Presentation Notes of the 2005 Workshop on

Compact Modeling for RF/Microwave Applications (CMRF 2005)

Santa Barbara, California, USA
October 12, 2005

Organized and sponsored by

Delft Institute of Microelectronics and Submicron Technology

DIMES

Technically Co-Sponsored by

IEEE Electron Device Society

Workshop Organizers

Slobodan Mijalković
Hsien-Chang Wu
Joachim Burghartz

Laboratory of High Frequency Technology and Components (HiTeC)
Delft Institute of Microelectronics and Submicron Technology (DIMES)
Delft University of Technology
Netherlands

Advisory Committee

Jörg Berkner	*Infineon Technologies AG, Germany*
Didier Celi	*ST Microelectronics, France*
John R. Long	*Delft University of Technology, Netherlands*
David Harame	*IBM Microelectronics, USA*
Dirk Klaassen	*Philips Research, Netherlands*
Colin McAndrew	*Freescale Semiconductor, USA*
Michael Schröter	*Dresden University of Technology, Germany*
Ming-Ta Yang	*TSMC, Taiwan*

2005 Workshop on Compact Modeling for RF/Microwave (CMRF 2005)

Verilog-A for Compact Modeling: Best Practices for High-Quality Model Authoring

Geoffrey Coram

ANALOG DEVICES

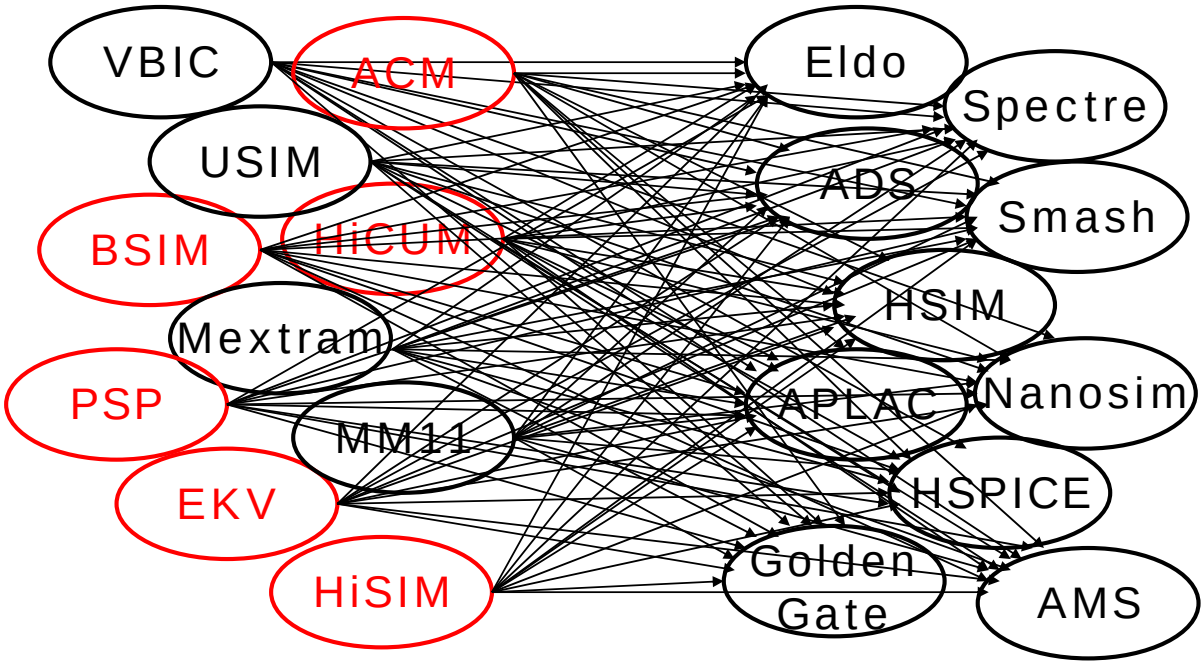
Colin McAndrew

Launched by Motorola
freescale semiconductor

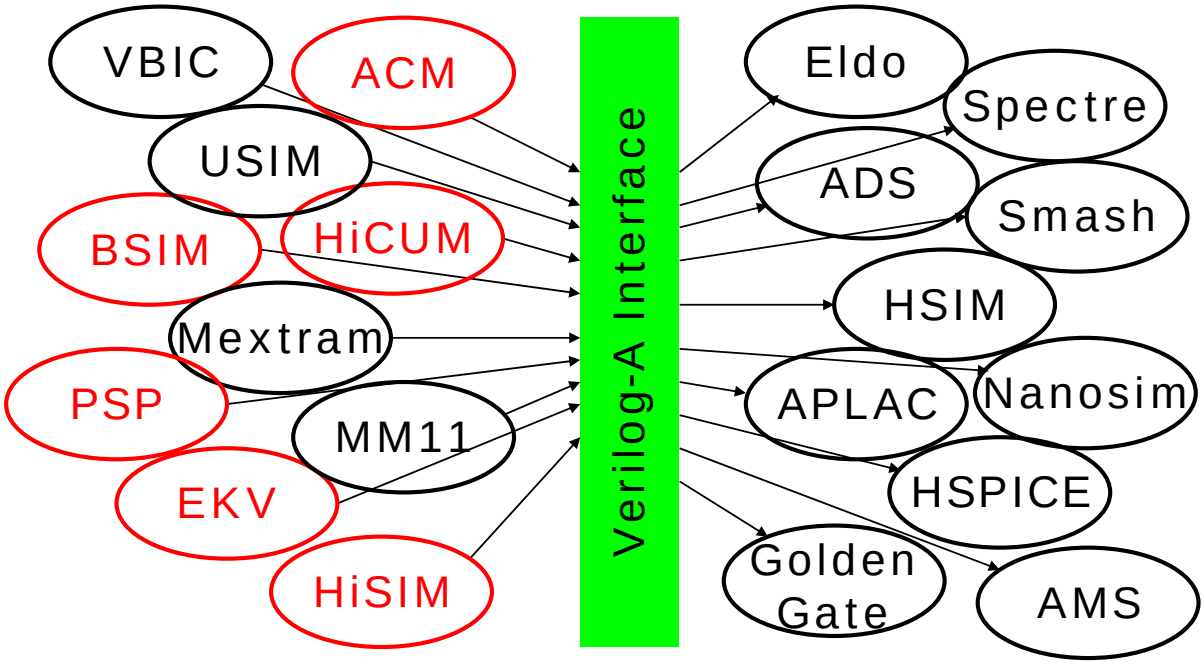
Outline

- **The Problem**
- **History**
- **Guidelines**
- **Admonishments**
- **Future**

Many models, many simulators



The Solution



```
SUBROUTINE BJT
…
   LOC=LOCATE(12)
10 IF ((LOC.EQ.0).OR.(NODPLC(LOC+36).NE.0)) RETURN
   LOCV=NODPLC(LOC+1)
   NODE1=NODPLC(LOC+2)
   NODE2=NODPLC(LOC+3)
   NODE3=NODPLC(LOC+4)
…
   AREA=VALUE(LOCV+1)
   BFM=VALUE(LOCM+2)
   BRM=VALUE(LOCM+8)
   CSAT=VALUE(LOCM+1)*AREA
   RBPR=VALUE(LOCM+18)/AREA
   RBPI=VALUE(LOCM+16)/AREA-RBPR
…
   ICHECK=1
   GO TO (100,20,30,50,60,70),INITF
   20 IF(MODE.NE.1.OR.MODEDC.NE.2.OR.NOSOLV.EQ.0) GO TO 25
      VBE=TYPE*VALUE(LOCV+2)
      VCE=TYPE*VALUE(LOCV+3)
```



```
SUBROUTINE DCOP
…
C  DIODES
C
  210 IF (JELCNT(11).EQ.0) GO TO 300
      ITITLE=0
  211 FORMAT(1H0,/,'0**** DIODES')
      LOC=LOCATE(11)
…
C  BIPOLAR JUNCTION TRANSISTORS
C
  300 IF (JELCNT(12).EQ.0) GO TO 400
      ITITLE=0
  301 FORMAT(1H0,/,'0**** BIPOLAR JUNCTION TRANSISTORS')
      LOC=LOCATE(12)
…
C  JFETS
C
  400 IF (JELCNT(13).EQ.0) GO TO 500
      ITITLE=0
  401 FORMAT(1H0,/,'0**** JFETS')
```

```
SUBROUTINE TMPUPD
…
C  DIODE MODEL
C
  100 LOC=LOCATE(21)
      IF (LOC.EQ.0) GO TO 200
      WRITE (IOFILE,101)
…
C  BIPOLAR TRANSISTOR MODEL
C
  200 LOC=LOCATE(22)
      IF (LOC.EQ.0) GO TO 300
      WRITE (IOFILE,201)
…
C  JFET MODEL
C
  300 LOC=LOCATE(23)
      IF (LOC.EQ.0) GO TO 400
      WRITE (IOFILE,301)
```



```
SUBROUTINE NOISE(LOCO)
…
C  DIODES
C
  200 IF (JELCNT(11).EQ.0) GO TO 300
      ITITLE=0
  201 FORMAT(//'0**** DIODE NOISE (SQ V/HZ)')
  210 LOC=LOCATE(11)
…
C  BIPOLAR JUNCTION TRANSISTORS
C
  300 IF (JELCNT(12).EQ.0) GO TO 400
      ITITLE=0
  301 FORMAT(//'0**** TRANSISTOR NOISE (SQ V/HZ)')
  310 LOC=LOCATE(12)
…
C  JFETS
C
  400 IF (JELCNT(13).EQ.0) GO TO 500
      ITITLE=0
  401 FORMAT(//'0**** JFET NOISE (SQ V/HZ)')
```

History of CM Interfaces

- **SPICE2**
  - **Excellent FORTRAN for the 1970's, but …**
  - **… the GOTO statement only goes so far**
  - **Ian Getreu:**
    - "the problem with SPICE2 was it was so good it stifled circuit simulator development for decades"
- **Improved model interfaces**
  - **ADMIT (dynamic linking, SPICE-like simulator)**
  - **iSMILE (similar interface)**
  - **CMC Type-I interface**

History of CM Interfaces

- **Early symbolic interfaces**
  - **WATAND**
  - **Tektronix**
  - **Saber/MAST**
  - **CMC Type-II interface**
- **Analog behavioral modeling languages**
  - **VHDL-AMS, first ABM language working group, painfully slow to come to fruition …**
  - **Verilog-A, spear-headed by Cadence, came to market earlier**

History of CM Interfaces

- **ABMs hijacked by device modelers**
  - **Verilog-A has become the de facto standard**
    - changing modeling paradigms is hard enough without arguing about which language is better
  - **Steve Hamm CICC2005 Panel:**
    - "most usage of Verilog-A models is for device modeling, not analog behavioral modeling"
- **Verilog-AMS LRM2.2 was driven by the requirements for compact modeling**

Verilog-A Jump Start

- **Looks much like many other languages**
- **Intuitive and easy to read and learn**
  - **start with an existing model and modify**
- **Based on through and across variables**
  - **set up is for KCL and KVL**
- **Understand the "contribution" operator**

```
I(di,si) <+ Ids;          // current di to si
V(d ,di) <+ I(b_rd)*rd; // voltage d  to di
```

- **Dynamic flows are done via `ddt()`**

Best Practices

- **Models formulated in currents I(V) and charges Q(V)**
  - **most natural for modified nodal analysis (MNA)**
  - **Q(V) not C(V) to ensure conservation of charge**
    - ddt(Q(V)) != ddt(C(V) * V) != C(V) * ddt(V)
- **No access to previous timesteps**
  - **watch non-quasi-static formulations**
  - **model may run in RF simulator**
- **Noises as current sources**

Cautions

- **Don't use `log()` where you mean `ln()`**
- **Watch for integer division: 1/2 = 0**
  - **but 1/2.0 = 1.0/2 = 1.0/2.0 = 0.5**
- **Don't introduce discontinuities:**
  - **`abs(x)` when x is bias-dependent**
  - **`if()` clauses that don't join up**
  - **`sqrt(x)` is OK at x=0, derivative is not**
- **Keep it simple**
  - **Verilog-AMS has many features not appropriate for compact modeling**
  - **Don't use `analysis()` or events**

`log()` vs `ln()`

- **Languages that use `log()` to mean the natural logarithm:**

C, C++	FORTRAN	Perl
MATLAB	VHDL-AMS	Awk
HSpice	Spectre	…

- **Languages that use `ln()`**

Verilog-A	Saber

- **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^2I/dV^2 continuous for homotopy**
  - **d^nI/dV^n continuous for distortion simulations**
  - **physical charges and currents are well-behaved**

Discontinuities

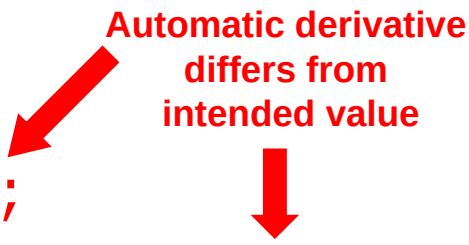
- **Consider this code:**

```
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

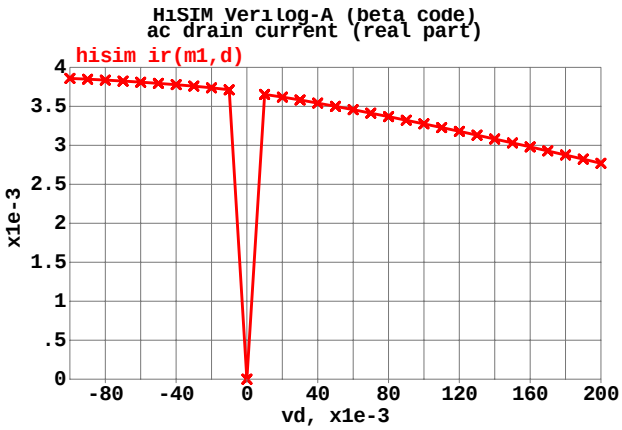
- **Resulting C code:**

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



Discontinuities

- **HiSIM Verilog-A (beta code)**
- **Clipping in C code may affect values and derivatives differently**




```
if ( Vds <= `epsm10 ) begin
    Pds = 0.0 ;
    Psl = Ps0 ;
```


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);
  …
```

- **But what about PNOISE, HBNoise, …?**

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

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**

Compiler 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;
```


Software Practices

- **Documentation**
  - **comments in model code, may reference external report for complicated models**
- **Regression tests**
  - **a model should come with test specifications and reference results**
  - **complete and comprehensive**
  - **DC, AC, noise, temperature, geometry, …**
  - **automate testing of N/P polarity flip, source-drain reversal, shrink, scale, m-factor**
  - **one-by-one testing of each “effect” in a model**

Software Practices

- **Align code vertically on =**
- **Use meaningful names**
  - **use maximum size (8) to help vertical alignment**
- **Physical constants are *not* dated and could change**
  - **model results would then change**
  - **define physical constants for a model**
- **Use `x*0.5` rather than `x/2.0`**
  - **it is more computationally efficient**
  - **compiler should catch this**

Software Practices

- **Reusability of common components**
  - **e.g. junction diodes in MOS models**
  - **use macros**
    - would be nice to have a common set
  - **declare local variables for re-usability**
    - block names must be passed down hierarchy

Software Practices

```
//
//  Built-in potential temperature mapping
//  (smoothly goes to zero as temperature increases).
//

`define psibi(psibi,p,ea,vtv,rt,blockName) \
begin : blockName \
    real psiio, psiin ; \
    psiio = 2.0*(vtv/rt)*ln(exp(0.5*p*rt/vtv)-exp(-0.5*p*rt/vtv)); \
    psiin = psiio*rt-3.0*vtv*ln(rt)-ea*(rt-1.0); \
    psibi = psiin+2.0*vtv*ln(0.5*(1.0+sqrt(1.0+4.0*exp(-
  psiin/vtv)))); \
end
```


Software Practices

```
//
//  pn junction charge, area and perimeter components
//

`define pnjQap(pnjQ,v,a,cja,pa,ma,fca,aja,
                      p,cjp,pp,mp,fcp,ajp,blockName,dba,dbp) \
begin : blockName \
    real acja,pcjp,arga,argp; \
    acja = a*cja; \
    pcjp = p*cjp; \
    if (acja>0.0) begin \
        `qj(arga,v,pa,ma,fca,aja,dba); \
    end else \
        arga = 0.0; \
    if (pcjp>0.0) begin \
        `qj(argp,v,pp,mp,fcp,ajp,dbp); \
    end else \
        argp = 0.0; \
    pnjQ = acja*arga+pcjp*argp; \
end
```

Software Practices

- **Coding style**
  - **use consistent indentation**
  - **use a rational block structure**
    - compilers should optimize code partitioning
    - structuring is for readability and maintainability
  - **keep “begin” and “end” as last and first directives in a line**
  - **watch for adding a line to a single line conditional that does NOT have “begin end”**
    - indenting does not mean it gets grouped with the line
  - **declare all branches (start with b_ for clarity)**
- **Start by modifying an existing model**

Debugging

- **`$strobe` to print values to the screen**
  - **`$write` in Eldo? (`$display`, `$monitor` also)**
- **`$fstrobe` to write to a file**
- **$debug added in LRM 2.2 – print every iteration**
- **How to get derivative information?**
  - **ddx() in LRM 2.2.**
  - **AC (small-signal) test circuit …**

Debugging

```
module mymos(d,g,s,b);
```

becomes

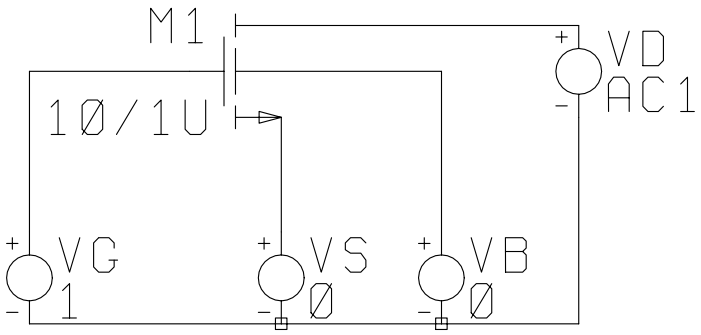
```
module mymos(d,g,s,b,test);
```

add this:

```
V(test) <+ Vdsat;
```

then AC voltage on test node is

$$\frac{\partial Vdsat}{\partial V_d}$$



Examples

- **Diode – presented here**
- **Other sources of examples:**
  - **Verilog-A model library at http://www.designers-guide.org/VerilogAMS/**
    - VBIC, MOS11, JFET, etc.
  - **Silvaco "public domain" models (non-commercial use) https://src.silvaco.com/ResourceCenter/en/downloads/verilogA.jsp**
    - BSIM3, BSIM4, BJT, etc.
  - **PSP, Mextram, HiCUM web sites**

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
    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);
```

one *analog* block per module; describes behavior

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

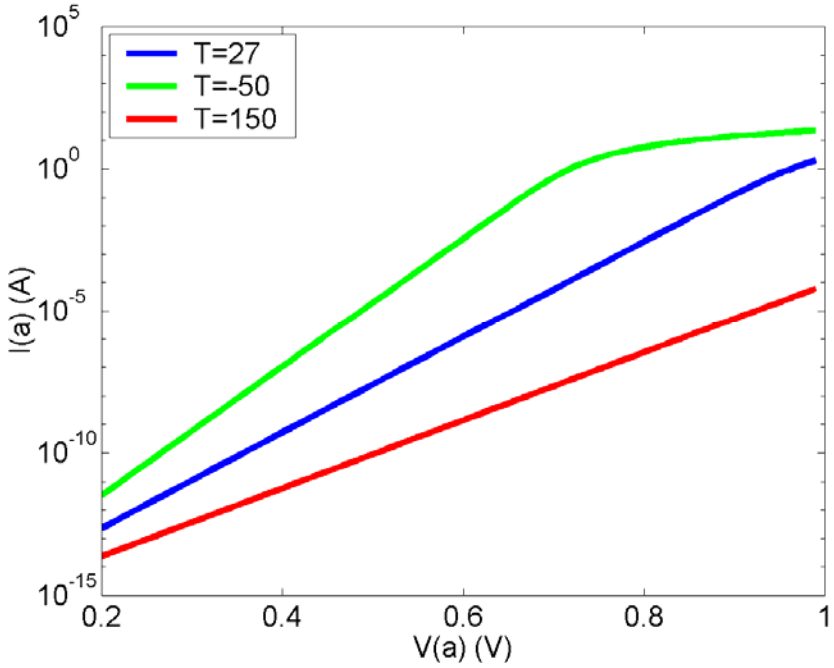
QA

```
verilogaFilename:    gcDiode.va
verilogaModulename:  diode
keyletter:           a
pins:                a c  // names not important
temperature:         27 -50 200

test:                defaultDcSweep
biases:              V(c)=0.0
biasSweep:           V(a)=0.2 1.0 0.01
outputs:             I(a)
modelParameters:     is=1.0e-16 rs=0.01
```

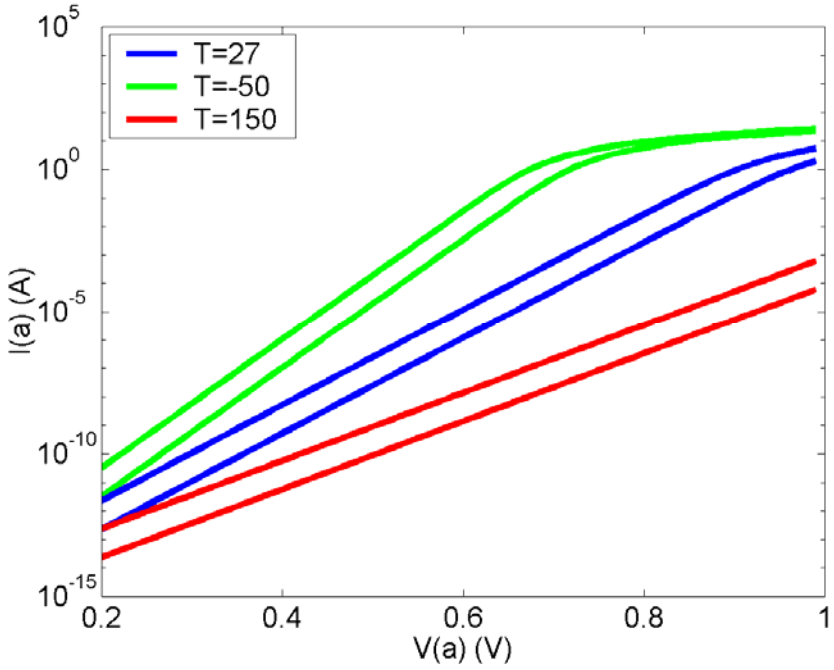

QA

```
test:                tenIsDcSweep
biases:              V(c)=0.0
biasSweep:           V(a)=0.2 1.0 0.01
outputs:             I(a)
modelParameters:     is=10.0e-16 rs=0.01

test:                r10DcSweep
biases:              V(c)=0.0
biasSweep:           V(a)=0.2 1.0 0.01
outputs:             I(a)
modelParameters:     is=1.0e-16 rs=10
```

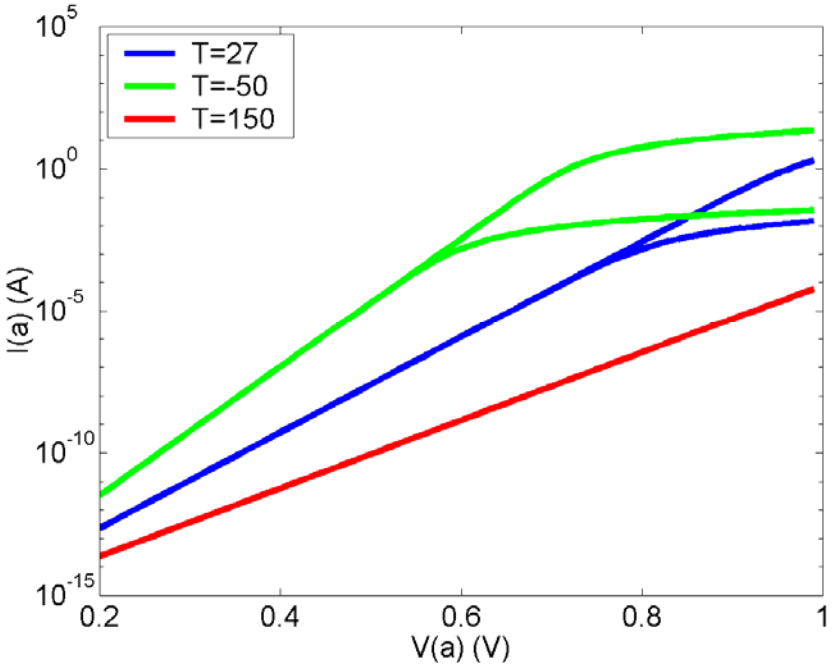
defaultDcSweep



tenIsDcSweep



r10DcSweep



QA

- **All models should come with complete and comprehensive tests defined, and with reference simulation results**
  - **DC**
  - **AC**
  - **noise**
  - **geometry**
  - **temperature**
  - **for all combinations of polarity flip, terminal flip, m-factor, shrink, scale**

Future

- $table_model in LRM2.2 is anemic
  - need to define algorithms for splines in multiple dimensions
- Would like to see model debugging tools
- Verilog-AMS will be aligned with SystemVerilog, the new IEEE digital Verilog standard

Conclusion

- Verilog-A is a powerful and easy-to-use compact modeling language
- Writing a good compact model still requires care and rigor
- Many examples now available