

2004 IEEE Behavioral Modeling and Simulation Conference (BMAS2004)

Tutorial: How to (and How NOT to) Write a Compact Model in Verilog-A

Geoffrey Coram
Analog Devices, Inc.



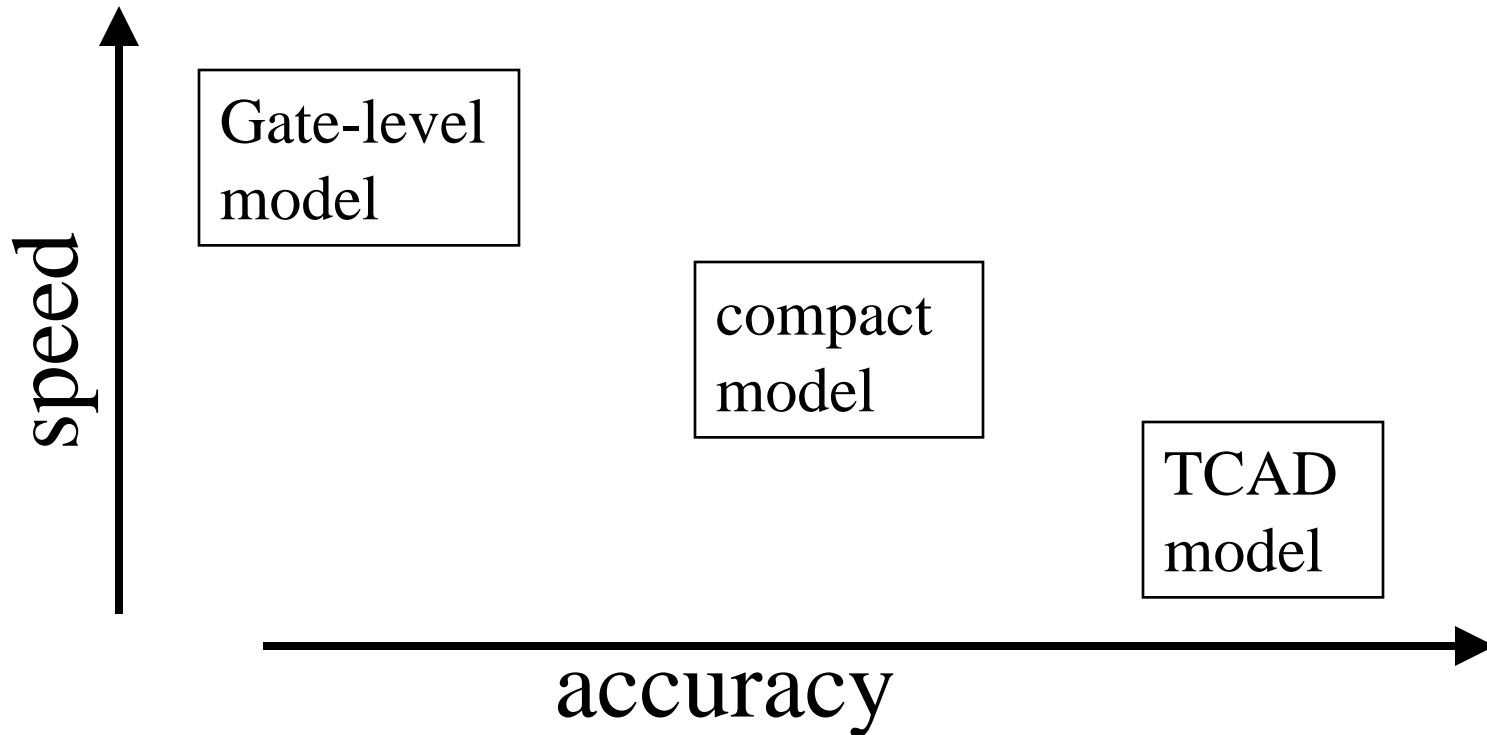
The World Leader in High Performance Signal Processing Solutions

Introduction

- ◆ **High-level language for compact modeling is long overdue**
- ◆ **Verilog-A is becoming the standard**
 - **Analog-only subset of Verilog-AMS**
 - **Compact modeling extensions in LRM 2.2**
- ◆ **Remaining steps:**
 - **Compact model developers need to become comfortable**
 - **Compilers must generate fast and reliable code**

What is a Compact Model?

- ◆ A model of transistor currents & voltages
- ◆ Built from physically-motivated equations
- ◆ Intended for use in an analog circuit simulator



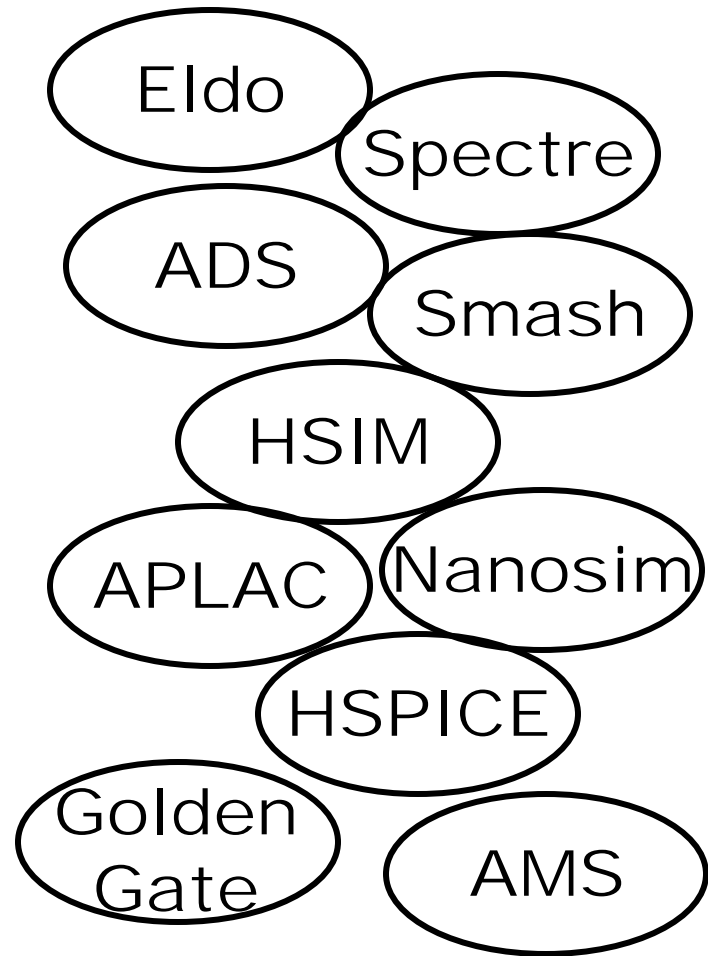
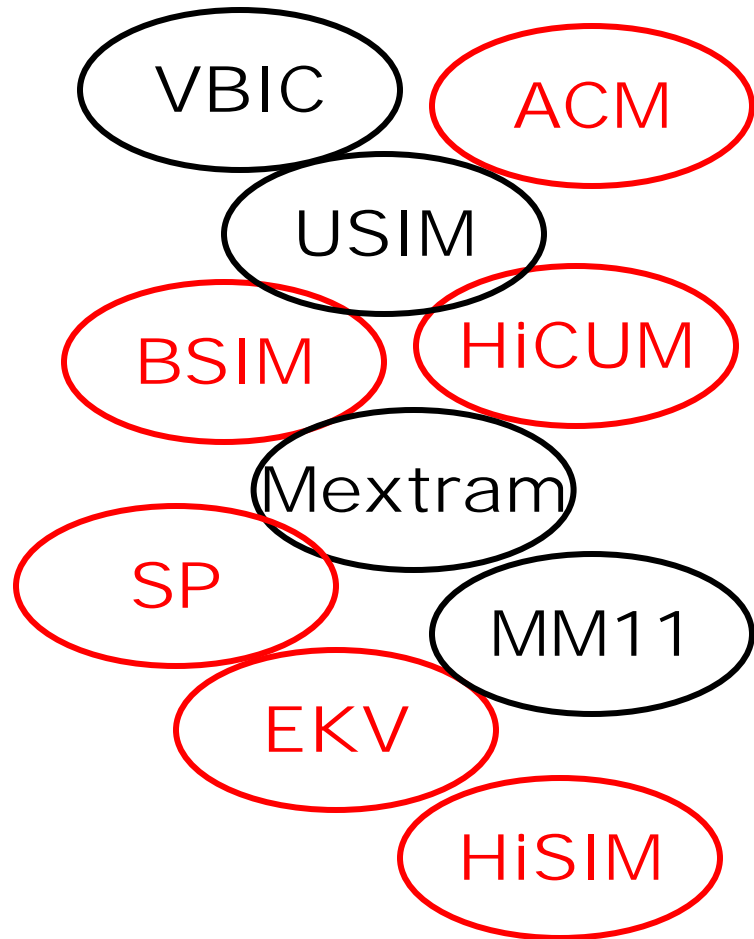
Why Verilog-A?

- ◆ **Faster implementation compared to C (or FORTRAN)**
 - **BSIM3 self-heating: 1-2 days in Verilog-A versus 2-3 weeks in C**
 - **Derivatives coded automatically**

Why Verilog-A?

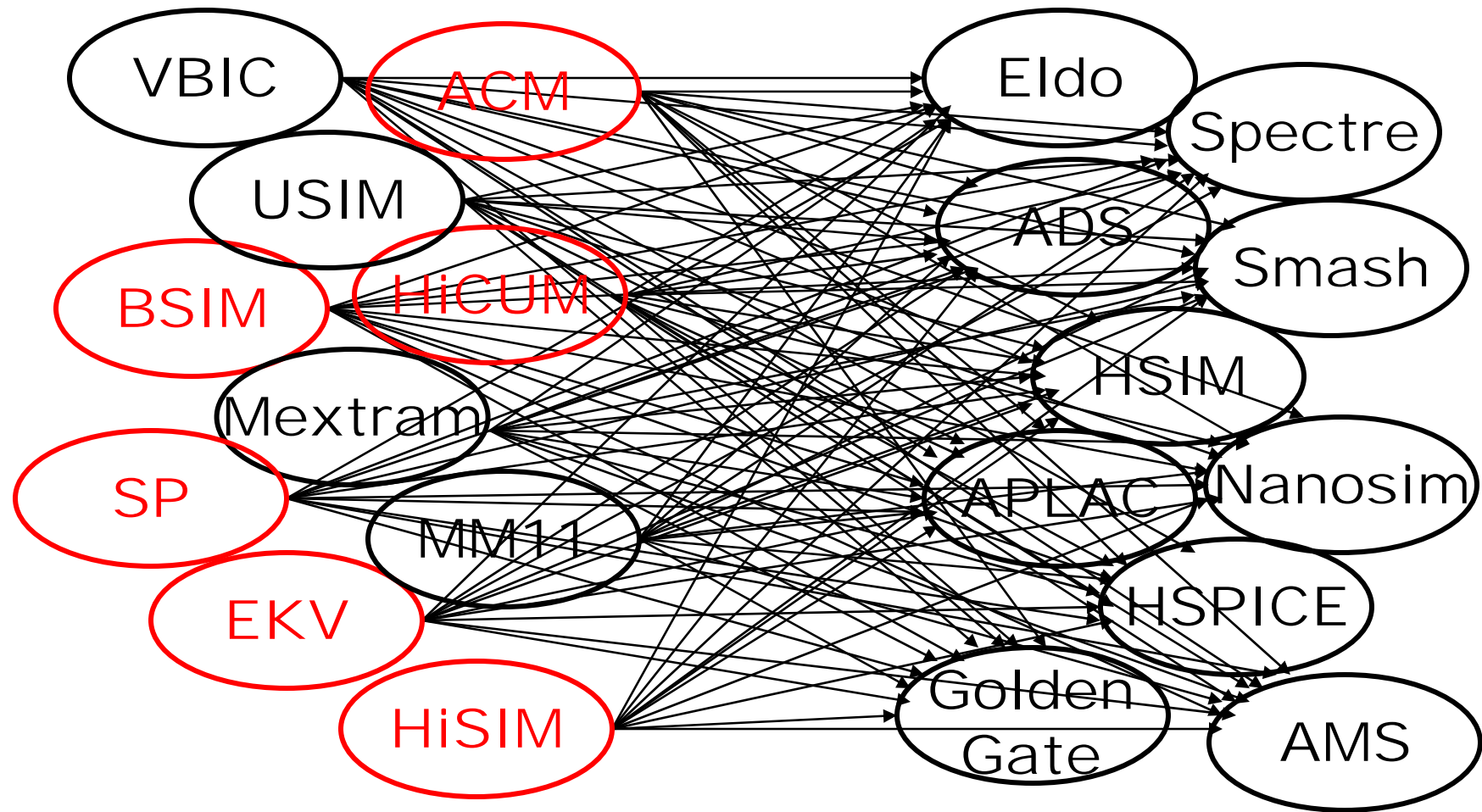
- ◆ **Faster implementation compared to C (or FORTRAN)**
 - **BSIM3 self-heating: 1-2 days in Verilog-A versus 2-3 weeks in C**
 - **Derivatives coded automatically**
- ◆ **Multiple simulator support**
 - **Analog Devices:Adice, Motorola/Freescale:Mica**
 - **Cadence:Spectre, MentorGraphics:Eldo, Synopsys:NanoSim, Agilent:ADS & ICCap, Silvaco:SmartSpice & UTMOST, ...**
 - **No* simulator-specific details**

Many models, many simulators



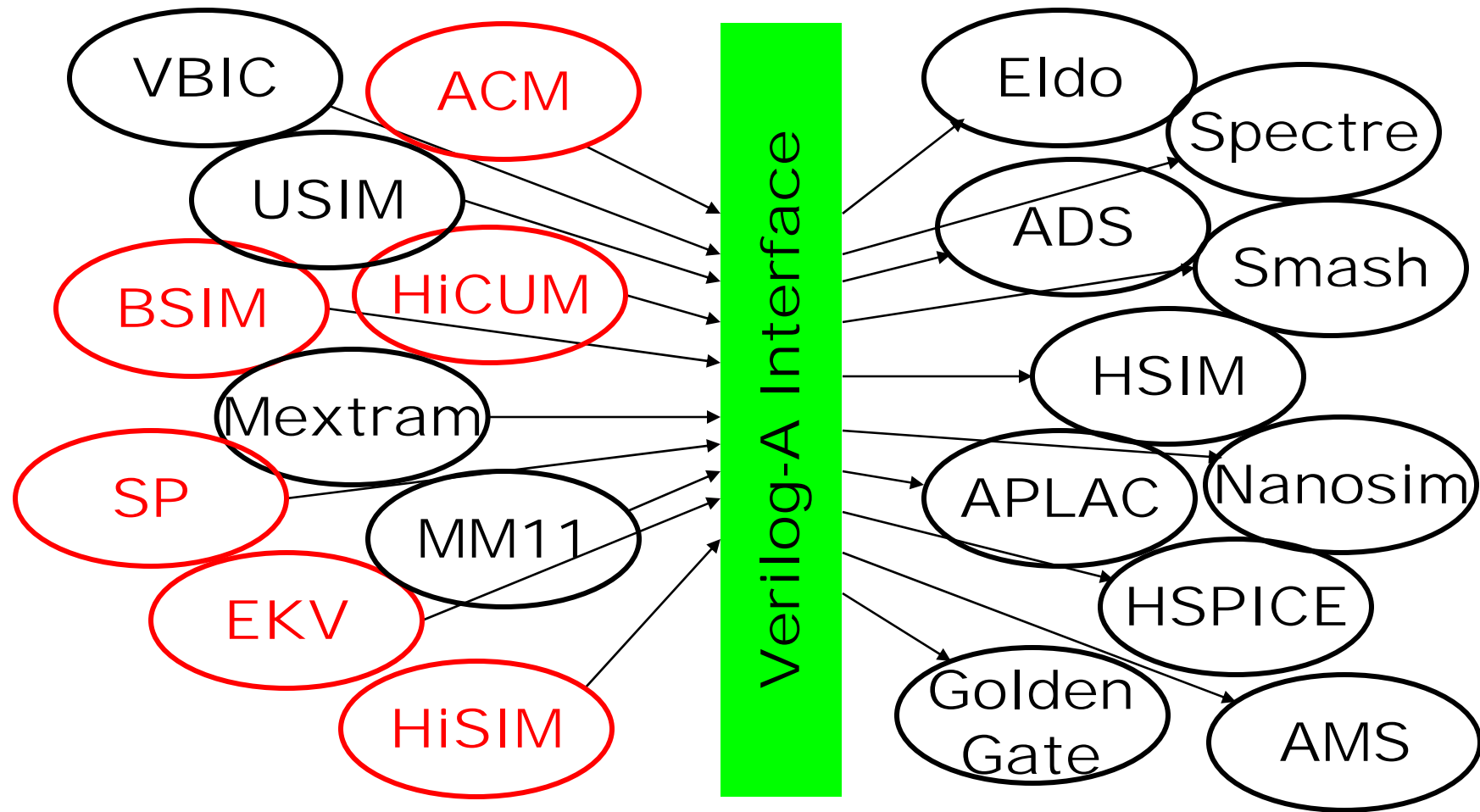
Source: C. McAndrew, *et al*, "Extensions to Verilog-A to Support Compact Device Modeling"

Many models, many simulators



Source: C. McAndrew, *et al*, "Extensions to Verilog-A to Support Compact Device Modeling"

The Solution



Source: C. McAndrew, *et al*, "Extensions to Verilog-A to Support Compact Device Modeling"

Why not Verilog-A?

- ◆ **Implementation problems**
 - **Slow performance**
 - **Poor convergence**
 - **Inconsistent results between simulators**
- ◆ **Missing language constructs**
- ◆ **Too easy to create (bad) models**
 - **Discontinuities**
 - **Non-physical equations that “blow up” outside expected range**
 - **(but at least the derivatives are always right)**

Why not Verilog-A?

- ◆ Implementation problems
 - ~~Slow performance~~ → **Compiled interfaces**
 - Poor convergence
 - Inconsistent results between simulators
- ◆ Missing language constructs
- ◆ Too easy to create (bad) models
 - Discontinuities
 - Non-physical equations that “blow up” outside expected range
 - (but at least the derivatives are always right)

Why not Verilog-A?

- ◆ Implementation problems
 - ~~Slow performance~~
 - ~~Poor convergence~~ → **Mature implementations**
 - ~~Inconsistent results between simulators~~ →
- ◆ Missing language constructs
- ◆ Too easy to create (bad) models
 - Discontinuities
 - Non-physical equations that “blow up” outside expected range
 - (but at least the derivatives are always right)

Why not Verilog-A?

- ◆ Implementation problems
 - ~~Slow performance~~
 - ~~Poor convergence~~
 - ~~Inconsistent results between simulators~~
- ◆ ~~Missing language constructs~~ → **CM extensions**
- ◆ Too easy to create (bad) models
 - Discontinuities
 - Non-physical equations that “blow up” outside expected range
 - (but at least the derivatives are always right)

Why not Verilog-A?

- ◆ Implementation problems
 - ~~Slow performance~~
 - ~~Poor convergence~~
 - ~~Inconsistent results between simulators~~
- ◆ ~~Missing language constructs~~
- ◆ Too easy to create (bad) models
 - ~~Discontinuities~~ → **Verilog-A debuggers?**
 - Non-physical equations that “blow up” outside expected range
 - (but at least the derivatives are always right)

How TO

- ◆ **Verilog-A is a simple language**
- ◆ **Most concepts can be learned from studying a simple example**
- ◆ **Can write BSIM3 in Verilog-A using only the concepts discussed here**

Diode example, p1

```
`include "disciplines.vams"  
`include "constants.vams"  
module diode(a,c);  
  inout a,c;  
  electrical a,c,int;  
  branch (a,int) res;  
  branch (int,c) dio;  
  parameter real is = 10p from (0:inf);  
  parameter real rs = 0.0 from [0:inf);  
  parameter real cjo = 0.0 from [0:inf);  
  parameter real vj = 1.0 from (0:inf);
```

Diode example, p1

```
`include "disciplines.vams"  
`include "constants.vams"  
module diode(a,c);  
  inout a,c;  
  electrical a,c,int;  
  branch (a,int) res;  
  branch (int,c) dio;  
  parameter real is = 10p from (0:inf);  
  parameter real rs = 0.0 from [0:inf);  
  parameter real cjo = 0.0 from [0:inf);  
  parameter real vj = 1.0 from (0:inf);
```



***modules replace
Spice primitives***

Diode example, p1

```
`include "disciplines.vams"  
`include "constants.vams"  
module diode(a,c);
```

```
  inout a,c;
```

```
  electrical a,c,int;
```

```
  branch (a,int) res;
```

```
  branch (int,c) dio;
```

```
  parameter real is = 10p from (0:inf);
```

```
  parameter real rs = 0.0 from [0:inf);
```

```
  parameter real cjo = 0.0 from [0:inf);
```

```
  parameter real vj = 1.0 from (0:inf);
```



**terminals
aka *ports*
(int is internal)**

Diode example, p1

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

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

```
module diode(a,c);
```

```
  inout a,c;
```

```
  electrical a,c,int;
```

```
  branch (a,int) res;
```

```
  branch (int,c) dio;
```

```
  parameter real is = 10p from (0:inf);
```

```
  parameter real rs = 0.0 from [0:inf);
```

```
  parameter real cjo = 0.0 from [0:inf);
```

```
  parameter real vj = 1.0 from (0:inf);
```



branches

connect nodes

Diode example, p1

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

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

```
module diode(a,c);
```

```
  inout a,c;
```

```
  electrical a,c,int;
```

```
  branch (a,int) res;
```

```
  branch (int,c) dio;
```

```
  parameter real is = 10p from (0:inf);
```

```
  parameter real rs = 0.0 from [0:inf);
```

```
  parameter real cjo = 0.0 from [0:inf);
```

```
  parameter real vj = 1.0 from (0:inf);
```

***parameter* declarations
include default values
and ranges**



Diode example, p2

```
`ifdef VAMS_COMPACT_MODELING
  aliasparam phi = vj;
  (*desc="jct. voltage"*) real vd;
  (*desc="current"*) real id;
  (*desc="depl. charge"*) real qd;
  (*desc="depl. cap."*) real cd;
  (*desc="conductance"*) real gd;
  `define GMIN ($simparam("gmin"))
`else
  real vd, id, qd;
  `define GMIN (1.0e-12)
`endif
```

Diode example, p2

```
`ifdef VAMS_COMPACT_MODELING
  aliasparam phi = vj;
  (*desc="jct. voltage"*) real vd;
  (*desc="current"*) real id;
  (*desc="depl. charge"*) real qd;
  (*desc="depl. cap."*) real cd;
  (*desc="conductance"*) real gd;
  `define GMIN ($simparam("gmin"))
`else
  real vd, id, qd;
  `define GMIN (1.0e-12)
`endif
```



**compiler
directive
for CM
extensions**

Diode example, p2

```
`ifdef VAMS_COMPACT_MODELING
  aliasparam phi = vj;
  (*desc="jct. voltage"*) real vd;
  (*desc="current"*) real id;
  (*desc="depl. charge"*) real qd;
  (*desc="depl. cap."*) real cd;
  (*desc="conductance"*) real gd;
  `define GMIN ($simparam("gmin"))
`else
  real vd, id, qd;
  `define GMIN (1.0e-12)
`endif
```

output variables

Diode example, p3

```
analog begin
```

```
    V(res) <+ I(res) * rs;
```

```
    vd = V(dio);
```

```
    id = is * (limexp(vd/$vt) - 1.0);
```

```
    if (vd < 0) begin
```

```
        qd = 2.0 * cjo * vj * (1.0 - sqrt(1.0 - vd/vj));
```

```
    end else begin
```

```
        qd = cjo * vd * (1.0 + vd / (4.0 * vj) );
```

```
    end
```

```
    I(dio) <+ id + `GMIN * vd;
```

```
    I(dio) <+ ddt(qd);
```

Diode example, p3

```
analog begin ← one analog block  
    V(res) <+ I(res) * rs; per module;  
    vd = V(dio); describes behavior  
    id = is * (limexp(vd/$vt) - 1.0);  
    if (vd < 0) begin  
        qd = 2.0 * cjo * vj * (1.0 - sqrt(1.0 - vd/vj));  
    end else begin  
        qd = cjo * vd * (1.0 + vd / (4.0 * vj) );  
    end  
    I(dio) <+ id + `GMIN * vd;  
    I(dio) <+ ddt(qd);
```

Diode example, p3

```
analog begin
```

```
  V(res) <+ I(res) * rs;
```

```
  vd = V(dio);
```

```
  id = is * (limexp(vd/$vt) - 1.0);
```

```
  if (vd < 0) begin
```

```
    qd = 2.0 * cjo * vj * (1.0 - sqrt(1.0 - vd/vj));
```

```
  end else begin
```

```
    qd = cjo * vd * (1.0 + vd / (4.0 * vj) );
```

```
  end
```

```
  I(dio) <+ id + `GMIN * vd;
```

```
  I(dio) <+ ddt(qd);
```



**parasitic
resistance**

Diode example, p3

```
analog begin
```

```
  V(res) <+ I(res) * rs;
```

```
  vd = V(dio);
```

```
  id = is * (limexp(vd/$vt) - 1.0);
```

```
  if (vd < 0) begin
```

```
    qd = 2.0 * cjo * vj * (1.0 - sqrt(1.0 - vd/vj));
```

```
  end else begin
```

```
    qd = cjo * vd * (1.0 + vd / (4.0 * vj) );
```

```
  end
```

```
  I(dio) <+ id + `GMIN * vd;
```

```
  I(dio) <+ ddt(qd);
```

**diode dc
current**



Diode example, p3

```
analog begin
```

```
  V(res) <+ I(res) * rs;
```

```
  vd = V(dio);
```

```
  id = is * (limexp(vd/$vt) - 1.0);
```

```
  if (vd < 0) begin
```

```
    qd = 2.0 * cjo * vj * (1.0 - sqrt(1.0 - vd/vj));
```

```
  end else begin
```

```
    qd = cjo * vd * (1.0 + vd / (4.0 * vj) );
```

```
  end
```

```
  I(dio) <+ id + `GMIN * vd;
```

```
  I(dio) <+ ddt(qd);
```

depletion
capacitance



Diode example, p4

```
`ifdef VAMS_COMPACT_MODELING
    gd = ddx(id, V(int));
    cd = ddx(qd, V(int));
`endif

    I(dio) <+ white_noise(2 * `P_Q * id, "shot");
    V(res) <+ white_noise(4 * `P_K *
                          $temperature * rs,
                          "thermal");

end
endmodule
```

Diode example, p4

```
`ifdef VAMS_COMPACT_MODELING
    gd = ddx(id, V(int));
    cd = ddx(qd, V(int));
`endif

I(dio) <+ white_noise(2 * `P_Q * id, "shot");
V(res) <+ white_noise(4 * `P_K *
                    $temperature * rs,
                    "thermal");

end
endmodule
```

**small-signal
output variables**

Diode example, p4


```
`ifdef VAMS_COMPACT_MODELING
    gd = ddx(id, V(int));
    cd = ddx(qd, V(int));
`endif

I(dio) <+ white_noise(2 * `P_Q * id, "shot");
V(res) <+ white_noise(4 * `P_K *
                    $temperature * rs,
                    "thermal");

// I(dio) <+ flicker_noise(...);

end
endmodule
```

noise contributions



Parameter handling

◆ Default expressions

```
parameter integer mobmod = 1 from [1:3];  
parameter real uc =  
    (mobmod==3) ? -46.5e-3 : -46.5e-12;
```

◆ Testing if a parameter was specified with

```
$param_given()
```

◆ String parameters

```
parameter string type = "NMOS"  
    from { "NMOS", "PMOS" };
```

How NOT to

- ◆ **Don't use `log ()` where you mean `ln ()`**

- ◆ **Don't introduce discontinuities:**
 - **`abs (x)` when `x` is bias-dependent**
 - **`if ()` clauses that don't join up**

- ◆ **Don't use `analysis ()` or events**

$\log()$ VS $\ln()$

- ◆ Languages that use $\log()$ to mean the natural logarithm:

C, C++

FORTRAN

MATLAB

VHDL-AMS

...

- ◆ Languages that use $\ln()$

Verilog-A

- ◆ THREE independent model writers have been tripped up by this

Discontinuities

- ◆ **SPICE models (for analog simulators) must be well-behaved:**
 - **$I(V)$ continuous**
 - **dI/dV continuous for Newton's method**
 - **$d^n I/dV^n$ continuous for distortion simulations**
 - **physical charges and currents are well-behaved**

Discontinuities

◆ Consider this code:

```
if (vbs == 0.0) begin  
    qbs = 0.0;  
    capbs = czbs+czbssw+czbsswg;  
end else if (vbs < 0.0) begin  
    qbs = ...  
    ...  
    I(b,s) <+ ddt(qbs);
```

Discontinuities

◆ Resulting C code:

```
if (vbs == 0.0) {  
    qbs = 0.0;  
    dqbs_dvbs = 0.0;  
    // capbs = czbs + czbssw + czbsswg;  
} else if (vbs < 0.0) {  
    qbs = ...  
    ...  
}
```

Automatic derivative differs from intended value

analysis ()

◆ Consider this code:

```
if ( analysis ( "tran" ) ) begin
    qd = ...
    ...
```

analysis ()

◆ Consider this code:

```
if ( analysis ( "tran" ) ) begin  
    qd = ...  
    ...
```

- ◆ Capacitance in small-signal ac analysis, harmonic balance, envelope following
- ◆ Pseudo-transient homotopy

analysis ()

◆ **Consider this code:**

```
    if (analysis("noise")) begin
        flicker =
strongInversionNoiseEval(vds, temp);
        ...
    end
```

analysis ()

- ◆ Consider this code:

```
if (analysis("noise")) begin  
    flicker =  
    strongInversionNoiseEval(vds, temp);  
    ...
```

- ◆ But what about PNOISE, HBNoise, ...?

- **Compiler/Simulator MUST do this optimization**

Events

◆ Consider this code:

```
@(initial_step) begin  
    isdrain = jsat * ad;
```

Events

◆ Consider this code:

```
@(initial_step) begin  
    isdrain = jsat * ad;
```

◆ What happens for a dc sweep?

- don't want re-computing for bias sweep
- need re-computing for temperature sweep

◆ Even for transient, `initial_step` is true for every iteration at time=0

➤ **Compiler MUST do this optimization**

Optimizations

◆ Dependency trees

- replace `analysis()` and `initial_step`

◆ Common subexpressions

```
id = is * (exp(vd/vtm) - 1.0);  
gd = is/vtm * exp(vd/vtm);
```

◆ Eliminating internal nodes

```
if (rs == 0.0)  
    V(res) <+ 0.0;  
else  
    I(res) <+ V(res) / rs;
```

Conclusion

- ◆ **(Almost) everything you need has been demonstrated**
- ◆ **Hazards have been identified**
- ◆ **Leave optimizations to the simulator**

Conclusion

- ◆ **(Almost) everything you need has been demonstrated**
 - ◆ **Hazards have been identified**
 - ◆ **Leave optimizations to the simulator**
- **Go write a Verilog-A model!**