

Tiburon Design Automation Verilog-A Compiler and Run-Time Engine in UCB SPICE

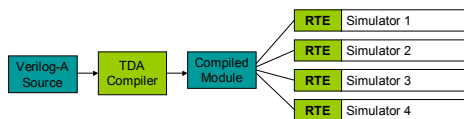
Tiburon's Verilog-A Compiler and Run-Time Engine (RTE) have been implemented in UC Berkeley's SPICE 3F5 program as a demonstration of capability. Key attributes of Tiburon's next-generation Verilog-A compiler will include

- Speed
- Complete language support
- Functionality for all analyses.

Tiburon's compiler currently supports most of the Verilog-A language, including

- User-defined analog functions
- Event operators
- Mathematical operators
- Delay operator
- Genvars.

Tiburon's solution uses two main components. The first is a compiler that compiles Verilog-A source code creating dynamically-linkable object libraries. The second component is a simulator-specific RTE that provides a consistent interface to various commercial simulators. The architecture is shown below.



TDA compiler creates modules that can operate with all simulators (assuming same OS, otherwise the same source code would generate independent modules).

The compiler and simulator-specific RTEs work together, supporting a complete range of analysis types.

As a demonstration, many standard models have been, or will be, implemented in Verilog-A.

Verilog-A offers a natural and precise method for model developers to describe their model behavior. The developer does not need to be concerned with simulator-specific issues such as matrix stamps, derivatives, and noise correlation matrices.

The appendix shows how the Gummel-Poon model can easily be written in Verilog-A. A similar implementation in SPICE requires thousands of lines of C-code, much of it irrelevant to the model.

Many models, such as the BSIM series of models, require even more code (~40k lines of C). Tiburon's compiler and RTE allow such complex models to be implemented in Verilog-A, while retaining the speed of hand-coded C/C++ code. To demonstrate this capability, the BSIM3 is currently being ported into Verilog-A and the RTE has been added to SPICE 3F5.

The SPICE integration was designed so that netlists using Verilog-A models have a consistent SPICE syntax and can share existing built-in model cards. This allows simple comparison of performance.

Shown here is the netlist for a BSIM3 DC-IV curve.

```

Test BSIM3

.load ../../veriloga/spice/bsim3.cml

VCC Drain 0 5.0
VG Gate 0 0.0
.dc VG 0 5.0 0.1

aM2 bsim3 Drain Gate 0 0 aVerilogDevice w=100 l=1
nrs=10 nrd=10

M1 Drain Gate 0 0 aVerilogDevice w=100 l=1 nrs=10
nrd=10

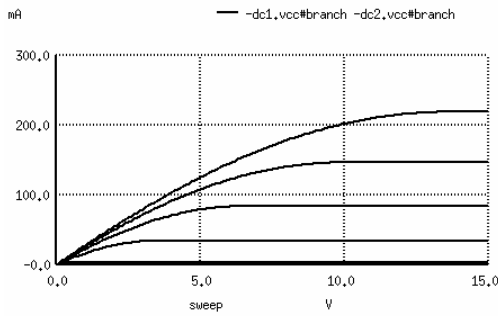
.model aVerilogDevice nmos level=8 rsh=1
tox=1.00000E-08 xj=1.00000E-07 lint=8.195860E-08
wint=-1.821562E-07 vth0=-.86094574 k1=.341038
k2=2.703463E-02 k3=12.24589 dvt0=.767506
dvt1=.65109418 dvt2=-0.145 nlx=1.979638E-07
w0=1.1e-6 k3b=-2.4139039 vsat=60362.05
ua=1.348481E-09 ub=3.178541E-19 uc=1.1623e-10
rdsw=498.873 u0=137.2991 prwb=-1.2e-5 a0=.3276366
keta=-1.8195445E-02 a1=.0232883 a2=.9 voff=-
6.623903E-02 nfactor=1.0408191 cit=4.994609E-04
cdsc=1.030797E-3 cdsbc=2.84e-4 eta0=.0245072 etab=-
1.570303E-03 dsub=.24116711 pclm=2.6813153
pdiblc1=4.003703E-02 pdiblc2=.00329051 pdiblc3=-
2.0e-4 drout=.1380235 pscbel=0 pscbe2=1.0e-28
pvag=-.16370527 prwg=-0.001 ags=1.2 dvt0w=0.58
dvt1w=5.3e6 dvt2w=-0.0032 kt1=-.3 kt2=-.03 prt=76.4
at=33000 ute=-1.5 ual=4.31E-09 ubl=7.61E-18 ucl=-
2.378e-10 kt1l=0 wr=1 b0=1.0e-7 b1=1.0e-7 dwg=5e-8
dwb=2e-8 delta=0.015 cgd1=1e-10 cgs1=1e-10 cgbo=1e-
10 cgd0=0.4e-9 cgso=0.4e-9 c1c=0.1e-6 c1e=0.6
ckappa=0.6

.end
  
```

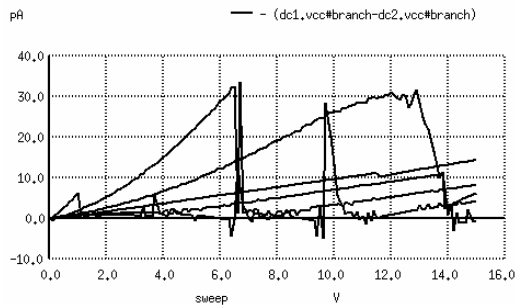
The `load` directive instructs the simulator where to find the Verilog-A compiled library, the new component identifier `A` is added to denote a Verilog-A model, otherwise the netlist is unchanged.

The philosophy of Verilog-A is to only ask the model developer to describe the model behavior, without worrying about simulator-specific issues such as model cards and instances. Both the Verilog-A model (M2) and the SPICE version (M1) point to the same model card; however, the Verilog-A code does not differentiate between an instance and a model card. Internally, the compiler and RTE manage dependencies for optimum efficiency.

The compiled Verilog-A model provides the same numerical results as its native SPICE equivalent. IV curves generated by both models using this netlist overlay exactly as shown below.



The difference is less than 40 pA out of 100 mA and is set by simulation convergence tolerance parameters. The differences are shown in the plot below:



Tiburon Design Automation will continue to port popular compact device models as a means to demonstrate the capabilities of Tiburon's Compiler and RTE.

Appendix A

A slightly compressed version of the SPICE Gummel-Poon BJT model that illustrates the main attributes of

```
'include "const.va"
'include "std.va"

module bjt(c,b,e,s);

  inout      c,b,e,s;    // external nodes
  electrical c,b,e,s;    // external nodes
  electrical ci,bi,ei;    // internal nodes

  parameter real AREA = 1 exclude 0;
  parameter real AF = 1.0 from (0.0:inf); // Flicker noise exp
  parameter real BF = 1.0 from (0.0:inf); // Ideal max fwd beta
  ...
  parameter real XTI = 3.0 from [0.0:inf]; // IS temperature coefficient
  real Vbe, Vbc, Vbs, Vbx, Vce, Vci, Vt, Vbn, Vbcx, Vbci;
  real Ib, Ibe, Ibe1, Ibe2, Ibc1, Ibc2, Irb, Ire, Irc, Iepi;
  ...
  real F1, F2, F3;

// Analog functions
analog function real K;

  input kV, gamma, Vth;
  real kV, gamma, Vth;
  begin
    K = sqrt(1+gamma*exp(kV/Vth));
  end

endfunction

analog begin

  Vt = $temperature();
  Vbe = V(bi, ei);
  ...
  Vbn = V(b, ci);
  Ibe1 = IS * (exp(Vbe/(NF*Vt)) - 1);
  Ibc2 = ISC * (exp(Vbc/(NC*Vt)) - 1);
  Ib = AREA * (Ibe1/BF + Ibe2 + Ibc1/BR + Ibc2);
  Kq1 = 1/(1-Vbc/VAF - Vbe/VAR);
  Kq2 = Ibe1/IKF + Ibc1/IKR;
  Kqb = Kq1 * (1 + pow(1 + 4*Kq2, NK))/2;
  if (IRB == 1e9 || IRB == 0)
    Rb = (RBM + (RB - RBM)/Kqb)/AREA;
  else if (IRB > 0) begin
    x = sqrt((1 + (144/PI_SQUARED) * Ib/(AREA * IRB)) - 1) / (24 /
PI_SQUARED) * sqrt(Ib/(AREA*IRB));
    tanx = tan(x);
    Rb = (RBM + 3 * (RB - RBM) * (tanx-x)/(x * pow(tanx,2)))/AREA;
  end // if (IRB > 0)
  // Extrinsic
  if (Vbcx <= FC * VJC)
    Qjcx = CJC * (1 - XCJC) * (VJC - Vbx) * ( pow(VJC, MJC) -
pow(VJC - Vbx, MJC) ) - VJC/(MJC - 1);
  else
    Qjcx = CJC * (1 - XCJC) * F1 + CJC * (1 - XCJC)/F2 * (VJC -
VJC + MJC * Vbx)/VJC - F3 * FC * VJC - MJC * FC * FC * VJC/2;
  ...
  Qbc = TR * Ibc1 + Qjci;

  I(bi,ei) <- AREA * Ibe1/BF + AREA * ddt(Qbe);
  I(ci,ei) <- AREA * ((Ibe1 - Ibc1)/Kqb);
  I(e,ei) <- AREA * Ire;
  ...
  I(c, cx) <- white_noise(4 * P_K * $temperature() / (RC * AREA), "thermal");
  I(bi, ei) <- flicker_noise(KF * pow(I(bi), AF), AF, "flicker");
end
endmodule
```

Parameters defined over range. Values can be automatically excluded.

Analog functions allow modular programming

Main code is very C-like.

Currents, charge, and noise contribution is simple.