Verilog-A Standardization for Compact Modeling

Marek Mierzwinski

Tiburon Design Automation
Santa Rosa, CA

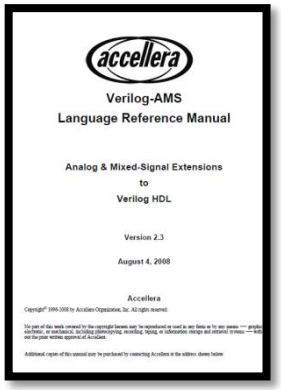
MOS-AK /GSA Workshop
December, 2011 Washington, DC

Outline

- A quick history
- Why Verilog-A has become the language of choice for compact model development
- Why some standardization still remains
- Panel introduction

Some history

- Verilog-A is a Hardware Description Language
- Verilog-A is a subset of Verilog-AMS
  - Verilog-AMS is a superset of Verilog
  - Subset is explicitly defined
- Developed in mid 1990s.
- Originally an interpreted language
- Specific extensions for compact models were developed in 2006 (LRM v 2.2)


accellera

Verilog-AMS
Language Reference Manual

Analog & Mixed-Signal Extensions
to
Verilog HDL

Version 2.3

August 4, 2008

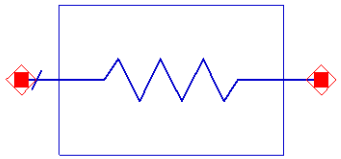
Accellera

Why Verilog-A?

- **Verilog-A is a natural language for analog model development**
  - **succinct**
  - **derivatives, calls to simulator code all handled by compiler**
  - **simple parameter support**
  - **powerful modelcard language**

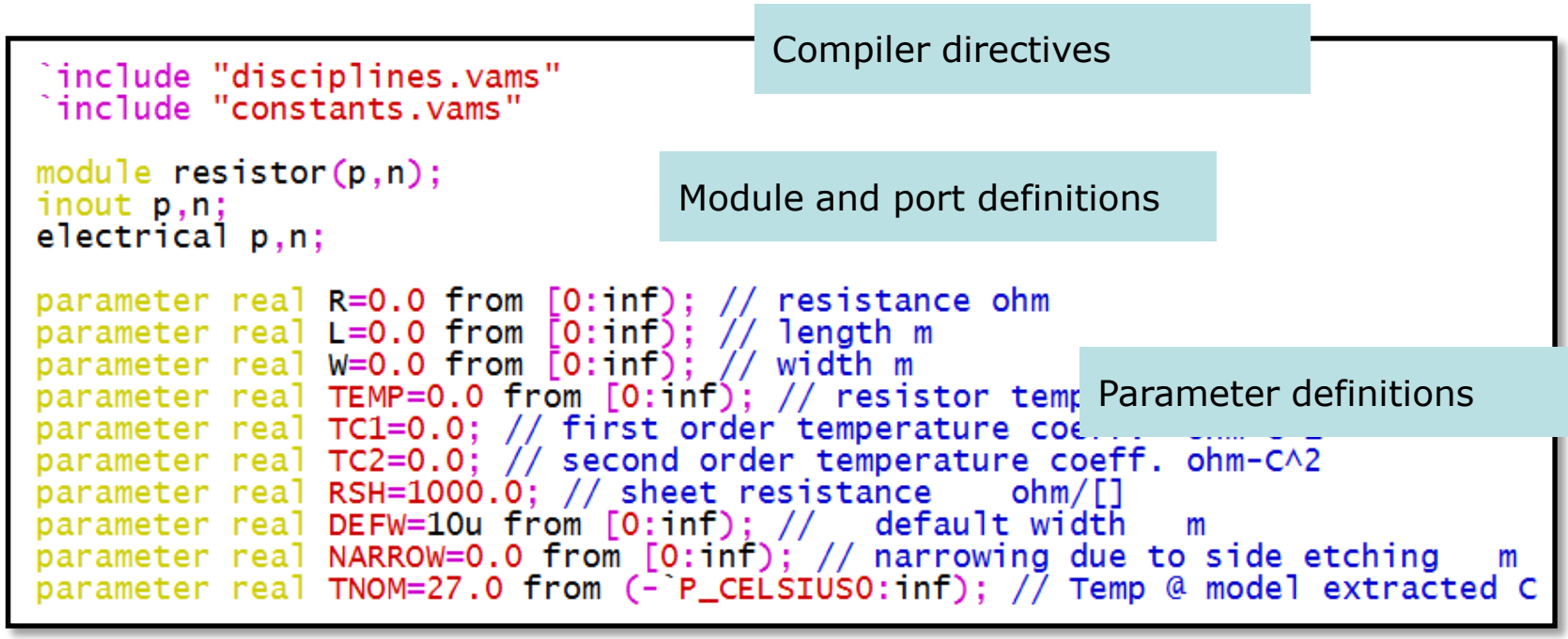
Geometric Resistor Example

- Illustrates how Verilog-A components can access netlist parameters and system parameters (like circuit temperature) , check range and if parameter is even changed
- Analysis-agnostic
- Simple functions for complex things like noise contributions



resistor
resistor1
R=0.0
L=0.0
W=0.0
TEMP=0.0
TC1=0.0
TC2=0.0
RSH=1000.0
DEFW=10u
NARROW=0.0
TNOM=27.0

Verilog-A: Example



Verilog-A: Example

```
analog begin

    if ($param_given(R))
        Rscaled = R;
    else begin // If R is not specified, calculate it from geometry
        if (!$param_given(L) || !$param_given(RSH))
            Rscaled = 1k;
        else begin
            if ($param_given(W))
                Width=W;
            else
                Width=DEFW;
            Rscaled = RSH * (L - NARROW) / (Width - NARROW);
        end
    end

    // Calculate thermal offset, if necessary:
    if ($param_given(TEMP)) begin
        Tdev = TEMP + `P_CELSIUS0;
        DeltaT = TEMP - TNOM;
    end
    else begin
        Tdev = $temperature;
        DeltaT = Tdev - (TNOM + `P_CELSIUS0);
    end
    Reff = Rscaled * (1 + TC1 * DeltaT + TC2 * DeltaT * DeltaT);

    if (Reff > 0.0)
        I(p,n) <+ V(p,n)/Reff +
             white_noise(4*`P_K*Tdev/Reff, "thermal");
    else
        V(p,n) <+ 0.0;
end
endmodule
```

Analog behavior

Verilog-A: Example

```
`include "disciplines.vams"
`include "constants.vams"

module resistor(p,n);
inout p,n;
electrical p,n;

parameter real R=0.0 from [0:inf); // resistance ohm
parameter real L=0.0 from [0:inf); // length m
parameter real W=0.0 from [0:inf); // width m
parameter real TEMP=0.0 from [0:inf); // resistor temperature C
parameter real TC1=0.0; // first order temperature coeff.  ohm-C^1
parameter real TC2=0.0; // second order temperature coeff. ohm-C^2
parameter real RSH=1000.0; // sheet resistance    ohm/[]
parameter real DEFW=10u from [0:inf); //   default width    m
parameter real NARROW=0.0 from [0:inf); // narrowing due to side etching    m
parameter real TNOM=27.0 from (-`P_CELSIUS0:inf); // Temp @ model extracted C

real Width, Tdev, DeltaT, Rscaled, Reff;

analog begin

    if ($param_given(R))
        Rscaled = R;
    else begin // If R is not specified, calculate it from geometry
        if (!$param_given(L) || !$param_given(RSH))
            Rscaled = 1k;
        else begin
            if ($param_given(W))
                Width=W;
            else
                Width=DEFW;
            Rscaled = RSH * (L - NARROW) / (Width - NARROW);
        end
    end

    // Calculate thermal offset, if necessary:
    if ($param_given(TEMP)) begin
        Tdev = TEMP + `P_CELSIUS0;
        DeltaT = TEMP - TNOM;
    end
    else begin
        Tdev = $temperature;
        DeltaT = Tdev - (TNOM + `P_CELSIUS0);
    end
    Reff = Rscaled * (1 + TC1 * DeltaT + TC2 * DeltaT * DeltaT);

    if (Reff > 0.0)
        I(p,n) <+ V(p,n)/Reff +
             white_noise(4*`P_K*Tdev/Reff, "thermal");
    else
        V(p,n) <+ 0.0;
end
endmodule
```

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

Verilog-A: Simulation

- **Most simulators include via a netlist instruction to load the source file**
  - `.hdl "~/myModels/mysource.va"`
  - `ahdl_include "~/myModels/mysource.va"`
- **During simulation, the Verilog-A module is instantiated like a primitive device:**
  - `X1 in out myAmp gain=10 ro=100k m=1`
  - `Amp1 (in out) myAmp gain=10 ro=100k m=1`
  - `R2 n1 n2 resistor Rsh=250`

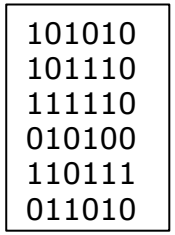
Behind the Scenes

Verilog-A source

Verilog-A compiler

C-code

C compiler



Object code

Analog Simulator

Why Verilog-A?

- **It's PORTABLE!**
  - **Verilog-A supported by all major commercial vendors**
  - **Write once, use everywhere**
  - **Models can be archived with the circuit design (not the simulator)**
- **It's in use**
  - **PSP, HiCUM, VBIC, EKV, MEXTRAM, Angelov…**

Related uses for compact models

- Wrapper for node monitoring
- Monitor device usage
  - Reliability
  - Power
- Extend
- Control instantiation by parameters
  - `x1 1 2 rcline N=10`

Example

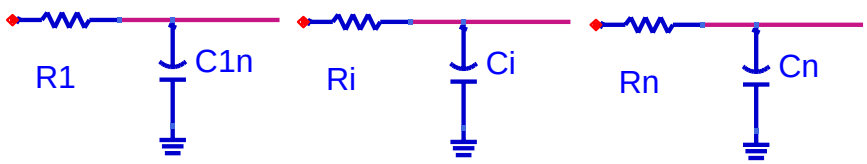
- **Variable number of instances**

```
module rcline(in, out);
parameter N=2 from [2:inf];
electrical in, out;
electrical [1:N-1] segments;
electrical gnd;
ground gnd;
rc seg[0:N-1]({in,segments}, {segments, out});
endmodule

/* 1 segment */
```

```
module rc(p,n);
inout p,n;
electrical p,n;
electrical gnd;
ground gnd;
resistor  #(.r(1))   r(p,n);
capacitor #(.c(0.1)) c(n,gnd);
endmodule
```

One module can instantiate a variable number of other modules (or primitives) based on parameter values



Current State

- Verilog-A models tend to run 10-100% slower
- Use more memory

Current State

- No theoretical reason for Verilog-A to be inferior in performance to built-ins
- Model coding can have a big influence
  - Execution speed
  - Memory use
  - Convergence/numerical stability
  - MOS-AK 2009/2010
- There have been demonstrations of equal or better performance

Current State

- Some compilers prefer variables are initialized in a particular way
- Some compilers need special coding patterns to determine special cases, like where a node might be collapsed
- Perceived limitations
  - Noise
  - NQS

Modeling Burst Noise

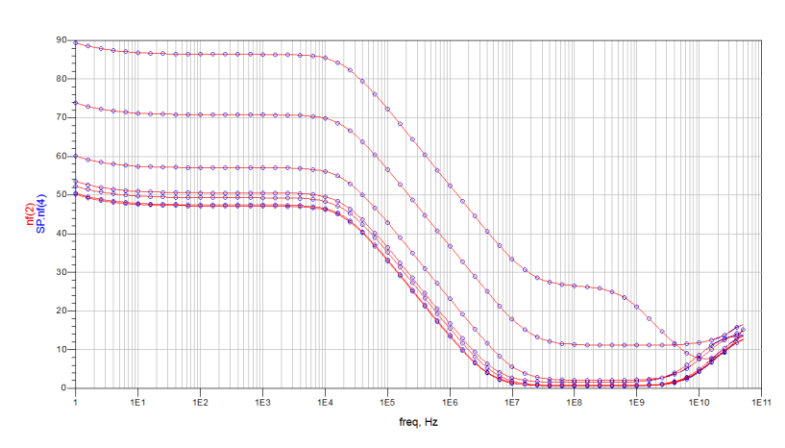
Implementing burst noise of the form can be done using the non-standard extension, `$realfreq`:

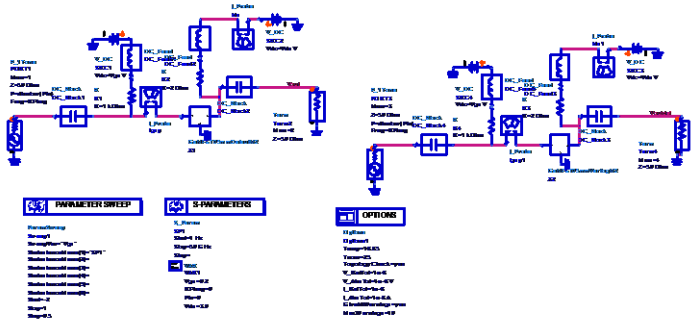
$$\langle i^2 \rangle = \frac{Kb \times Idc^{Ab}}{1 + \left(\frac{f}{Fb}\right)^2}$$

```
I(di,si) <+ white_noise(NoisePwr * Klf, "Burst") * sqrt(1/(1+pow($realfreq/Fgr,2)));
```

Or it can be done using standard Verilog-A functions by coloring the noise via a Laplace transform.

```
I(di,si) <+ laplace_nd(white_noise(Klf*NoisePwr, "burst"), {1}, {1,0,-1/(`M_TWO_PI*Fgr*`M_TWO_PI*Fgr) });
```





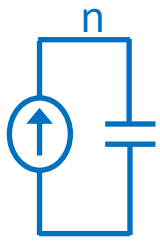
Compare built-in (c-code) version of Angelov to Verilog-A

Modeling NQS Effects in RF

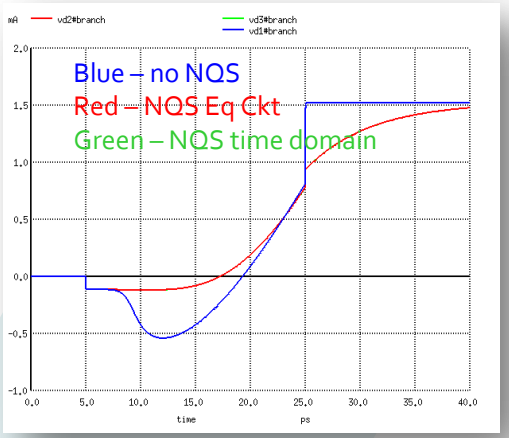
- Non-quasi-static effects are typically modeled with a delay on the charge

$$q(t_i) = \frac{q(t_{i-1}) + \frac{\Delta t}{\tau} Q(t_i)}{1 + \frac{\Delta t}{\tau}}$$

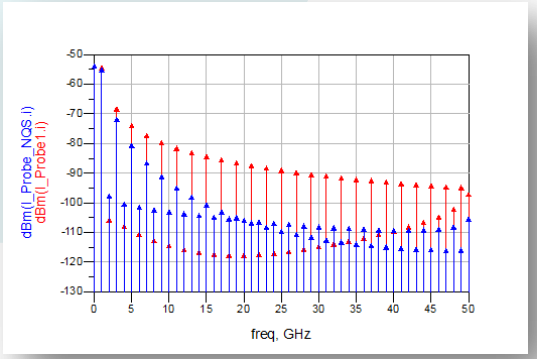
- Instead, use intermediate node




```
I(n) <+ (Qb_nqs - Qb) / taub + ddt(V(n));
Qb_nqs = V(n);
```



- NQS model now works in time and frequency domain



Conventions?

- **Model versus Instance parameters**
  - **Or should the compiler figure this out?**
- **Initialization and dependency**
  - **Is there a good way to suggest this to the compiler?**
    - analog initial block, block names,…
  - **Or should the compiler figure this out?**
- **Analysis-specific code**
  - **The language supports it, should compact models be allowed to use it?**

Google Group

Panelists

- Geoffrey J. Coram, Analog Devices (Moderator)
- Walter R. Curtice, WRC Consulting
- Carlos Galup-Montoro, Universidade Federal de Santa Catarina, Brazil
- Keith Green, Texas Instruments, Compact Modeling Council
- Benjamin Iniguez, University Rovira i Virgili, Spain, COMON Network