9     branch (t ,b )              cap;
10     branch (t ,s )              ts;
11     branch (b ,s )              bs;
12
13     parameter real L = 4.0e-5   from (0:inf);
14     parameter real W = 4.0e-5   from (0:inf);
15     parameter real D = 0.6e-6   from (0:inf);
16     parameter real EPSR = 3.9   from (0:inf);
17     parameter real CTS =1.0e-15 from (0:inf);
18     parameter real CBS =1.0e-14 from (0:inf);
19
20     real C;
21
22     analog begin
23         C = `P_EPS0 * EPSR * L * W / D;
24         I(cap) <+ ddt(C * V(cap));
25         I(ts)  <+ ddt(CTS * V(ts));
26         I(bs)  <+ ddt(CBS * V(bs));
27     end
28 endmodule
```

**Nodes and Branches:**

- **top**
- **bottom**
- **substrate**

**Parameters:**

\[ q(v) = CV \]

\[ i(t) = \frac{d}{dt} q(v(t)) \]

\[ C = \frac{\varepsilon_r \varepsilon_0 L \cdot W}{d} \]

or

\[ C = \frac{\varepsilon_r \varepsilon_0 L \cdot W}{d} \left[ 1 + \frac{\frac{d}{\pi W}}{\frac{d}{\pi W}} \ln \left( \frac{2\pi W}{d} \right) \right] + \cdots \]

\[ I_{ts}(t) = \frac{d}{dt} (C_{p,t} \cdot V_{ts}(t)) \]

\[ I_{bs}(t) = \frac{d}{dt} (C_{p,b} \cdot V_{bs}(t)) \]

**complicated equations for** \( C_{p,t} \) **and** \( C_{p,b} \)**

H. B. Palmer 1927
Writing Verilog-A: Example

- How to write a Verilog-A model
- Consider a simple example →

Topology:

| Terminals: | a, b, c; |
| Internal node: | i; |
| Branches: | res, isrc, cap, ind, dio; |

Parameters:

Parameter real R = 1.0e3 from [0,inf) (* desc="Resistance of R", units="Ohm" *)

Equations:

\[
\begin{align*}
I(\text{res}) & \leftrightarrow V(\text{res})/R; \\
I(\text{cap}) & \leftrightarrow \text{ddt}(C \cdot V(\text{cap})); \\
V(\text{ind}) & \leftrightarrow \text{ddt}(L \cdot I(\text{ind}));
\end{align*}
\]
Writing Verilog-A: Code

Common Includes
Model Name
External/internal Nodes
Branch Declarations
Parameters
Local Variables
Equations

good

```verilog
equations
 module RLCdioIs0rc(a, b, c);

inout a, b, c;
electrical a, b, c, i;

branch (a, i ) res, isrc;
branch (b, i ) cap;
branch (c, i ) ind;
branch (a, b ) dio;

parameter real R = 1.0e3 from (0:inf);
parameter real L = 1.0e-9 from (0:inf);
parameter real C = 1.0e-6 from (0:inf);
parameter real IS= 1.0e-12 from (0:inf);
parameter real I = 1.0e-3 from (0:inf);

real Id, Vt;

kT
---------
q

analog begin

Vt = $Vt;
Id = 1.5*limexp(V(dio)/Vt) - 1);

I(res) <+ V(res)/R;
I(isrc)+< I;
I(cap) <+ ddt(C * V(cap));
V(ind) <+ ddt(L * I(ind));
I(dio) <+ Id;

end

endmodule
```
Writing Verilog-A: Recommendations

• **Topology**
  ➔ *define branches explicitly*
  • ALWAYS use branch voltages instead of individual node voltages
  • NEVER HAVE GLOBAL GROUND INSIDE DEVICE

• **Parameters**
  ➔ use constants from constants.vams
  ➔ another useful parameter statement [2]:
    • parameter string type="NMOS" from {"NMOS","PMOS"}
  ➔ $param_given()[2]: useful for testing if parameter is assigned

• **Equations**
  ➔ make all contributions right at the end
  ➔ charge/flux-based modeling: \( \text{ddt}(C \ast V(\text{cap})) \)
  ➔ if possible, model in terms of currents: \( I(\text{branch})<+\text{function}(\ldots); \)
Writing Verilog-A: Example

A more complicated example: VBIC model

Equivalent circuit:

Similar structure, only larger!

Topology:

11 nodes, 17 branches ...

Parameters:

100+ parameters ...

Equations:

400+ lines, convert branch voltages to currents.

Can code a BSIM model using only concepts studied here!
Potential Hazards (an incomplete list)

- Floating nodes
- Use of events $\rightarrow$ initial_step(), final_step()
- Use of functions $\rightarrow$ absdelay(), analysis()
- Memory states
- Implicit contribution
- Use of loops
- log() vs. ln()
- ddt() vs. idt()
- Discontinuity $\rightarrow$ if clauses, functions such as abs()
- Numerical hazards $\rightarrow$ division by 0, too large results
Floating Nodes

Consider this device:

- Jacobian becomes singular, DC analysis may fail in open-source simulators
- Affects portability across simulators
- Don't put floating nodes!

\[
\begin{align*}
KCL: & & \begin{pmatrix} a & b & c & i & f \end{pmatrix} & \begin{pmatrix} l(dio) \\ l(res) \\ l(isrc) \\ l(cap) \\ l(ind) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
\end{align*}
\]
Events

- Consider this code:
  ```
  @(initial_step) begin
    Tini = `TABS + TNOM;
  ...
  ```

- What will happen in transient analysis?
  - will be executed for every iteration at time = 0

- What will happen in DC sweep?
  - Temperature sweep: should recompute
  - Other sweeps: need not recompute

- Other events: final_step, cross, timer

- Just provide the equations, no events!

Functions

- Consider this function: \( \text{absdelay}(\text{input, td}) \)

- Not physical. How can a simulator deal with it?

- **Approximate using RC or RLC filter instead**

- Consider this function:
  
  ```
  if (analysis("tran"))
    q = ...
  ```

- Results in zero capacitance in AC analysis

- **Leave this optimization to simulators!**

ddt() vs idt()

- Consider this code:
  \[ I(\text{ind}) <+ \text{idt}(V(\text{ind})) / L, \]
- Not a differential equation
  - extra unknown needed
  - not supported by all simulators
- Change to \[ V(\text{ind}) <+ \text{ddt}(L*I(\text{ind})); \]

- Consider this code:
  \[ dI = \text{ddt}(\text{ddt}(Qd)); \]
- Add an extra variable/node instead

- Provide differential equations, avoid idt() and ddt of ddt
Memory States

• Consider:
  ➔ Can we use a variable without initializing it?

• “Memory States”:
  • If a variable is used before it is assigned a value:
    ➔ initializes it to 0 at the beginning
    ➔ takes on the value from the previously-converged time point.

• What will happen in AC analysis?
  ➔ no such concept as “time point”

• The whole concept of memory states is problematic!

  usually mistakes in variable usage

• Declare and initialize variables before use!
Implicit Contribution

- Consider this code:
  
  ```
  branch (p, n) dbranch;
  ...
  I(dbranch) <+ is*(exp((V(dbranch)-r*I(dbranch))/vt)-1);
  I(dbranch) <+ 1;
  ```

  ➔ MATLAB : `A = A + 1`? No!
  ➔ C++ : `A += A + 1`? No!
  ➔ implicit equation : `A = f(A)`? Yes!

- What is happening here:
  ➔ solve implicit equation: `Id = f(Vd, Id)`
  ➔ make contribution to KCLs at `p` and `n`

- Can be very confusing!
Implicit Contribution

- **How the code is interpreted:**
  
  \[ I(d\text{branch}) \leftarrow \text{is} \times \exp\left(\frac{(V(d\text{branch}) - r \times I(d\text{branch}))/vt - 1}{\text{allocate unknowns}}\right) \]

  \[
  \begin{bmatrix}
  \vdots \\
  \vdots \\
  \vdots \\
  I_d - f(V_d, I_d) \\
  \text{KCL}_p - I_d \\
  \text{KCL}_n + I_d \\
  \end{bmatrix}
  \]

  \[
  \begin{bmatrix}
  \vdots \\
  \vdots \\
  V_d \\
  I_d \\
  \end{bmatrix}
  \]

  contribution to KCLs at the nodes

  a clearer way
Loops

- Consider this code:
  
  ```
  A=1;
  while max(abs((Idx-Idxx)./Idx))>1e-10;
    A=A+1;
    if A>500, break, end
  ...
  // steps to refine Idx
  end
  ```

- This is basically a NR algorithm INSIDE the model to solve for Idx!

- while loops are generally not physical

- **Avoid using loops, provide an equation for Idx and let the simulator solve it**
Consider this code:

```java
if (x==0)
    Y = 1;
else
    Y = x+1;
end
```

- Discontinuity in function derivatives
- Be careful with `if` clauses
- `abs()` may also introduce discontinuity
- **Check that derivatives are continuous**

**log() vs. ln()**

- Verilog-A uses `log()` as base-10 logarithm and `ln()` as natural logarithm.
- Languages that use `log()` as natural logarithm:
  - C & C++
  - MATLAB
  - FORTRAN
  - python
  - java
  - VHDL-AMS

- **Don't use log() when you mean ln()!**

Numerical Hazards

- Division by 0
  - consider this code:
    \[ y = \sqrt{x}; \]
  - no problem calculating \( y \) when \( x=0 \), but
    \[ \frac{dy}{dx} = \frac{0.5}{\sqrt{x}}; \]
  - the same may happen to \( \text{pow}(a,b) \) when \( a=0, b<0 \)

- Too large results
  - \( \exp() \rightarrow \limexp() \)
  - \( \text{pow}(a, b) \rightarrow \limexp(b \times \ln(a)) \)

Examples in this page are adapted from G. Coram BMAS2004 Tutorial [3]
Summarizing Potential Hazards

- Don't put floating nodes
- Avoid use of events such as: `initial_step()`
- Avoid inappropriate functions: `absdelay()`, `analysis()`
- Don't use memory states
- Don't use implicit contributions
- Don't use loops to solve equations inside model
- `log()` is base-10 logarithm
- Avoid `idt()` and `ddt of ddt`
- Discontinuity → if clause, functions such as `abs()`
- Numerical hazards → `division by 0, too large results`
References


Conclusion

• NEEDS-compatible Verilog-A enables NEEDS-SPICE flow

• Introduction to Verilog-A
  ➔ Topology, Parameters, Equations

• Initial recommendations for making Verilog-A models NEEDS-compatible
  ➔ some potential hazards in Verilog-A identified
    • confusing constructs, numerical hazards, etc