MicroSim Application Notes

MicroSim Corporation
20 Fairbanks
Irvine, California 92618
(714) 770-3022

Version 7.1, October, 1996.

MicroSim Trademarks

Referenced herein are the registered trademarks used by MicroSim Corporation to identify its products. MicroSim is the exclusive owner of: “MicroSim,” “PSpice,” “PLogic,” “PLSyn” and “Polaris.”

Trademarks of MicroSim include: “DesignLab” and “Setting the Standard for Desktop EDA.”

Additional marks of MicroSim include: “StmEd,” “Stimulus Editor,” “Probe,” “Parts,” “Monte Carlo,” “Analog Behavioral Modeling,” “Device Equations,” “Digital Simulation,” “Digital Files,” “Filter Designer,” “Schematics,” ”MicroSim PCBoards,” “PSpice Optimizer,” and variations thereon (collectively the “Trademarks”) are used in connection with computer programs. MicroSim owns various trademark registrations for these marks in the United States and other countries.

All Other Trademarks

MicroSoft, MS-DOS, Windows, Windows NT and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Exchange and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

ShapeBased is a trademark and SPECCTRA and CCT are registered trademarks of Cooper & Chyan Technologies Inc. (CCT). Materials related to the CCT SPECCTRA Autorouter have been reprinted by permission of Cooper & Chyan Technology, Inc.

Xilinx is a registered trademark of Xilinx Inc. All, X- and XC- prefix product designations are trademarks of Xilinx, Inc.

EENET is a trademark of Eckert Enterprises.

All other company/product names are trademarks/registered trademarks of their respective holders.

Copyright Notice

Technical Support

Internet	Tech.Support@MicroSim.com
Phone	(714) 837-0790
FAX	(714) 455-0554
AUTOFAX	(714) 454-3296

Sales Department

Internet	Sales@MicroSim.com
Phone	800-245-3022

Contents

Preface

Preface

This collection of Application Notes has been specifically compiled for users of the MicroSim applications. It contains articles to show you how a particular task can be accomplished using MicroSim‘s applications and examples that demonstrate a new or different approach to solving an engineering problem.

Many of these Application Notes have been compiled as a result of customer requests. They provide a variety of real-world applications and practical examples for many of the functions and features found in MicroSim applications. Some of them have been taken from the quarterly issues our newsletter, *MicroSim Source*.

Analog Behavioral Modeling

MicroSim Corporation Newsletter, October 1989

Let's take a look at examples of how the *Analog Behavioral Modeling* feature of PSpice can cope when generic SPICE fails.

First, let's say you need to create a signal whose voltage is the square root of another signal's voltage. Calculating square roots is simple, even for SPICE, through the use of a feedback circuit. However, this technique fails if the reference signal ever goes negative. In this case the functional form of Analog Behavioral Modeling works nicely:

```
Esqrt out_hi out_lo value={sqrt(abs(v(input)))}
```

This takes the absolute value of the ground-referenced signal "input" before evaluating the square-root function (you could also use a floating signal-pair by replacing v(input) with v(in_hi)-v(in_lo) or v(in_hi,in_lo), for example). The absolute-value function is a nonlinear function difficult to perform in generic SPICE.

We can also introduce ideal nonlinearities using the table lookup form of Analog Behavioral Modeling. For example, the one-line, ideal opamp model:

```
Eamp out 0 table {200K*(v(in_hi)-v(in_lo))}=
+ (-15,-15) (15,15)
```

has high gain, but its output is clamped between ±15 volts. The input to the table is the differential gain formula, but the lookup table has only two entries: so the output of the table is interpolated between these two endpoints and clamped when the input exceeds the table's range. This is a convenient use of the table lookup form, which is not available in generic SPICE.

Small systems of behavioral models are easy to design, also. For example, a true-RMS circuit can be built by decomposing the RMS function: (i) square the signal, (ii) integrate over time, and (iii) take the square-root of the time average. These three operations can be bundled in a tiny subcircuit for use as a module:

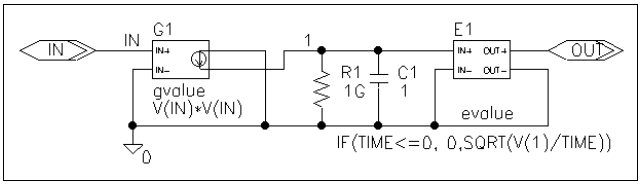


Figure 1 *RMS subcircuit*

```
.subckt RMS in out
G1 0 1 VALUE {V(IN)*V(IN)}
C1 1 0 1
R1 1 0 1G
E1 out 0 VALUE {IF(TIME<=0, 0, SQRT(V(1)/TIME))}
.ends
```

The current source, G1, squares the signal, which is then integrated in the capacitor. The voltage on the capacitor is time averaged, and the square-root is taken (the resistor is a dummy load that satisfies the SPICE algorithms). The voltage source E1 shows that the value of simulated “time” is available in Analog Behavioral Modeling, and may be used as a variable in a formula. Notice that the if-than-else function is used. If time is less than or equal to zero then the output of E1 is sqrt(v(1)/time). This prevents convergence problems. When sqrt(v(1)/time) is evaluated at time = 0.

Parameter passing into subcircuits also works with Analog Behavioral Modeling, which makes your models more flexible. Here is a small system that is a voltage follower with hysteresis, which would be useful in simulating, say, a mechanical system with gear backlash:

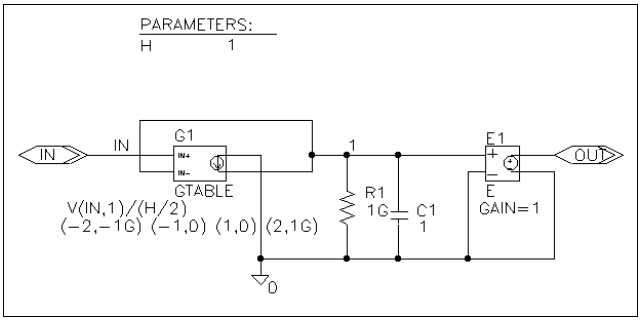


Figure 2 *Hysteresis subcircuit*

```
.subckt HYS in out params: H=1
G1 0 1 TABLE {V(IN,1)/(H/2)} (-2,-1G) (-1,0) (1,0) (2,1G)
C1 1 0 1
R1 1 0 1G
E1 out 0 1 0 1
.ends
```

The parameter H defines the size of the hysteresis, and is used in the formula input to the table. The combination of the formula and table defines a dead-band outside of which the output follows the input with an offset of H/2. The capacitor serves as memory for the circuit and is nearly ideal except for the DC-bias resistor, which provides a droop time constant of one billion(!) seconds. The voltage follower, E1, prevents output loading problems. E1 could also have gain representing the gear ratio of a mechanical system; then voltage would represent the total turn angle of each gear, and H the amount of angular backlash.

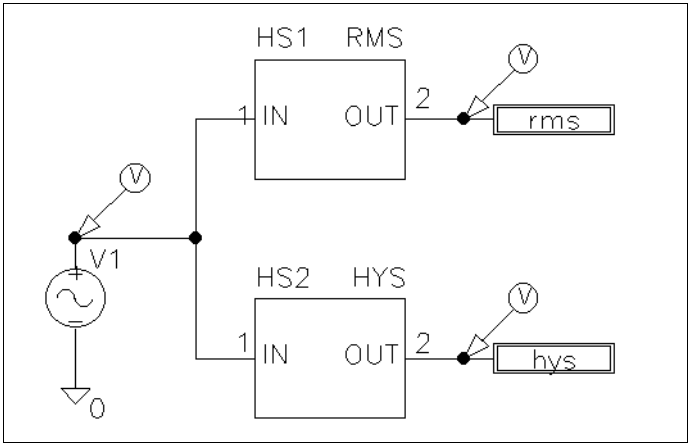


Figure 3 *Circuit using RMS and hysteresis subcircuits*

```
.param H=1
*
V1 in 0 SIN (0 1 1)
Xrms 1 rms RMSXhys 1 hys HYS
param: H=1
*
.tran 10m 1
.end
```

A 1 Hz sine wave was used for the stimulus to the RMS and HYS circuits.

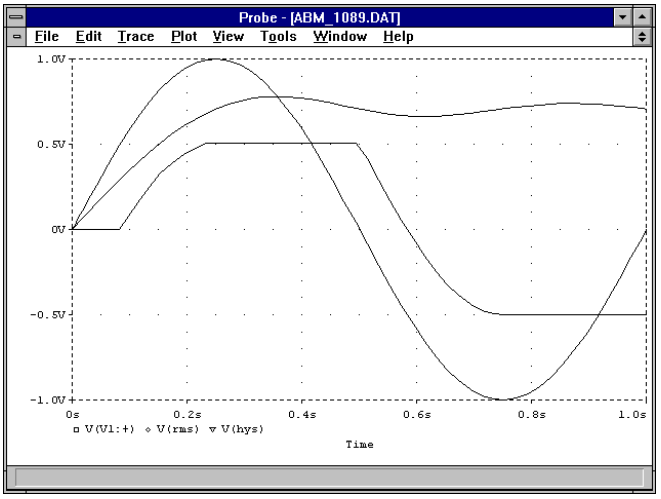


Figure 4 *Output from RMS and HYS circuits*

In Figure 4 we see a Probe plot of the input, and the outputs from each circuit. Note that the RMS circuit outputs the well-known result of 0.707 volts after one input cycle, while the HYS circuit lags the input by a half volt in each direction for a total hysteresis of one volt. Perhaps these examples will give you ideas for other functions which would be “most difficult” to create with generic SPICE.

Reprinted from the article, “Analog Behavioral Modeling using PSpice,” by I. M. Wilson, from the *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, 1989, IEEE.

Analog Behavioral Modeling Using PSpice

Abstract-Modeling new device types requires more than the polynomial sources provided by SPICE. PSpice extensions allow arbitrary equations and/or table lookup. These extensions are also useful for black-box system level modeling. Examples are presented of both types of behavioral modeling.

Introduction

Behavioral Modeling is the process of developing a model for a device or system component from the viewpoint of externally observed behavior rather than from a microscopic description.

Two important applications of Behavioral Modeling in the domain of analog simulation are: modeling new device types; and black-box modeling of complex systems.

This paper discusses extensions made to PSpice to support these applications and presents detailed examples of each.

Extending Simulators

Analog simulators generally contain built-in models for a limited number of devices. Simulating a circuit containing a device not contained in the intrinsic set requires extending the simulator in some way. There are three ways to extend SPICE and related simulators: use a polynomial controlled source, modify the simulator code to add a new model, or build a macro model. PSpice Behavioral Modeling provides an alternative way to extend simulator capability.

SPICE Polynomial Controlled Sources

SPICE includes controlled voltage and current sources (E, F, G, and devices). These have a POLY variant that define the output as a polynomial function of one or more input voltages or currents. Each polynomial term is weighted by a coefficient.

This approach can be used to represent simple ideal devices such as multipliers, squarers, etc. In opamp-type feedback loops it can be used to implement buffers, square root devices and so on. Modeling more complex devices usually requires a combination of curve-fitting and macro modeling techniques.

Polynomial approximations work best when the function modeled satisfies the following criteria: it must be smooth (the function and its derivatives must be continuous); and it must go to plus or minus infinity with the independent variable(s). Functions that do not behave in this fashion may be approximated over a restricted range of values. It may be impossible to get a usable model of a function whose inputs span a large range and where the output must be accurately specified in a small region.

Code Modification

Any simulator can be extended to include new types of device by writing code similar to that already in place for the basic SPICE set. A few vendors provide a mechanism for users to do this. Languages are typically Fortran or C.

There are significant problems with this approach. The environment in which SPICE device code operates is far from simple. Expert programming skills are required both to ensure that the additional code operates as expected and that the simulator continues to operate correctly. Additionally, detailed understanding of the SPICE implementation is required. This is likely to be a feasible approach only in academic environments or device foundries.

Once a simulator has been modified, it is (self-evidently) nonstandard. This is a potential problem if the simulation ever has to be run on another simulator. Additionally, each time a new version of the (unmodified) simulator becomes available, the porting and validation effort will have to be repeated.

Macro Models

A device can be modeled by constructing a macro model using existing primitives. This approach works well for composite devices such as optocouplers. There is typically a one-to- one correspondence between components of the composite device and those of the macro model, although some functions may be abstracted using controlled voltage or current sources.

The approach does not work so well when the device characteristics are given in equation form or as a set of measured values. In these cases it may be necessary to resort to techniques such as synthesizing a log function by converting voltage to current, passing this through an ideal diode and sensing the voltage across the diode. Macro models built using these techniques soon become complex, difficult to maintain, and can be slow and inaccurate.

Functional Approach

The capabilities of a simulator can be extended by including the ability to evaluate expressions which are functions of circuit variables (voltages, currents, simulation time). The microgrammar that defines the language may include constructs such as assignment and explicit control statements (“procedural”); or it may exclude these (“nonprocedural”).

The functional approach works well when device characteristics are known in equation form, and the device is state-free. It is not so useful when only a physical model of device behavior is available or when the device has several internal states.

A significant advantage over code modification is that no changes have to be made when new versions of the simulator are released. A disadvantage is that there is no standardization across the various simulators offering a behavioral modeling capability.

PSpice expressions are nonprocedural. This means that there are no assignments and no if-then-else type constructs. The issue of procedural versus nonprocedural is called religion in software jargon. SPICE syntax is nonprocedural: the input netlist contains facts about device node connections and parameter values. PSpice extensions follow this precedent, for consistency and to meet the requirements of the user base, who are primarily non-programmers.

A summary of PSpice extensions, together with some simple examples, is given below.

Time domain

- arbitrary expressions, can include constants, parameters, node voltages & currents, TIME, math functions including log, exp, and trig

  ```
  E1 1 0 VALUE {sin(twopi * fc * TIME)}
  ```

- table lookups; value of a controlling expression is linearly interpolated in a table

  ```
  E2 2 0 TABLE {1.0 + v(4)} (0 0)
  + (0.1 0.2) (0.2 0.25)
  ```

Frequency domain

- Laplace expressions, including constants, parameters, and math functions in S including log, exp, and trig

  ```
  E3 3 0 LAPLACE {v(5)+v(6)} {1/(1 + t1*S)}
  ```

- table lookups; magnitude and phase are linearly interpolated in a table

  ```
  E4 4 0 FREQ {v(7)} (0 0 0)
  + (5k 0 -5760) (6k -60 -6912)
  ```

Device Modeling

Modeling Tunnel Diode

The tunnel diode has frequently been used as an example of SPICE device modeling using polynomials. The static current/voltage characteristic of the device contains a region of negative dynamic resistance. The transitions from positive to negative resistance and back again are smooth - there are no discontinuities in slope and the device does not exhibit hysteresis. The device is only operated in the vicinity of the negative resistance region; typically a span of one or two volts.

These attributes make the device eminently suitable for polynomial representation (it is no coincidence that this device has been used for illustration so often in the past).

Main characteristics of a tunnel diode current/voltage curve are peak voltage and current (Vp, Ip), valley voltage and current (Vv, Iv) and projected peak voltage (Vpp). Specific device parameters for this example:

```
Vp = 50mv; Ip = 4.2ma; Vv = 370mv;
+ Iv = 370ua; Vpp = 525mv
```

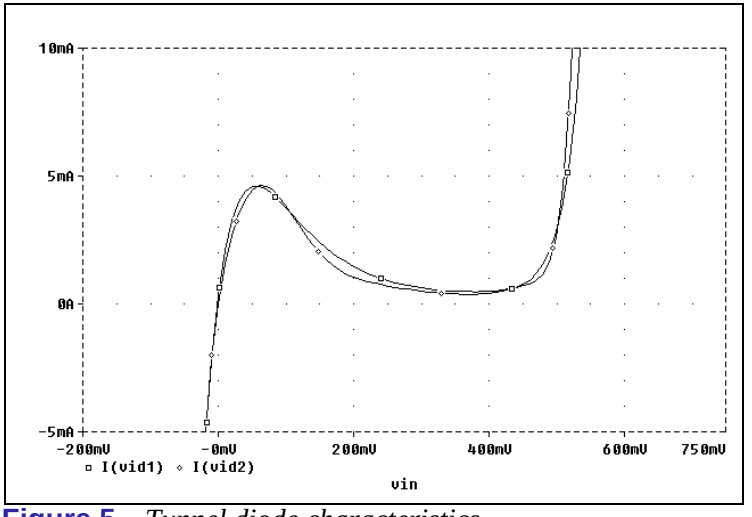


Figure 5 *Tunnel diode characteristics*

Using Polynomials: Reference [1] provides a set of coefficients for a 12th-order (!) polynomial of one dimension giving current as a function of voltage between the anode and cathode of the device:

```
Gtd a k POLY(1) a k
+  -3.95510116e-17
.... etc ....
+  +1.68527934e+05
```

The coefficients were obtained by taking a set of (x,y) values from a manufacturer's data sheet and using a curve-fitting program to perform a linear regression on the coefficient values. The resulting static current/voltage characteristic is shown in Figure 5.

Using Behavioral Modeling: Current flow in a tunnel diode is due to three distinct effects [2]: thermal current (analogous to a conventional diode), tunnel current (due to direct tunneling) and excess current (due to indirect tunneling). Writing these three terms in PSpice's extended syntax:

```
Gthermal a k VALUE
+ {Ip*exp(- Vpp/ Vt)*(exp(v(a,k)/Vt)-1)}
Gtunnel a k VALUE {Ip*(v(a,k)/Vp)*exp
+ (1- v(a,k)/Vp)}
Gexcess a  k VALUE {Iv * exp(v(a,k) - Vv)}
```

where Vt is 26 mv at 300 K.

The resulting current/voltage characteristic is shown in Figure 5.

Parameterization

Consider modeling devices with parameters different from the example set used above, for example to produce a library of devices for general use. The polynomial approach would require a set of coefficients for each distinct device. This becomes impractical for anything more than a handful of devices. It may be possible to define a "*generic*" tunnel diode device and map

inputs and outputs appropriately, but it is not intuitively obvious what mapping to use.

The functional approach is much better suited to setting up libraries of devices owing to the presence of parameters in the equations. To model a device with a different value for Vp, for example, only that parameter's value needs to be updated. Note also that in the basic equations above, the temperature dependence is included (Vt). A subcircuit definition can be used to package the tunnel diode model and its parameters:

```
.SUBCKT TD a k PARAMS: Vp=50mv Ip=5ma Vv=0.3v Iv=0.3ma
+ Vpp=500mv
Gthermal a k ....
Gtunnel  a k ....
Gexcess  a k ....
.ENDS
```

usage:

```
X1 4 5 TD PARAMS: Vp=55mv ;
+ override 50mv default
```

For more difficult devices, where straightforward equations may not be available, or where the relationship between the parameters in the equations and data sheet values for the device is not obvious, a lookup table approach may be used. Where possible, a normalized device characteristic can be modeled by the table, with parameterized expressions used to transform inputs and outputs.

System Modeling

Behavioral Modeling as Abstraction

In the early stages of system design, the emphasis is on high-level issues rather than on low-level details. Behavioral models allow systems to be simulated with reduced complexity and with improved computational efficiency.

A simple example is using a controlled source as a gain block rather than using a complete operational amplifier model:

```
Eamp 1 0 TABLE {1e6*(v(pos,neg))}
+ (-15 -15) (+15 +15)
```

PSpice extensions allow black-box simulation of many high-level circuit elements. The use of arbitrary expressions, lookup tables and Laplace formulations are powerful tools.

Modeling a Phase-Locked Loop

To contrast the high-level and low-level modeling approaches, consider a simple phase-locked loop (Figure 6). Phase locked loops contain three major components: a voltage-controlled oscillator (VCO); a Phase Detector which compares the output of the VCO with the input (target) signal to derive an error signal; and a Loop Filter. The inverted output of the Loop Filter becomes the controlling voltage for the VCO, thus forming a negative feedback control loop.

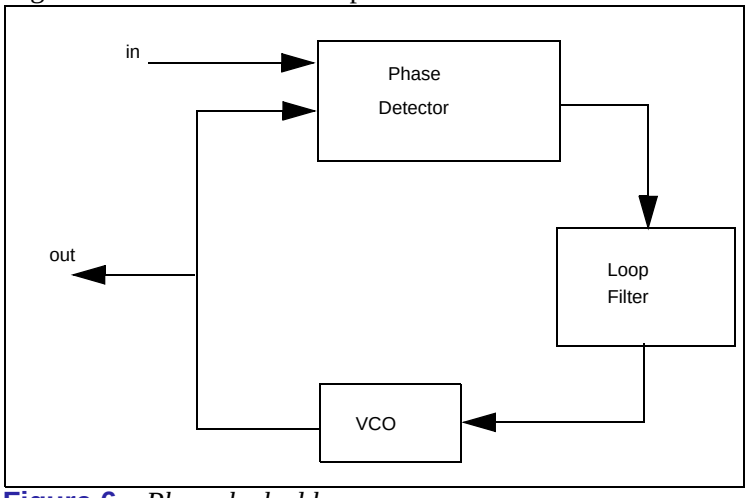


Figure 6 *Phase-locked loop*

Mathematical Description: The general time-domain equation for a phase-locked loop can be written as:

$$\phi o' = K \sin[\phi i(t) - \phi o(t)] \Theta f(t)$$

The input signal yi and the VCO output signal yo are given by:

$$yi(t) = A \sin[wt + \phi i(t)] \quad yo(t) = B \sin[wt + \phi o(t)],$$

The symbol Θ represents convolution, and f(t) is the impulse response of the filter.

This nonlinear differential equation is not solvable in the general case. Approximate solutions may be found when the equation is linearized. The typical case where the loop filter is a simple RC network when linearized gives rise to a second-order linear differential equation.

Behavioral Model: Each of the three main components of the PLL can be expressed succinctly in PSpice's extended Behavioral Modeling syntax.

The Phase Detector is a multiplier with the output range constrained to [-1,+1]. This is written as a TABLE device, with the controlling expression being the product of the input voltages and a 2-element lookup table being used to limit the output:

```
Epd 3 0 TABLE {v(1)*v(2)} (-1 -1)(+1 +1)
```

The VCO is described as a sinusoidal function of time with an additional term controlling the phase:

```
Evco 5 0 VALUE {sin(2*pi*fc*TIME + v(4))}
```

The Loop Filter is conveniently described by giving its Laplace Transform, using s-domain notation. For example, for a one-pole filter with phase lead correction:

```
Elpf 7 0 LAPLACE {v(6)} {(1+t2*s)/(1+t1*s)}
```

A complete phase-locked loop description consists of these three "devices," an integrator and a few dummy loads.

Circuit Level Model: A model of the same PLL was developed using bipolar transistor circuits.

The VCO was an astable multivibrator with the charging current proportional to the VCO control voltage. The multiplier was a double-balanced modulator using 6 BJTs. The Loop Filter consisted of two resistors and a capacitor.

A complete description consists of these circuit fragments, together with power supplies, bias resistors, bypass capacitors, etc.

Comparing the two Approaches: Table 1 contrasts the two approaches to modeling the PLL.

Compared with the Circuit model, it took about 20% of the time to develop the Behavioral model, and the transient analysis ran in about 4% of the time.

The time required to run the analysis is significant. The Behavioral model allows many more analyses to be run in a given time, permitting a higher degree of design refinement and/or test.

Table 1 *Comparison of Modeling Approaches*

Model	Behavior	Circuit
Devt. Time	1 day	5 days
Simulation Time	24 sec*	606 sec*
* lines	9	43

* run times measured on a Sun 4/110

Future Challenges

Modeling state behavior

Many real devices exhibit two or more stable states. Transitions between these states occur under well-defined circumstances. For example, consider a spark gap. This has two persistent states. An arc may be present, in which case the device is in its ON (low resistance) state. Or there may be no arc, in which case the device is in its OFF state. A combination of applied voltage and dV/dt causes the device to transition from its OFF to its ON state, via a transitory “arc forming” phase. If the arc current falls below a holding value, the device turns OFF, via an “arc extinguishing” phase.

The question arises, how to model this kind of device with SPICE-based simulators. Macro models are difficult to construct. Representing the state variable requires some component with memory. Possibilities include hysteresis blocks and digital primitives (if a mixed-mode simulator is available). Neither of these offers an easy or elegant solution.

Let us consider how procedural and nonprocedural Behavioral Modeling approaches might look. The procedural approach would assign device state to some variable local to a device instance (e.g., STATE).

Code such as the following might then express the state transition from state 0 (OFF) to state 1 (ARC-FORMING):

```
if(STATE == 0 && f(V, Vdot) > VTON) {
Tarc = time          ; -- remember when
 STATE = 1           ; -- new state
}
```

and to represent the device's I/V characteristic:

```
if(STATE == 1) {
Iout = Vin / (RON + (ROFF-RON)* (1 -
+ (time - Tarc)/TON))
}
```

An attractive nonprocedural alternative would be to define a state machine for the device. Each state is associated with a functional form giving the device's behavior in that state. A set of conditions allow the state transitions to be specified. Here are fragments of what such a device representation might look like:

```
Eesa  5  0 STATE (4)
+ {v(in) / (RON + (ROFF-RON)* (1-(time- TENTRY)/TON))}
+ 0,1: {f(V,Vdot) > VTON}
```

The second line gives the State 1 behavior of the device. The third line gives a transition (0->1) and a condition to be met for this transition to occur.

Note that although the substance of the two descriptions is the same, the nonprocedural form hides most of the housekeeping operations (such as assigning a new value to the state variable). This is clearly a desirable state of affairs.

Managing Convergence and Time-Step Control

Behavioral models are not restricted to well-behaved devices like the tunnel diode. Devices with abrupt behavior can be readily modeled using the TABLE forms and the logarithmic and exponential functions. Convergence control to ensure that the simulator takes small enough steps may be necessary in these models. For TABLE devices this can be implemented by setting the internal non-convergence indicator if an attempt is made to skip from a section of the TABLE device to another section that is not an immediate neighbor.

For other forms the proposed output at a given time step could be compared with the previous output and absolute and/or relative delta criteria applied. If the test failed, the time step would be reduced. The criteria could be specified per device, with global default values.

Controlling the time step may be necessary not only for convergence, but also from sampling considerations. Interpolation schemes are used for graphical display of the simulation results. The time step must be constrained so that voltage/current changes are within the scope of the interpolator.

It is not possible to deduce the frequency domain behavior of devices specified by arbitrary expressions. There is a risk of aliasing occurring if the initial and subsequent choices of timestep produce "*reasonable*" (but subsampled) values of a periodic function. For example, suppose there is a 1 MHz source in the circuit and the initial time step is chosen as 10 µS. If each subsequent time step is also 10 µS, the apparent value of the source will be 0.

In practice, this kind of subsampling will be readily noticed. It can be avoided by manually setting the step ceiling.

Summary

Analog Behavioral Modeling has two immediate, highly practical uses:

- It can be used to extend the capability of an existing simulator to model new devices and sources, without modifying the simulator's source code.
- It can also be used to design systems at an abstract level, ensuring that the concepts are correct, before proceeding with the detailed circuit-level design.

SPICE's syntax is nonprocedural. This approach has proven to be convenient and powerful. Presenting Analog Behavioral Modeling as functional or state-machine forms fits in naturally with existing SPICE usage and is to be preferred over procedural, programmatic, extensions.

Acknowledgments

The author would like to thank his colleagues at MicroSim for their helpful suggestions and creative ideas.

References

[1] PSpice User's Guide, Appendix B. (See Note below)

[2] S. M. Sze, *Physics of Semiconductor Devices*, Wiley & Sons, 1981, ch. 9, p529.

Note *Appendix B no longer exists, therefore refer to the example circuit below.*

```
TDO - TUNNEL DIODE OSCILLATOR
VBIAS 0 2 -120MV
LS    2 1  2.5UH
CS    1 0  100PF
GTD   1 0  POLY(1)  1 0
+   -3.95510115972848E-17
+   +1.80727308405845E-01
+   -2.93646217292003E+00
+   +4.12669748472374E+01
+   -6.09649516869413E+02
+   +6.08207899870511E+03
+   -3.73459336478768E+04
+   +1.44146702315112E+05
+   -3.53021176453665E+05
+   +5.34093436084762E+05
+   -4.56234076434067E+05
+   +1.68527934888894E+05
.DC VBIAS 0 -600MV -5MV
.PLOT DC I(VBIAS) (0,5MA)
.TRAN 5NS 500NS 0 5NS
.PLOT TRAN V(1)
.OPT ACCT LIST NODE OPTS NOPAGE
.WIDTH IN=80 OUT=80
.END
```

Analyzing Amplifier's Settling Time

The Design Center Source newsletter, originally titled "Using Performance Analysis to Analyze Your Amplifier's Settling Time"

Settling time is a key performance parameter for an amplifier. The standard simulation methodology to test for this parameter steps the input voltage over the relevant input range and measures the time taken for the output to settle to some defined value close to its steady state value. The defined value depends upon the resolution of the system. For example, a 12 bit system in a range of ten volts will probably need to settle to within 1.2 mV (1/2 lsb) of its final value.

During the design of such an amplifier, many parameters are varied to optimize the settling time. It can become extremely tedious moving along the response curves to find the exact settling time. Performance Analysis (available in Probe version 5.0 and later) by means of "goal function" definition, can facilitate this investigation. To demonstrate the implementation of the relevant goal functions, the settling time of an LF411 in unity gain configuration will be computed as a function of load capacitance. The circuit file for the demonstration is shown below.

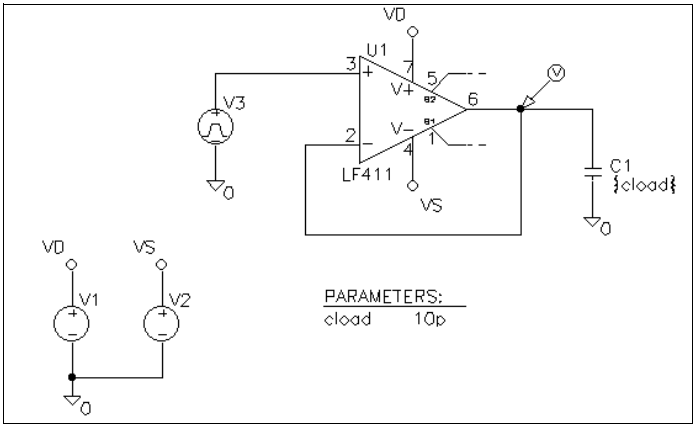


Figure 7 *Amplifier schematic*

```
* step response of LF411
.options reltol=.0001
.param cload=10p
.probe v(2)
.step param cload 100p 700p 7p
.lib linear.lib
vd vdd 0 15
vs vss 0 -15
v1 1 0 pulse (0 1 .1u .01u 1u 1 2)
x1 1 2 vdd vss 2 lf411
cload 2 0 {cload}
.tran 1ns 5ms
.end
```

Figure 8 shows the response of the system for three different load capacitors, to a one volt step at the input. The method normally used to estimate the settling time from these curves is fairly straightforward. We simply start at the end point and scan backwards along the curve until we find a point where the response curve intersects the defined settled value

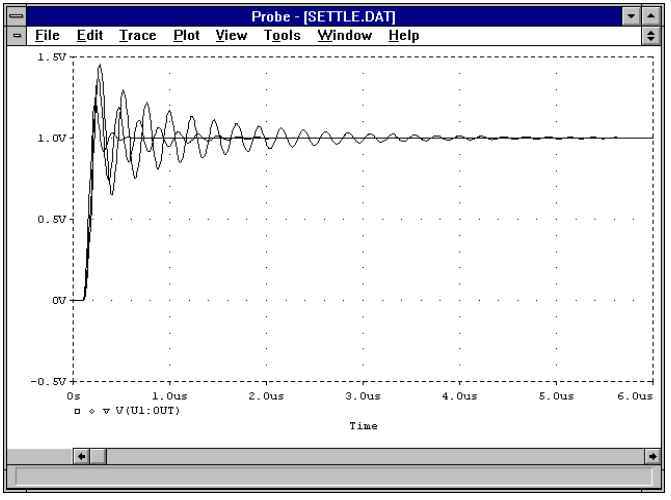


Figure 8 *System response for three load capacitor values*

The goal function shown below demonstrates the backwards search from the end of the run to where the defined value (1.01 volts in this case) intersects the curve.

```
settle(1) = x1
 {
 1|
 search backward /End/ level(1.01) !1
 ;
}
```

Using this goal function, we can now examine the settling time versus load capacitance. The curve in Figure 9 shows the trend, but several inconsistent discontinuities are noticeable. To appreciate where the discontinuities come from, we must first visualize the oscillation which intersects the defined level. With increasing load, this oscillation will increase in amplitude as will the cycle after it. At some point, however, the succeeding cycle will grow enough to intersect the defined level, giving a jump of half the oscillation period.

To offset this effect, we first detect the peaks of the cycles in the neighborhood of the defined value. We can then fit a polynomial to these points and use this to predict the settling time. The goal functions to implement this are shown below as S1, S2, and S3, which are three components of the Lagrangian polynomial. In the example shown, the Lagrangian components are evaluated at a defined level of 1.01, which, when added together, will produce the settling time curve to 10 mV.

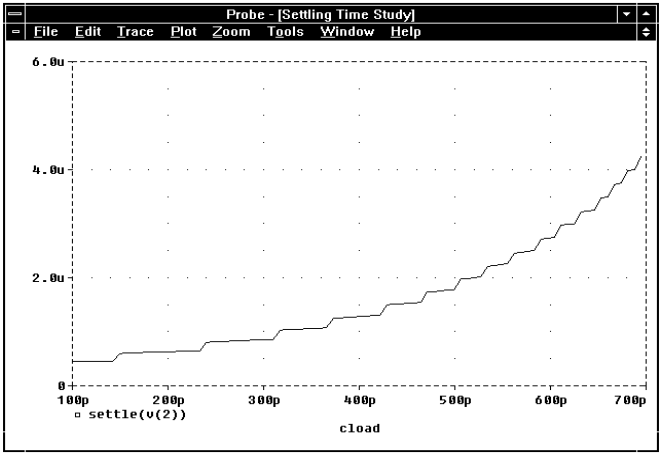


Figure 9 *Settling time curve produced by an inadequate goal function*

```
s1(1) =(1.01-y2)*(1.01-y3)*x1/((y1-y2)*(y1-y3))
{
1|
search backward /end/ LEVEL(1.01)
search forward peak !1
search backward peak !2
search backward peak !3
;
}

s2(1) =(1.01-y1)*(1.01-y3)*x2/((y2-y1)*(y2-y3))
{
1|
search backward /end/ LEVEL(1.01)
search forward peak !1
search backward peak !2
search backward peak !3
;
}


s3(1) =(1.01-y1)*(1.01-y2)*x3/((y3-y1)*(y3-y2))
{
1|
search backward /end/ LEVEL(1.01)
search forward peak !1
search backward peak !2
search backward peak !3
;
}
```

Different settling time curves can be produced by modifying the goal functions for different defined levels. Figure 10 shows the curves for the settling time to 20 mV, 10 mV, and 5 mV, by setting the defined level at 1.02, 1.01, and 1.005, respectively. Note that both the “marked point expression” and the LEVEL function are modified for each distinct defined level. In Figure 10, each settling time curve has been defined as a macro expression. For instance:

```
settle_10mV = s1(v(2)) + s2(v(2)) + s3(v(2))
```

To smooth the curves even further, a more appropriate function could be defined using goal functions.

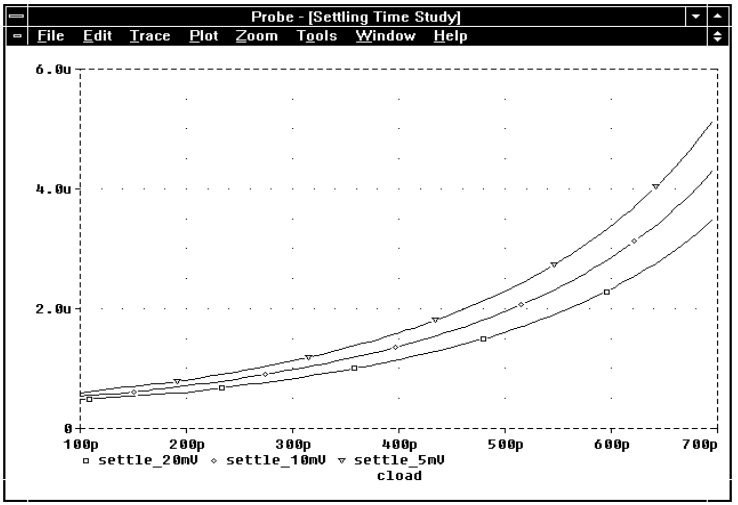


Figure 10 *Smoothed settling time curves to 20 mV, 10 mV, and 5 mV*

Analyzing Ground Bounce in High Speed Designs

The Design Center Source newsletter, January 1994

Introduction

Advances in device technology are progressing at such a rate that the functionality of integrated circuits (ICs) doubles every year. As the functional blocks that comprise digital systems become more complex, so do the printed circuit boards (PCBs) that make up the systems. The evolution of device technology has yielded devices with clock rates in the 50 to 100 MHz range and rise/fall times on the order of 1 to 2 nsec. At these speeds, digital designers can no longer assume that a design is immune to parasitic effects. Transmission line effects and package interconnect parasitics must be taken into account to insure reliable operation. This article examines how package parasitics can affect high speed designs. In particular, we will illustrate how Polaris and PSpice can be used to simulate parasitics and how existing digital library device models can be modified to model package parasitics that can predict ground bounce.

What Is Ground Bounce?

Any integrated circuit can experience ground bounce under the right conditions. As rise times decrease, ground bounce is more likely to occur.

The devices that are connected to the output pin of an IC impart a capacitive load that the output driver must overcome in order to cause the input driver to switch from a low to a high logic level. Conversely, when the output switches from a high to low logic level, the output must discharge the capacitive load on the

input(s) being driven in order for the input to change state. The output driver therefore becomes a current sink, channeling the current to the chip ground. The amount of current channeled through the driver obeys the relation $i = C\,dv/dt$. Inside the chip, the bonding wires that connect the output driver on the chip die to the package pin are inductively coupled to the chip ground. As the current is channeled to the chip ground, a voltage is induced on the chip ground that follows the relationship $v = L\,di/dt$. Therefore, if the rise/fall times are short, the output driver will be forced to sink a large amount of current in a short time. This will induce an equally large voltage on the chip ground causing the ground voltage to *bounce*. Since the output drivers use the chip ground as the reference for a logic low, any outputs that are low at the time will also experience the bounce; the inputs will be similarly affected. The amount of ground bounce experienced follows the relation

$$\Delta V_{GND} = \frac{L_{pkg} \cdot C_{load} \cdot \Delta V \cdot n_{switch}}{t_{rise}^2}$$

where n_{switch} is the number of simultaneously switching outputs, and ΔV is the voltage swing between logic levels. If more than one output is switching at the same time, the effect is magnified. Furthermore, if this induced voltage is large enough, an output that is low can be unintentionally driven high, possibly causing data corruption.

Ground bounce mechanism device output stage with its associated capacitive loading.

The gates driven by a digital output are not the only circuit components that impart a capacitive load on the output driver. The PCB traces, which can be modeled as transmission lines, also contribute to capacitive loading as do the paths that route signals across different layers. Polaris can be used to extract these kinds of transmission line parasitics from the PCB interconnect and to insert them into the simulation netlist so that they can be analyzed with PSpice.

Example Circuit

Figure 11 shows a schematic diagram of the circuit that we will use to demonstrate ground bounce analysis. This circuit uses components from the FAST logic family. In the example circuit, the device of interest is the 74F543 octal bus transceiver which is associated with the F543R symbol. The 74F543 output drivers are capable of sinking large amounts of current (24 mA for the A-side drivers and 64 mA for the B-side drivers). Switching large currents is one cause of ground bounce as will be demonstrated by the simulation.

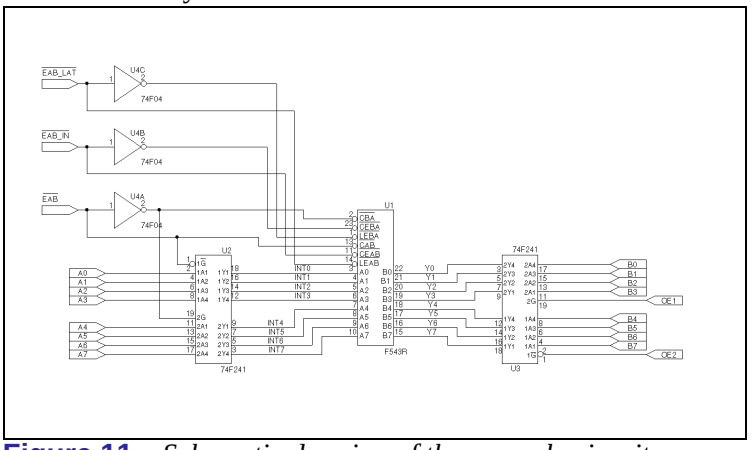


Figure 11 *Schematic drawing of the example circuit*

Creating Package Models and Symbols

The digital device models supplied in the Model Library are device-oriented models rather than package-oriented models. Device-oriented models are desirable for digital simulation because they address the transitive aspects of the device (i.e., does the state of the outputs change in the expected manner for a given input) rather than their analog behavior. However, as rise times and clock speed become faster, more digital designers must consider analog characteristics, such as ground bounce, to insure reliable operation of their circuits.

Since the inductive coupling of the bonding wires to the chip ground contributes to ground bounce, these models must be modified to reflect package inductance. For our example circuit, we must create a subcircuit model for a 74F543 that includes package inductance for the chip power and ground pins. To create the new model, we will first create a new symbol (MY543) referencing the existing 74F543 model available in the Model Library file “dig_3.lib.” We will then use this symbol in a schematic and incorporate the required inductance components (see Figure 12). Using Schematics’ automated Create Subcircuit feature, we can then generate a corresponding subcircuit definition (see Figure 13). Finally, we will create another new symbol (F543R) that will reference the subcircuit definition we just created and be used in the schematic for our example circuit as shown in Figure 11.

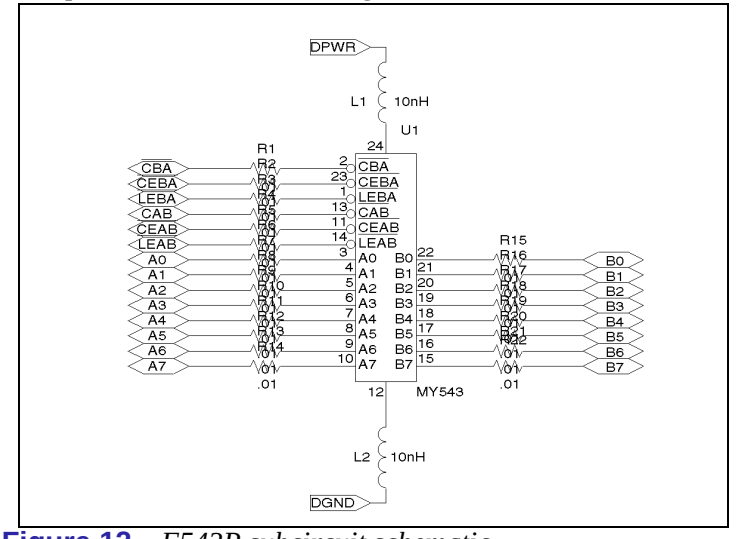


Figure 12 *F543R subcircuit schematic*

We must remove the ipin attributes of the 74F543 symbol so that the power and ground pins will not be connected to the analog/digital power and ground interface defaults. To accomplish this, we will make a copy of the generic 543 symbol found in “dig_3.slb” using the File/Edit Library command to start the Symbol Editor, and the Part/Copy command to make a copy of the 543 symbol which we will name MY543. Using the Part/Pin List command, the PWR and GND pins are modified such that they are no longer hidden pins (by disabling the Hidden check box for the selected pin), thus making them available for use in

the subcircuit schematic. The MODEL attribute is changed to reference the 74F543 model. The TEMPLATE attribute is left unchanged. The Part/Save Changes command is used to save the changes to the symbol definition. The File/Save command is used to save the symbol definition to a Symbol Library file. This file must exist in a directory on the configured library search path so that the symbol is available to the Schematic Editor; or, if this is a new file, the Symbol Editor will offer to automatically configure it.

After the symbol is created and saved, the subcircuit schematic can be generated. The subcircuit schematic is shown in Figure 12. Inductors are added to the power and ground pins to represent the package inductance. For a 24-pin DIP with the V_{CC} and GND pins at pins 24 and 12 respectively, the package lead inductance is 10 nH.

In the schematic, the power and ground pins (DPWR and DGND) are connected to global ports while interface ports are used for the remaining signal pins. Thus, the Tools/Create Subcircuit command creates signal pins for the interface ports and OPTIONAL pins for the global ports. After the subcircuit definition is created (see Figure 13), a symbol can be defined in the Symbol Editor using techniques similar to those used for MY543; this new symbol is named F543R. The MODEL attribute of this symbol must be set to F543R to reference the subcircuit model just created.

```
.SUBCKT F543R CABbar CBAbar CEABbar CEBAbar
+ LEABbar LEBAbar
+ A0 A1 A2 A3 A4 A5 A6 A7
+ B0 B1 B2 B3 B4 B5 B6 B7
+ OPTIONAL: DPWR=$G_DPWR DGND=$G_DGND
+ PARAMS: MNTYMXDLY=0 IO_LEVEL=0

X_U1 $N_0008 $N_0007 $N_0003 $N_0005 $N_0004 $N_0002
+ $N_0009 $N_0010 $N_0011 $N_0012 $N_0013 $N_0014 $N_0015
+ $N_0016 $N_0017 $N_0018 $N_0019 $N_0020 $N_0021 $N_0022
+ $N_0023 $N_0024 $N_0006 $N_0001 74F543
+ PARAMS: IO_LEVEL=0 MNTYMXDLY=0

L_L1 $N_0006 DPWR 10nH
L_L2 DGND $N_0001 10nH
R_R1 CBAbar $N_0007 .01
R_R2 CEBAbar $N_0005 .01
R_R3 LEBAbar $N_0002 .01
R_R4 CABbar $N_0008 .01
R_R5 CEABbar $N_0003 .01
R_R6 LEABbar $N_0004 .01
R_R7 A0 $N_0009 .01
R_R8 A1 $N_0010 .01
R_R9 A2 $N_0011 .01
R_R10 A3 $N_0012 .01
R_R11 A4 $N_0013 .01
R_R12 A5 $N_0014 .01
R_R13 A6 $N_0015 .01
R_R14 A7 $N_0016 .01
R_R15 $N_0017 B0 .01
R_R16 $N_0018 B1 .01
R_R17 $N_0019 B2 .01
R_R18 $N_0020 B3 .01
R_R19 $N_0021 B4 .01
R_R20 $N_0022 B5 .01
R_R21 $N_0023 B6 .01
R_R22 $N_0024 B7 .01

.ENDS F543R
```

Figure 13 *Subcircuit definition incorporating package inductance characteristics*

Simulating the Design

Polaris is used to extract the parasitics due to PCB interconnect. Because the rise times for FAST logic are on the order of a nanosecond or two, any PCB traces longer than 4.5 inches will have delays that are longer than the rise time of the FAST circuits. This means that transmission line effects, such as reflections, will eat into the noise budget for the design and must be taken into account. Although there are many techniques for terminating the interconnect to reduce noise due to reflections, that discussion is beyond the scope of this article.

Figure 14 shows the layout used for this example. Because we do not know which nets are susceptible to ground bounce problems, the parasitics for all of the nets in the database will be included in the simulation. The Polaris Setup button within the Tools/Polaris dialog is used to set the parameters for parasitic extraction. The Crosstalk Setup mode is set to "All Nets" to force Polaris to extract parasitics for all of the nets in the layout database. Because we're including all of the design parasitics, the simulation time will be long.

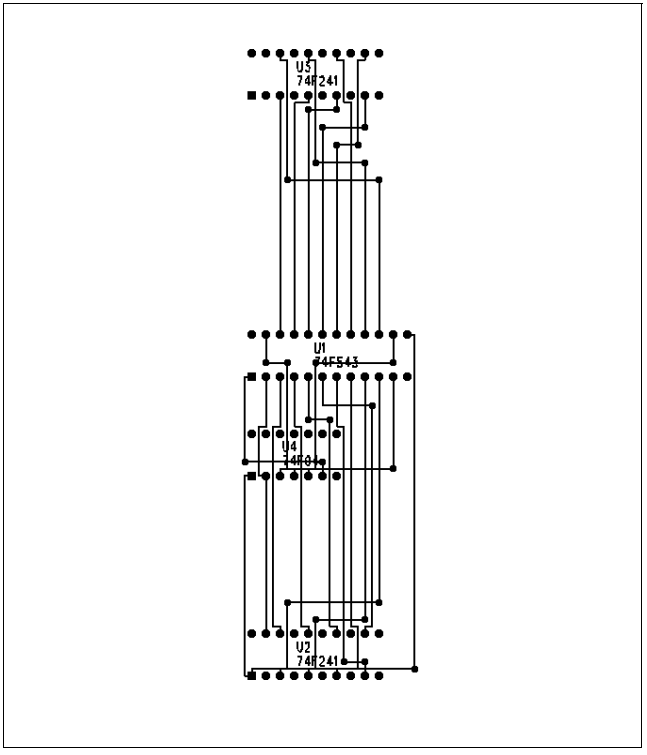


Figure 14 *Board layout for the example circuit*

The stimulus for the simulation is designed to first latch all of the inputs of the 543 in the high state, then transition the inputs low, and re-enable the 543's transparent mode. About 10 nsec after transparent mode is enabled, the inputs are again driven high. This represents a legal mode of operation for the 543, but it also sets up conditions for ground bounce problems.

Figure 15 shows a plot of the simulation results. The analog trace is the chip ground of the 74F543 subcircuit model. The most interesting occurrences in the simulation happen between the 40 and 70 nsec points. At 40 nsec (Event 1), the input bus of the 543 (INT) makes a high to low transition. The trace of the chip ground shows a corresponding spike of about 0.4 volts. This elevation of the ground voltage is ground bounce. The 543 inputs, which are using the chip ground as the reference for the low logic level, subsequently experience an increase in voltage. Considering that the maximum input low voltage for FAST logic is 0.8 volts, there is cause for concern. Any noise on the

input, such as ringing from the transmission lines of the PCB interconnect, could drive the input voltage over the maximum V_{IL} and into the undefined range. This is precisely what happens at 57 nsec (Event 2) when the control inputs, all of which should be at the low state, spend a brief period (1.5 nsec) in the undefined range. At the same time, the input bus goes to an unknown state and eventually settles out. The low to high transition of the input bus causes the output to rapidly start sourcing current which causes a -1.25 V spike in the ground voltage. If the load capacitance is high enough (for example, DRAMs usually impart a highly capacitive load), it is possible to cause other faults in the 543 to appear. For this experiment, the 543 is driving one standard FAST load plus the loading of the transmission line (also about 50 pF).

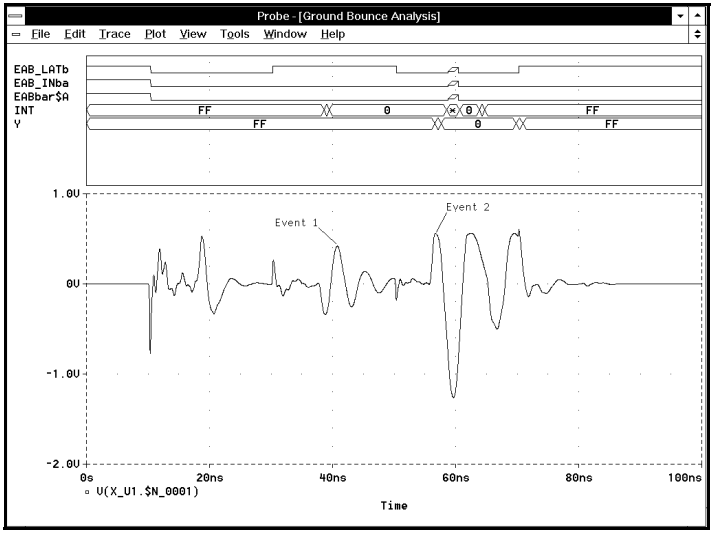


Figure 15 *Simulation results demonstrating ground bounce events at 40 and 57 nsec*

Summary

Problems in high speed digital circuits due to parasitic effects are a fact of life for designers. Those digital designers who were able to enjoy the luxury of being able to ignore parasitic effects in the past will now have to contend with problems associated with the analog behavior of digital components as clock speeds and rise times get ever faster. Through the use of a simple circuit with relatively conservative timing, we have shown that ground bounce can present problems for designers. The example used in this article has a small data bus (8 bits). As next-generation systems incorporate more and more functionality, data buses are getting wider as clock speeds increase. Since ground bounce is a function of the number of simultaneously switching lines, the problems faced by designers will continue to get worse.

Polaris and PSpice allow you to accurately simulate the behavior of circuit designs with consideration for parasitics due to interconnect—Polaris extracts the parasitics and PSpice simulates circuit behavior with parasitic values fully merged into the circuit design. In addition, the Model Library is fully user-accessible allowing custom models exhibiting new behaviors to be easily derived from existing library models, and subsequently used in simulations. Thus, compromises to operational reliability can be detected and corrected before the board is built, thereby reducing the overall cost of high-speed digital designs.

Brushless DC Motor Model

Although PSpice is designed as an electronic circuit simulator, some of our customers use it to simulate mechanical or electromechanical systems. Analog Behavioral Modeling makes simulating mechanical systems much simpler. An interesting example of an electromechanical system which can benefit from PSpice simulation is a brushless DC motor. Brushless DC motors are used in computer disk drives and other applications where precise control of motor operation is required.

A brushless DC motor is built like a stepping motor. It has a permanent magnet rotor attached to the motor shaft, and several electromagnets arranged around the stator. Each of these electromagnet windings must be driven in sequence to make the motor shaft turn, a process called *commutation*. Commutation must be synchronized with the motor shaft angle to make the motor turn at the desired speed and direction. Implementing a commutation strategy and motor control system usually requires both analog and digital circuit elements. Because the motor is part of this closed loop control system, it is important to have an accurate model of its electrical and mechanical behavior.

The equations which we will use to describe the motor's behavior come from the book *Brushless Motor System Design and Analysis* by Professor Charles K. Taft, Dr. R. G. Gauthier, S. R. Huard, and Dr. T. J. Harned. It covers brushless motor operation and commutation strategies in much more detail than this article, and is highly recommended if you have further questions. The book may be obtained by contacting Professor Taft at the Mechanical Engineering Department, University of New Hampshire, Kingsburg Hall, Durham, New Hampshire 03824. The book costs $100, including postage. The same author also has *Stepping Motor System Design and Analysis* available for $105.

The first step in modeling the motor is to develop an electrical equivalent to the mechanical system. The basic equation which describes the mechanical system is:

$$T_{total} = J\frac{d^2\theta}{dt^2} \qquad (1)$$

where

T_{total}	is total torque (including friction) applied to the motor shaft from all sources (g·cm),
J	is the mechanical system moment of inertia (g·cm·sec^2),
θ	is the motor shaft angle (radians).

This equation can also be expressed as the following two equations:

$$T_{total} = 2\pi J\frac{dS}{dt} \qquad (2)$$

$$S = \frac{1}{2\pi}\frac{d\theta}{dt} \qquad (3)$$

where

S is the shaft speed (rev/sec).

Noting that the circuit equation for a capacitor is:

$$I = C\frac{dv}{dt} \qquad (4)$$

We can implement equation (2) by modeling torque as a current and the moment of inertia, 2πJ, as a capacitor. This will give the shaft speed as the voltage across the capacitor. This is convenient because we can model any additional mechanical system moment of inertia as an additional capacitor in parallel with the first one. Also, we can add various torque and drag forces as parallel current sources. This makes the model easier to use in a system.

We can use equation (4) again on equation (3) to give the shaft angle as the voltage across a capacitor which has a current equal

to the shaft speed applied to it. Here is a PSpice subcircuit for the motor's mechanical system:

```
.subckt motor_mech shaft_speed shaft_angle
+ params:
+       J= .30   ; moment of inertia of rotor (g*cm*sec*sec)
+       twopi = {2 * 3.141596}

Cmotor shaft_speed 0 {J*twopi}                ; Inertia
Gintegrate 0 shaft_angle_intg VALUE = {V(shaft_speed)}
Cintegrate 0 shaft_angle_intg {1/twopi} IC=0.0
Rdummy2          0 shaft_angle_intg 1e12 ; (otherwise floating)
Ecopy shaft_angle 0 VALUE = {V(shaft_angle_intg)}
                                              ; Copy the voltage
Rdummy3          shaft_angle 0 1              ; Make sure there is a load

.ends
```

To use the model, we apply a current proportional to the shaft torque between nodes SHAFT_SPEED and 0 (1 amp = 1 g·cm). The voltage on that node will correspond to the shaft speed (1 volt = 1 rev/sec), and the voltage on SHAFT_ANGLE will be the shaft angle (1 volt = 1 radian).

Now we need to model the mechanical losses of the motor. The simplest are linear losses: damping and eddy current losses. They are described by the equation:

$$T_{damping} = 2\pi(B \cdot S) \qquad (5)$$

where

B is the damping and eddy current losses (g·cm·sec/rad)

S is the shaft speed (rev/sec)

We translate into our model units and get:

$$I(SHAFTSPEED) = 2\pi(B \cdot V)(SHAFTSPEED) \qquad (6)$$

This is just the equation for a resistor attached to node SHAFT_SPEED and ground, with a value of $1/(2\pi B)$.

```
.PARAM B=.36;Damping and eddy current losses (g*cm*sec/rad)
Reddy shaft_speed 0 {1/(B*twopi)}
```

Another mechanical loss is the frictional loss. This loss is a fixed torque which opposes the direction of rotation. To model this loss we use a table to specify the shape of the loss curve, and an Analog Behavioral Modeling current source to multiply the loss curve by the loss factor F (g·cm).

```
.PARAM F = .72 ; Friction losses (constant
torque loss) (g*cm)
```

```
Gdrag shaft_speed ld1 VALUE = {F * V(drag)} nonlinear drag
Ldummy1 ld1 0 100mH  ; force timestep control
Edrag2 drag 0 TABLE {V(shaft_speed)}=(-.001, -1) (.001, 1)
Rdummy1 drag 0 1
```

The Ldummy1 inductor is inserted in series with the Gdrag current source to help PSpice do timestep size control. Since it is in series with the current source it has no effect on the current (torque) produced by Gdrag, but the voltage across it reflects the derivative of the current and allows PSpice to do timestep size control on that derivative. When using the Analog Behavioral Models, it is often a good idea to place an inductor in series with a controlled current source, or a capacitor in parallel with a controlled voltage source, to help PSpice with timestep control.

Another torque is the magnetic detent torque which tends to align the rotor magnetic poles with the stator poles. This torque is periodic, and is described by the equation:

$$T_{detent} = D \cdot \sin(N_d \cdot A \cdot \theta) \qquad (7)$$

where

D	is the magnetic detent torque (g·cm)
A	is the number of north poles on the rotor
N_d	is an integer determined by the number of stator slots and the structure of the rotor.

We can translate this directly into a behavioral modeling current source:

```
.PARAM D = 2.9          ; Magnetic detent torque (g*cm)
.PARAM A = 2            ; number of north poles on the rotor
.PARAM P = 3            ; number of stator phases
Gdetent shaft_speed ld2 VALUE={D *sin(2*A*P*V(shaft_angle))}
```

All of these together give us the following model for the mechanical part of the motor.

```
.subckt motor_mech shaft_speed shaft_angle
+ params:
+           J= .30    ; moment of inertia of rotor (g*cm*sec*sec)
+           B= .36    ; Damping and eddy current losses
                      ; (linear torque with speed) (g*cm*sec/rad)
+           F= .72    ; Friction/drag losses (constant torque losses)
                      ; +(g*cm)
+           D= 2.9     Magnetic detent torque (g*cm)
+           A= 2      ; Number of north poles on the rotor
+           P= 3      ; Number of phases (if you change this you need
                      ; to add more windings to the motor subckt.)
+           twopi = {2 * 3.141596}

Cmotor shaft_speed 0 {J*twopi}            ; Inertia
Reddy shaft_speed 0 {1/(B*twopi)}         ; Linear losses
Gdrag shaft_speed ld1 VALUE = {F * V(drag) ; non-linear drag
Ldummy1 ld1 0 100mH                       ; force timestep control
Gdetent shaft_speed ld2 VALUE = {D * sin(2*A*P*V(shaft_angle))}
                                          ; detent
Ldummy2 ld2 0 100mH                       ; force timestep control
Edrag2 drag 0 TABLE {V(shaft_speed)} = (-.001, -1) (.001, 1)
Rdummy1 drag 0 1

Gintegrate 0 shaft_angle_intg VALUE = {V(shaft_speed)}
Cintegrate 0 shaft_angle_intg {1/twopi} IC=0.0
Rdummy2 0 shaft_angle_intg 1e12           ; (otherwise floating)
Ecopy shaft_angle 0 VALUE = {V(shaft_angle_intg)
                                          ; Copy the voltage
Rdummy3 shaft_angle 0 1                   ; Make sure there is a load

.ends
```

Now we need to model the electrical properties of the stator windings. The properties which are required for a first order model are winding inductance, winding resistance, winding capacitance, winding mutual inductance, winding back EMF, and the torque on the rotor from the winding current. The first four of these are simple electrical properties of the winding which are modeled directly by PSpice. The second two require a behavioral model. Dr. Taft *et al* provide us with the following equations for back emf and torque:

$$V_{bn} = Cb \cdot S \cdot \sin\left(A \cdot \theta - (N-1)\frac{2\pi}{P}\right) \qquad (8)$$

$$T_{dn} = Ct \cdot i_n \cdot \sin\left(A \cdot \theta - (N-1)\frac{2\pi}{P}\right) \qquad (9)$$

where

V_{bn}	is the back emf voltage for the phase n winding
Cb	is the back emf voltage constant (volts·sec/rev)
T_{dn}	is the drive torque from the phase n winding
Ct	is the torque constant (g·cm/amp)

i_n	is the current in the phase n winding (amp)
S	is the shaft speed (rev/sec)
A	is the number of north poles on the rotor
N	is the phase number (1, 2, 3 in our example)
P	is the number of motor phases.

Keeping in mind that the sine terms of equations (8) and (9) are the same, and adding the other four electrical properties of the motor windings, we come up with the following model for the motor:

```
* The motor with both ends of each coil available.
*
* Phase 3 coil -----------------+---+
* Phase 2 coil --------+---+ | |
* Phase 1 coil +---+ | | | |
* | | | | | |
.subckt bldcmtr p1a p1b p2a p2b p3a p3b shaft_speed shaft_angle
+ params:
+       J= .30            ; moment of inertia of rotor (g*cm*sec*sec)
+       B= .36            ; Damping and eddy current losses
*                (linear torque with speed) (g*cm*sec/rad)
+       F= .72   ; Friction/drag losses (constant torque losses)
*                (g*cm)
+       D= 2.9            ; Magnetic detent torque (g*cm)
+       A= 2              ; Number of north poles on the rotor
+       P= 3     ; Number of phases (if you change this you need
*                to add more windings to the motor subckt.)
+       CL=3mh            ; winding inductance (Henrys)
+       CR=6ohm           ; winding resistance (Ohms)
+       CC=.001uf         ; winding capacitance to ground (Farads)
+       CM=.5             ; adjacent winding mutual coupling factor
+       Cb=.12            ; Back EMF constant (Volt*sec/rev)
+       Ct=300            ; Torque constant (g*cm/amp)
+       twopi = {2 * 3.141596}
```

The Motor Model is continued.

The Motor Model (continued)

```
* Model each winding. The inductor must be here so we can include
* mutual inductance. The other effects are modeled in
* motor_winding
Lwinding1 p1a p1x {CL}
R_snub_1 p1a p1x {1K*twopi*CL}
*                 ; snubbing resistor to limit coil Q
x1 p1x p1b shaft_speed shaft_angle motor_winding
+ params: N=1 A={A} P={P} CL={CL} CR={CR} CC={CC}
+                  CM={CM} Cb={Cb} Ct={Ct} twopi={twopi}
Lwinding2 p2a p2x {CL}
R_snub_2 p2a p2x {1K*twopi*CL}
*                 ; snubbing resistor to limit coil Q
x2 p2x p2b shaft_speed shaft_angle motor_winding
+ params: N=2 A={A} P={P} CL={CL} CR={CR} CC={CC}
+ CM={CM} Cb={Cb} Ct={Ct} twopi={twopi}
Lwinding3 p3a p3x {CL}
R_snub_3 p3a p3x {1K*twopi*CL}
*                 ; snubbing resistor to limit coil Q
x3 p3x p3b shaft_speed shaft_angle motor_winding
+ params: N=3 A={A} P={P} CL={CL} CR={CR} CC={CC}
+                  CM={CM} Cb={Cb} Ct={Ct} twopi={twopi}

* Model the mutual inductance here.
* (For three phase, all windings are adjacent to each other.)
k1 Lwinding1 Lwinding2 {Cm}
k2 Lwinding2 Lwinding3 {Cm}
k3 Lwinding3 Lwinding1 {Cm}

* Model the motor mechanical system.
x4 shaft_speed shaft_angle motor_mech
+ params: J={J} B={B} F={F} D={D} A={A} P={P} twopi={twopi}
.ends

*
* The motor winding
*
* This models the electrical properties of the windings,
* and creates the torque "current" which is delivered to
* the mechanical model.
* Mutual inductance is modeled in the motor subcircuit,
* so the inductance must be there also. The inductance
* must be in series with this model.
.subckt motor_winding winding1 winding2 shaft_speed shaft_angle
+ params: N=1 A=2 P=3 CL=3mh CR=6ohm CC=.001uf CM=.5 Cb=.12 Ct=300
+ twopi={2*3.141596}

* The electrical model: backemf, resistance, and capacitance
Ebackemf winding1 3 VALUE = {Cb * V(shaft_speed) * V(factor)}
Vsense 3 4 0v               ; measure winding current
Rwinding 4 winding2 {CR}
* Place half the winding capacitance at each end of the winding
C1 winding1 0 {CC/2}
C2 winding2 0 {CC/2}

* The mechanical model: torque created by this winding
Gtorque 0 shaft_speed VALUE = {Ct * I(Vsense) * V(factor) }

* The shaft angle function for this phase.
Efactor factor 0 VALUE = + {sin(A*V(shaft_angle) - (N-1)*(twopi/P))}
Cdummy factor 0 10uf        ; force timestep control

.ends
```

To test the motor model we need to implement a commutation strategy. We will use a simple one which works like electronic brushes and drives only one stator phase at a time. To do this, we take the sine of the shaft angle with respect to each stator phase (p1x, p2x, and p3x). The sine wave is used to control switches for each phase, so that voltage is applied only to the stator phase which has the highest torque-to-current ratio. There are many other commutation strategies which can be used. The phases can be connected in a “Y” or delta, with two or more of the terminals connected to a supply or ground during each commutation interval. Professor Taft’s book describes several of these commutation strategies in detail.

```
* A test circuit for the motor
.param twopi = {2*3.141596}
.param P = 3                    ; the number of phases
.param A = 2                    ;the number of north poles on the rotor

* Connect one end of each phase winding to ct.
x1 p1 ct p2 ct p3 ct shaft_speed shaft_angle bldcmtr
+ params: J=.30 B=.36 F=.72 D=2.9 A= {A} P= {P} CL=3mh CR=6ohm CC=.1pf
+ CM=.5 Cb=.12 Ct=300 twopi={twopi}
rct ct 0 1 ;hook ct to ground through current measuring resistor

* Make some brushes
Ep1x p1x 0 VALUE = {V(on) * sin(A*V(shaft_angle) - (1-1)*(twopi/P))}
Ep2x p2x 0 VALUE = {V(on) * sin(A*V(shaft_angle) - (2-1)*(twopi/P))}
Ep3x p3x 0 VALUE = {V(on) * sin(A*V(shaft_angle) - (3-1)*(twopi/P))}

r1 p1x 0 1
r2 p2x 0 1
r3 p3x 0 1


S1p ppwr p1 p1x 0 switchp
S1n npwr p1 p1x 0 switchn
S2p ppwr p2 p2x 0 switchp
S2n npwr p2 p2x 0 switchn
S3p ppwr p3 p3x 0 switchp
S3n npwr p3 p3x 0 switchn

* 5v to drive, 0v to brake
Vppwr ppwr 0 PWL (0 5v .9 5v .901 0v 2s 0v)
Vnpwr npwr 0 PWL (0 -5v .9 -5v .901 0v 2s 0v)

* Clamping diodes to keep the kickback voltage down
D1p p1 ppwr dmod
D1n npwr p1 dmod
D2p p2 ppwr dmod
D2n npwr p2 dmod
D3p p3 ppwr dmod
D3n npwr p3 dmod

.model switchp vswitch (RON = .1 ROFF = 1e5 VON= .86 VOFF= .84)
.model switchn vswitch (RON = .1 ROFF = 1e5 VON=-.86 VOFF=-.84)
.model dmod D (RS = 10)

* "on" is used to enable the "brushes": 0 disconnects, 1 connects
* brushes to power.
Von on 0 PWL( 0,0 10ms,0 20ms,1 .8s,1 .81s,0 .9s,0 .91s,1)
ron on 0 1
.watch tran V([Shaft_Speed])
.tran 10ms 2s
.probe
.options acct
.end
```

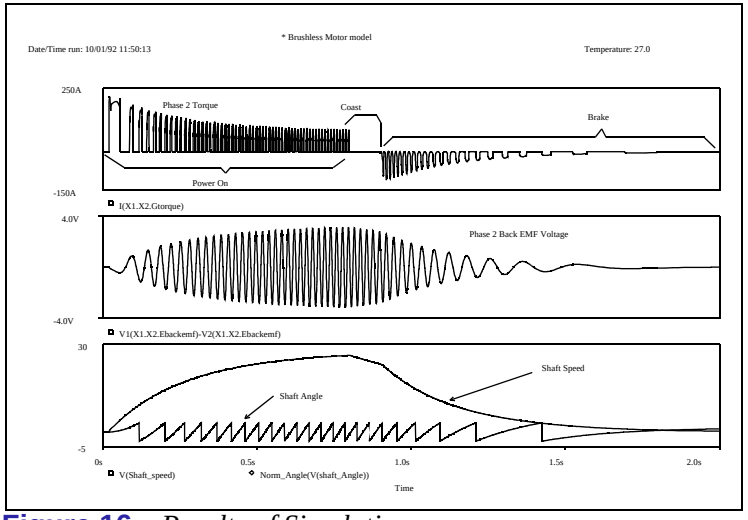


Figure 16 *Results of Simulation*

Be aware that simulating the motor takes lots of computer time (15 minutes on a 486-33 Mhz PC), and using the .PROBE command to save all the internal voltages and currents takes up lots of disk space (2.5 Mb).

The simulation run applies 5 v to the motor brushes from 20 ms to 0.8 s. During this time the motor accelerates to about 25 rev/sec. Power is disconnected from 0.8 s to 0.9 s, and the motor slows down due to friction losses. From 0.9s to the end of the simulation, the motor brushes are connected to 0 v making the motor slow quickly, as the back emf provides reverse torque. The top plot shows the motor torque delivered by the phase 2 stator winding. The middle plot shows the back emf voltage for the phase 2 winding. Notice that as the motor speed increases, the back emf increases and the torque decreases. The bottom trace shows the motor shaft speed (1 volt = 1 rev/sec) and the motor shaft angle (1 volt = 1 radian). The shaft angle has been normalized to the range $-\pi$ to $+\pi$ by the Probe macro Norm_Angle. (Otherwise, the shaft angle increases by 2π each revolution.) This macro is defined as follows:

```
Norm_Angle(a) = 2*atan(sin(a/2)/cos(a/2))
```

A brushless DC motor is a relatively complex electromechanical system to simulate in detail, as we have seen. But PSpice's Analog Behavioral Modeling feature allows us to model it in

enough detail to verify the operation of an electronic motor control system.

Note *This model is included in the Model Library in "misc.lib."*

Create Analog Random Noise Generators for PSpice Simulation

by Steven C. Hageman, CALEX MFG. CO., INC. October 1993 G132

This is a revision of Mr. Hageman's 1993 article, in which the programming was DOS-based. With the trend away from DOS in favor of Windows, it seemed desirable to develop an approach suitable for a Windows environment. Although the form of the author's PWLNOISE program is new, the substance is unchanged.

Introduction

With the great increase in computer speeds, a variety of systems can be simulated effectively and quickly with using time-domain random noise inputs. The brute-force method for creating a random noise input requires writing a small program to generate a set of random numbers, which can then be incorporated into a piecewise linear (PWL) source. This technique may require a considerable amount of time just to generate a single noise source.

The program described in this article, Pwlnoise.bas, provides a straightforward means for generating PWL noise sources in seconds rather than hours.

Noise Source Schematic

The schematic diagram for the random noise generator is shown in Figure 17.

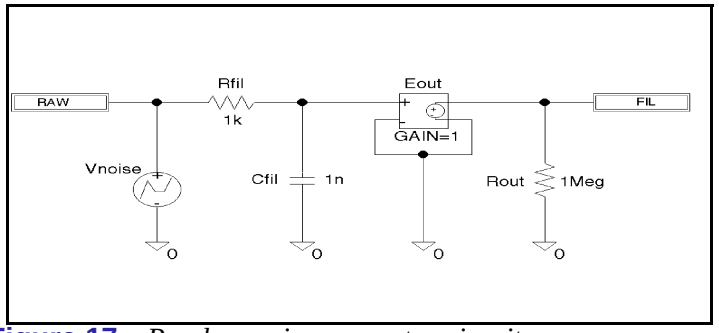


Figure 17 *Random noise generator circuit*

When entering this schematic, please note the following:

- Eout is a *voltage-controlled voltage source*. Enter it using the part name `E` (just a single letter *E*).
- Vnoise is a *file-input piecewise linear voltage source*. Use the part name `VPWL_FILE`. It has an attribute called File. Define this attribute as:

```
[path]\pwlnoise.{}
```

The [path] should be the one in which schematics (.sch files) are saved. Save the schematic as:

```
pwlnoise.sch
```

Noise Source Operation

The source, Vnoise, generates random values of voltage, scaled to 1 V_{RMS}. This signal is the “raw” noise source, RAW. Components Rfil and Cfil serve to filter the signal, and Eout scales it to the RMS value specified by the user. Hence, the output at FIL is scaled and filtered noise. Either the raw or the filtered noise source can be used within your circuit as required.

The filtering inherent in this circuit eliminates simulation problems caused by the discontinuous nature of the raw random

$$\text{Sinc } x = \frac{\sin x}{x}$$

noise. The raw noise (see Figure 18) is uniformly distributed across a range of values. The crest factor of this type of noise is approximately 1.8:1.

The noise is fundamentally triangular and discontinuous. This gives rise to a Sinc spectrum where the amplitude begins flat and goes through a series of bumps and valleys. The amplitude gradually decreases as frequency increases.

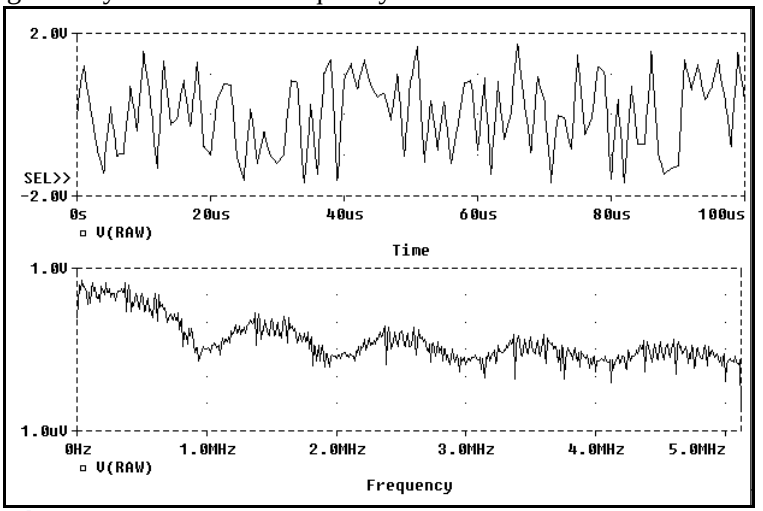


Figure 18 *(top plot) Unfiltered noise signal in the time domain; every corner is a discontinuity that can cause simulation problems and significantly increase the resulting noise bandwidth; (bottom plot) Spectrum of the unfiltered noise signal; the classic Sinc spectrum is present with harmonics extending to very high frequencies*

With such a signal, two kinds of simulation problems may occur:

- Aliasing errors arise from to the sampled nature of the transient simulation data. Also, real noise is not like the Sinc spectrum; rather, it is band-limited.
- Convergence problems arise from the discontinuous and large slope changes possible at the inflection points of the PWL noise table.

Our program solves these problems by calculating the 3-dB bandwidth of the first spectral lobe noise. This bandwidth is then applied to a single-pole RC filter on the output of the noise generator circuit. This filtering solves the above stated

problems by rounding the corners of the noise (thereby removing the discontinuities), and by band-limiting the noise source.

Figure 19 shows the effect of filtering the random noise signal shown in the previous figure.

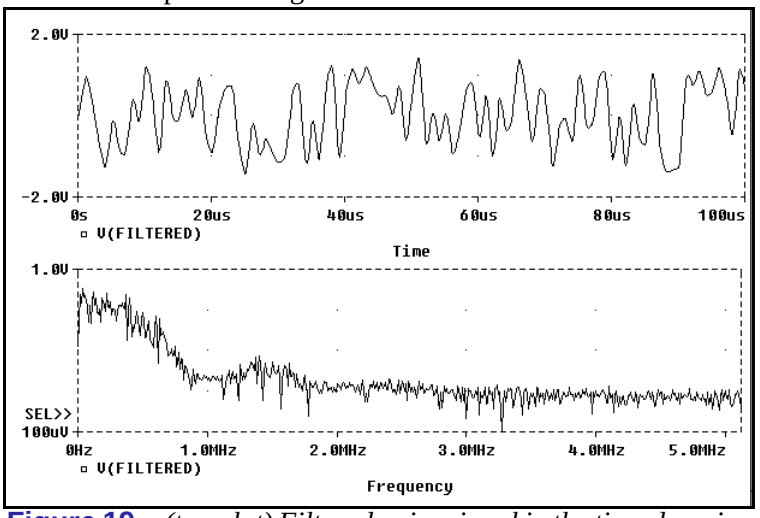


Figure 19 *(top plot) Filtered noise signal in the time domain, exhibiting realistic "soft, rounded" corners; simulation is more efficient because the slope changes are not as abrupt; (bottom plot) Spectrum of the filtered noise signal, exhibiting more well behaved characteristics than the unfiltered case*

Program Operation

Line	Action
10	Defines PI (π); defines RFIL (the filter resistor value) as 1000Ω.
20	Defines the path used by software; change if necessary.
60	Determines TIME STEP (in seconds) from user input.
80	Determines FINAL TIME (in seconds) from user input.
90	Determines RMS noise voltage (in V_{RMS}) from user input.
100	Computes the number of points, and dimensions array A(I) accordingly.
110	Computes the maximum slew rate, source bandwidth, and value of the filter capacitor.
210	Fills the array with nonnegative random numbers, each less than 1, and computes the sum of all array values. Note that A(1) remains equal to 0.
230	Computes the arithmetic mean ("average") of the array values.
250	Computes the variance (square of the standard deviation) of the array values.
270	Computes the standard deviation of the array values.
290	Redefines the array values so that they have a mean of 0 and a standard deviation of 1.

Using the Program

The Pwlnoise.bas program is given at the end of this article . It runs under any conventional Basic interpreter, such as the Microsoft GW-BASIC bundled with many computers.

It isn't necessary to leave the Windows environment to create or run the program. Just use any text editor or word processor to copy the program as it appears here, and save it in ASCII — repeat, *ASCII* — format under the name `pwlnoise.bas`. (Do *not* save it as a file "formatted" for your word processor. For example, if you're using Microsoft Word, save Pwlnoise.bas as a **`.txt`** file rather that a **`.doc`** type.)

A typical path might be:

```
"c:\microsim\"
```

Note especially line 20 of the program. In it you define the path for your working directory. When completing this line, put the path in quotations marks, and don't forget the final backslash.

When you're ready to run the program, open the File menu in the Windows Program Manager, and choose Run. Enter this line:

```
[path]\gwbasic [path]\pwlnoise
```

The program will ask you to enter three values:

- **TIME STEP in seconds**: the time value to be used between steps in the PWL source. This parameter partially controls the bandwidth and slew rate of the source. For example, if the time step is decreased, the random noise values change more rapidly with time, thereby increasing the bandwidth of the resulting noise, and increasing its slew rate.
- **FINAL TIME in seconds**: the time at which the random noise generator is to be stopped. Increasing this number increases the number of steps that are included in the PWL noise source.
- **RMS NOISE in volts(RMS)**: the RMS value of the noise voltage to be simulated. This parameter also affects the slew rate and bandwidth of the resulting noise generator. If the RMS value is increased for a given time step, the slew rate increases thereby increasing the signal's bandwidth.

Note *Be sure to make a note of the FINAL TIME value that you enter. You will need it later.*

Since each prompts names the unit involved (seconds for the first two and V_{RMS} for the third), enter *numbers only* (without units).

For example, `0.0024` is okay, as is `2.4E-3`, but not `2.4msec` or `2.4mV.`

Once you've entered these values, the program will display four computed results:

- **Points**: the number of points that will be included in the PWL noise source.
- **Bandwidth**: the 3-dB bandwidth of the first spectral lobe noise.
- **Maximum Slew Rate**: the approximate maximum slew rate that the signal can achieve from one time step to another.
- **CFIL**: the computed value of the filter capacitor.

Note *Be sure to make a note of the value displayed for CFIL. You will need it later.*

When all the necessary computations have been made, the program will remind you to make a note of parameters FINAL TIME, RMS and CFIL, which you will need later.

For details of program operation, see the sidebar.

Schematics and PSpice

Enter the schematic editor and open Pwlnoise.sch. Now make the following changes:

- Set the value of the capacitor equal to CFIL, the value previously displayed by Pwlnoise.bas.
- Set the GAIN attribute of Eout equal to the RMS value that you entered when running Pwlnoise.bas. Be sure to enter only the number, without any units (the gain is dimensionless). For example, enter “0.125,” not “0.125V.”
- Set duration of the Transient analysis equal to the FINAL TIME value that you entered when running Pwlnoise.bas.

You can now run PSpice and Probe in the usual way. The global ports (RAW and FIL) make it easy to use this circuit as part of another schematic, one requiring a noise source. But remember to rerun Pwlnoise.bas whenever you need to change the noise source’s parameters.

Entering `10` produces noise of 10 V_{RMS}, while `10E-9` yields 10 nV_{RMS}.

Note *Be sure to make a note of the RMS value that you enter. You will need it later.*

```
10 CLS:RANDOMIZE TIMER:PI=4*ATN(1):RFIL=1000
20 PATH$=
30 PRINT:PRINT"INPUT VALUES"
40 PRINT:PRINT"***  Use numerical notation without units.  "
50 PRINT"***  For example, enter 1.2E-3 or 0.0012, but not 1.2msec or 1.2mV."
60 PRINT:PRINT"Enter TIME STEP in seconds:     ";:LINE INPUT TS$:TS=VAL(TS$)
70 PRINT:PRINT"***  The next value must be larger than TIME STEP."
80 PRINT:PRINT"Enter FINAL TIME in seconds:    ";:LINE INPUT FT$:FT=VAL(FT$)
90 PRINT:PRINT"Enter RMS NOISE in volts(RMS):  ";:LINE INPUT RMS$:RMS=VAL(RMS$)
100 NP=INT(FT/TS)+1:DIM A(NP)
110 SLEW=2*SQR(2)*RMS/TS:BW=3.2/(2*PI*TS):CFIL=1/(2*PI*RFIL*BW)
120 '
130 PRINT:PRINT:PRINT"OUTPUT VALUES"
140 PRINT:PRINT" Points";TAB(16);"Bandwidth";TAB(36);
150 PRINT"Maximum slew rate";TAB(61);"CFIL"
160 PRINT STRING$(75,45)
170 PRINT NP;TAB(15);BW;"Hz";TAB(35);SLEW;"V/sec";TAB(60);CFIL;"F"
180 PRINT:PRINT:PRINT"***  Working...";
190 '
200 FOR I=2 TO NP
210 A(I)=RND:SUM=SUM+A(I)
220 NEXT I
230 MEAN=SUM/NP
240 FOR I=1 TO NP
250 V=V+(A(I)-MEAN)^2
260 NEXT I
270 SD=SQR(V/NP)
280 FOR I=2 TO NP
290 A(I)=(A(I)-MEAN)/SD
300 NEXT I
310 '
320 OPEN PATH$+"PWLNOISE.{}"FOR OUTPUT AS 2
330 PRINT#2,"0,0   ";
340 FOR I=2 TO NP
350 IF (I-1)/3=INT((I-1)/3) THEN PRINT#2,"":PRINT#2,"+ "; ELSE PRINT#2,"   ";
360 X$=STR$(I*TS):X$=RIGHT$(X$,LEN(X$)-1)
370 Y$=STR$(A(I)):Y$=RIGHT$(Y$,LEN(Y$)-1)
380 PRINT#2,X$;",";Y$;
390 NEXT I
400 '
410 CLOSE:LOCATE 23,1:BEEP
420 PRINT">>>> After making a note of FINAL TIME, RMS and CFIL, ";
430 PRINT"press any key to exit <<<<";
440 IF INKEY$=""THEN 440 ELSE SYSTEM
```

Be sure to complete line 20.

Figure 20 *The PWLNOISE.BAS program*

Create S-Parameter Subcircuits for Microwave and RF Applications

by John S. Gerig Wideband Associates

The article "Obtain S-Parameter Data from Probe" (April 1994 issue of *The Design Center Source)*, described useful subcircuits which permit both the transmission (S_{21} and S_{12}) and the reflection (S_{11} and S_{22}) parameters of a given circuit to be calculated and displayed using Probe. In microwave and RF design work, a PSpice subcircuit whose S-parameters are defined as *tables* is also useful; the data corresponds to that provided by many manufacturers for microwave transistors and other microwave devices. [See, for instance, the Hewlett-Packard *Communications Components Designer's Catalog.*]

The S-Parameter Subcircuit

A 2-port S-parameter subcircuit can be easily implemented as shown within the dashed lines of Figure 21. The voltage-controlled voltage sources—E11, E21, E12, and E22—use the frequency-response table feature provided with PSpice Analog Behavioral Modeling. Figure 21 also shows the setup for measuring S_{11} and S_{21} for this subcircuit. Node 3 is typically grounded, and nodes 1 and 2 correspond to ports 1 and 2.

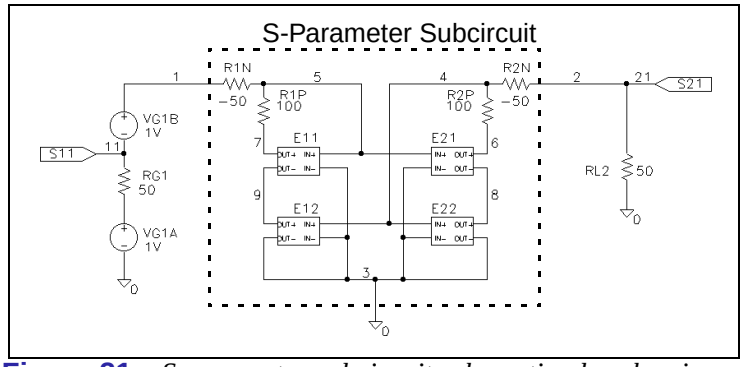


Figure 21 *S-parameter subcircuit schematic also showing setup for evaluating S_{11} and S_{21}. Z_0=50 ohms is assumed.*

As noted in the April article (repeated in the theory sidebar at left), the S_{ij} coefficients are the dimensionless ratios of scattered wave amplitudes, *b1* and *b2*, to incident wave amplitudes, *a1* and *a2*; i.e., S_{21}=*b2/a1*. The wave amplitudes are usually normalized so that their squared magnitudes measure power. In the present application, however, it is convenient to normalize *a1* or *a2* to 1 volt; the corresponding S-parameters then become the voltages at certain nodes.

The voltage generator at the left side of Figure 21 is set up as an open-circuit voltage of 2 in series with a generator impedance of Z_0 ohms. This delivers a voltage wave *a1=1* volt to a matched or Z_0 ohm load. By implementing the 2-volt generator as two 1-volt generators—VG1A and VG1B in series with the generator resistance RG1 as shown—we achieve the particularly simple result that the voltage at node 11 is the reflected or scattered wave *b1*. Since *a1* equals one, this is S_{11} itself. This can be verified by inspection. If the generator is terminated at node 1, then the voltage at node 11 is +1 for an open circuit, 0 for a matched load, and -1 for a short circuit. This is in agreement with equation (8) of the April article (see side bar).

Similarly, the voltage at node 21 to the right of Figure 21 is simply S_{21} = *b2/a1*. Thus, for example, the Probe expressions VDB(21) and VP(21) will display the dB amplitude and phase of S_{21}.

Note *In the case of S-parameters S_{12} and S_{22} associated with a2=1, the test generators and termination of Figure 21 must be transposed. See the example schematic in Figure on page 65; the nodes are labelled 22 and 12, accordingly.*

E11 and E21, when connected to matched terminations, deliver the scattered waves *b1* and *b2* resulting from the incident wave *a1* modified by appropriate response tables. Thus the required control input to these generators is the voltage *2*a1*—the total voltage generated by VG1A and VG1B in series. By introducing the negative resistance, $R1N=-Z_0$, between nodes 1 and 5, we can force node 5 to this voltage since the voltage drop across R1N due to any input current exactly cancels the drop across RG1. When E11 and E12 are turned off, the further addition of $R1P=2*Z_0$ then offsets R1N and produces the required Z_0 input impedance at port 1.

E12 and E22 similarly generate the appropriate scattered waves for an incident wave *a2* at port 2.

Converting Manufacturer's Data

It is completely feasible, though tedious, to prepare table-based subcircuits by manually editing S-parameter data published by the manufacturer. However, the manufacturer's data almost always presents S-parameters in terms of magnitude and wrapped phase, whereas the frequency-response tables used in PSpice Analog Behavioral Modeling (version 6.0 and earlier) require decibel levels and unwrapped phase in degrees. (PSpice version 6.1 can optionally accept magnitude in raw form and unwrapped phase in radians.)

Phase wrapping refers to reducing the phase data to principal values in the range -180 to +180 degrees. This creates a problem for Analog Behavioral Models which linearly interpolate the phase data. For example, if the unwrapped phase data steps from -175 to -195 degrees, the wrapped version would step from -175 to +165 degrees. In the unwrapped case, the interpolated midpoint is a realistic -185 degrees; in the wrapped case, however, the nonphysical interpolated value is -5 degrees.

The S2P2LIB1Conversion Program

It is much easier to convert the manufacturer's S-parameter data to PSpice-compatible form using a program that can interpret S2P files. Figure 22 shows the logic to do so; this program—S2P2LIB1—is written in QuickBasic.

```
'S2P2LIB1-- converts S2P tables to
+ PSpice-compatible frequency-response tables.
'Written in QuickBASIC (v4.5). See 'HelpMessage'
+ SUB for additional comments.
DECLARE SUB ParseDataLine ()
DECLARE SUB HelpMessage ()
DIM D(16), M(50, 9)
COMMON SHARED D(), M(), L$
False = 0: True = NOT False
DataLine = 0: EODat = False
C$ = COMMAND$
'C$ = "10236N.S2P"  'Decomment and edit in uncompiled version.
IF UCASE$(RIGHT$(C$, 4) = ".S2P" THEN  'Plausible command$ found.
 ufn$ = LEFT$(C$, LEN(C$) - 4)
ELSE
 CALL HelpMessage: SYSTEM
END IF
OPEN C$ FOR INPUT AS #1
OPEN ufn$ + ".LIB" FOR OUTPUT AS #2
PRINT #2, ".SUBCKT " + ufn$ + " 1 2 3; Port1 Port2 Common"
PRINT #2, "*Subcircuit generated by S2P2LIB1.EXE on " + DATE$

WHILE NOT (EOF(1) OR EODat)
LINE INPUT #1, L$
L$ = LTRIM$(L$)        'Get rid of any leading spaces.
IF LEFT$(L$, 1) = "!" THEN PRINT #2, "*" + RIGHT$(L$ + " ", LEN(L$))
IF LEFT$(L$, 1) = "#" THEN HDR$ = UCASE$(L$): PRINT #2, "*" + L$
IF VAL(L$) > 0 THEN
 InTheData = True
 ParseDataLine
 DataLine = DataLine + 1
 FOR k = 1 TO 9: M(DataLine, k) = D(k): NEXT
END IF
IF VAL(L$) = 0 AND InTheData = True THEN EODat = True
WEND
CLOSE #1
PRINT #2, " ": PRINT #2, "R1N 1 5 -50": PRINT #2, "R1P 5 7 100"
PRINT #2, "R2N 2 4 -50": PRINT #2, "R2P 4 6 100"
S$(1) = "*S11 FREQ DB PHASE": S$(2) = "*S21 FREQ DB PHASE"
S$(3) = "*S12 FREQ DB PHASE": S$(4) = "*S22 FREQ DB PHASE"
E$(1) = "E11 7 9 FREQ {V(5,3)}=": E$(2) = "E21 6 8 FREQ {V(5,3)}="
E$(3) = "E12 9 3 FREQ {V(4,3)}=": E$(4) = "E22 8 3 FREQ {V(4,3)}="
G$ = "_+_(###.##gHz_,+####.##_, +####.##)"    'Format for gHz data
M$ = "_+_(######.##MHz_,+####.##_, +####.##)"  'Format for MHz data
IF INSTR(HDR$, "MHZ") THEN P$ = M$ ELSE P$ = G$

FOR P = 1 TO 4         'Build S11, S21, S12, and S22 blocks in sequence.
PRINT #2, " ": PRINT #2, S$(P): PRINT #2, E$(P)
```

Figure 22 *QuickBasic logic for S2P2LIB1 program.*
(Continued on the next page)

```
Offset = 0: PrevPh = 0 'Clear variables used to unwrap phase.
FOR F = 1 TO DataLine  'Successive frequency values.
Ph = M(F, 2 * P + 1)   'Current phase data.
IF ABS(Ph-PrevPh)>180 THEN Offset=Offset-360*SGN(Ph-PrevPh)
PrevPh = Ph
UWP = Ph + Offset                'UnWrapped Phase.
DB=20*LOG(M(F,2*P))/LOG(10): 'Convert magnitude to DB values.
PRINT #2, USING P$; M(F, 1); DB; UWP
PRINT ".";   'Something to look at.
NEXT F: NEXT P
PRINT #2, " ": PRINT #2, ".ENDS"
PRINT "Finished. Result saved in file "  ufn$  ".LIB"
CLOSE #2
END

SUB HelpMessage
 PRINT"WBA 7/94. This utility converts an"
 PRINT"S-parameter file *.S2P having"
 PRINT"a magnitude and angle (degrees) format"
 PRINT"into a PSPICE subcircuit"
 PRINT"file *.LIB which uses the Analog Behavioral"
 PRINT"Model option."
 PRINT"The source file should be in the current directory."
 PRINT"A 50 ohm reference impedance is assumed."
 PRINT"gHz frequency units are assumed unless the"
 PRINT""*.S2P files contains"
 PRINT"a header record beginning with '#' and followed by 'MHz'"
 PRINT" "
 PRINT"USAGE: S2P2LIB1 *.S2P "
END SUB

SUB ParseDataLine
k=1: D$=""
FOR CH=1 TO LEN(L$)+1
CH$=MID$(L$+ " ", CH, 1)
IF CH$ <> " " THEN D$ = D$ + CH$: SP = 0
IF CH$=""AND SP=0 THEN D(k)=VAL(D$): k=k+1:SP=1:D$=""
NEXT

IF k<>10 THEN
 PRINT"Error: didn't find 9 numbers parsing the "
 PRINT"following data line:"
 PRINTL$
 PRINT"Check *.S2P file": END
END IF
END SUB
```

Figure 23 *QuickBasic logic for S2P2LIB1 program continued from prior page.*

S2P2LIB1 assumes that the S-parameter data is in magnitude and phase format, and that the reference impedance is 50 ohms. (Although other formats are allowed in S2P files, they are less frequently encountered.) Default frequency units are GHz, though MHz is also accepted if the header line (‘#’ in column one) contains “MHZ.” S2P comment lines (‘!’ in column one) are converted to PSpice comment lines (‘*’ in column one). Noise data is ignored.

Example: AC Analysis of the 10236N S-Parameter Model

Figure 24 lists an abridged version of the 10236N subcircuit which implements the S-parameter model for an Avantek/ Hewlett-Packard ATF10236 transistor. (The frequency-response tables have been truncated to conserve space.) Instead of manually setting up the S-parameter tables, the full-table version of this subcircuit can be automatically created from its corresponding S2P file (see Figure 24) using the S2P2LIB1 conversion program described above. Simply type

```
S2P2LIB1 10236N.S2P
```

at the command line prompt. This produces the unabridged 10236N subcircuit model which implements the circuitry shown in Figure 21 in both the forward and reversed directions. The subcircuit model is saved to the “10236N.LIB” file which can then be used in a PSpice simulation.

```
!  ATF-10236 S AND NOISE PARAMETERS
!  Vds=2V Id=25mA
!  LAST UPDATED 06-28-89

!FREQ  S11    S21    S12    S22
!GHZ MAG   ANG   MAG   ANG   MAG   ANG   MAG   ANG

0.5  .97   -20   5.68  162   .023  76    .47   -11
1.0  .93   -41   5.58  143   .050  71    .45   -23
2.0  .77   -81   4.76  107   .086  51    .36   -38
3.0  .59   -114  4.06  80    .120  35    .30   -51
4.0  .48   -148  3.51  52    .149  18    .23   -67
5.0  .46   166   3.03  26    .172  3     .10   -67
6.0  .53   125   2.65  1     .189   -14  .09   48
7.0  .62   96    2.22  -20   .191  -28   .24   55
8.0  .71   73    1.75  -39   .189  -41   .37   51
9.0  .75   54    1.47  -55   .184  -46   .46   42
10.0 .78   39    1.28  -72   .180  -59   .51   34
11.0 .82   26    1.04  -86   .179  -71   .54   26
12.0 .84   12    .95   -101  .177  -82   .54   17

!FREQ Fopt GAMMA OPT  RN/Zo
!GHZ  dB  MAG  ANG  -

0.5  0.4   .94   16    0.78
1.0  0.5   .87   36    0.63
2.0  0.6   .73   74    0.33
4.0  0.8   .45   148   0.15
6.0  1.0   .42   -137  0.12
8.0  1.3   .49   -80   0.45
12.0 1.6   .65   -20   1.16
```

Figure 24 *S2P file used to derive the 10236N S-parameter subcircuit model shown in Figure 23.*

Schematic Setup:

Editor's Note: To benefit users running the Design Center with Schematics, MicroSim added this section to the author's original manuscript.)

1 *Within the Symbol Editor,* create a 10236N symbol similar to that shown in Figure 25. This symbol will represent the circuitry shown within the dashed lines of Figure 25.

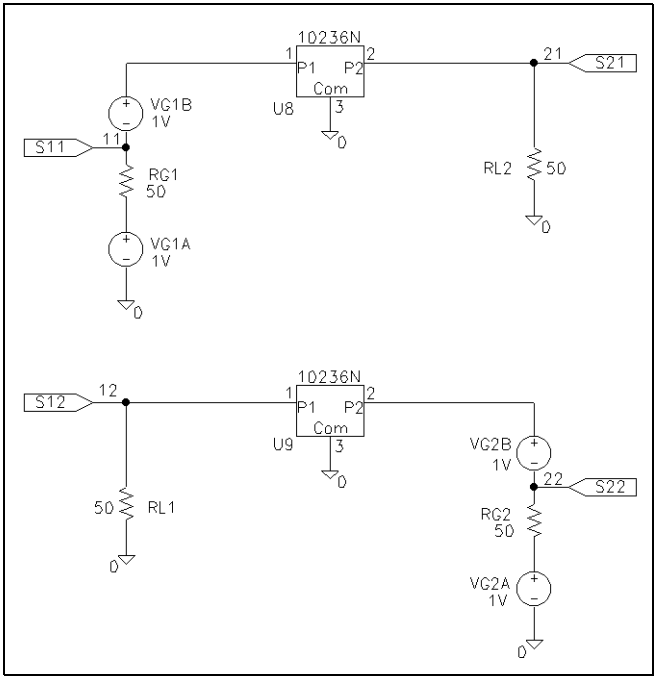


Figure 25 *Schematic setups for evaluating the S-parameters of the 10236N.*

2 Define symbol attributes (Part/Attributes), including the MODEL attribute to associate the 10236N subcircuit model with the symbol:

MODEL = 10236N
TEMPLATE = X^@REFDES %P1 %P2 %COM @MODEL

Note The TEMPLATE value is a one-line string; do not insert newline characters when entering this value.

3 Save the symbol to a new Symbol Library file using Part/Save Changes and File/Save As. Be sure to select Yes when asked to add this to the list of configured files for use with Schematics. (Alternatively, you can save the symbol to an existing library file using Part/Save to Library.)

4 Within the Schematic Editor, configure the model file, “10236N.LIB,” for use with your example schematic (Analysis/Library and Include Files). Use the Add Library

button if you wish to have it available to the current schematic only; otherwise use Add Library* for global availability.

5 Place VSRC (or VAC voltage source), GLOBAL (global port), R (resistor), AGND (analog ground), and 10236N symbols to create the test schematic shown in Figure 25 on page -65. Define the voltage sources so that AC magnitude equals 1; define the resistors equal to 50 ohms.

6 Enable and set up the AC sweep analysis under Analysis/ Setup with the following characteristics:

AC Sweep Type	Linear
Total Pts	100
Start Freq.	0.5E9
End Freq.	12E9

Note *If you expect to run a series of simulations which test different S-parameter subcircuits, you can create one generic S2P symbol rather than a custom symbol for each model (as was demonstrated for the 10236N). To do so, create a* MODEL *attribute for the S2P symbol and leave it undefined (blank). Then, from within the Schematic Editor, you can change the model reference for the S2P part instance in your schematic. Simply select the S2P instance, select Edit/Model, select Change Model Reference in the dialog, then type in the name of the S-parameter subcircuit to be used in the next simulation.*

7 Circuit File Setup: Create a circuit file as shown in Figure 26. The “10236N.LIB” file is referenced using a .LIB statement (.INC will also work). Subcircuit declarations, XQF and XQR, can then reference the 10236N subcircuit model. The .AC statement defines the parameters for the AC sweep analysis as described earlier for the schematic setup.

```
SPARTST1.CIR S-Parameter Demo 10236N.LIB
*JG 7/25/94 This file illustrates a simple method for analyzing
* the S-parameters of a subcircuit. The subcircuit has been
* created from a standard S2P
* S-parameter file using the utility S2P2LIB1.EXE.

.OPT NOMOD RELTOL=.0002
.AC LIN 100 .5E9 12E9

.LIB 10236N.LIB  ;Subcircuit 10236N for ATF10236 transistor

*Separating the conventional 2 volt EMF generator into two 1 volt
*generators (VG1A and VG1B) connected by the RG1=Z0 as shown,
* produces a voltage at node 11 equal to S11. S21 appears at node 21.
VG1A 101 0 AC 1
RG1 101 11 50
VG1B 1 11 AC 1
XQF 1 21 0 10236N
RL2 21 0 50

*The following makes S22 appear at node 22, and S12 at node 12.
VG2A 201 0 AC 1
RG2 201 22 50
VG2B 2 22 AC 1
XQR 12 2 0 10236N
RL1 12 0 50

.PROBE V(11) V(12) V(21) V(22)
.END
```

Figure 26 *Circuit file to simulate the 10236N subcircuit and analyze its S-parameters.*

After having simulated the circuits with PSpice, Probe can be used to display the various S-parameters in any desired form. Sample results are shown in Figure 27.

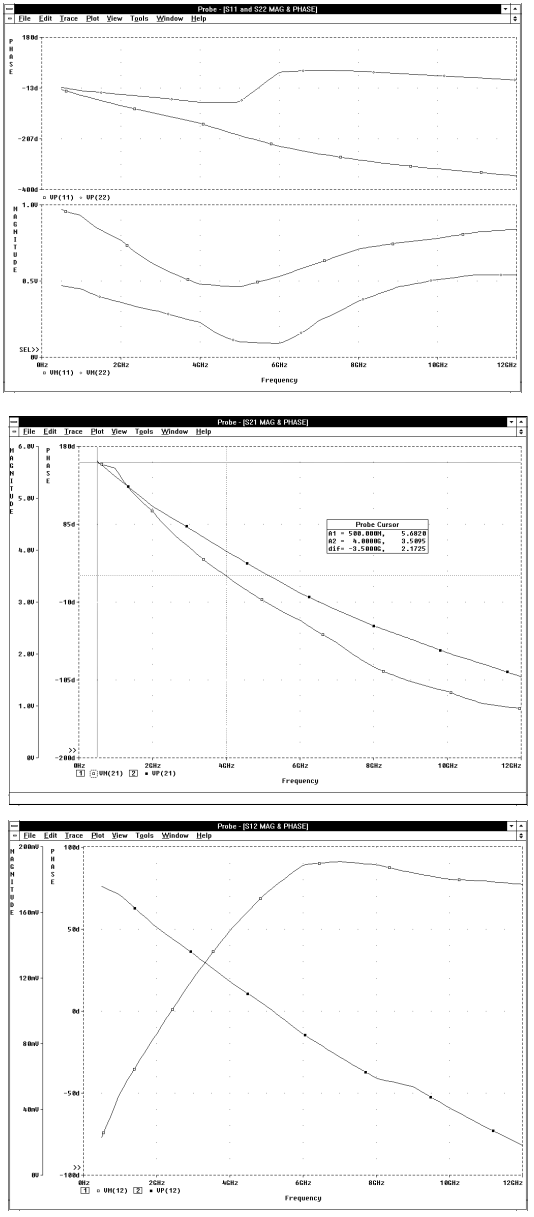


Figure 27 *Simulation results showing S-parameter magnitude and phase.*

If the AC sweep analysis is modified to reproduce the frequency steps occurring in the original S2P file, and appropriate print commands are added for VM(11), VP(11), etc. (using VPRINT1

pseudocomponents in the schematic or .PRINT statements in the circuit file), the resulting PSpice output file (".out") should numerically reproduce the original data.

Transient Analysis Considerations

Data for an S-parameter model is usually measured in the frequency domain, and normally represents only the linear or small-signal behavior of the device under specified DC operating conditions. Effects in a nonlinear transistor or diode are not modeled. The frequency-response table option in Analog Behavioral Modeling allows transient analysis, but the cautions discussed in the PSpice user's guide should be reviewed.

To Download Files from the BBS

The files referenced in this article are available on the MicroSim bulletin board in one self-extracting file named "sparam.exe." This file contains the conversion program, S2P2LIB1 as a DOS executable (".exe") and as uncompiled QuickBasic source (".bas"). This file also contains the unabridged version of "10236N.LIB," the S2P file—"10236N.S2P," and the corresponding symbol file—"S2P.SLB" (which can be imported into Schematics from within the Symbol Editor using Part/Import and Part/Save to Library).

To download the self-extracting file, select [T]ech Support from the main menu, [6] File Transfer, [1] Download User Requested Files, and indicate that you wish to download "sparam.exe." The BBS number in the U.S. is (714) 830-1550 (14.4k-1200, N-8-1).

Biography: John S. Gerig is a consultant specializing in microwave circuit design, frequency filters, and frequency synthesizers. He owns the company, Wideband Associates and may be reached by phone at (703) 391-*5619 or FAX at (703) 391-0318.*

Create Schematic Symbols for New Vendor Models

The Design Center Source newsletter, October 1994

If you are a Schematics user, you might find it necessary to create new symbols to augment those that are available in the standard Model and Symbol Libraries supplied by MicroSim. For instance, you may wish to use a new vendor model, and need a corresponding symbol to represent the part in your schematic. This application note explains the steps required to add and configure new vendor model definitions, and to create corresponding base and AKO symbols for parts when you already have an existing Model Library file with other definitions from the same vendor.

Overview

For every new device that you want to add to the Design Center, you will need to add two and sometimes three distinct items. These are: a model or subcircuit definition, a symbol, and a package definition. The model and symbol are required in all cases, but the package is required only if you will be packaging your design to go to a PCB layout program.

Model Library

All of the .MODEL and .SUBCKT statements are kept in Model Library files. By convention, each file has a ".lib" extension. If you receive new models from a vendor, they may be in a single file, or in many files that each contain one model or subcircuit. If they are in a single file, simply rename it to "<*anything*>.lib." If each model is in its own file, then you can copy them all into

a single file. For example, if they are named "<*device name*>.mod," then:

- in DOS, type:

```
copy *.mod mylib.lib
```

- in UNIX, type:

```
cat *.mod >mylib.lib
```

Be careful that none of the new model or file names duplicate those already in use.

Now you need to tell the simulator that this file exists. In Schematics, select Analysis/Library & Include Files. In the field labeled File Name, type the name of your file with the extension. If you want these models to be visible to every design, select Add Library*. If you want them to be visible to the current design only, select Add Library.

Symbol Library

Now you need to add a symbol for each of the models or subcircuits that you intend to use. Switch to the Symbol Editor by selecting Edit Library from the File menu. The status bar at the top of the screen should say <new>:<new>. This means that you are editing a new Symbol Library file and a new symbol.

- If you want to add symbols to an existing library, select Open from the File menu. Then navigate to the library file, and open it.
- If you are starting a new library, select Save As from the File menu, and enter a name. When asked whether to add the new library to the list of Schematics' libraries, click Yes to make your new library visible to Schematics.

Before we start, a quick overview of the structure of a Symbol Library file may be helpful. Most Symbol Library files are composed of base parts and AKO (A Kind Of) parts. The base part usually contains the graphical information for the symbol as well as the minimum attributes required to make that symbol functional. The AKO symbol inherits all of the graphics and attributes of the base part, but may alter them or add to them.

Note that an AKO symbol can only reference base parts contained in its own library file. This will be important as we proceed.

Adding Parts

The quickest way to add a new symbol is to copy an existing one and customize its attributes. Let's assume you have a new set of opamps from a vendor. For this example we will assume that each device has five interface pins. Since all of the symbols will look the same, we want a Symbol Library file that has one base part and multiple AKO parts—one for *each* device to be used.

To add a base part

1. Select Copy from the Part menu.
2. Click Select Lib on the Part dialog box to display a list of available ".slb" files.
3. Scroll down the list of available ".slb" files and select "opamp.slb." Then click OK or press <Enter> to display a list of symbols.

   Notice that most symbols have an AKO reference after them.

The base parts are always at the bottom of the library file.

4. Scroll down the list nearly to the bottom, and select op5; op5 then appears in both the New Part Name box and Existing Part Name box in the Copy Part dialog box.
5. Click OK to display the symbol for op5.
6. Select Save from the File menu.

To add an AKO symbol

Your new library file now has one base part. Now we will add the first AKO part (i.e., the first symbol that will be usable for simulation). Assume that the first opamp that you want to add is called LM557.

1. Select Part/New. This opens the Definition dialog box.

2 Enter a description for the part. This can be anything meaningful to you—for example, inverting OPAMP.

3 Enter the name of the part. This usually matches the name of the model (i.e., LM557).

4 For AKO name, enter OP5.

5 Click OK. The status bar at the top of the screen should now list LM557 as the symbol being edited.

6 Select Part/Attributes. This opens the Attributes dialog box. You will see four attributes listed for the symbol: REFDES, TEMPLATE, PART, and MODEL. REFDES equals U? by default which denotes that it will be referencing an IC definition. The TEMPLATE attribute is defined in detail in the Schematic Capture User's Guide, but for most cases, you will not need to edit TEMPLATE. Both PART and MODEL are blank.

7 Single-click on PART. The word PART now appears in the Name field.

8 Place the cursor in the Value field by single-clicking in the box to the right of the field name.

9 Enter the name of the symbol. In this case, LM557.

10 Click Save Attribute.

11 Do the last three steps for the MODEL attribute as well.

Note The value of MODEL must exactly match the name of the part as defined in the .SUBCKT definition found in your ".lib" file. (Case is not important.)

12 Click OK.

13 Select Part/Save Changes.

14 Select File/Save.

You now have one base part and one AKO symbol in your library file. At this point you can return to the Schematic Editor (File/Return to schematic). You are now ready to place the LM557 symbol in your drawing, and simulate.

For each additional AKO part that you want to add, simply repeat the steps in the preceding section. If other new devices

have different graphics than the op5, simply add a new base part and proceed as described above. If you need to add symbols in subsequent editing sessions, be sure to first open the library file. That is, after invoking the Symbol Editor, select File/Open and type in the name of the library file to which you will be adding the new symbols. The status bar at the top of the screen shows you which library file you currently have opened.

Creating “Eye” Displays Using Probe

The Design Center Source newsletter, January 1993

In communications work, the “eye” display is frequently used to illustrate the voltage and timing margins present in a system transmitting digital data. In an eye display the signal is shown with a time axis that is a small number of data periods long. The signal “wraps” from the right edge of the display back to the left edge, thereby depicting a large number of overlapping data periods. If a random bit pattern is simulated and then displayed in this way, the required margins can be seen as the opening in the “eye” on the display.

Probe’s macro feature can be used to create a sweep function that implements the wrap feature required. You can then change the time axis variable from time to this sweep function to obtain an eye display.

To use the macro feature, select Macros from the Trace menu in Probe. Then enter these three macros:

```
pi=4*atan(1)
mod(a, b) = (b)*(atan(tan(((a)/(b))*pi-pi/2))+pi/2)/pi
eye_sweep(p, d) = mod(time+(p)/2+(d), p) - ((p)/2+(d))
```

The first macro (pi) calculates the value of π. The second macro (mod) is a floating-point modulo function, implemented using the tangent and arctangent functions. The third macro (eye_sweep) implements an “eye sweep” function; the display is centered at one half the period (p/2), plus a time delay (d).

To use the macros, simply change the time axis variable from time to eye_sweep(p, d). In this function, p is the data period of the system (or a multiple of it), and d is the time interval by which the start of the display is delayed.

As an example, consider a simple system that sends a “random” stream of bits through an Analog Behavioral Model of a bandpass filter. The bit stream is generated by a ten-bit shift register (U1) with feeedback provided by an **XOR** gate (U2). The signal is processed by a tristate buffer (U3) and a filter (E1).

This arrangement generates a pseudo-random sequence 1023 bits long. The simulation runs for 1.5 usec, which is 300 bit periods of 5 nsec.

Here is a schematic diagram of the circuit:

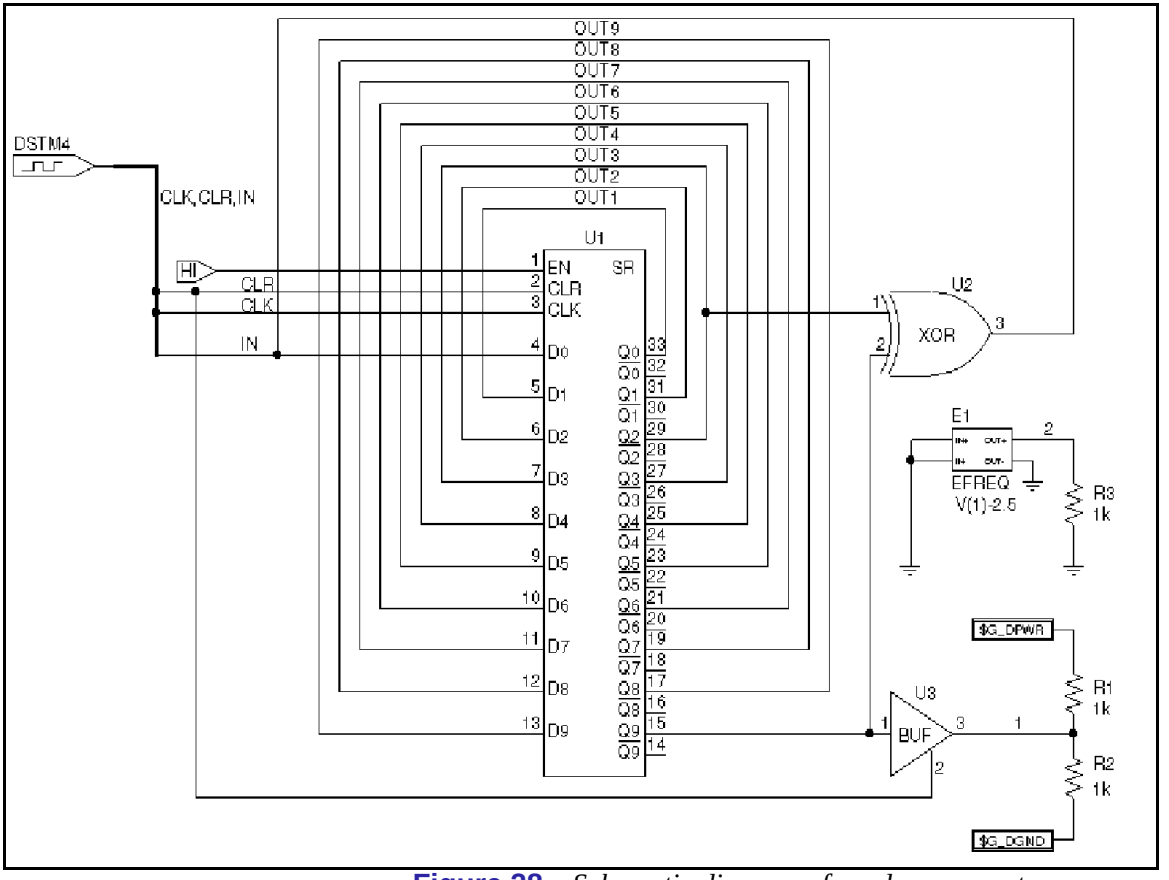


Figure 28 *Schematic diagram of random generator.*

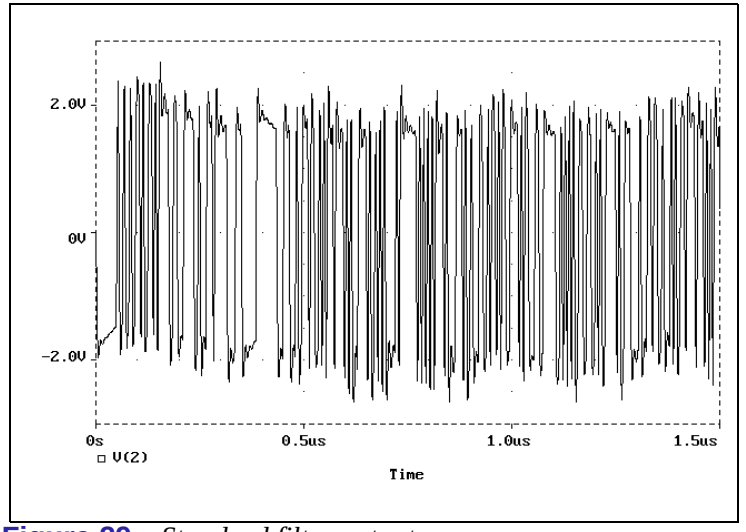


Figure 29 *Standard filter output*

Figure 29 shows the standard display of the filter’s output versus time. It is very difficult to use this display to determine the voltage and timing margins for data recovery from the system. The eye display, on the other hand, superimposes the results for all three hundred data periods on a shortened time axis, enabling both margins to be gauged easily.

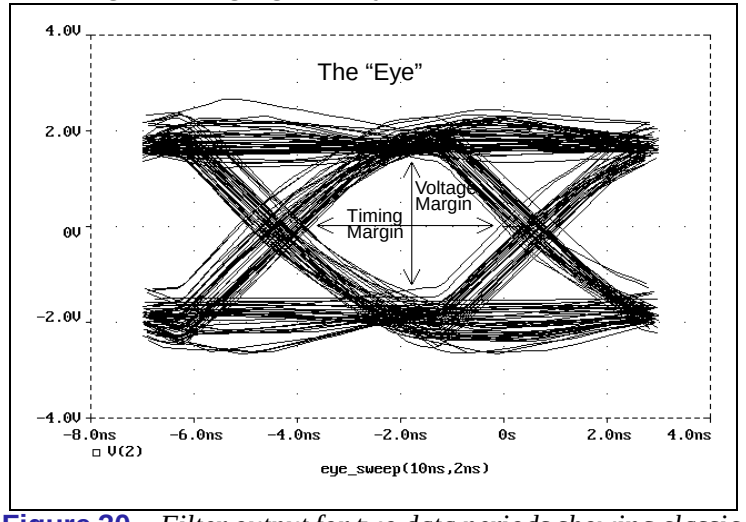


Figure 30 *Filter output for two data periods showing classic eye shape*

Figure 30 shows the eye display of the filter output with two data periods for the time axis duration, and shows the classic eye shape. The problem is that half of the simulation data is displayed, not in the central eye, but in the two half-eyes on either side. To include all of the simulation data in one eye, we need to use a single data period for the sweep period. We also need to offset the display slightly to make sure that the eye is centered in the smaller display. The results are shown below.

.

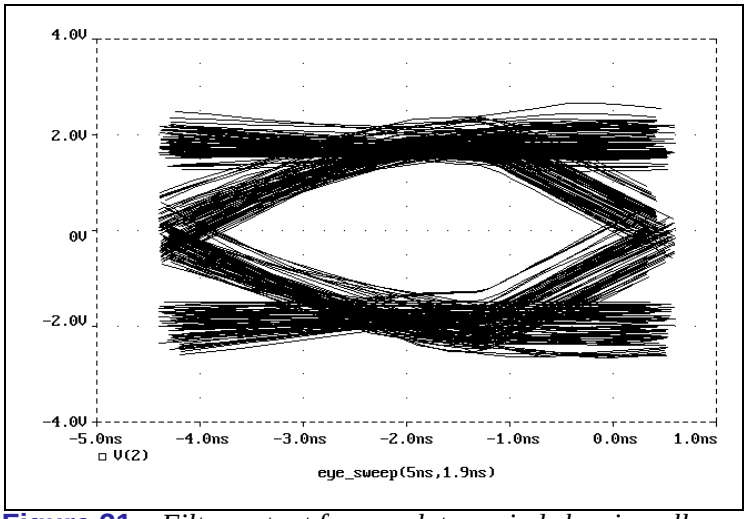


Figure 31 *Filter output for one data period showing all simulation data in one eye*

To implement eye displays, download EYE.EXE from the MicroSim bulletin board at:

(714) 830-1550

or the ftp site: ftp.microsim.com

Then be sure to install the three macros (given earlier) when running Probe.

Creating Impedances with Behavioral Modeling

MicroSim Corporation Newsletter, October 1990

We regularly receive questions on how to create nonlinear resistors with the Analog Behavioral Modeling feature. The method for doing this can be illustrated by creating the transfer function for a linear conductance. A conductance can be thought of as a voltage-controlled current source: the current between its nodes is a constant, times the voltage across those same nodes. For example:

```
GCOND 7 4 VALUE = {V(7,4)*.001}
```

is a linear conductance with a value of 1 milli-mho (i.e., a 1 kilo-ohm resistor). The controlling nodes are the same as the output nodes. For a nonlinear conductance the appropriate nonlinear function is used, but the device still has the same controlling and output nodes:

```
GSQ 7 4 VALUE = {V(7,4)*V(7,4)*V(7,4)*.001}
```

GSQ has a small-signal conductance of $3 \times .001 \times V(7,4)^2$. (The small-signal conductance is the derivative of the transfer function.)

Any nonlinear resistance can be expressed as a nonlinear conductance by inverting the transfer function. Sometimes, however, it is convenient to implement it directly. This can be done by noting that a resistor is a current-controlled voltage source. For example,

```
ERES 7 4a VALUE = {I(VSENSE)*1K}
VSENSE 4a 4
```

is a linear resistor with a value of 1 kilo-ohm. VSENSE is needed to measure the current through ERES. A quadratic resistor is then:

```
ERES 7 4a VALUE = {PWRS(I(VSENSE),2)*1K}
VSENSE 4a 4
```

The PWRS (signed power) function is used instead of $I(VSENSE)^2$ because we want the sign of the voltage across ERES to become negative when the current through VSENSE is negative.

There are a couple of things to watch for when creating nonlinear devices this way. First, all physical impedances have zero current at zero voltage.

Second, one needs to be careful of the asymptotic behavior of the device. It is very easy to create devices which generate power at high voltages. Even though the real circuit may not operate at such voltages, there is nothing to prevent PSpice from finding an unrealistic solution at a high voltage. In general, it is good practice to use the TABLE form to limit the output of devices. For example, here is a constant-power load:

```
GCONST 7 4 TABLE {100/V(7,4)} = (-100,-100) (100,100)
```

GCONST tries to dissipate 100 watts of power regardless of the voltage across it. For very small voltages the formula 100÷V(7,4) can lead to unreasonable values of current. The TABLE limits the current to be between -100 and +100 amps.

This approach can also be used to create frequency-dependent impedances. The main difference is that the LAPLACE or FREQ type is used. For example, a capacitor can be written as:

```
GCAP 7 4 LAPLACE {V(7,4)} = {s}
```

The current through GCAP is the integral of V(7,4). However, the LAPLACE device uses much more computer time and memory than does the built-in capacitor (C) device. We recommend the LAPLACE form only for cases where its flexibility is needed. Note that, in general, frequency-dependent impedances have varying phases as well as varying magnitudes of impedance. For example, the formula for a wire with skin effect is:

```
EWIRE 7 4a LAPLACE {I(VSENSE)}={R0 + R1*sqrt(s)}
VSENSE 4a 4
```

The wire’s impedance is constant (and real) at low frequencies. At high frequencies, its impedance behaves as the square root of the frequency and becomes half real and half imaginary (i.e., it becomes half inductive).

Digital Frequency Comparator

The Design Center Source newsletter, April 1993, originally titled “Simulate an All-Digital Frequency-Comparator Circuit Using PSpice”

This article illustrates how a hierarchical all-digital design with two implementation views, can be defined in Schematics, and subsequently simulated in PSpice. This discussion is relevant to version 5.2 and later of the MicroSim software. The example circuit is a basic frequency-comparator (see Figure 32). All parts used in the schematic are provided in the standard Symbol and Model Libraries. One implementation is chosen for PSpice simulation to demonstrate the circuit’s behavior.

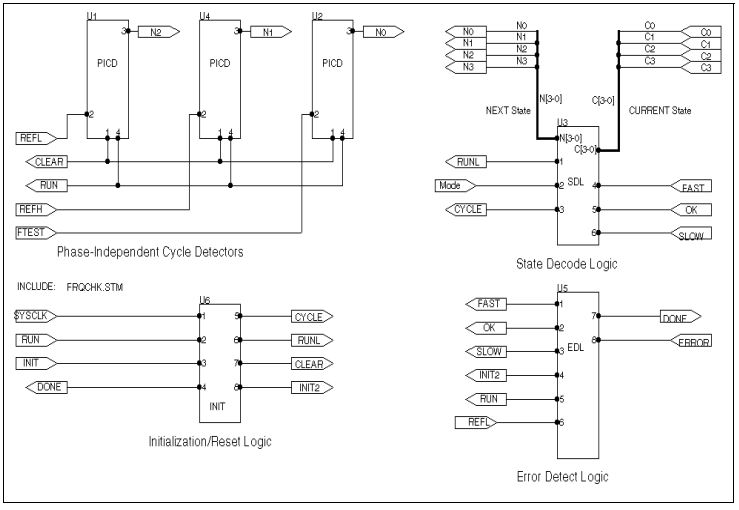


Figure 32 *Top-level schematic for the frequency-comparator circuit*

The frequency-comparator circuit accepts two reference frequency inputs, and a test frequency input which is compared to the references. After initialization and start-up, the circuit produces fast, slow, OK, and error indications. Operation is continuous as long as both of the reference signals are applied.

Initialization is accomplished by applying a low pulse to the INIT input, having a minimum width of 40 nsec. At least 40 nsec after the negative-going edge of the INIT input, circuit operation

commences upon applying a negative-going edge to the RUN input.

Outputs of the circuit—SLOW, FAST, OK, and ERROR—are pulses indicating the result of comparing the test frequency signal, FTEST, to the low and high frequency reference signals, REFL and REFH, respectively. The ERROR pulse is generated if more than 7 complete periods of the REFL signal are observed with no activity on the FTEST input during that time.

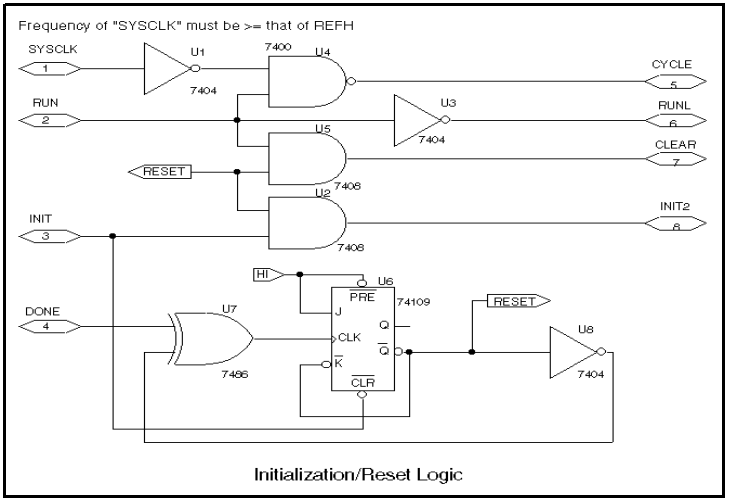


Figure 33 *INIT block implementation*

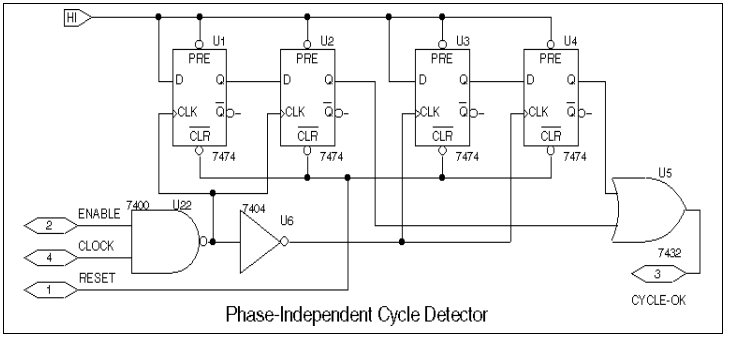


Figure 34 *PICD block implementation*

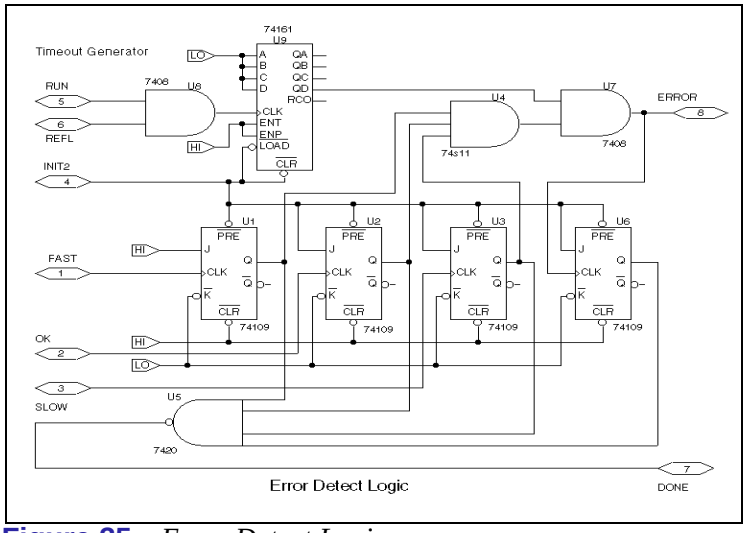


Figure 35 *Error Detect Logic*

Implementation

The frequency-comparator circuit is designed in Schematics using hierarchical blocks for the initializer (INIT block), cycle-detectors (PICD blocks), state-decoder (SDL block), and error-detector (EDL block). The design has two alternative implementations: a gate-level implementation using off-the-shelf 74xx parts (see Figure 36), and a functionally equivalent implementation using a mixture of 74xx parts and a commonly available Programmable Array Logic (PAL) device, PAL20RP4B (see Figure 37). Both implementations use the digital stimulus include file, “freqchk.stm,” providing definitions for the INIT, RUN, MODE, REFH, REFH, FTEST, and SYSCLK input signals (see Figure 21 on page -58).

The design alternatives are implemented as two views of the SDL block, with the DEFAULT view being the gate-level implementation, and the PAL-IMPL view being the PAL implementation. For the PAL-IMPL view, the data required to program the PAL20RP4B device is supplied in a JEDEC file, “frqchk.jed,” generated using OrCAD/PLD (see Figure 1 on page 87).

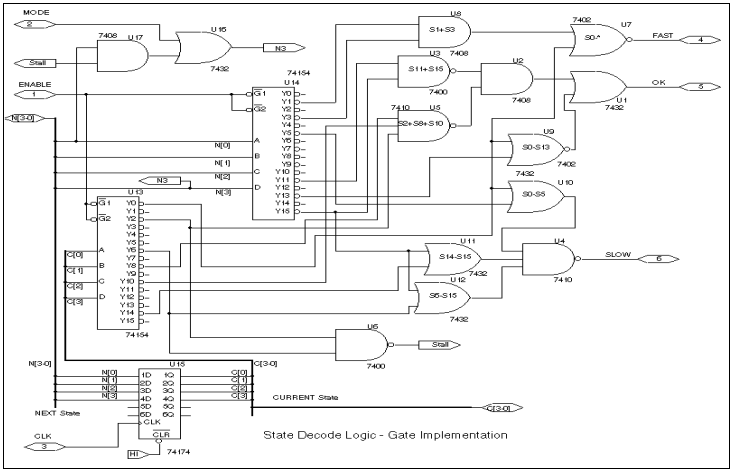


Figure 36 *One implementation of the SDL block—the gate-level view*

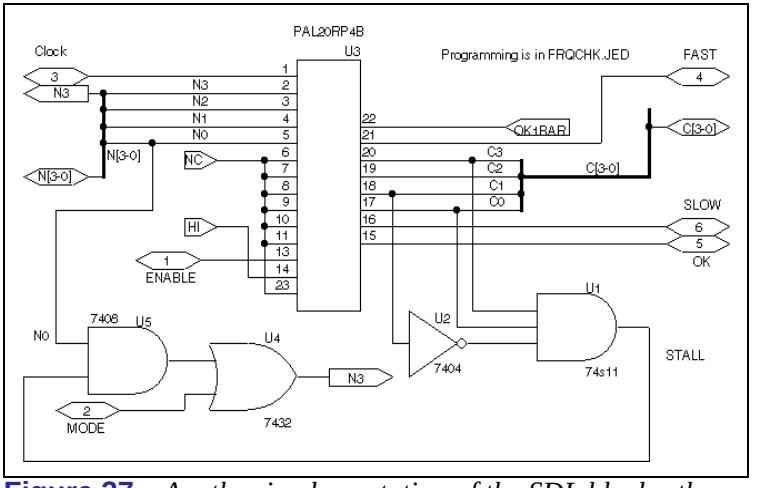


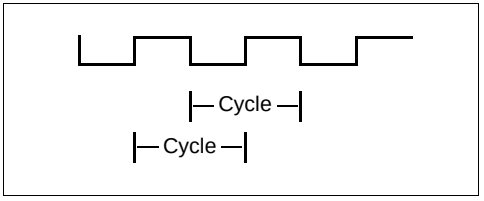
Figure 37 *Another implementation of the SDL block—the PAL view*

Operation

The three frequency inputs—REFL, REFH, FTEST—each drive a separate instance of a cycle-detector circuit (PICD

blocks). Each cycle-detector is made up of two pairs of D-type flip-flops and a few basic gates. After having been "reset" and "enabled," the cycle-detectors output a HI level as soon as two similar edges (e.g., falling) have been applied. This indicates that one complete period of the input signal has been observed.

The circuit implements a simple finite-state machine (see Figure 38) that recognizes the order in which the individual frequency inputs make complete cycles.



For example, suppose that the REFH signal period is observed first (generating N1), followed by the REFL signal period (generating N2), then the FTEST period (generating N0). This indicates that the FTEST frequency is too low and that the SLOW output should be pulsed. But if the FTEST period is observed before the REFL cycle, an OK pulse is produced. The state machine "current state" simply represents the order of activity that has been observed since the last "initialization" or "reset," which occurs every time any kind of output pulse is generated.

The cycle-detectors monitor the input activity and produce the "next state" value (N3, N2, N1, N0), which is fed to the state-decoder (SDL block). At a rate determined by the system clock, SYSCLK, this "next state" becomes the "current state"; the 74154 4/16 decoders in the gate-level view of the state-decoder, continually provide unary logic indications of the next/current "transitions" (since "next state" values are not synchronized to SYSCLK). The random combinational logic in this same view recognizes the specific transitions that comprise the conditions of interest, i.e., FAST, SLOW, and OK, as per the state-transition diagram. (In the PAL view, the PAL20RP4B device replaces all of the decoding logic as well as the 4-bit register representing the "current state" value. The alternative implementations are functionally identical.) Note that the output indicators are not "static" state assignments; they are derived

from selected state transitions. Thus, S14 → S15 recognizes a SLOW condition, while S10 → S15 signifies an OK condition.

The error-detector logic (EDL block) waits for the TIMEOUT signal output by the timeout generator. The timeout generator is simply a counter whose Q3 output indicates that the 8th rising edge of the low frequency reference, REFL, has occurred. If none of the normal output indicators (SLOW, FAST, or OK) have occurred before TIMEOUT, the ERROR output is asserted. The error-detector also asserts its DONE output whenever any of FAST, SLOW, OK, or ERROR have occurred.

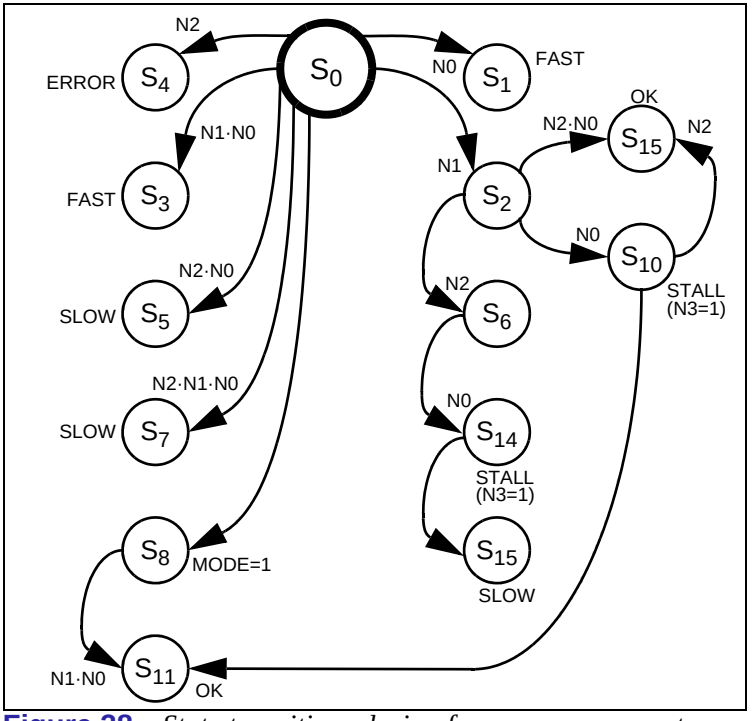


Figure 38 *State transitions during frequency-comparator operation*

The initialization/reset logic (INIT block) performs two functions. One distributes the effects of the INIT and RUN inputs, as defined in the stimulus include file, “frqchk.stm” (see Figure 40). The other uses the DONE signal from the error-detector to generate a RESET pulse; this has the same effect as the external RUN pulse—to restore the state machine to its starting state (0) as well as reset the cycle-detectors, timeout

generator, and flip-flops in the error-detector. Normal operation then resumes.

```
OrCAD PLD 386
Type: PAL20RP4B
*
QP24* QF2568* QV1024*
F0*
L0000 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 01 *
L0040 11 11 11 11 11 11 11 01 11 11 11 01 11 11 11 11 11 11 11 11 *
L0080 11 11 11 11 11 11 11 11 11 10 11 11 11 11 11 11 11 11 11 11 *
L0120 11 11 11 11 11 11 11 11 11 11 11 11 11 10 11 11 11 11 11 11 *
L0320 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 01 *
L0360 10 11 10 11 11 11 01 01 11 01 11 01 11 01 11 11 11 11 11 11 *
L0640 10 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 *
L0960 11 11 10 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 *
```

```
L1280 11 11 11 11 10 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 *
L1600 11 11 11 11 11 11 10 11 11 11 11 11 11 11 11 11 11 11 11 11 *
L1920 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 01 *
L1960 10 11 01 11 10 11 01 01 11 01 11 01 11 01 11 11 11 11 11 11 *
L2000 01 11 01 11 01 11 01 10 11 10 11 10 11 11 11 11 11 11 11 11 *
L2240 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 01 *
L2280 01 11 01 11 10 11 01 01 11 01 11 01 11 01 11 11 11 11 11 11 *
L2320 01 11 11 10 01 11 01 11 11 11 11 11 11 11 11 11 11 11 11 11 *
L2560 11 11 11 11 *
C4B0E*
CCF0
```

Figure 39 *JEDEC file containing the programming for the PAL20RP4B*

```
* “frqchk.stm” stimulus file
*
uh1 stim (4,1111) $g_dpwr $g_dgnd
+ INIT RUN MODE REFL
+ IO_STM IO_LEVEL=0
+ 0s 1100
+ 2055ns 0100
+ 2135ns 1000
+ 2175ns 1000
+ 2215ns 1000
+ 2255ns 1100
+ 5us 1101
+ label=loop1
+ +10us 1100
+ +10us 1101
+ +10us goto loop1 -1 times

uh2 stim (1,1) $g_dpwr $g_dgnd
+ REFH
+ IO_STM IO_LEVEL=0
+ 0s 0
+ +3us 1
+ label=loop1
+ +5us 0
+ +5us 1
+ +5us goto loop1 -1 times

uh3 stim (1,1) $g_dpwr $g_dgnd
+ FTEST
+ IO_STM IO_LEVEL=0
+ 0s 0
+ label=loop1
+ +20us 1
+ +20us 0
+ +20us goto loop1 5 times
+ +0s 1
+ label=loop2
+ +3us 0
+ +3us 1
+ +3us goto loop2 20 times
+ +0s 1
+ label=loop3
+ +6us 0
+ +6us 1
+ +6us goto loop3 10 times
```

Figure 40 *Stimuli for the INIT, RUN, MODE, REFL, REFH, FTEST, and SYSCLK inputs*

```
uh5 stim (1,1) $g_dpwr $g_dgnd
+ SYSCLK
+ IO_STM IO_LEVEL=0
+ 0s 0
+ +2us 0
+ label=loop1
+ +800ns 1
+ +800ns 0
+ +800ns goto loop1 -1 times
```

Figure 41 *Stimuli for the INIT, RUN, MODE, REFL, REFH, FTEST, and SYSCLK inputs (continued)*

PSpice Simulation—PAL View

To simulate the PAL implementation of the frequency-comparator circuit, these steps must be followed:

The transient analysis must be enabled under the Analysis/Setup command. The transient analysis is defined with: `Print Step = 1us, Final Time = 1ms.`

All flip-flops must be initialized in the 0 state (rather than the default X state). This is set with the DIGINITSTATE option under the Analysis/Setup/Options command. This allows the simulator to properly initialize the circuit by forcing the reset logic to a deterministic state (non X; the hardware implementation would eventually sync itself to the input stimuli and operate correctly).

In the top-level schematic, the SDL block is the only block with more than one view. Without further setup, Schematics will generate the PSpice netlist using the DEFAULT gate-level view for SDL. To use the PAL view instead, the PAL-IMPL view name must be specified in the View field of the Translators dialog (under the Configure/Translators command).

After running the simulation by selecting Analysis/Run PSpice, the state-machine operation is viewed in Probe by placing markers on the appropriate wires and buses (using the Markers/

Mark Voltage/Level command), or by typing the signal names in the Probe dialog under the Trace/Add command as follows:

```
SYSCLK, REFH, REFL, FTEST
FAST, SLOW, OK, ERROR
{N3, N2, N1, N0}            ;NEXT
{C3, C2, C1, C0}            ;CURRENT
```

Figure 42 demonstrates the correct response of the circuit to the digital stimulus at FTEST. If you are currently running version 5.3 of the MicroSim application—System 3 (under Microsoft Windows or Sun OpenWindows), and would like a copy of the schematic files for the frequency-comparator circuit, please contact MicroSim Technical Support.

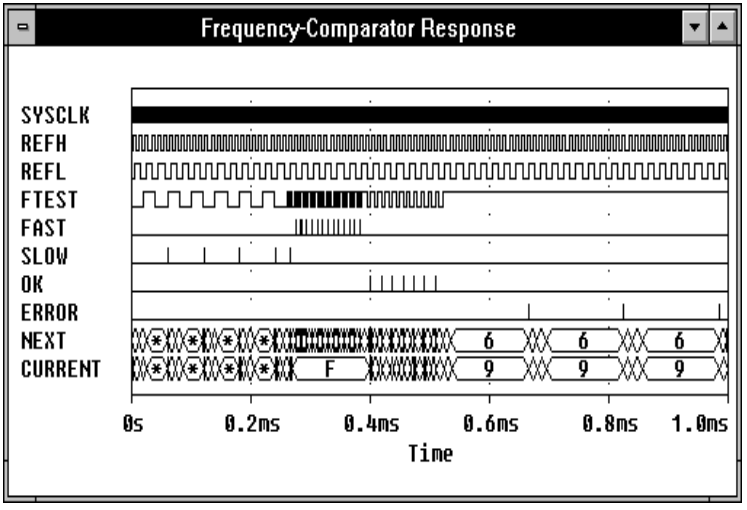


Figure 42 *Frequency-comparator output as FTEST input is varied*

Filter Models Implemented with ABM

by Bashir Al-Hashimi, PhD, School of Engineering, Staffordshire University, Stafford, ST180AD England

Analog behavior modeling (ABM) allows the simulation of analog circuits using mathematical equations. This article shows how filter behavioral models are developed and implemented using the Laplace function of MicroSim PSpice, version 6.2. Given the filter bandwidth and order, the models simulate lowpass, highpass, bandpass, and band-reject filters. For ease of use, the models are developed as parameterized subcircuits. Simulation examples are included to demonstrate the use of these models.

Introduction

Filters are often described in terms of a number of parameters including type, order, and response. There are four filter types:

- Lowpass
- Highpass
- Bandpass
- Band-reject

The order of the filter usually determines the amount of attenuation the filter provides—the higher the order, the more the attenuation. There are a number of filtering responses available. The most commonly used are Butterworth, Chebyshev, and Bessel. Each response has its advantages and disadvantages.

The Butterworth response, for example, has a maximally flat magnitude passband, while the Chebyshev has steeper attenuation characteristics than the Butterworth. The Bessel has a linear phase response and therefore an excellent pulse response. More information on filters is available in the

Electronic Filter Design Handbook by A.B. Williams. See reference [2].

Lowpass Filter Behavioral Models

A block diagram of a general lowpass filter is shown in Figure 43. The diagram consists of one first order and a number of second order sections, allowing different filter orders to be simulated. For example, connecting one first and two second-order sections yields a fifth-order filter. The overall voltage transfer function of the circuit shown in Figure 43 is obtained by multiplying the transfer functions (TF) of the individual sections:

$$(V_{out}/V_{in})=(1st\text{-}order\,TF) * (2nd\text{-}order\,TF)^N$$

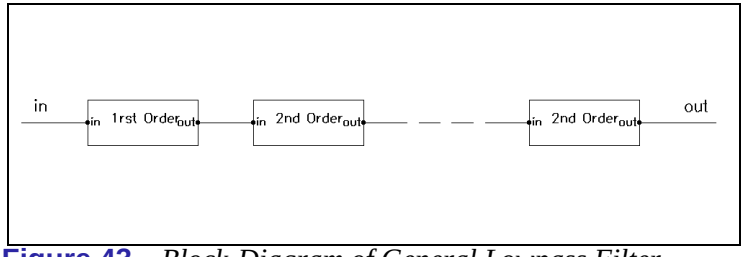


Figure 43 *Block-Diagram of General Lowpass Filter*

The first- and second-order section transfer functions, H(s), are

$$H(s)=x/(s+x)\ ;\ x=2\pi Fc\alpha$$
$$H(s)=x^2/(s^2+(x/Q)s+x^2)\ ;\ x=2\pi Fc\varpi_0$$

where *s* is the Laplace variable, and *Fc* is the filter cutoff frequency or bandwidth. The parameters α, ϖ_0, and *Q* define the pole positions of the various filtering responses (Butterworth, Chebyshev, and Bessel). To simulate the three different responses, it is necessary to define three models, one for each response.

Figure 44 shows the behavioral Butterworth lowpass filter model.

```
.SUBCKT Butt_LP 1 2 params: Fc=1 ord=1  ; subcircuit description
.PARAM pi=3.14159                       ; constant
.FUNC lp_1(x) {x/(s+x)}                 ; 1st-order lowpass transfer
                                        ; function
.FUNC lp_2(x,Q) {(x*x)/(s*s+x/Q*s+x*x)} ; 2nd-order lowpass transfer
                                        ; function
* a1-a4 and b1-b4 determine which filter sections are selected, given
* the filter order.
*
.PARAM a1={table(ord,1,1,2,0,3,1,4,0,5,1,6,0,7,1,8,0,9,1)}
.PARAM a2={stp(ord-1.5)} a3={stp(ord-3.5)} a4={stp(ord-5.5)}
+ a5={stp(ord-7.5)}
.PARAM b1={1-a1} b2={1-a2} b3={1-a3} b4={1-a4} b5={1-a5)}
*
* alpha, omega and Q values of the Butterworth response for each section
* are looked up from these tables, based on the filter order. Up to
* 9th-order (ord) filter is allowed.
.PARAM alpha_b={table(ord,1,1,2,0,3,1,4,0,5,1,6,0,7,1,8,0,9,1)}
.PARAM omega1_b={table(ord,1,0,2,1,3,1,4,1,5,1,6,1,7,1,8,1,9,1)}
.PARAM Q1_b={table(ord,1,0,2,0.707,3,1,4,1.307,5,1.618,6,1.932,
+ 7,2.247,8,2.564,9,0.532)}
.PARAM omega2_b{table(ord,3,0,4,1,5,1,6,1,7,1,8,1,9,1))}
.PARAM Q2_b={table(ord,3,0,4,0.541,5,0.618,6,0.707,7,0.802,
+ 8,0.90,9,0.653)}
.PARAM omega3_b={table(ord,5,0,6,1,7,1,8,1,9,1)}
.PARAM Q3_b={table(ord,5,0,6,0.518,7,0.555,8,0.601,9,1)}
.PARAM omega4_b={table(ord,7,0,8,1,9,1)}
.PARAM Q4_b={table(ord,7,0,8,0.509,9,2.879)}
*
E 2 0 laplace {V(1)}={                  ; VCVS with laplace description
+ (b1+a1*lp_1(2*pi*alpha_b*Fc))*        ; 1st order
+ (b2+a2*lp_2(2*pi*omega1_b*Fc,Q1_b))*  ; 2nd order
+ (b3+a3*lp_2(2*pi*omega2_b*Fc,Q2_b))*
+ (b4+a4*lp_2(2*pi*omega3_b*Fc,Q3_b))*
+ (b5+a5*lp_2(2*pi*omega4_b*Fc,Q4_b))}
.ends Butt_LP                           ; end of subcircuit description
```

Figure 44 *Behavioral Butterworth Lowpass Filter Model*

The model implements the overall voltage transfer function of the filter as shown in Figure 43 using a controlled voltage source (E component) that has the Laplace description. Although the concept is general in that it allows an nth-order lowpass filter to be simulated, Figure 44 shows that the model is limited to simulating a maximum of a ninth-order filter, which is made up of one first-order section and four second-order sections. It is often considered that a ninth-order filter is adequate for most applications. The filter model can easily be increased to simulate filters greater than 9th order by adding second-order sections.

The transfer functions are specified through .FUNC statements. The parameters a1 through a4 and b1 through b4 determine which filter sections of the model are selected by using a simple selection algorithm, based on the order of the filter specified by the user. The selection algorithm is implemented using .PARAM statements as shown in the listing. The filter order is defined by the subcircuit parameter, ORD. Note that the MicroSim PSpice function “stp,” which is used in the model, describes a step function where stp(x) is 1 if x>0 and is 0

otherwise. The α, ϖ_0, and Q values of the Butterworth response are defined as lookup tables using .PARAM statements. These values are looked up automatically given the order of the filter.

The model is described as a subcircuit called Butt_LP with the input at node “in” and the output at node “out” as shown in Figure 45.

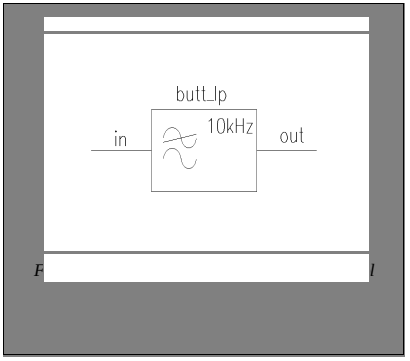


Figure 45 *Behavioral Butterworth Lowpass Filter Symbol*

The subcircuit has two parameters: the filter cutoff frequency (FC) and its order (ORD). These subcircuit parameters are given default values, which are arbitrarily set to 1. They will be changed to the required cutoff frequency and order when the subcircuit is called.

Similar Chebyshev and Bessel lowpass filter models can be easily developed. The Chebyshev filter model is called *Ch2p5_LP*, while the Bessel model is called *Bessel_LP*. Chebyshev filtering response exists for a range of passband ripple [1]. The ripple has been fixed at 0.25dB in the case of the *Ch2p5_LP* model. To develop Chebyshev models with different values of ripple, the values α, ϖ_0, and Q are needed for the required ripple, which are readily available [1]. The filter models can be obtained from the MicroSim BBS (714-830-1550) in “filters.exe” (a self-extracting zip file). Note that both of these models are capable of simulating up to ninth-order filters.

Example 1

To illustrate the use of the models, consider the following example. Here, the Butterworth lowpass filter model is used to obtain a family of curves for second- through ninth-order responses. Assume the filter has a cutoff frequency of 10 kHz. Using MicroSim Schematics, the circuit of Figure 46 is drawn.

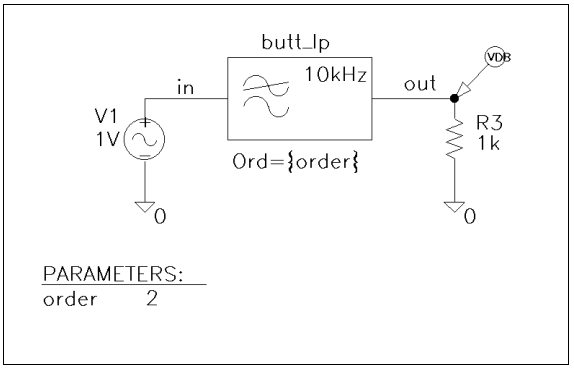


Figure 46 *Butterworth Lowpass Filter Circuit*

The filter Butt_LP has an AC source on its input and a 1K load resistor on its output. On the Butt_LP symbol, the attribute FC has a value of 10K, and ORD is set to the global variable defined in the global parameter block (PARAM). Finally, a VDB voltage marker is placed on the output node to view the results.

In MicroSim Schematics, be sure to configure the symbol library “filters.slb” (obtained from the MicroSim BBS) in the Schematic Editor through Options/Editor Configuration/Library Settings/Add, and the model library “filters.lib” through Analysis/Library and Include Files/Add.

The analysis consists of a 1000-point linear AC sweep and a parametric analysis. The parametric analysis steps the global parameter, ORD, from 2 to 9. The simulated frequency response of the filter for various orders is shown in Figure 47.

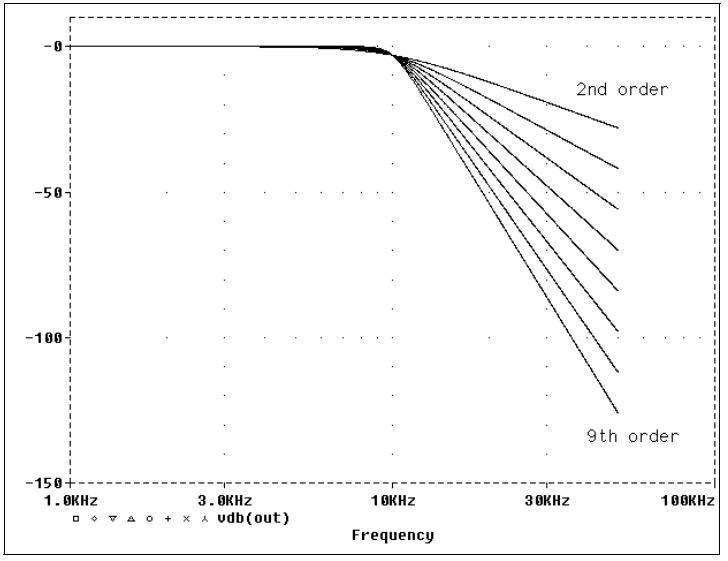


Figure 47 *Butterworth Lowpass Filter FrequencyResponse for Various Orders*

Highpass Filter Behavioral Models

Three highpass filter models are developed similarly. The three models are:

- Butt_HP (Butterworth filter)
- Ch2p5_HP (0.25dB ripple Chebyshev filter)
- Bessel_HP (Bessel filter)

Each model is capable of simulating up to a ninth-order filter and is used in a similar manner to that of the lowpass as described in Example 1.

Bandpass Filter Behavioral Models

One approach to obtain bandpass filters is to cascade lowpass and highpass circuits [2]. The cutoff frequency of the lowpass circuit defines the lower –3dB point of the bandpass filter bandwidth, while the cutoff frequency of the highpass filter defines the upper –3dB point of the filter bandwidth. A behavioral bandpass filter model representation is shown inFigure 48, based on this approach and by using the lowpass and highpass filter models discussed earlier.

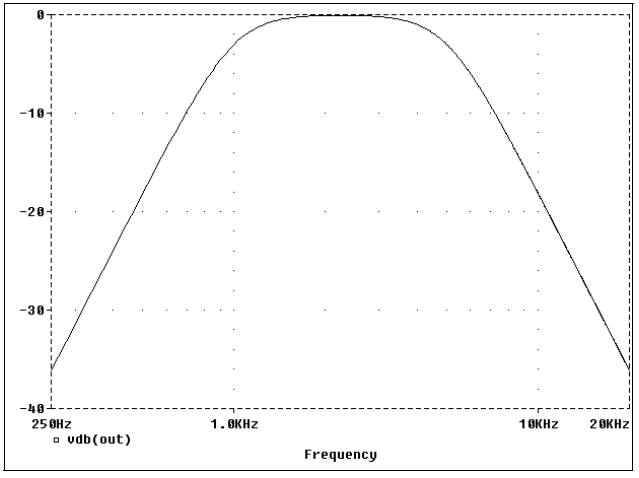


Figure 48 *Butterworth Bandpass Filter Frequency Response*

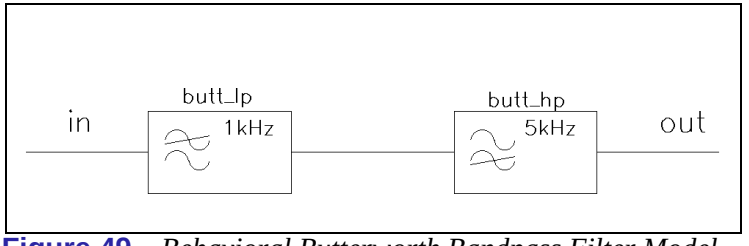


Figure 49 *Behavioral Butterworth Bandpass Filter Model*

The three bandpass filter models are:

- Butt_BP (Butterworth filter)
- Ch2p5_BP (0.25dB ripple Chebyshev filter)
- Bessel_BP (Bessel filter)

Example 2

To demonstrate the use of the bandpass filter models, consider simulating a bandpass circuit with the following specifications:

lower –3dB point=1kHz
upper –3dB point=5kHz
30dB minimum at 0.3kHz and 20kHz

Assume a Butterworth response is required. To meet the specifications, third-order lowpass and highpass filters are required [2].

The circuit is shown in Figure 50.

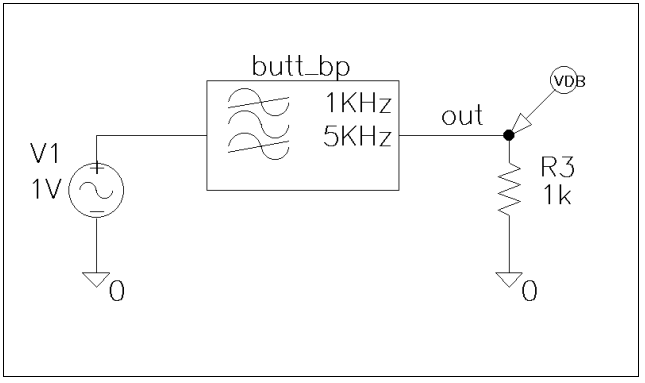
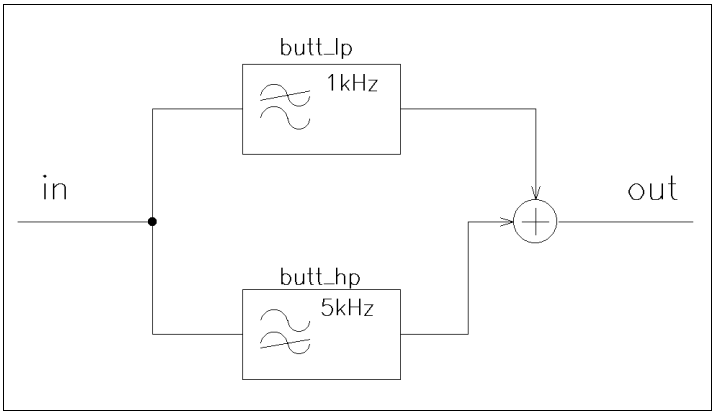


Figure 50 *Butterworth Bandpass Filter Circuit*

The *Butt_BP* filter symbol has attributes FCL=1kHz and FCH=5kHz for low and high cutoff frequencies, respectively. A single AC source is frequency swept over 0.25kHz to 50kHz (1000 points). The simulated frequency response of the filter is shown in Figure 48.

Band-Reject Filter Behavioral



Models

Figure 51 is a block diagram representation of how a band-reject filter can be realized [2].

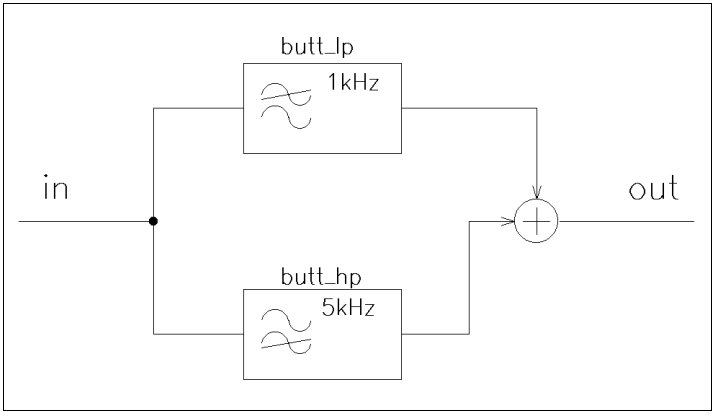


Figure 51 *Butterworth Band-Reject Filter Behavioral Model*

This model is based on summing outputs of the lowpass and highpass filter models. The cutoff frequency of the lowpass circuit defines the lower –3dB point of the bandpass filter bandwidth, while the cutoff frequency of the highpass circuit defines the upper –3dB point of the filter bandwidth.

Three band-reject filter models are contained in “filters.lib.” Each subcircuit has three parameters, FCL, FCH, and ORD. The three models are:

- Butt_BR (Butterworth filter)
- Ch2p5_BR (0.25dB ripple Chebyshev filter)
- Bessel_BR (Bessel filter)

Example 3

Figure 52 contains a fifth-order, 0.25dB ripple Chebyshev band-reject filter with a lower –3dB point at 1kHz and the upper –3dB point at 5kHz.

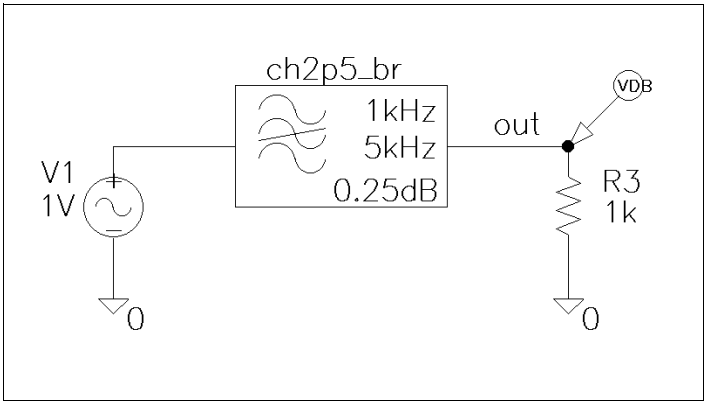


Figure 52 *Chebyshev Band-Reject Filter Circuit*

The simulated frequency response is shown in Figure 53.

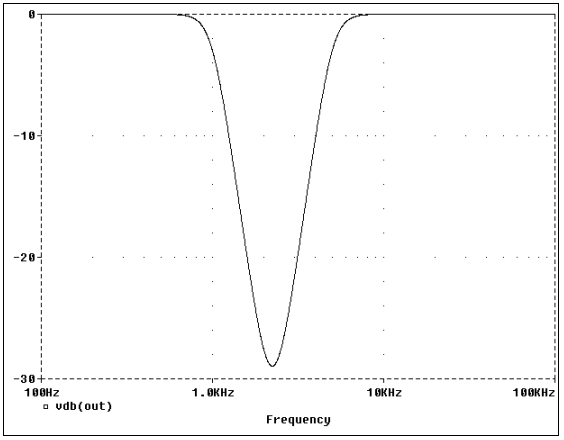


Figure 53 *Chebyshev Band-Reject Filter Frequency Response*

Library Availability

The symbol and model libraries used in this article are contained in a self-extracting zip file which can be downloaded from the MicroSim BBS. To download the file, dial the BBS at (714) 830-1550, and from the main menu select [T]ech Support, then [6] File Transfer, then [1] Download User Requested Files, and then download “filters.exe.”

References

[1] Van Valkenburg, M.E., *Analog Filter Design*, Holt, Rinehart & Winston, New York, 1982.

[2] Williams, A.B., *Electronic Filter Design Handbook*, McGraw-Hill Book Company, USA, 1981.

[3] Al-Hashimi, B.M. The Art of Simulation Using PSpice: Analog & Digital, CRC Press, USA, 1995.

Frequency-Domain Modeling of Real Inductors

MicroSim Corporation Newsletter, January 1991.
Originally titled SPICE is SPICE...(part 6 in a series)

SPICE is SPICE... right? At least until limits get in your way, keeping you from creating effective simulations. In this discussion, let's look at frequency-domain modeling of real inductors using the Analog Behavioral Modeling option to PSpice. Other vendors are crowing that generic SPICE always had behavioral models and no extensions are needed. They support this approach with a library of control theory models and motor/servo equivalents. This example will demonstrate a sample of the useful capabilities Analog Behavioral Modeling provides for real electrical circuits.

Wide band inductors combine with capacitances to make frequency-selective circuits. Ideally, these inductors should have low winding resistance, low core loss, and low distributed capacitance. In practice, these ideals are approximated only over a limited range by real inductors. Various losses limit the quality (Q) factor of an inductor, which is defined as the ratio of inductive reactance to series resistance. As the Q factor increases, frequency-selective circuits can realize sharper cut-off, more defined resonance, and higher attenuation ratios.

For small signals, ferromagnetic hysteresis loss is low, as the inductor flux density is low. Also, for a constant-induced voltage, this loss decreases with increasing frequency. The remaining losses may be modeled as an effective series resistance whose value is frequency dependent. The formula for Q, neglecting hysteresis loss, becomes

$$Q = \frac{\omega \cdot l}{R_{dc} + R_{ac} + R_d}$$

where "R_{dc}" is winding resistance, "R_{ac}" is resistance due to core losses not involving hysteresis (e.g. eddy currents), and "R_d" is resistance due to winding dielectric losses.

Winding resistance is a straightforward calculation based on wire resistance per unit length, and how much wire is required to obtain the desired inductance. For a nominal utilization of the space around a core to wind the wire, there is a direct relation between winding resistance and inductance. The manufacturer may even specify this relationship in ohms/mH, or will plot maximum number of turns versus wire size, thereby forcing you to check a wire table to calculate the resistance.

The resistance due to core losses is described by V. E. Legg as

$$R_{ac} = \mu \cdot L \cdot e \cdot (2\pi\omega)^2 + c \cdot 2\pi\omega$$

where “e” represents the eddy current loss, which increases with the square of frequency, and “c” represents a residual loss that is proportional to frequency. The factors “μ” and “L” are the relative permeability of the core material and dc inductance of the device, respectively.

Dielectric loss resistance is more important at higher frequencies, and may even be negligible at frequencies substantially below the self-resonant frequency of the inductor. This loss is calculated as

$$R_d = d \cdot \omega^2 \cdot L^2 \cdot C_d$$

where “d” is an empirical factor and “C_d” is the distributed capacitance of the winding. Some manufacturers provide these values.

Combining the above effects into a subcircuit model for a “real” inductor is simple using Analog Behavioral Modeling. The model will consist of an ideal inductor in series, with three resistors to represent the various losses. The inductor and first resistor have constant values, which are computed from the parameters used by the subcircuit. The other two resistors are modeled as frequency-dependent voltage sources, whose voltages depend on the current through the subcircuit and the frequency, following Ohm’s Law

$E(freq) = I \cdot R(freq)$

Generating a frequency-dependent voltage, with a zero phase angle, is done simply with the Laplace form in Analog Behavioral Modeling: multiplying the Laplace variable “s” by its negative “-s” provides a value proportional to frequency squared, but the phase has been canceled. To get something

proportional to frequency, just use the square root function on the previous result.

Since values for the various parameters are specified to only two digits, we can afford to approximate *pi* as the square root of 10 (a 0.7% error) to simplify the formulas. In particular

$$\text{freq}^2 = \frac{-(s)\cdot s}{4\cdot\pi^2} \approx \frac{(-s)\cdot s}{40}$$

Now, the subcircuit can be written directly by following the previous formulas:

```
.subckt L N1 N2 params:
+ t = 1                          ; no. of turns
+ mu = 100                       ; relative permeability
+ nHpt = 1K                      ; nH per turn
+ rpt = 1m                       ; ohms per turn
+ c = 10u                        ; residual loss
+ e = 10n                        ; eddy current loss
+ d = .05                        ; power factor
+ cd = 50p                       ; distributed capacitance

lw N1 a                          ; winding inductance
+ { 1n * nHpt * t*t }

rw a b                           ; winding resistance
+ { rpt * t }

vc b c dc 0                      ; current sense
```

```
ec c d                   ; core loss (resistance)
+ laplace { i(vc) } =
+ { mu * (1n*nHpt*t*t) * (e*(-s*s/40) + c*sqrt(-s*s/40)) }
ed d N2          ; winding dielectric loss (resistance)
+ laplace { i(vc) } =
+ { d * (-s*s/40)*sqrt(-s*s/40) * pwr(1n*nHpt*t*t,2) * cd }
.ends
```

Testing the model is a matter of measuring Q versus frequency. For this test, we build a series LC circuit, stimulate it with a unity source, and measure the magnitude of the voltage across the capacitor. The peak value occurs at resonance and this value is Q, by definition. By choosing capacitance values to provide resonant frequencies that match one of the increments in the frequency analysis, the corresponding Q value is correctly simulated. In addition, if we specify more frequency steps than

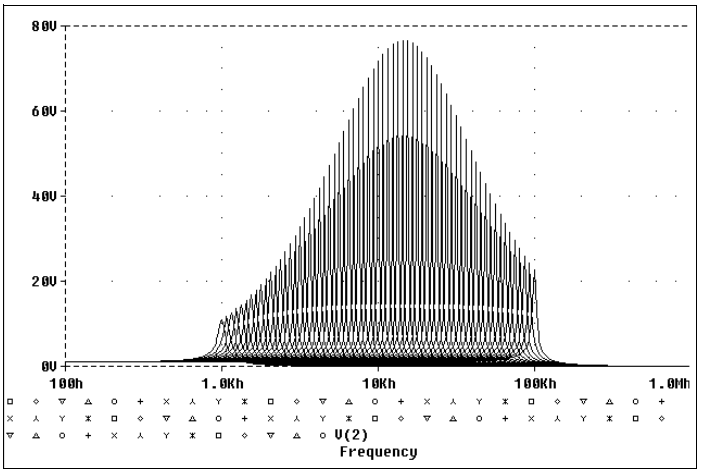
horizontal dots in our display, the resonant curves have maximum smoothness.

```
* Wideband Inductor: Magnetics, Inc. MPP core #55278
.param
+ kcap=1                        ; step multiplier
+ mH=50                         ; design value = 50mH
+ mHpKt=68                      ; mH/(1000 turns)
+ rpmH=.486                     ; ohms/mH

.step dec param kcap .01 100 16
.ac dec 160 100 1Meg

v1 1 0 ac 1
x1 1 2 L params:
+ t ={ sqrt(mH/mHpKt) * 1000 }
+ nHpt={ mHpKt }
+ rpt ={ mHpKt * rpmH / 1000 }
+ mu=160 c=25u e=17n d=.012 cd=50p
c1 2 0
+ {kcap*5.06605918212n} ; nom. 10kHz resonance
```

The Q of the inductor, versus frequency, is shown by the envelope of the resonance curves. Since an ideal inductor has infinite Q, it is helpful to understand what limits Q in a real inductor. Starting at low frequencies and working upward, the “leading edge” of Q is limited by the winding resistance. The “trailing edge” is usually, as in this example, limited by eddy current losses, which increase with frequency squared. The “peak” in Q could be limited by residual losses, which increase proportionally to frequency, but often the eddy current losses dominate. The winding dielectric losses are not noticeable in this example, but would show up with cores using higher resistivity materials that reduce the eddy current losses.

Working from standard equations, we developed an accurate model for a wideband inductor. The simulated results match the manufacturer’s data. Analog Behavioral Modeling provides more than convenience: using functional descriptions by formula and/or look-up table, in both instantaneous and frequency domains, and with automatic transformation between

these domains, PSpice handles a broader range of modeling problems than generic SPICE.



Improve Simulation Accuracy When Using Passive Components

The Design Center Source newsletter, April 1994, by Steven C. Hageman Applied DC

This is a slightly abridged version of Mr. Hageman's 1994 article.

Introduction

Everyone seems to get on the band wagon when it comes to making better and better IC models. Today opamp models exhibiting five to ten poles or zeros are common. While these models add to PSpice simulation accuracies, the importance of other components should not be overlooked.

In particular, passive components can have as much influence on simulation accuracy as do IC models. This article surveys the effect of frequency and temperature on the behavior of selected common passive components. Suggested techniques for improving simulation accuracy using these components are also presented.

Frequency Effects

Resistor models

At first glance, resistors appear to operate in a straightforward manner. In a DC circuit that does not overheat, they mostly all work. However, as operating frequency increases, their behavior can change. Whether the resistor behaves in a resistive,

capacitive, or inductive manner depends on the resistor value and frequency of operation.

As can be seen from Figure 55, a standard 0.25-watt carbon or metal film resistor has three major elements associated with it. The first is the resistor element itself; this is the value that is read from the side of the resistor. Two major parasitic elements are also included free of charge with every resistor that you buy. The first is a parasitic capacitor that is formed across the resistor; this capacitor reduces the resistor's impedance at high frequency. The second parasitic is the lead inductance; this inductance tends to increase the impedance as frequency increases.

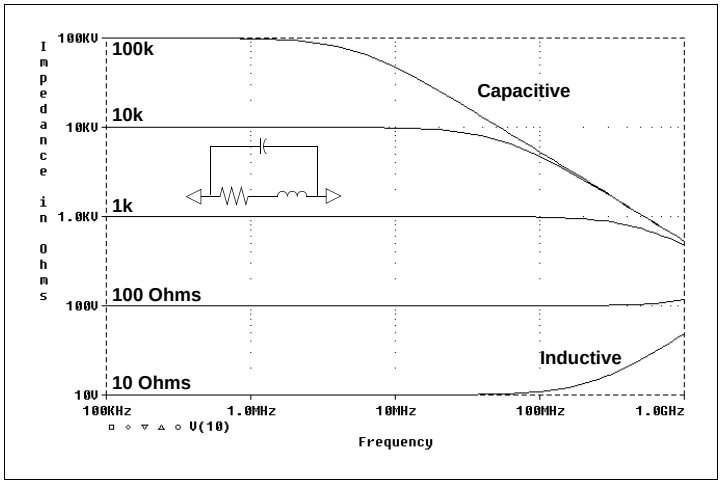


Figure 55 *Small film resistors two major parasitic components affecting frequency response*

The question arises, "How can the shunt capacitance reduce impedance while the series inductance simultaneously increases it?" The answer is that one of these effects will dominate, depending on the resistor value. A small resistor does not exhibit much shunt capacitance, but its impedance increases with frequency due to the series inductance. Just the opposite happens with large resistors; the capacitance reduces the impedance as frequency increases, whereas the inductance is negligible.

Figure 55 shows a plot of impedance versus frequency for several resistor values. This plot clearly illustrates that the resistance value has a profound effect on how the resistor behaves at higher frequencies.

The figure also shows small film resistors have at least two major parasitic components. How the parasitics affect the frequency response of the resistor depends on the resistor value. Surface Mount Technology (SMT) types have about ten times greater frequency response.

Figure 56 shows a corresponding model for the 0.25-watt resistor. This model is for a resistor with 0.1-inch leads placed on an insulated printed circuit board (PCB). A resistor that is lying down flat on a solid ground plane with one lead attached to the ground has an additional 0.3 pF shunt capacitance. About 20 nH of additional series inductance should be added for each additional inch of lead or trace length.

```
*---------------------- RES SUBCIRCUIT --------------------
* 0.25-W CARBON FILM / METAL FILM RESISTOR INCLUDING ESL AND
* SHUNT CAPACITANCE EFFECTS.
* Example Call:    XRES 10 20 RES PARAMS: R=4.99K
*-----------------------------------------------------------
*
*                  +---------------- +NODE
*                  |      +--------- -NODE
*                  |      |        +---- PASSED RESISTOR VALUE
*                  |      |        |
.SUBCKT     RES    1      3        PARAMS: R=1
R    1      2      {R}             ; RESISTOR ELEMENT
L    2      3      7N              ; ESL
C    1      3      0.3P            ; SHUNT CAPACITANCE
.ENDS
```

Figure 56 *PSpice model for 0.25-watt film resistor*

Ceramic Capacitor Models

Multilayer ceramic capacitors have much the same parasitic elements as a resistor, just slightly rearranged. Figure 57 illustrates the behavior of a frequency-dependent model for a basic leaded ceramic capacitor of COG, X7R, or Z5U dielectric construction, plotting impedance versus frequency for several typical part values. The equivalent series resistance (ESR) limits the Q of the capacitor; the parasitic inductance and capacitance value set the self-resonant frequency.

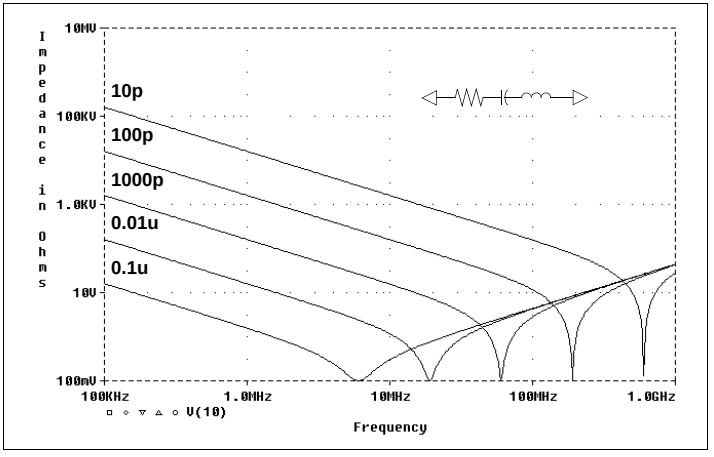


Figure 57 *Capacitors self-resonate at some frequency*

All capacitors self-resonate at some frequency, after which the impedance starts to climb inductively; 805 and 1206 SMT ceramics have approximately double the frequency response of the leaded type shown.

Figure 58 shows the PSpice model for a leaded ceramic capacitor with 0.1-inch leads. As with the resistor model, an additional 20 nH of series inductance should be added for each inch of additional lead or PCB trace length that the capacitor has in a circuit. Additional lead length lowers the self-resonant frequency of the capacitor and limits its high frequency effectiveness.

```
*---------------- CER_CAP SUBCIRCUIT -------------------
* CERAMIC CAPACITOR SUBCIRCUIT WITH ESR AND ESL EFFECTS.
* Example Call:    XCAP 10 20 CER_CAP PARAMS: C=0.01U
*-----------------------------------------------------------
*
*                        +--------------- +NODE
*                        |     +---------- -NODE
CAPACITOR VALUE          |     |      +---- PASSED
*                        |     |      |     |
.SUBCKT     CER_CAP      1     4      PARAMS: C=1
C      1     2      {C}                ; CAPACITOR ELEMENT
R      2     3      0.1                ; ESR
L      3     4      7N                 ; ESL
.ENDS
```

Figure 58 *PSpice model for a leaded ceramic capacitor with 0.1-inch leads*

Design Example

For commonly used resistor values of 100 kilohms or less, the frequency response effects don’t change the circuit’s overall behavior when the frequency is less than about 1 MHz, or when the time steps in a transient analysis are greater than 1 usec. However, outside of these limits, using the frequency-correct models can significantly enhance accuracy.

A 60-dB123 pad was simulated using the PSpice models developed in this article; the simulated circuit closely approximates measurements made on a real circuit; the ideal versus real is off by more than 20 dB at 500 MHz, while the simulation using the frequency-adjusted models is off less than 3 dB at the same frequency.

Other Common Components

Resistors and capacitors are only part of the problem in making accurate PSpice simulations. Conductors (PCB traces and wires) and inductors also deviate from ideal as the frequency increases.

Conductors

A conductor that looks like a small resistor at DC has an increasing impedance with frequency that is dependent on the physical dimensions of the conductor. Its inductance can be approximated by an inductor of about 20 nH per inch of length in series with the DC resistance. Thus, a conductor looks inductive at frequencies as low as

10 kHz up to the length that is about a quarter-wavelength long. At longer lengths, the conductor undergoes multiple pole and zero resonances like an antenna. The frequency where a conductor stops looking inductive and starts to act like an antenna can be found using the formula,

$$F = 2850/L$$

where L is a quarter of the wavelength in inches, and F is in MHz. Thus, a conductor that is 10 inches long will behave like an antenna when the frequency is 285 MHz or greater. Most PCB traces are not long enough to act as antennas, but ribbon cables can be.

On controlled impedance PCB's, the traces look like transmission lines. These can be modeled with the PSpice transmission line models.*

Even power and ground planes used in PCB design don't escape frequency effects. The impedance of a ground plane (or any

large, flat, low inductance trace or plane) doesn't look inductive at higher frequencies; it looks *lossy*. At higher frequencies, the skin effect of the plane starts to dominate and increase the planes impedance. The skin effect is proportional to the square root of the frequency, so it doesn't rise as fast as does that of a wire that is behaving inductively (+20 dB per decade slope of impedance versus frequency).

Inductors

Inductors vary greatly in shape and size depending on the exact job that they are to perform. Power inductors, like the type used in switching power supply output filters, are usually large structures that may self-resonate at frequencies from 500 kHz to 75 MHz. These power inductors are sometimes designed for low loss so they may have a large Q at resonance (small resistive term). This is especially true when the core material of the

inductor is ferrite. (MPP and iron powder core materials have more loss and lower Q values.) The high Q gives rise to a rather narrow, sharp resonance. Above the resonance frequency, the inductor's shunt capacitance dominates. The shunt capacitance is usually large for a big power inductor because of the capacitive coupling among the many turns used.

When modeling power inductors, the resonant frequency is based on the size of the core. Generally, the larger the core, the lower the self-resonant frequency.

Ferrite beads used for EMI control are at the other end of the spectrum. Beads are designed for lossy operation and have very low Q values with relatively low inductance. The self-resonance peak is low and very broad, extending for several octaves of frequency. Beads are best modeled as an inductor with a small shunt resistance on the order of 50 to 100 ohms and a low shunt capacitance of 1-5 pF or less.

---

*. Using the Polaris signal integrity analysis tool (an optional integrated feature of the software), PSpice transmission line models can be automatically derived for traces on a PCB from information in the layout database.

Temperature Effects

Ambient

Passive component values can be subject to temperature effects that are dependent upon the circuit's operational temperature. To account for these effects when simulating, each relevant component needs a .MODEL statement specifying how the particular component value varies with temperature. The built-in PSpice models for resistors, capacitors, and inductors have two temperature effect terms—linear and quadratic.* These terms may be curve fit to a component's actual temperature characteristics.

The linear term can be used alone to give a single-slope fit to any component by specifying the TC1 temperature coefficient in the PSpice .MODEL statement. For example, a 100 ppm per °C resistor could be specified as

```
.MODEL R_100 RES (R=1, TC1=0.0001)
```

TC1=0.0001 relates to 100 ppm/°C. For a 250 ppm/°C resistor, TC1 would equal 0.00025. If the slope is negative, a minus sign must precede the TC1 value.

A single-slope temperature curve is usually sufficient for resistor and inductor simulation; hence, these components are typically governed by a first-order term. In practice, they may *wobble* around the temperature curve, but this wobble is usually within 20% of the expected value. However, this may not be totally accurate over the military temperature range; therefore, it is important to find out how the parts in question actually perform when simulating over very large temperature ranges.

Resistors are not the only components with temperature effects; most capacitors, especially ceramics, have very well-defined temperature curves depending on the dielectric used in their construction. However, a single-slope temperature curve is not sufficient for simulating the most common types of capacitors used in analog circuits.** The X7R and Z5U dielectric types have a fairly large upside-down parabolic curve shape. Their

*. Resistors have an additional exponential temperature coefficient which can be used instead of the linear and quadratic coefficients.

capacitance values fall off (appreciably for the Z5U type) at temperatures greater and lower than 25°C. Therefore, both the linear and quadratic temperature coefficients must be specified in their PSpice .MODEL statements as

```
.MODEL X7R CAP (C=1, TC1=5.75E-5, TC2=-1.285E-5)
.MODEL Z5U CAP (C=1, TC1=2.38E-3, TC2=-1.48E-4)
```

The X7R model is valid for temperatures from -55°C to +125°C while the Z5U model is valid for temperatures from -40°C to +85°C. Both of these models are fairly accurate over these temperature ranges (i.e., within +/- 20%). The temperature curves for these capacitors are normalized to 25°C.

The PSpice-modeled temperature dependence of capacitance for two popular types of ceramic capacitors closely matches the real parts.

Per Component

Beginning with version 5.3 of PSpice, passive components can also be characterized for temperature effects that override (1) the circuit's operational temperature and (2) the temperature, TNOM, at which model parameters are assumed to have been measured. Individual device temperature behaviors can be customized by specifying either the T_ABS, T_REL_GLOBAL, or T_REL_LOCAL parameter in a .MODEL statement. A new measurement temperature can also be defined by setting the T_MEASURED model parameter.

Suppose that a resistor's resistance multiplier is unity when measured at 0°C. To signify this, T_MEASURED can be specified in the resistor's corresponding .MODEL statement as

```
.MODEL RMOD RES(R=1, TC1=0.0001,
T_MEASURED=0)
```

**. Temperature-compensating capacitors and COG capacitors are approximated by a single-slope temperature curve; their behavior is similar to that for the resistors described earlier.

When the circuit is operating at 0°C, R evaluates to 1. At 100°C, R evaluates to 1.01 which is the resistance multiplier (R=1) plus the first order operational temperature effect (TC1 * (TEMP-T_MEASURED)).

T_ABS allows specification of an absolute device temperature. If T_ABS is specified as T_ABS=25, the model is held at 25°C no matter what the circuit's operational temperature is doing. Given the above resistor example, adding T_ABS=25 to the model definition causes R to evaluate to 1.0025 at all times, even if the operational temperature is varied within parametric or DC sweep analyses.

T_REL_GLOBAL is used to specify a device temperature that is relative to the circuit's operational temperature. For example, a power resistor might be dissipating power and be warmer than its surrounding global ambient by 10°C. This can be specified in a .MODEL statement as

```
.MODEL RMOD RES(R=1, TC=0.0001,
T_REL_GLOBAL=10)
```

T_REL_LOCAL is used in the AKO ("a kind of") .MODEL statement. An AKO model references an existing model, thus inheriting the existing model's parameter definitions. Parameter values can be overridden or added by specifying them in the AKO .MODEL statement. Using this technique, the device temperature defined in a new model can be calculated relative to the absolute device temperature specified in a base model. The base model must define the absolute device temperature using the T_ABS parameter. The AKO model must define the relative change to the T_ABS temperature using the T_REL_LOCAL parameter.

For example, a model, RMOD, whose device temperature is 20°C greater than that specified in the RBASE model statement can be defined as

```
* Base Model
.MODEL RBASE RES(R=1, TC1=0.0001, T_ABS=10)
* AKO Model
.MODEL RMOD AKO:RBASE RES(T_REL_LOCAL=20)
```

RBASE sets a resistor's absolute temperature to 10°C. RMOD evaluates to 30 °C. If the T_ABS parameter had not been

defined in the RBASE model, the T_REL_LOCAL definition in RMOD would have been ignored.

Biography: Steve Hageman is an analog designer specializing in power conversion. He owns the consulting firm, Applied DC, and may be reached by FAX/phone at (510) 687-0483.

Including Relays in PSpice Simulations

The Design Center Source newsletter, January 1993

Some systems require simulating the operation of electromechanical relays in order to accurately model system behavior. The model libraries for release 5.2 do not include relays. Therefore creating an accurate relay model can take some time. The models discussed here are included in the 5.3 (and later) software release. An April 1990 application note titled “Modeling an Electromechanical Device” may in some ways add to the confusion, since it discusses modeling the mechanical (and electrical) behavior of a relay. In this application note, we will discuss two approaches to modeling the relay: the mechanical approach taken in the earlier application note, and two purely electrical (behavioral) models.

The mechanical model for the relay is described in more detail in the above mentioned application note. The focus of that application note is modeling the mechanical part of electromechanical devices in general, using the relay as an example. This model constructs an electrical analogy to the mechanical operation of the relay. To do this, it calculates the magnetic and mechanical forces acting on the contact arm of the relay, and simulates the acceleration, velocity, and position of the arm in response to these forces. The electrical contacts of the relay are simulated by switches controlled by the position of the contact arm. There are two problems with this modeling approach: first it requires information about the physical construction of the relay (spring force, contact arm moment, magnetic permeance as a function of contact arm position) which are not normally available to the user of a relay, and second, it takes a lot of computer time to simulate the exact position of the contact arm. Most of this time is wasted if all the user needs to know is whether the contacts are open or closed. This type of physical model could be useful for designing a relay, but it is overkill for simulating its electrical behavior.

If you are interested in a complete physical model for a relay you should take a look at the RELAY_SPDT_PHY model in “misc.lib.”

The second and third approaches simply model the electrical behavior of the relay coil and contacts. Rather than requiring physical construction parameters, these models require behavioral parameters. The first behavioral model does not include contact bounce, and is the fastest to simulate. It requires parameters for coil inductance and resistance, contact resistance, pull-in and dropout coil currents, and make and break times. The model uses a PSpice digital buffer's propagation time to model the make and break times, and uses the AtoD conversion device to model the pull-in/dropout current hysteresis. It uses the DtoA conversion device in an unusual configuration to model the contacts. By using the digital devices it is easy to set the delays using subcircuit parameters, and there are no time step problems which can be caused by very high gain analog switches. The following circuit file shows the simple behavioral model of a relay.

```
* Behavioral model of a relay. (No contact bounce)
.subckt RELAY_SPDT_BHV coila coilb no nc com
+ PARAMS:
+ T_make = 20mSec  ; Time for contact to close when current
                   ; is turned off/on
+ T_break= 10mSec  ; Time for contact to open when current
                   ; is turned off/on
+ I_pull = 35ma    ; Pull-in current
+ I_drop = 25ma    ; Drop-out current
+ R_coil = 100     ; Coil resistance
+ L_coil = 5mH     ; Coil inductance
+ R_open = 100MEG  ; open circuit contact resistance
+ R_close= .05 ; closed circuit contact
```

```
resistance
* electrical model of coil
v_winding coila a1 0
r_winding a1 a2 {R_coil}
l_winding a2 coilb {L_coil}
* make a voltage from the current
e_cc cc 0 value = {Limit(I(v_winding),-3*I_pull,3*I_pull)}
r_cc cc 0 1k
* use digital to create a switch with hysteresis
o_mag cc 0 relay_1 DGTLNET=d digio_1
.model relay_1 doutput (
+ s0name="0" s0vlo={-I_pull} s0vhi={I_pull}
+ s1name="1" s1vlo={I_drop} s1vhi={4*I_pull}
+ s2name="1" s2vlo={-4*I_pull} s2vhi={-I_drop}
+ timestep={T_make/1000}
+ )
* by using min/max delay we can use a single
* N-device to simulate both the no and nc contacts.
u_dly buf dpwr dgnd d cnt relay_2 digio_1 MNTYMXDLY=4
.model relay_2 ugate (
+ tplhmn={T_break           tphlmn={T_break}
+ tplhmx={T_make}           tphlmx={T_make}
+ )
n_cnt com no nc relay_3 DGTLNET=cnt digio_1
.model relay_3 dinput (
+ s0name="0" s0tsw={T_make/1000} s0rlo={R_open} s0rhi={R_close}
+ s1name="1" s1tsw={T_make/1000} s1rlo={R_close} s1rhi={R_open}
+ s2name="R" s2tsw={T_make/1000} s2rlo={R_open} s2rhi={R_open}
+ s3name="F" s3tsw={T_make/1000} s3rlo={R_open} s3rhi={R_open}
+ s4name="X" s4tsw={T_make/1000} s4rlo={R_open} s4rhi={R_open}
+ s5name="Z" s5tsw={T_make/1000} s5rlo={R_open} s5rhi={R_open}
+ )
.model digio_1 uio
.ends
```

In some systems the previous model is too simple, since it does not include contact “bounce.” (Contact bounce is caused by the physical bouncing of the electrical contacts as they close. It looks electrically as if the relay contacts close and open several times in quick succession before they remain closed.) The last model includes contact bounce for a specified period of time after the contacts close. The contact bounce is created by taking a digital contact close signal and converting it to an analog ramp using a DtoA conversion device. The analog ramp forms the input to a table-controlled voltage source. The table creates a “bounce” output voltage which is then converted to digital to square it up. The digital value is used to control another DtoA conversion device which models the contacts.

```
* Behavioral model of a relay with contact bounce.

.subckt RELAY_SPDT_BHV_BOUNCE coila coilb no nc com
+ PARAMS:
+ T_make = 20mSec         ; Time for contact to close when current
                          ; is turned off/on
+ T_break= 10mSec         ; Time for contact to open when current
                          ;is turned off/on
+ T_bounce=5mSec          ; bounce time (after T_make)
+ I_pull = 35ma           ; Pull-in current
+ I_drop = 25ma           ; Drop-out current
+ R_coil = 100            ; Coil resistance
+ L_coil = 5mH            ; Coil inductance
+ R_open = 100MEG         ; open circuit contact resistance
+ R_close= .05            ;closed circuit contact resistance

* electrical model of coil
v_winding coila a1 0
r_winding a1 a2 {R_coil}
l_winding a2 coilb {L_coil}

* make a voltage from the current
e_cc cc 0 value = {limit(I(v_winding),-3*I_pull,3*I_pull)}
r_cc cc 0 1k

* use digital to create a switch with hysteresis
o_mag cc 0 relay_1 DGTLNET=d digio_1
.model relay_1 doutput (
+ s0name="0" s0vlo={-I_pull} s0vhi={I_pull}
+ s1name="1" s1vlo={I_drop} s1vhi={4*I_pull}
+ s2name="1" s2vlo={-4*I_pull} s2vhi={-I_drop}
+ timestep={T_make/1000}
+ )
```

The model of relay with contact bounce is continued.

The model of relay with contact bounce (continued).

```
u_dly buf dpwr dgnd d cn_no relay_2 digio_1
u_inv inv dpwr dgnd d cn_nc relay_2 digio_1
.model relay_2 ugate (
+ tphlmn={T_break} tphlmx={T_break}
+ tplhmn={T_make-0.126*T_bounce}
+ tplhmx={T_make-0.126*T_bounce}
+ )

* model contact bounce with subckt
x_no cn_no com no RELAY_CONTACT_BOUNCE
+ PARAMS:
+ T_make = {T_make}
+ T_break= {T_break}
+ T_bounce= {T_bounce}
+ R_open = {R_open}
+ R_close= {R_close}
x_nc cn_nc com nc RELAY_CONTACT_BOUNCE
+ PARAMS:
+ T_make = {T_make}
+ T_break= {T_break}
+ T_bounce= {T_bounce}
+ R_open = {R_open}
+ R_close= {R_close}
.model digio_1 uio
.ends


.subckt RELAY_CONTACT_BOUNCE ctrl contact1 contact2
+ PARAMS:
+ T_make = 20mSec     ; Time for contact to close when current
                      ; is turned off/on
+ T_break=10mSec      ; Time for contact to open when current
                      ; is turned off/on
+ T_bounce= 5mSec     ; bounce time
+ R_open = 100MEG     ; open circuit contact resistance
+ R_close= .05        ; closed circuit contact resistance
```

The relay contact bounce subckt is continued.

The relay contact bounce subckt (continued).

```
* Simulate a fixed pattern of bounces. Convert the digital
* ctrl (1=closed) into an analog signal which has a rise time
* of T_bounce. Use a table to convert this ramp into a "bounce"
* pattern. Then use an AtoD to square it off.
* Another DtoA makes the contact itself.
n_1 c_a 0 1 relay_4 DGTLNET=ctrl digio_1
.model relay_4 dinput (
+ s0name="0" s0tsw={T_make/1000} s0rlo=1 s0rhi=20
+ s1name="1" s1tsw={T_bounce/.665} s1rlo=20 s1rhi=1
+ s2name="R" s2tsw={T_make/1000} s2rlo=1 s2rhi=20
+ s3name="F" s3tsw={T_make/1000} s3rlo=1 s3rhi=20
+ s4name="X" s4tsw={T_make/1000} s4rlo=1 s4rhi=20
+ s5name="Z" s5tsw={T_make/1000} s5rlo=1 s5rhi=20
+ )
r_1 c_a 0 1MEG
V1 1 0 1
e_bounce c_b 0 TABLE {v(c_a)}=
+ (.05,0 .1,.55 .2,0 .3,.6 .4,0 .5,.7 .6,0 .7,.8 .8,0 .9,1)
o_no c_b 0 relay_5 DGTLNET=c_1 digio_1
.model relay_5 doutput (
+ s0name="0" s0vlo=-1 s0vhi=.5
+ s1name="1" s1vlo=.5 s1vhi=2
+ timestep={t_bounce/1000} )

n_contact contact1 contact2 contact2 relay_3 DGTLNET=c_1 digio_1
.model relay_3 dinput (
+ s0name="0" s0tsw={T_make/1000}
+ s0rlo={R_open*2} s0rhi={R_open*2}
+ s1name="1" s1tsw={T_make/1000} s1rlo={R_close*2}
+ s1rhi={R_close*2}
+ s2name="R" s2tsw={T_make/1000} s2rlo={R_open*2}
+ s2rhi={R_open*2}
+ s3name="F" s3tsw={T_make/1000} s3rlo={R_open*2}
+ s3rhi={R_open*2}
+ s4name="X" s4tsw={T_make/1000} s4rlo={R_open*2}
+ s4rhi={R_open*2}
+ s5name="Z" s5tsw={T_make/1000} s5rlo={R_open*2}
+ s5rhi={R_open*2}
+ )
.model digio_1 uio
.ends
```

Here is an example simulation using each of the three relay models

:

```
* Test Relay models
X1 a 0 no_b nc_b com RELAY_SPDT_BHV
+ PARAMS:
+ T_make = 20mSec
+ T_break= 1.5mSec
+ I_pull = 35ma
+ I_drop = 25ma
+ R_coil = 100
+ L_coil = 5mH
+ R_open = 100MEG
+ R_close= .05
X2 a 0 no_bb nc_bb com RELAY_SPDT_BHV_BOUNCE
+ PARAMS:
+ T_make = 20mSec
+ T_break= 1.5mSec
+ T_bounce=10mSec
+ I_pull = 35ma
+ I_drop = 25ma
+ R_coil = 100
+ L_coil = 5mH
+ R_open = 100MEG
+ R_close= .05
X3 a 0 no_phy nc_phy com RELAY_SPDT_PHY_MSRD
+ PARAMS:
+ T_drop = 20mSec
+ I_pull = 35ma
+ I_drop = 25ma
+ R_coil = 100
+ L_coil = 5mH
* Drive all the relays with a 5v pulse
vdrive a 0 pwl (0,0 .100,0 .101,5V .700,5V .701,0v)
* connect a load resistor to each contact
v1 1 0 1
r1 1 no_b 1K
r2 1 nc_b 1K
r3 1 no_bb 1K
r4 1 nc_bb 1K
r5 1 no_phy 1K
r6 1 nc_phy 1K
r7 com 0 .001
.tran .1 1
.probe
.options acct
* use the new relay library
.lib relay.lib
.end
```

Figure 59 shows the results of running this simulation. Note that it takes a few minutes to simulate this circuit, but only few

seconds if you do not include the physical relay model in the simulation.

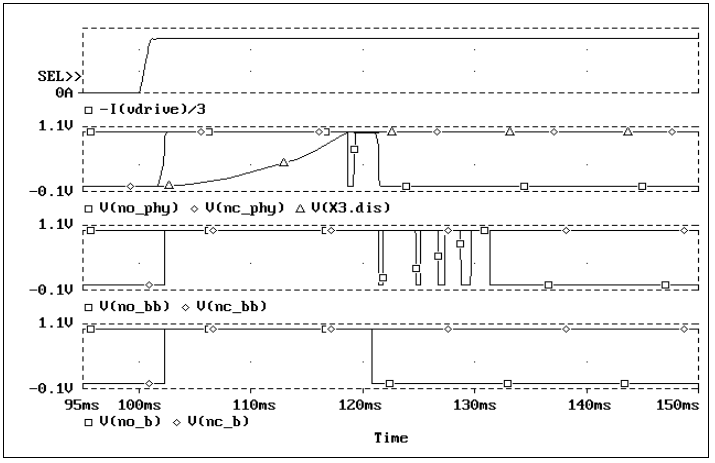


Figure 59 *Results of the relay behavioral model*

We can see that the new behavioral models for relays give the user better control over the operation of the relay model, and give good simulation results with much less simulation time. These models have been included in the release libraries since version 5.3, along with schematic symbols for them.

Note *These models are included in the model library "misc.lib."*

Minimize DAC Switch Area with MicroSim's Analog Optimizer

A standard technique for building integrated circuit digital-to-analog converters (DACs) is to use an R-2R ladder configuration with current-steering switches. A high-accuracy DAC can be constructed even if the switches have nonzero ON resistance, provided that the switch resistances obey a binary weighting law. Paragon, MicroSim's analog performance optimizer, can be used to reduce the area occupied by the switches by trading off accuracy against area.

This example focuses on a 12-bit DAC design. Using Paragon to optimize the design gives a better than 40% reduction in switch area while meeting nonlinearity requirements of $\leq$0.2 least significant bit (LSB).

Background

Figure 60 on page -125 shows the basic configuration of an R-2R current-steering DAC. R-2R ladders are frequently used in IC applications because only the ratio of component values is important, and only two resistor values are required. The currents flowing in the *rungs* of the ladder reduce by a factor of 2 per stage. Each stage has a switch which directs the current either to ground or to a virtual earth at the input of an opamp configured as an
I-to-V converter. Operation of the n^{th} switch causes a change in total current of 2^n times the bit_0 current, resulting in the desired binary weighting of the bit contributions.

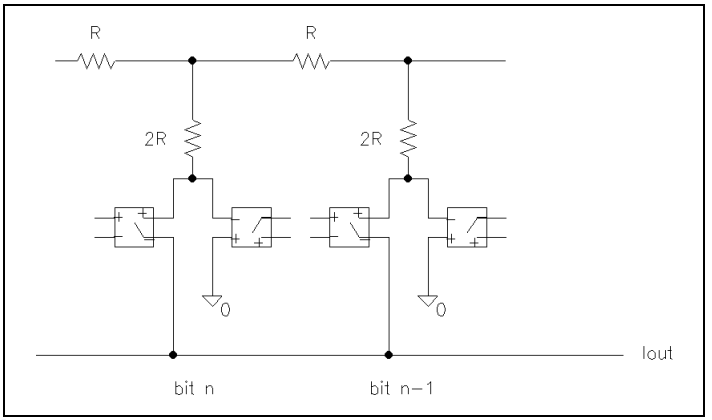


Figure 60 *R-2R current-steering DAC.*

The above analysis assumes switches with zero ON resistance. Using real switches with finite ON resistance, the binary weighting of bit currents can be preserved provided that the ON resistance of the switches decreases by a factor of 2 per stage of the ladder. This can be demonstrated by noting that the required relationship holds true if the lower end of the rung is not zero, but is at some constant voltage instead. For this condition to hold, the ON resistances of the switches must halve as the currents double. (Another way to arrive at this result is to write down the conditions at the end of the ladder and work back up the ladder.)

D/A Converter Linearity

DAC performance is typically specified by stating the maximum permissible differential nonlinearity (DNL) and integral nonlinearity (INL). Two adjacent digital codes should produce output values which are one LSB apart (2^{-n} of full scale). DNL is defined as the maximum deviation of the actual step from the ideal for all pairs of adjacent input codes. INL is defined as the maximum deviation of the output values from a straight line from 0 to full scale.

DNL can be measured by taking two identical DACs, driving one with a digital ramp from 0 to 2^n-2 and the other with a ramp

from 1 to 2^n-1. Subtracting the outputs and normalizing provides DNL.

INL can be measured by driving a DAC with a digital ramp from 0 to 2^n-1. Subtracting the output from a corresponding analog ramp between 0 and full scale and normalizing provides INL.

Figure 61 shows the test schematic used to measure INL and DNL. Four identical DACs are used: two to produce the adjacent values for DNL measurement, one to convert a ramp for INL measurement, and one fed with all 1's to produce full scale output for normalization.

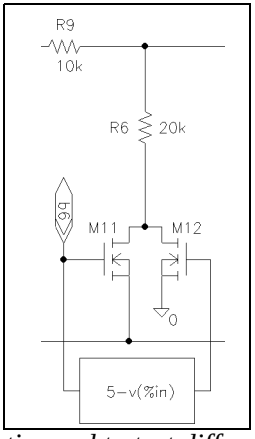


Figure 61 *Schematic used to test differential and integral nonlinearity.*

Ideal ADCs are used to generate the *digital* ramps input to the DACs. The ADCs are implemented using Analog Behavioral Modeling (ABM) expressions. A single DC source provides a ramp from 0 to 2^n-1 volts. This is then offset, scaled, and limited to drive the ADCs and produce the required digital patterns. This approach allows the entire simulation and optimization to be done using DC sweeps only. An alternative approach is to use digital stimuli and PSpice's A/D converter (UADC) primitives. This would require a transient analysis rather than a DC sweep. This particular example shows 12-bit DACs. This means that the input ramp is from 0 to 4095.

Additional ABM devices perform the arithmetic required to form outputs corresponding to DNL and INL.

DAC and ADC Implementation

The 12-bit DAC design is a slight variation on the R-2R idea described previously. The first four stages have individual weighting resistors. The remaining eight stages use an R-2R ladder. The current-steering switches are implemented using pairs of N-channel MOSFETs. The ON resistance of the MOSFET used as a switch is inversely proportional to the width of the device, so the required binary scaling of ON resistances is achieved by an equivalent scaling of the device widths.

Figure 62 shows a section of the R-2R ladder and its associated switch. An ABM device is used as an inverter between MOSFET gates so that when one device is on, the other is off. With a +5 volt reference voltage, and a 2.5 kohm feedback resistor on the output opamp, the full scale output is about -10 volts.

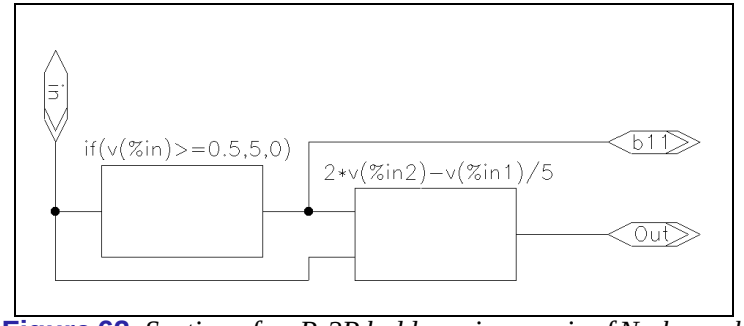


Figure 62 *Section of an R-2R ladder using a pair of N-channel MOSFETs as the current-steering switches*

An ideal ADC is constructed by cascading 12 identical 1-bit cells. Each cell contains a comparator which outputs a logic ‘1’ if the input is equal to or above the reference voltage, a logic ‘0’ otherwise (logic levels in this example are +5 volts and 0 volts). The cell then forms a residue by multiplying the input by 2 and subtracting the reference voltage if the cell’s comparison output is a ‘1’. Figure 63 shows one of the 1-bit ADC cells. It comprises two ABM devices.

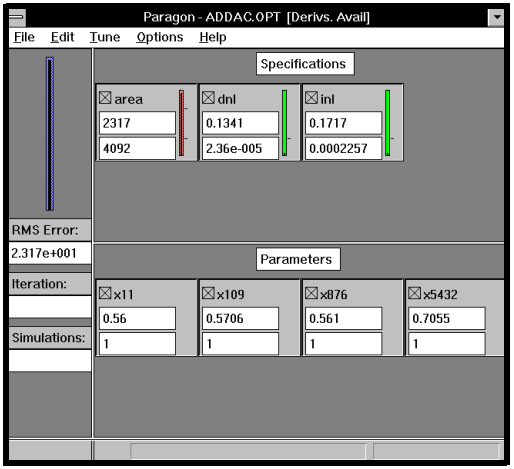


Figure 63 *One bit ADC cell.*

Setting Up

A number of points need to be addressed before proceeding with details of the optimization. First, all analysis is done with a DC sweep. This causes the Vramp device to be swept between 0 and 4095 volts. A simulation using all 4096 data points is too long for convenience, so most of the work is done using a step of 25 (i.e., the ramp is 0, 25, 50, ...). The results are checked periodically by performing measurement using a step of 1.

Secondly, the parameters which will be varied by the optimizer are the ratios of widths of the transistors in the DAC switches. It is convenient to define a set of parameters each with nominal value of 1.0 and then set the width of the n^{th} cell to be $2^n \times parameter_n$. The optimizer supports up to eight independent parameters. There are, however, eleven independent widths in this example.

The following scheme is used to set the widths from a smaller number of parameters ('x' parameters):

x11 controls stage 11 (MSB)
x109 controls stages 10, 9
x876 controls stages 8, 7, 6
x54321 controls stages 5, 4, 3, 2, 1

The widths of the transistors are scaled using the M parameter of the MOSFET model. The M parameter of the transistors in a stage is set to $\{M_n\}$ for the n^{th} stage (e.g., {M8} for the 8^{th} stage). The relationship between each M parameter and the associated 'x' parameter is defined using an ABM expression. For example, M8 is set by x876 as follows:

.PARAM M8 {x876 * 256} ; 256 is 2^8

Optimization

The purpose (goal) of the optimization in this example is to minimize the area of silicon occupied by the switches. There are requirements on the amount of nonlinearity which can be introduced by altering the ratios of switch sizes. For this example, an upper bound of 0.2 LSB is acceptable for both INL and DNL. These are constraints for the optimization.

To optimize the design, Paragon requires definitions of the goal and constraints, together with the parameters which are to be varied. The area can be computed directly by Paragon as a function of the parameter values (a *Paragon expression*). No simulations or goal function evaluations are required for this. INL and DNL for each step of the input ramp are output by the test schematic directly. A Probe intrinsic goal function is then used to find the absolute maximum values. For example, for INL:

MAX(ABS(V(INL))

The target values and ranges for the goal and the constraints are set to 0,100 (for the goal) and 0.2,0.01 for the constraints. Each of the *x parameters* is set up with an initial value of 1.0 and a permissible range of between 0.1 and 10.0.

The type of optimization required is to minimize a single, positive value (the total area). To set Paragon up for this, select Options/Advanced and select the Minimization button. Selecting Tune/Udpate Values performs a simulation and displays initial values for the specifications. Selecting Tune/Auto/Start runs the optimization. The optimized results show a reduction in total area to 2317, with DNL of 0.13 LSB and INL of 0.17 LSB. The achieved reduction in area is greater than 40%.

The optimized parameter values are 0.56, 0.5706, 0.561, and 0.7055. The optimization takes about 66 minutes on a SPARC station 2 with a Weitek Power µP chip.

Running a simulation with these values, but with the step value in the DC sweep set to 1, produces INL and DNL figures which are somewhat degraded but which still meet the requirements.

Summary

In an integrated circuit DAC design, silicon area can be traded off against output nonlinearities. Performing the trade-off manually is difficult because the relationships are highly nonlinear and are difficult to establish analytically.

In this particular example, Paragon was used to provide a 40% reduction in switch area while meeting requirements of 0.2 LSB on both differential and integral nonlinearity. The approach described in this article can be easily adapted to other weighting schemes, voltage-mode DACs, and more.

We would like to thank John Horan of Regional Technical College in Cork, Ireland for providing the original idea presented in this article.

Model Ferrite Beads in SPICE

By Michael A. Wyatt, *Electronic Design*, October 15, 1992

Ferrite beads used for power supply decoupling represent a difficult modeling problem for those involved in SPICE circuit simulations. The bead's frequency-dependent nonlinear and non-monotonic behavior are particularly difficult to model. Eventually, the best recourse is to use simple L-R type equivalent circuit models. But they're only accurate over a narrow frequency range.

Now with the advent of behavioral modeling packages; such as PSpice from MicroSim Corp., accurate modeling and simulation of ferrite beads over a wide range is possible. To model the bead, a voltage-controlled current source (VCCS) behavioral model is used in such a way that the source's terminals appear as an impedance. The source's current is controlled by its terminal voltage as follows:

$$I_{source} = K(V_{source})$$

$$Z = V_{source}/I_{source}$$

Then:

$$Z = 1/K$$

Thus, the terminal impedance (Z) is the reciprocal of the source's scale factor (K). The listing illustrates this concept for a subcircuit model of the Fair-Rite 2673000101 ferrite bead. GBEAD is the VCCS and is controlled by V(1,2), the source's terminals. FREQ is the behavioral model keyword for a frequency-table-controlled source in PSpice. Data is entered into the table as frequency, magnitude of 1/Z in decibels, and the angle of 1/Z in degrees. Note that the frequency-dependent nonlinear nature of the bead's inductive component would be very difficult to model with discrete circuit elements. If ferrite-bead data is unavailable in an impedance magnitude and angle format, then the impedance magnitude and angle should be computed as:

$$1/Z = \frac{1}{\sqrt{Real^2 + Img^2}}$$

```
.SUBCKT         BEAD73          1       2
GBEAD           1               2       FREQ    {V(1,2)} =
*               Freq            1/Z db  1/Z ang
+               (1k,            40.5,   -89.9)
+               (10k,           20.6,   -89.5)
+               (100k,          0.9,    -80.5)
+               (1meg,          -20.4,  -59.5)
+               (2meg,          -23.4,  -46.0)
+               (3meg,          -24.7,  -39.1)
+               (5meg,          -25.4,  -36.9)
+               (7meg,          -26.2,  -35.9)
+               (10meg,         -27.7,  -34.6)
+               (20meg,         -30.0,  -23.0)
+               (30meg,         -30.3,  -16.5)
+               (40meg,         -30.0,  -12.0)
+               (50meg,         -29.7,  -08.0)
+               (60meg,         -29.6,  -06.5)
+               (70meg,         -29.6,  -06.3)
+               (80meg,         -29.6,  -06.2)
+               (100meg,        -29.6,  -06.6)
+               (200meg,        -29.7,  -09.5)
+               (1000meg,       -32.0,  -26.0)
RBEAD           1               2       200
.ENDS           BEAD73
```

$$\angle 1/Z = -\text{Arctan}\frac{\text{Img}}{\text{Real}}$$

The listing entries are Frequency, 20Log(1/Z), and Angle(1/Z).

Simulation results of the bead's effectiveness on power supply coupling between two local Vcc points fed from a main supply source verified the model's effectiveness.

In another test, the supply network was subjected to a 10 ns, 100 mA current pulse at one Vcc point. When the simulation was performed, a time-domain plot showed reduced ringing with the ferrite beads in the circuit.

Some convergence and time-domain simulation problems were experienced with different bead models. Resistor RBEAD in the listing was included to reduce the Q of the bead and supply a real shunt impedance around the VCCS. RBEAD is determined empirically with the VCCS frequency table coefficients to provide a good representation of the bead's frequency-dependent behavior.

```
Voltage Controlled Capacitor Test Circuit
* Table controlled variable capacitor.
* A power curve models the device's capacitance
* -voltage curve.
* The controlling voltage is restricted to a defined range.

* From the 'D' device capacitance equations
* Cj = CJO * (1 + Vr/Vj)**-M, where
*      CJO = zero-bias junction capacitance
*      CJO = p-n potential
*      M = p-n grading coefficient

*      Cj is junction capacitance for reverse voltage Vr.
* Specify capacitance @4v reverse voltage
.subckt talbecap 1 2 PARAMS: C4 = 1pf, M = 5.0, VJ = 1.0
 Ecopy 3 6 1 2 1.0
 Vsense0 6 0v
 Cref  3 0 {C4 * pwr(vj+4,M)}     ; computes CJO from C4
 Hsense 10 0 Vsense 1.0           ; converts I(Cref) to V(10)
 Rdummy 10 0 1Meg
 Gout 1 2 VALUE =                 ; capacitance/voltage modeling
+ {v(10)/pwr(TABLE(v(1,2), 1,1, 60,60)+VJ,M)}
.ends

* 1v/s slew rate stimulus:
Vdc 2 0 pwl(0,1v 59s,60v)

* Parameters chosen for a 1N5144
X1 2 0 table cap PARAMS:C4 = 22p

.tran/op .1s 59 0 .1s
.probe

.end
```

Model Transient Voltage Suppressor Diodes

by Steve Hageman Applied DC

Transient Voltage Suppressor (TVS) diodes present some unique modeling challenges. These devices are unlike standard Zener diodes because of their high power clamping capability. During a power transient, the junction temperature may change up to 350°C or more for a short period of time. For an accurate result, this extreme change in junction temperature must be modeled during the simulation. The major junction temperature considerations are (1) effect on clamping voltage, and (2) ability of the device to survive the transient.

Modeling Goals

In this article, models are developed for the popular 500 watt [1] and 1500 watt [2] unidirectional devices. The 500 watt model is reviewed in detail. The 1500 watt model is electrically the same except for some parameter changes to accurately reflect the larger device.

The basic goal when modeling the TVS is to represent the mean behavior of the device with an overall error of less than 15%. Modeling better than 15% accuracy is not worthwhile. It would serve to complicate the model, thus slowing down the simulation. Also, it would tend to give a false sense of security because the real device parameters vary by better than 20% from device to device. Thus, it is sufficient to partition the TVS diode model into three parts (see Figure 64 and Figure 65):

1. forward diode characteristics
2. reverse diode or breakdown characteristics
3. device thermal model

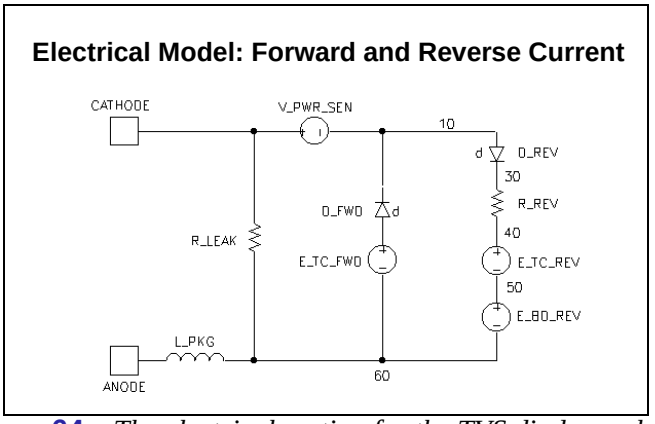


Figure 64 *The electrical section for the TVS diode can be completely modeled with these few parts. Two basic sections make up the electrical model: one section for the forward current direction, and another for the breakdown or reverse current direction.*

The TVS model described in this article accurately represents the following behaviors:

- forward V/I characteristics
- reverse V/I characteristics
- junction temperature and effects during power pulses
- junction capacitance

Most of the TVS parameters are dependent on device breakdown voltage, so heavy use is made of the ‘parameterization’ feature provided in PSpice. This capability allows a model to be developed that needs only one parameter—the device breakdown voltage. All other variable device parameters are related to the breakdown voltage by equations which are evaluated at run time.

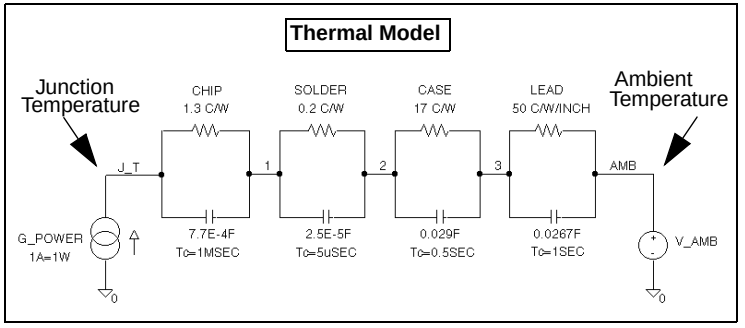


Figure 65 *Heat flow is a three dimensional problem. To get reasonable accuracy, the thermal model for the TVS device is broken into four component parts, starting at the chip and ending with the leads. The values shown here are for a 500 watt SA type of device.*

Forward Characteristics

A TVS that is biased in the forward direction behaves exactly like a normal diode. Parts, the PSpice model parameter extraction program, was used to obtain the proper parameters for the D_FWD diode (see Figure 64 and the .MODEL statement for D_FWD_SA in Figure 65).* The forward diode characteristics do not change appreciably with TVS breakdown voltage, so one model is used for all devices.

*. To extract the model parameters related to forward diode characteristics, forward voltages and currents from the manufacturer's device data sheet are used as input to Parts. Parts adjusts the model parameters to provide a good fit to the device curve.

```
*****************************************************************
* SA SERIES TRANSIENT VOLTAGE SUPPRESSOR MODEL
* Model valid for 6.8 to 100 volt breakdown voltages
*****************************************************************
* ANODE    ANODE input node
* CATHODE  CATHODE input node
* J_T      Junction temperature output node
* BDV      Breakdown voltage in volts
* AMB_T    Starting ambient temperature
*****************************************************************
.SUBCKT SA ANODE CATHODE J_T PARAMS: BDV=1, AMB_T=27
* * * * ELECTRICAL MODEL * * * *
V_PWR_SEN CATHODE 10  DC 0.0 ;Current sense
RLEAK  CATHODE ANODE 100MEG  ;Leakage resistor
* FORWARD SECTION
D_FWD  20  10  D_FWD_SA      ;Forward diode
E_TC_FWD 20,60 VALUE = {-2.2E-3 * (V(J_T) - AMB_T)} ;Forward diode TC
* REVERSE (BREAKDOWN) SECTION
D_REV  10  30  D_REV_SA      ;Reverse (Breakdown) diode
R_REV  30  40  {0.00032*PWR(BDV,1.93)} ;Resistance
* Reverse TC
E_TC_REV 40,50 VALUE = {(0.00016*PWR(BDV,1.48)) * (V(J_T) - AMB_T)}
V_BD_REV 50  60  {BDV - 0.3} ;Reverse voltage
* PACKAGE
L_PKG  ANODE 60  5N          ;Package inductance
* * * * THERMAL MODEL * * * *
* Power to current converter, 1A = 1W dissipated in device
G_POWER  0,J_T VALUE = {ABS( I(V_PWR_SEN) * V(ANODE,CATHODE))}
V_AMB_T  AMB  0  {AMB_T}     ;Ambient temp node
* DISTRIBUTED THERMAL MODEL
R_CHIP J_T 1 1.3             ;Diode chip
C_CHIP J_T 1 7.7E-4
R_SOLD 1  2 0.2              ;Solder joints
C_SOLD 1  2 2.5E-5
R_CASE 2  3 17               ;Package
C_CASE 2  3 0.029
 R_LEAD 3  AMB 37.5          ;Leads
 C_LEAD 3  AMB 0.0267
* * * * DIODE MODELS * * * *

* Forward diode model - includes forward capacitance
.MODEL D_FWD_SA D(T_ABS=27, IS=2.15N, N=1.63, RS=14M
+ IKF=0.782, XTI=3, EG=1.11
+ CJO={1.7E-8*PWR(BDV,-0.99)};Junction capacitance equation
+ M=0.333, VJ=0.75, FC=0.5
+ ISR=100P, NR=2, BV=1000, IBV=100U, TT=100N)

* Reverse diode model
.MODEL D_REV_SA D(T_ABS=27, IS=10N)

.ENDS ;----- END OF SA SUBCIRCUIT MODEL
```

Figure 66 *SA series transient voltage suppressor diode model.*

The forward diode is also a good place to model the TVS device capacitance. The relatively large junction capacitance behaves like a regular diode and is included in D_FWD. The capacitance

is a function of breakdown voltage (*BV*) and is easily related by the exponential equation

$$C = 1.7^{-8} \times (BV)^{-0.99} \quad (1)$$

where *C* is in farads at zero volts bias. This equation is added to the D_FWD_SA model statement (referenced by D_FWD) using PSpice parameters; thus, the proper junction capacitance is calculated for any TVS device at the start of a PSpice analysis.

To account for the diode's voltage temperature dependence, the D_FWD_SA model includes the `T_ABS=27` model parameter clause; that is, the D_FWD diode is defined to have an absolute temperature of 27°C regardless of the circuit-wide temperature set during simulation by the TNOM option (.OPTIONS), a temperature analysis (.TEMP), or a parametric analysis (.STEP). The diode's temperature coefficient is added back with E_TC_FWD. This voltage-controlled voltage source adds -2.2 mV/°C of the calculated junction temperature.

Reverse Characteristics

The D_REV diode, R_REV, E_TC_REV, and V_BD_REV make up the reverse or breakdown characteristics of the TVS. D_REV functions to block forward current from this section and works in conjunction with V_BD_REV; these two parts set the basic breakdown voltage.

At higher currents, R_REV and E_TC_REV become predominant factors in the breakdown voltage. R_REV is the high current resistance of the device, modeled as a function of breakdown voltage expressed as

$$R = 3.2^{-4} \times (BV)^{1.93} \quad (2)$$

where *R* is in ohms. E_TC_REV models the breakdown voltage temperature dependence. The magnitude of E_TC_REV used during simulation is a function of the breakdown voltage expressed as

$$TC = 1.6^{-4} \times (BV)^{1.48} \quad (3)$$

where TC is in volts/°C. The value of E_TC_REV is also dependent on the junction temperature during simulation.

Thermal Model

It is a fairly simple task to model the thermal characteristics of a device which is in equilibrium. However, when the junction temperature changes within a one-second time span, modeling becomes nontrivial. Dynamic heat flow is a three-dimensional problem that is not directly solvable with SPICE. Instead, the heat flow paths of the device must be broken apart and modeled individually as linear RC circuits. Even this approach only works as long as the rise time in temperature is longer (slower) than the thermal transient time across the material [3] [4].

To model TVS devices adequately, a four part thermal model is used as shown in Figure 65. The junction temperature node, J_T, is fed from the power-to-current converter, G_POWER. G_POWER is an analog behavioral device that calculates the instantaneous power that the TVS is dissipating by multiplying the absolute value of the voltage across the device by its current (current is sensed by V_PWR_SEN). The units are scaled so that 1 ampere equals 1 watt dissipation. The various thermal resistances of the TVS are modeled along with their thermal capacitance values. Each RC pair makes up that section's thermal time constant.

Heat flows from the junction to the ambient via the RC paths. The ambient temperature is represented by a voltage source called V_AMB.

For slow heating times (> 10 seconds), the thermal time constants don't matter; only the resistances matter. When fast power pulses are absorbed by the TVS, the thermal time constants have a significant effect on the instantaneous junction temperature.

The thermal model is coupled to the electrical model via the voltage at node J_T. The voltage here represents the junction temperature with a 1 volt = 1°C scale factor. This voltage is used by the forward and reverse voltage temperature coefficient generators during simulation.

Testing the TVS Model

To test the TVS model for an SA 5.0 device, a standard 10 X 1000 waveform (see Figure 68 on page -141) is generated using the circuit file shown in Figure 67. This waveform is a double exponential that is described nicely by the PSpice EXP (EXPonential) stimulus as:

```
I_DRV +node -node EXP(0, {IPP}, + 0, 250n, 10u, 1.4m)
```

```
SA 5.0 TEST CIRCUIT

* * INCLUDE THE TVS MODEL * *
.INC SA.CIR

* * SETUP THE 10 X 1000 PULSE ANALYSIS * *
.PARAM IPP=56          ;SET THE IPP LIMIT FOR AN SA 5.0
.TRAN 1U 100M 0 100U   ;RUN ANALYSIS FOR 100 mSEC
.PROBE                 ;SAVE RESULTS TO A PROBE FILE

* * 10 X 1000 PULSE GENERATOR * *

I_PW 0 10 EXP(0, {IPP}, 0, 250E-9, 10E-6, 1.4E-3)

* * SET UP THE SA SUBCIRCUIT * *

XSA 0 10 J_TEMP SA PARAMS:BDV=6.8

.END
```

Figure 67 *Circuit file to test the SA 5.0 device with a 10 X 1000 pulse.*

The value, {IPP}, is a parameter that is passed to I_DRV; this sets the peak current to be used during simulation. The maximum value of IPP is given on the manufacturer's device data sheet. The breakdown voltage is the only parameter that needs to be passed to the TVS subcircuit. This voltage is the mean voltage listed on the data sheet at the onset of breakdown (low current region). For an SA 5.0 device, the breakdown voltage is listed as 6.4 to 7.3 volts. So the mean is about 6.8 volts.

The resulting junction temperature and voltage clamping waveforms are shown in Figure 68. The results of the simulation compare very favorably with the device's actual performance.

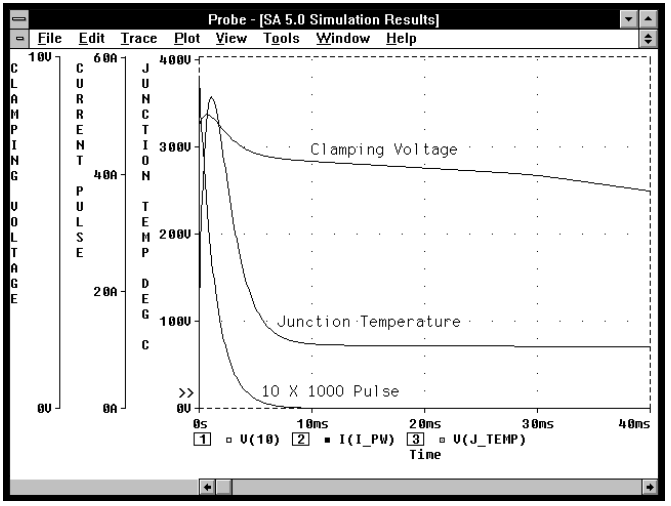


Figure 68 *Simulation results when pulsing an SA 5.0 device with a 56 amp 10 X 1000 pulse. The junction temperature climbs very fast to around 350°C. The clamping voltage compares favorably to an actual device.*

Other Common Transient Waveforms

TVS devices are used to clamp many types of transient waveforms. A very common type of transient that happens billions of times every day is Electro Static Discharge (ESD). Figure 69 shows a typical “Human Body” discharge and its model. The capacitor can have an initial voltage from 0 to upwards of 20 kV. Of note, the threshold for human detection of the discharge is on the order of 2,500 volts. This is the reason for wearing wrist straps when working in electronics manufacturing. Though the static discharge cannot always be felt, it can still exist.

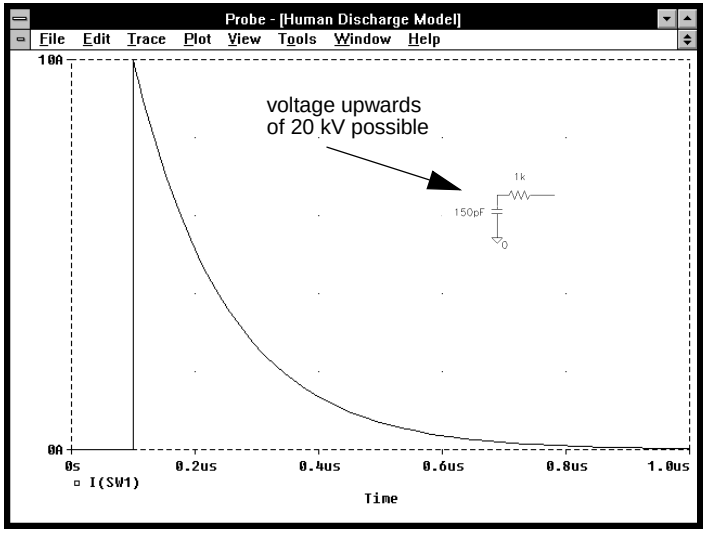


Figure 69 *Electro Static Discharge (ESD) events happen billions of times a day The human body model of an ESD source may have an initial charge of upwards of 20 kV. This may produce up to 20 amps in a short, low inductance discharge path. The ESD current rise time is on the order of 1 nsec.*

Another common ESD transient occurs when a charged piece of furniture is discharged to an object. Figure 70 shows a typical discharge profile and the equivalent circuit. Again, the initial voltage on the capacitor can range from 0 to 20 kV. The furniture discharge has series inductance associated with it so there is a backswing and negative current as well as a positive current. The frequency of the resonant circuit is typically on the order of 20 MHz.

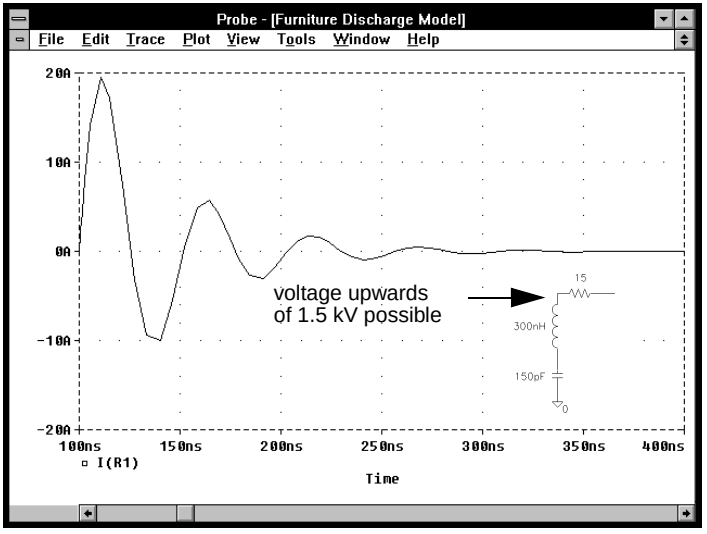


Figure 70 *Another common ESD discharge is the 'furniture' model. This discharge happens when a piece of furniture is charged. The furniture model typically has an inductive ring that will produce both a positive and a negative current swing.*

When nature causes an ESD event, the result is lightning discharge. Figure 71 shows the waveform for the induced circuit current due to a lightning discharge that is 0.1 km from the receptor circuit. The current may approach 25 amps with an initial open circuit voltage of around 1,100 volts on unprotected data lines according to IEC standard 801-5. A direct hit may be on the order of kA; the TVS devices modeled here are not capable of surviving such a hit. This is the realm of gas discharge tubes and the like [5]. When modeling lightning discharges, special care must be taken to include all stray inductance and capacitance effects in the circuit. These strays are quite large in long cable runs and can significantly affect the actual current waveform input to the receptor circuit [6].

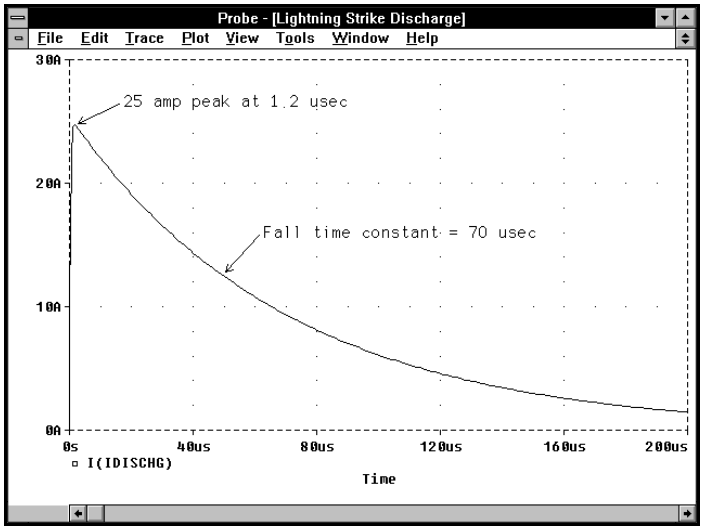


Figure 71 *Having a data line 0.1 km away from a lightning strike may cause up to 25 amps and 1,100 volts to be impressed on an unprotected data line according to IEC standard 805-1. A direct strike may produce currents in the kA range. Direct strikes are best handled by gas discharge tubes and lightning countermeasures.*

Conclusion

The TVS model presented here accurately predicts the most important device parameters. The resulting models require only one device parameter to be passed to them; equations relate all other parameters to the breakdown voltage. The common 500 watt, SA series devices were presented in the Figure 66 circuit file and described in the text. Figure 72 on page -145 shows the circuit file for the 1.5KE and 1N62xx series, 1500 watt TVS devices. The 1.5KE model has the same circuit layout as the SA series devices; only the circuit parameters have changed as needed to accurately predict device operation.

```
*********************************************************
* 1.5KE SERIES TRANSIENT VOLTAGE SUPPRESSOR MODEL
* Model valid for 6.8 to 100 volt breakdown voltages
*********************************************************
* ANODE              ANODE input node
* CATHODE            CATHODE input node
* J_T                Junction temperature output node
* BDV                Breakdown voltage in volts
* AMB_T              Starting ambient temperature
*********************************************************

.SUBCKT KE ANODE CATHODE J_T PARAMS: BDV=1, AMB_T=27

* * * * ELECTRICAL MODEL * * * *
V_PWR_SEN CATHODE 10  DC 0.0  ;Current sense
RLEAK  CATHODE ANODE 100MEG   ;Leakage resistor

* FORWARD SECTION
D_FWD  20  10  D_FWD_SA       ;Forward diode
E_TC_FWD 20,60 VALUE ={-2.2E-3*(V(J_T)-AMB_T)};Forward diode TC

* REVERSE SECTION
D_REV  10  30  D_REV_SA       ;Reverse (Breakdown) diode
R_REV  30  40  {0.00014*PWR(BDV,1.7)} ;Resistance

* Reverse TC
E_TC_REV 40,50 VALUE ={(0.00021*PWR(BDV,1.36))*(V(J_T) - AMB_T)}
V_BD_REV 50  60  {BDV - 0.3}  ;Reverse voltage

* PACKAGE
L_PKG  ANODE 60  7N           ;Package inductance

* * * * THERMAL MODEL * * * *

* Power to current converter, 1A=1W dissipated
G_POWER  0,J_T VALUE = {ABS( I(V_PWR_SEN) * V(ANODE,CATHODE))}
V_AMB_T  AMB  0  {AMB_T}      ;Ambient temp node

* DISTRIBUTED THERMAL MODEL
R_CHIP J_T 1 0.6              ;Diode chip
C_CHIP J_T 1 3.3E-3

R_SOLD 1  2 0.1               ;Solder joints
C_SOLD 1  2 1E-4

R_CASE 2  3 10                ;Package
C_CASE 2  3 0.36

R_LEAD 3  AMB 15.75           ;Leads
C_LEAD 3  AMB 0.127

* * * * DIODE MODELS * * * *

* Forward diode model - includes forward capacitance
.MODEL D_FWD_SA D(T_ABS=27, IS=59.4P , N=1.27, RS=6M, IKF=0.659
+ XTI=3, EG=1.11
+ CJO={7E-8/BDV}              ;Junction capacitance equation
+ M=0.333, VJ=0.75, FC=0.5
+ ISR=100P, NR=2, BV=1000, IBV=100U, TT=100N)

* Reverse diode model
.MODEL D_REV_SA D(T_ABS=27, IS=10N)

.ENDS ;----- END OF KE SUBCIRCUIT MODEL
```

Figure 72 *KE series transient voltage suppressor diode model.*

References

500 watt devices including General Semiconductor Industries™ and Motorola™ SA5 to SA100 devices.

1500 watt devices including General Semiconductor Industries™ and Motorola™ 1N62XX and 1.5KE6.8 to 1.5KE100 devices.

P. Antognetti, Editor, *Power Integrated Circuits*, McGraw Hill Company, New York, NY. 1986, ISBN 0-07-002129-5.

S. Clemente, "Transient Thermal Response of Power Semiconductors to Short Power Pulses," *IEEE Transactions on Power Electronics*, October, 1993.

O.M. Clark, "A Guide For Transient Suppression Using The Transzorb™ TVS," General Semiconductor Industries, Inc. application note, Tempe, AZ, 1986.

O.M. Clark and T.M. Dalsing, "Optimizing Placement Of Board Level Transient voltage Suppressors," *EMC-ESD International Conference Proceedings*, Denver, CO, April, 1990.

Biography: Steve Hageman is an analog designer specializing in power conversion. He owns the consulting firm, Applied DC, and may be reached by voice/FAX at (510) 687-0483.

Modeling Constant Power Loads

The Design Center Source newsletter, January 1992

In power systems, it is common to encounter loads which draw constant power. To model such a load, we can use a voltage-dependent current source. A first approximation looks like this:

```
gload n1 n2 value = {pload/v(n1,n2)}
```

With this formula, the power = v*i = v(n1,n2)*(pload/v(n1,n2)) = pload, as desired.

Unfortunately, this first approximation behaves badly near v = 0. When calculating the bias point for more difficult circuits, PSpice reduces the power supplies. PSpice relies on the assumption that, when the supplies are close enough to 0, all devices in the circuit are turned off. The above formula violates this assumption. Further, it is not a good model of a real constant-power load for low voltages.

A real load can only consume constant power over a limited range of applied voltage. When the voltage drops below this range, the load's impedance stops falling. For many loads, a good model is a series connection of two resistances: the fixed "minimum" resistance and the dynamic "constant-power" resistance. We can write

$$R_{total} = R_{min} + R_{var} = R_{min} + v^2/P$$
$$i = v/R_{total} = v/(R_{min} + v^2/P) = 1/(R_{min}/v + v/P)$$

For low v, $i = v/R_{min}$. For high v, i = P/v. The corresponding PSpice statement is

```
gload n1 n2 value = {1/(RMIN/v(n1,n2) + v(n1,n2)/PLOAD}
```

This device behaves like a resistor of value RMIN at low applied voltages and like a constant-power load at high voltages. The crossover occurs at

$$R_{min}/v = v/P \text{->} v^2 = R_{min}P \text{->} v^2/R_{min} = P$$

when the power dissipated in R_{min} equals the desired power, P. This is the point of maximum power dissipation into R_{min}. For higher voltages the current falls and most of the power is dissipated by R_{var}.

Modeling Lossy Transmission Lines

The Design Center Source newsletter, January 1992

PSpice adds a lossy line parameter set to the T device to support modeling lossy transmission lines. You can specify a lossy line in terms of its electrical length and the resistance, inductance, capacitance, and conductance distributed along the length. This allows you to model loss and dispersion in a non-ideal line.

The internal model used in connection with these parameters is a “distributed” model. That is, the line response is computed using impulse responses instead of the commonly used ladder structures associated with lumped models. Though the lumped ladder model can be simulated in any SPICE simulator, the distributed approach has several advantages, especially accuracy.

The lumped model assumes that the line can be represented by a series of short segments which include a series resistor and inductor along with a shunt resistor and capacitor. If you string a large enough number of these segments together, you will have a discretized model of the distributed nature of the line (see Figure 73).

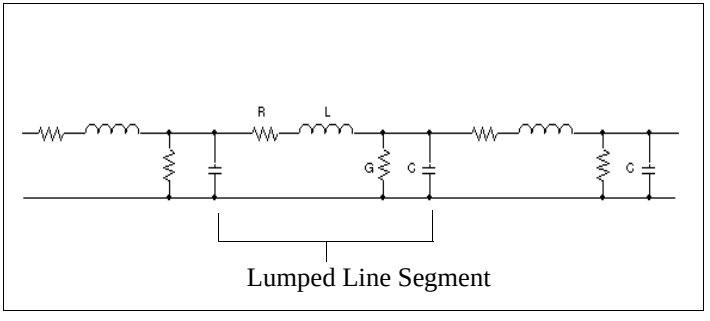


Figure 73 *Lumped segment model*

A problem with this approach is that the number of segments required for accurate results can become quite large, causing it to become the dominant factor in the simulation execution time. This is further compounded by the fact that the short

transmission line segments have short time constants, and this makes for short simulation time steps and long simulation times.

However, the most significant problem is that the reduction of the smoothly continuous line to a set of finite lumps results in frequency artifacts showing up around the natural frequencies of the approximating segments. These are seen as oscillations at points where abrupt changes occur in the signal traveling along the line.

The following circuit file demonstrates both the internal distributed model and the lumped approach just described. It also shows how these models differ in behavior from the ideal T device.

```
* Transmission line model
.Options ITL5=0 acct
.Tran 1n 20n 0 .1n
.Probe
.Lib
* Circuit description
Vin 1 0 PULSE (0 5 0 .1n .1n 5n 10n)
Rsrc 101 1 100
Rload 100 0 {2*sqrt(1n/5p)}
.Inc "Tline.inc"
.End
```

The standard PSpice device libraries, incorporate a set of lumped model subcircuits (of which TLUMP64, used below, is one). The include file “Tline.inc” contains a single line of netlist which is one of the following:

For the distributed model case,

```
Tdistrb 101 0 100 0 len=24 r=2 l=1n g=0 c=5p
```

For the lumped model case,

```
Xlumped 101 100 0 TLUMP64 params: len=24 r=2 l=1n
+ g=0 c=5p
```

For the ideal model case,

```
Tideal 101 0 100 0 z0={sqrt(1n/5p)td={24*sqrt(1n*5p)}
```

The circuit simulates each of the three transmission line models with a 100 ohm source resistor (Rsrc), a load resistance (Rload) equal to twice the characteristic impedance, and a 200 MHz input pulse train (Vin). The transmission line output is at node

100. V(100) is plotted using Probe for each of the three models. The results are shown in Figure 74.

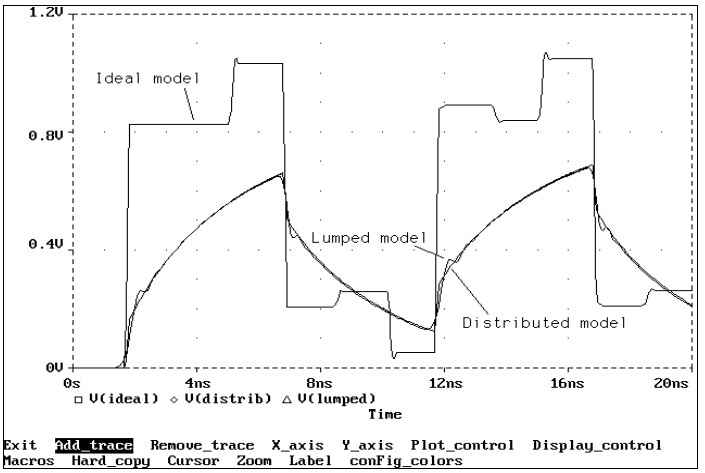


Figure 74 *Ideal, lumped, and distributed transmission line models*

It is clear that the ideal model is inadequate for the given transmission line characteristics. The lossy line exhibits significant attenuation as well as distortion. The lumped and distributed models are, however, in good agreement. The lumped model shows some spurious wobbling at the beginnings and endings of the pulses due to the natural frequency phenomenon discussed earlier.

Both of the lossy models, not surprisingly, use much more CPU time than the ideal model. Both models introduce overhead during circuit read-in, and both slow down the transient analysis. Most of the read-in overhead for the distributed model comes from computation of impulse responses, which is done up front in order to avoid redundant calculation during the transient run. The lumped model takes longer to read in because it results in a fairly large circuit after the lumps are expanded. Comparative CPU times in seconds, using a 486/50 PC, are given in Table .

Note *Figure 74 was obtained using MicroSim software package version 5.1. CPU times are less for subsequent versions.*

The lumped model's CPU time goes up very quickly when you improve the resolution of the ladder. Doubling the number of segments from 64 to 128 doubles the transient analysis time and quadruples the read-in time.

A special case worth noting is the distortionless line, which is characterized by having R/L equal to G/C. There is no dispersion when this is true, and PSpice's internal model doesn't need to perform any impulse response calculations. The CPU time will consequently be about the same as for the ideal case.

Table 2 *Comparative CPU Times*

	Read-in Time	Transient Analysis	Total Time
Ideal Model	.16	.83	.99
Lumped Model (64 lumps)	29.06	31.59	60.86
Lumped Model (128 lumps)	111.22	63.77	175.27
Distributed Model	24.83	10.76	35.59

Modeling Potentiometers and Variable Resistors

MicroSim Corporation Newsletter, October 1990

We have received several inquiries on how to model a potentiometer. These requests are closely related to other applications of variable resistors, such as strain gauges. We will review these as well.

Electrically, a potentiometer consists of two resistors connected in series. The specification for the potentiometer consists of:

- the total resistance (R), and
- the pot's "setting" (SET). That is, where the center tap is set. A convenient way to describe this is to define SET to be 0 when the tap is all the way at the "bottom" and 1 when it is all the way at the "top."

The potentiometer can then be implemented by this subcircuit.

```
.SUBCKT POT (TOP, BOTTOM, TAP) PARAMS: R=1K SET=.5
 RTOP TOP TAP    {(1.001-SET)*R}
 RBOT TAP BOTTOM {( .001+SET)*R}
.ENDS
```

The values 1.001 (instead of 1) and .001 (instead of 0) are used to prevent the resistors from having 0 ohms at the extremes.

So far in our discussion, the setting of the pot has been static. That is, it does not change with time. This is appropriate for almost all applications, since the time required for the movement of the pot is much longer than the electrical time constants of the circuit. In other words, there is no loss of information by running several transient analyses and varying the pot's setting with a .STEP command.

A typical usage would be:

```
.PARAM SET=.5
.STEP  PARAM(SET) 0, 1, .2
X1  3  5  17  POT PARAMS: R=10K SET={SET}
```

Here a 10k pot is used in 6 runs, having the settings 0, .2, .4, .6, .8, and 1.

In schematics there is a symbol for a potentiometer located in “breakout.slb”. The example circuit below shows how the pot may be used. The example shows an adjustable regulator. The pot R1 is swept to show the adjustment range of the regulator.

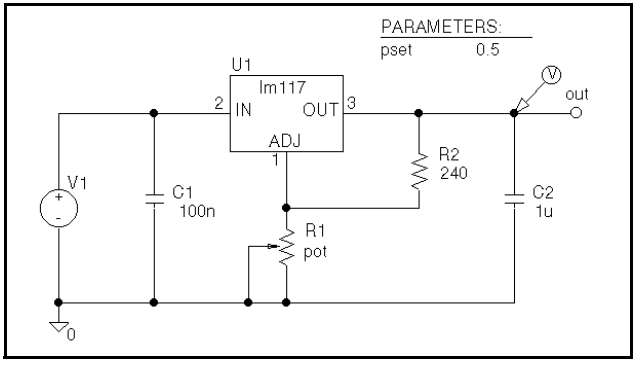


Figure 75 *Variable potentiometer test circuit*

For this analysis a DC sweep of the parameter pset is used.

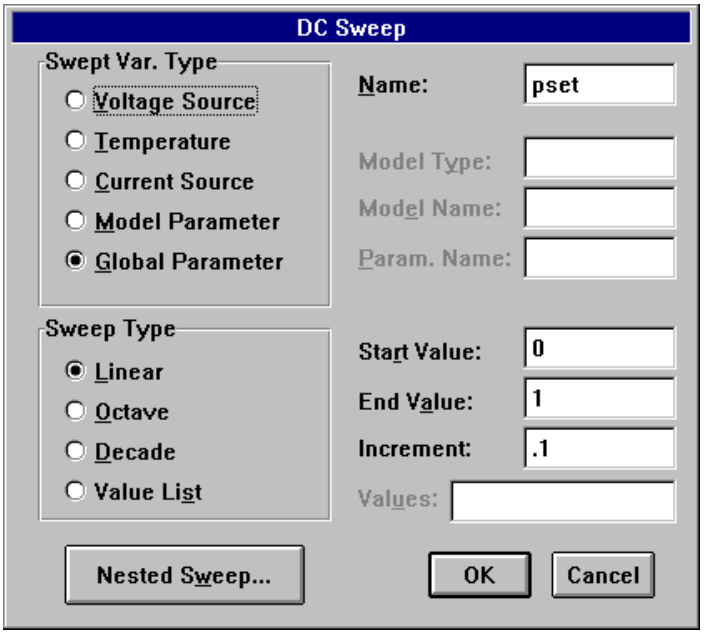


Figure 76 *DC Sweep dialog box*

Pset is swept from 0 to 1 in increments of 0.1.

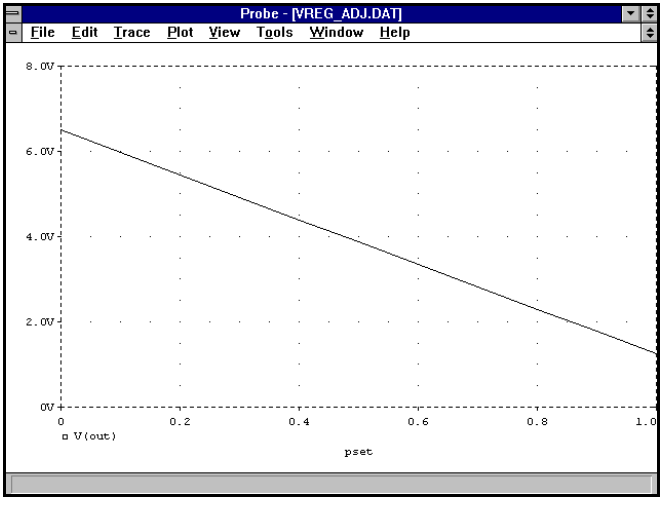


Figure 77 *Probe screen capture of V(out)*

So far, we have assumed that the pot is linear; for a logarithmic pot we need an extra parameter: the dynamic range of the pot.

As SET goes from 0 to 1, the value of RBOT goes

```
.SUBCKT POT (TOP, BOTTOM, TAP) PARAMS: R=1K RANGE=1000 SET=.5
 RTOP  TOP TAP    {R-(R/RANGE)*PWR(RANGE,SET)}
 RBOT  TAP BOTTOM {  (R/RANGE)*PWR(RANGE,SET)}
.ENDS
```

logarithmically from R÷RANGE to R. RTOP makes up the difference between RBOT and the total resistance of the pot.

If a time-varying potentiometer is needed, it can be built using the same ideas as above, but using the ZX subcircuit from the library file “misc.lib.” This subcircuit implements a voltage-controlled resistance. One then builds an appropriate controlling waveform (using, for instance, the PWL type) to vary the resistance as desired.

Variable resistors can also be used to implement many kinds of sensors. A strain gauge, for example, consists of a resistance bridge:

```
.SUBCKT GAUGE (IN+ IN- OUT+ OUT-) PARAMS: R=1K F=0 SENS=1e-3
 RUL IN+ OUT+ {R*(1+F*SENS/2)}                ; upper left
 RUR IN+ OUT- {R*(1-F*SENS/2)}                ; upper right
 RLL IN- OUT+ {R*(1-F*SENS/2)}                ; lower left
 RLR IN- OUT- {R*(1+F*SENS/2)}                ; lower right
.ENDS
```

R can be determined by the gauge’s current drain at the bias voltage (R = V÷I). SENS sets the sensitivity of the gauge and F is the applied force. Suppose, for example, that the strain gauge is part of a pressure sensor and that full-scale output is 50 mV with a bias voltage of 10 V and that full scale corresponds to a pressure of 500 psi. Within the normal operating range the bridge output is 10V×(F×SENS), in this case 50 mV at F = 500. So, SENS = 50mV÷(10V×500) = 10e-6.

Modeling Quartz Crystals

The Design Center Source newsletter, October 1987

We have received many questions about modeling quartz crystals. We recommend the following model for a quartz crystal:

```
LCRYSTAL      1       2       12henry
CSERIES       2       3       20Pfarad
RCRYSTAL      3       4       30Kohm
CPARALLEL     1       4       20Pfarad
```

This model includes both the series and parallel resonances. Note the very large value of the inductor. This yields the proper Q for the crystal, about 30000. The values in this case were chosen to give a frequency of 10 kHz.

The above model of a quartz crystal can be found in many places. For instance,

D. A. Fink, Editor, *Electronics Engineers' Handbook*, McGraw-Hill

contains a description in section 7-108, pages 7-61 through 7-65 and in section 13-38, pages 13-31 through 13-34.

Note that both series and parallel resonances are available. The parallel resonance can be "pulled" by an external capacitor, allowing the frequency to be adjusted slightly. Many crystals' resonant frequency have a quadratic temperature dependence. This can be correctly modeled by introducing a quadratic temperature coefficient for the inductor, LCRYSTAL.

Modeling Schottky Diodes

The Design Center Source newsletter, October 1987

Another question that is asked regularly is: how does one set the forward voltage drop of a diode, especially a Schottky diode? Sometimes people try to use the model parameter VJ to set the forward voltage drop. However, VJ only affects the depletion capacitance, not the DC current.

The correct parameter to change is IS. Referring to the forward DC current equation of the diode as listed in the *MicroSim PSpice Reference Manual*.

$$I = IS \cdot (e^{V/N \cdot Vt} - 1)$$

or, solved for V,

$$V = N \cdot Vt \cdot log(I/IS + 1)$$

The forward voltage drop, V, depends on the current, I, but only weakly. V increases by 60 millivolts for each factor of 10 that I increases. Alternatively, V increases by 60 millivolts for each factor of 10 that IS decreases. So, the forward voltage drop for your circuit's bias conditions can be set by changing IS in the diode's .MODEL statement. For Schottky diodes, values of IS are larger than for diffusion diodes of the same area.

The model parameter N can also be used to adjust the forward voltage drop, but changing N will make the I-V curve deviate from the normal slope of a decade per 60 millivolts. We do not recommend changing N in order to model Schottky diodes.

These comments also apply to the base-emitter junction of the bipolar transistor. Adjusting IS will change the b-e forward voltage drop. This is useful when modeling Germanium transistors, for example. One temptation to resist, however, is to model a Darlington transistor pair as one device with a very high beta. The forward b-e voltage drop for the pair would need to be about $2 \cdot 0.7 = 1.4$ volts. At a nominal forward current of 1 milliampere, this leads to IS = 1mA/2E23 = 5E-27. This value is so small that on the VAX, which has a limited exponent range, it can cause numerical problems. Also, for IS as low as 5E-27, the I-V curve does not have the correct shape at the low end: the

knee is much too sharp. The real device's forward voltage drop is the sum of two voltage drops with more rounded knees. All of this is to say that a Darlington should be modeled by connecting two transistors, as in the real device.

References

[1] D. A. Hodges and H. G. Jackson, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill.

Modeling Voltage-Controlled Resistors and Capacitors

MicroSim Corporation Newsletter, January 1991

The MISC.LIB library file contains subcircuit models for voltage-controlled reactances and admittances. These can be used to make voltage-controlled resistors and capacitors. The following two examples illustrate the use of a voltage-controlled resistor to control the Q of a series RLC filter network, and the use of a voltage-controlled capacitor to change the frequency of oscillation of a Wien bridge oscillator.

Note *This modeling technique is not applicable to capacitances whose values change slowly. It applies to cases where the capacitance changes very quickly between constant values. The application note "A Nonlinear Capacitor Model for Use in PSpice" illustrates a better model for capacitors whose value depends on their own terminal voltages.*

Variable Q RLC Network

In most circuits the value of a resistor is fixed during a simulation. While the value can be made to change for a set of simulations by using a .STEP statement to move through a fixed sequence of values, a voltage-controlled resistor can be made to change dynamically during a simulation. This is illustrated by the circuit shown in Figure 78, that employs a voltage-controlled resistor, X_VCRes. This special resistor is defined using the ZX subcircuit given in MISC.LIB. This subcircuit consists of two controlled sources and is described in detail in Appendix D of *SPICE, A Guide to Circuit Simulation and Analysis Using PSpice* by Paul Tuinenga, and will not be repeated here. This subcircuit employs an external reference component that is sensed. The output impedance equals the

value of the control voltage times the reference. Here, we will use Rref, a 50 ohm resistor as our reference. As a result, the output impedance is seen by the circuit as a floating resistor equal to the value of Vcontrol times the resistance value of Rref. In our circuit, the control voltage value is stepped from 0.5 volt to 2 volts in 0.5 volt steps, therefore, the resistance between nodes 3 and 0 varies from 25 ohms to 100 ohms in 25 ohm-steps.

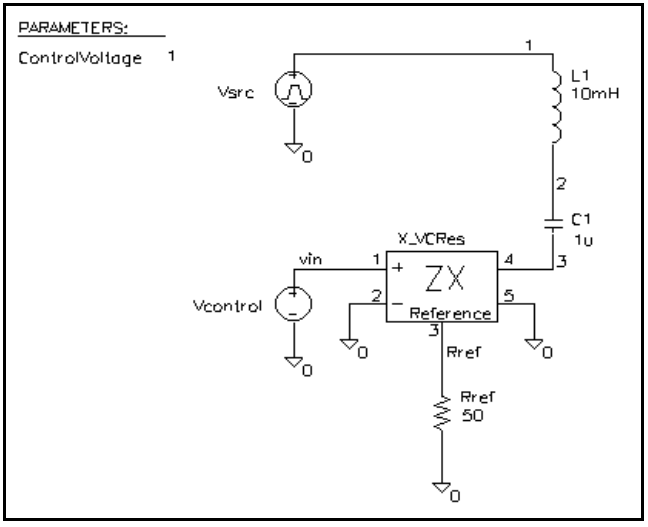


Figure 78 *Variable Q RLC circuit*

The first and second connections to the ZX subcircuit are the control input, followed by a connection to the reference component and then, finally, the two connections for the floating impedance.

```
L1 1 2 10mH
C2 2 3 1uF
*          Control   Reference         Floating Z  Subcircuit
X_VCRes vin 0 Rref   3 0                ZX
Rref Rref 0 50ohm
*
.STEP PARAM ControlVoltage 0.5, 2, 0.5
Vcontrol vin 0 DC {ControlVoltage}
```

A transient analysis of this circuit using a 0.5 ms wide pulse will show how the ringing differs as the Q is varied by X_VCRes.

```
Vsrc 1 0 PULSE( 0V, 1V, .5ms, 1us, 1us, .5ms, 4ms )
.TRAN .1m 4m 0 0.01m
.PROBE
.LIB
```

Using Probe, we can observe how the ringing varies as the resistance of X_VCRes changes. Figure 79 shows the input pulse and the voltage across the capacitor C2. Comparing the four output waveforms, we can see the most pronounced ringing occurs when X_VCRes has the lowest value and the Q is greatest. Any signal source can be used to drive our voltage-controlled impedance. If we had used a sinusoidal control source instead of a staircase the resistance would have varied dynamically during the simulation.

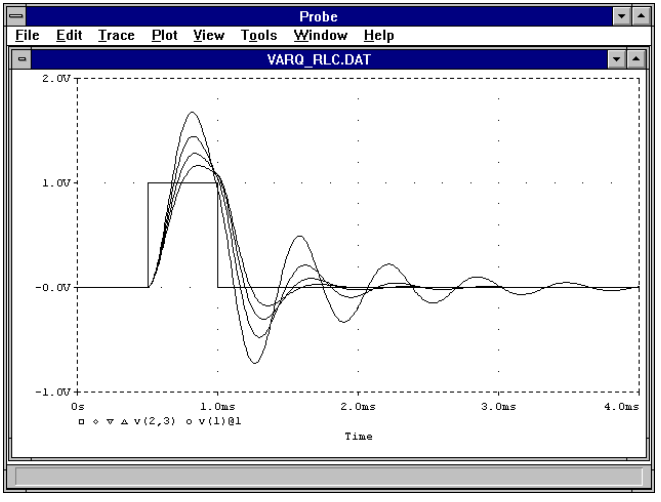


Figure 79 *Output waveforms of variable Q RLC circuit*

Voltage-Controlled Wien Bridge Oscillator

As a second example, we will use a voltage-controlled capacitor to adjust the frequency of oscillation of a Wien bridge oscillator.

First, a simplified operational amplifier is created using a voltage-controlled voltage source (an E device). Node 1 is the plus input, node 2 is the minus input and node 4 is the output of the opamp.

```
Eamp 4 0 Value {V(1,2) * 1E6}
```

A voltage divider network provides negative feedback to the amplifier. The closed-loop gain of the opamp must be at least 3 for oscillations to occur since the Wien bridge attenuates the output by 1/3 at the frequency of oscillation. The back-to-back Zener diodes limit the gain of the opamp as the oscillations build so that saturation does not occur.

```
Rg1 4 5 10k
Rg2 2 0 9.5k
Rg3 2 5 10k
D1 4 6 D1N750
D2 5 6 D1N750
```

As shown in Figure 80, the Wien bridge consists of two resistors and two voltage-controlled capacitors.

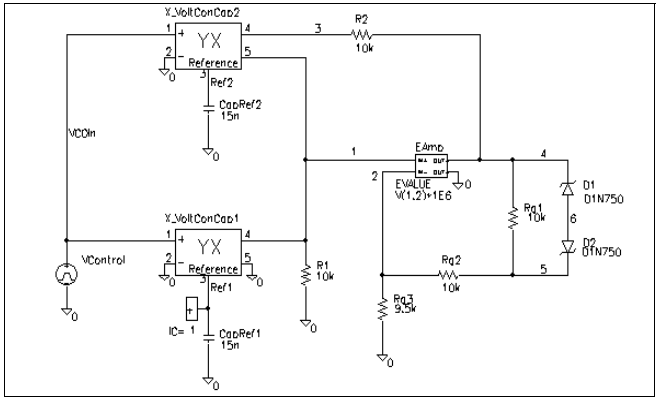


Figure 80 *Frequency controllable Wien Bridge oscillator*

Each of these capacitors uses the YX subcircuit from the file MISC.LIB and its own reference capacitor. In this example 15nF capacitors are used.

```
R1 1 0 10k
R2 3 4 10k
X_VoltConCap1 VcoIn 0 Ref1 1 0 YX
CapRef1 Ref1 0 15nF
X_VoltConCap2 VcoIn 0 Ref2 3 1 YX
CapRef2 Ref2 0 15nF
```

The voltage control of the oscillations is given by Vcontrol which is a pulse that moves from 1.0 volts to 1.2 volts, 25 ms into the run. This changes the admittance from that of a 15 nF capacitor to a 18 nF capacitor. The frequency of oscillation will then change. The .IC statement causes PSpice to begin simulation with an initial condition of 1 volt on node Ref1 so oscillation can begin.

```
Vcontrol VcoIn 0
+ PULSE( 1.0V 1.2V, 25ms, 1uS, 1uS, 50ms, 50ms )
.TRAN 500u 50m 0 50u
.IC V([Ref1])=1v
.LIB
.PROBE
```

Figure 81 shows the Fourier transform of voltage V(4), the output of the oscillator. Using this capability, we can easily see the transition from the first frequency to the second. The resonant frequency is given as $1/(2\pi*R*C*VcoIn)$. The first frequency is 1/(6.28 * 10k * 15n * 1.0V) = 1kHz. The second frequency is 1/(6.28 * 10k * 15n * 1.2V) = 0.886kHz. We can see the two peaks on the plot indicating the two resonant

frequencies. It can also be noted that the period of the oscillations is proportional to the control voltage VcoIn.

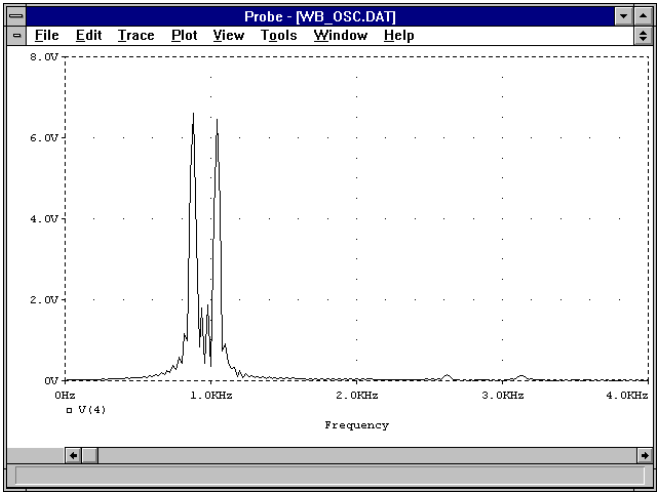


Figure 81 *Frequency controllable Wien Bridge oscillator output*

Modeling Voltage-Variable Capacitors

The Design Center Source newsletter, April 1992

Some of our customers have asked how to model a voltage-variable capacitor. The example circuit file shown on the next page describes a test circuit that contains a voltage-variable capacitor. This capacitor is constructed by way of a TABLE function embedded in the VALUE extension to the G (voltage-controlled current source) device. This model is a better representation of a varicap device than the commonly used YX device

The Probe plot in Figure 82, shows capacitance versus controlling voltage for a voltage-variable capacitor similar to 1N4155.

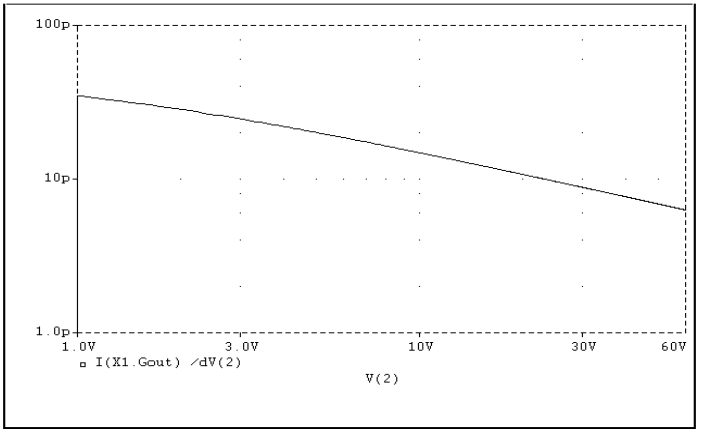


Figure 82 *Capacitance versus controlling voltage*

```
Voltage Controlled Capacitor Test Circuit
* Table controlled variable capacitor.
* A power curve models the device's capacitance
* -voltage curve.
* The controlling voltage is restricted to a defined range.
* From the 'D' device capacitance equations
* CJ = CJO * (1 + Vr/Vj)**-M, where
*        CJO = zero-bias junction capacitance
*        CJO = p-n potential
*        M = p-n grading coefficient
*        Cj is junction capacitance for reverse voltage Vr.
* Specify capacitance @4v reverse voltage
.subckt tablecap 1 2 PARAMS: C4 = 1pf, M = 0.5, VJ = 1.0
 Ecopy 3 6 1 2 1.0
 Vsense 0 6 0v
 Cref 3 0 {C4 * pwr(vj+4, M)} ; computes CJO from C4
 Hsense 10 0 Vsense 1.0 ; converts I(Cref) to V(10)
 Gout 1 2 VALUE = ; capacitance/voltage modeling
+ {v(10)/pwr(TABLE(v(1,2), 1, 1, 60, 60)+VJ, M)}
.ends
* 1v/s slew rate stimulus:
Vdc 2 0 pwl(0,1v 59s,60v)
* Parameters chosen for a 1N5144
X1 2 0 tablecap PARAMS: C4 = 22p
.tran/op .1s 59 0 .1s
.probe
.end
```

The subcircuit, tablecap, may be placed in a model library file and a symbol for the part may be created.

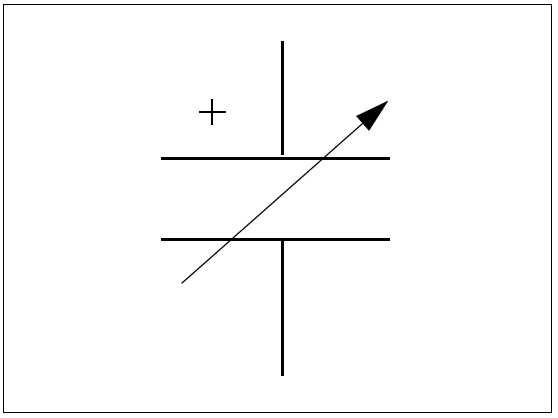


Figure 83 *Volt_cap symbol*

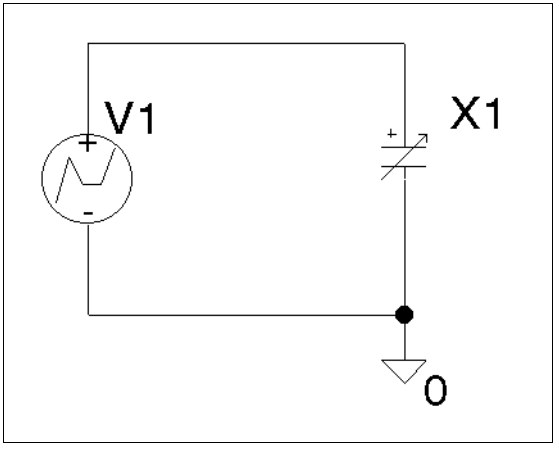


Figure 84 *Voltage-variable test circuit*

A Nonlinear Capacitor Model for Use in PSpice

Theory

The charge and current fomulas for a linear capacitor are:

$$Q = C * V \qquad (1a)$$

$$I = C * dV(t)/dt \qquad (1b)$$

For a nonlinear (voltage-dependent) time-independent capacitor these formulae become:

$$Q = \int C(V) * dV \qquad (2a)$$

$$I = C(V) * dV/dt \qquad (2b)$$

This applies to cases where the capacitance has been measured at different bias voltages.

Some would argue that for a nonlinear capacitor,

$$Q = C(V) * V \qquad (3a)$$

where V is a function of time. Therefore,

$$I = dQ/dt = C(V(t)) * dV(t)/dt + dC(V(t))/dt * V(t) \qquad (3b)$$

This is not correct. The flaw in this argument is equation (3a). Although (1a) holds true for linear capacitors, the generalized definition of charge is (2a). Capacitance is the partial derivative of Q with respect to V; which means

$$I = dQ/dt = \partial Q/\partial V * dV/dt = C(V) * dV/dt \qquad (4)$$

Given this relationship between the current through a nonlinear capacitance and the voltage applied to it, analog behavioral modeling can be used to model any nonlinear capacitor whose capacitance, C(V), is a function of the voltage applied to it.

The Model

The nonlinear capacitor is modeled by a subcircuit in which the capacitor is replaced by a controlled current source, Gout, whose current is defined by (2b). In the subcircuit, the time derivative, dV(t)/dt, is measured by applying a copy of the voltage across Gout to a known capacitance, Cref, and monitoring its current. The C(V) function in the subcircuit is arbitrary.

Example Model

The value of the nonlinear capacitor model in this example has a second order polynomial dependence on its voltage. This is equivalent to the standard PSpice capacitor model, whose linear and quadratic coefficients, VC1 and VC2, can be defined in a .MODEL statement.

```
* Polynomial Nonlinear Capacitor Model
.subckt polycap 1 2 params: C0=1u C1=0 C2=0
Ecopy  3 6 1 2 1.0       ; copy V(t)
Vsense 0 6 0V            ; Ammeter
Cref   3 0 1.0E-6        ; to get 1E-6*dv/dt
                         ; *1E-6 to avoid ridiculous currents
Gout   1 2 VALUE =
+ {(C0 + C1*V(1,2) + C2*V(1,2)*V(1,2)) * I(Vsense)*1E6 }
* ---------------------------------- -------------
*               C(V)                      dV(t)/d
.ends
*$
```

Figure 85 *Circuit file for polynomial nonlinear capacitor model*

The subcircuit model, shown in Figure 85, is placed in a model library file. This model is parameterized so that users can specify the polynomial coefficients in the circuit file, or on the symbol in the Schematic Editor (Figure 86). In the case of circuit file entry, the model library is referenced in the circuit file by a .LIB statement, and the capacitor is instantiated by an *X* device. In the Schematic Editor, the name of the model library file is added to the list of those configured in the Analysis/ Library & Includes dialog box, and a new symbol represents the

nonlinear capacitor, with the coefficients, C0, C1, and C2, as attributes.

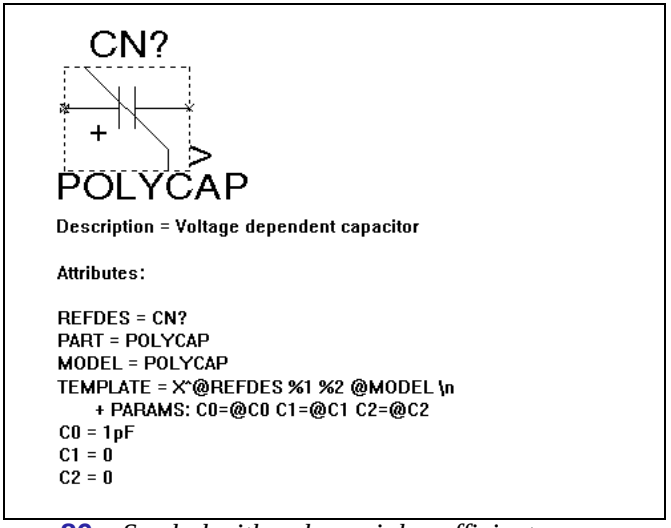


Figure 86 *Symbol with polynomial coefficients*

Example Circuit File

The example circuit in Figure 87, the schematic for which is shown in Figure 88, performs two types of analyses on the polycap model.

```
Voltage-Controlled Capacitor Test Circuit

.lib my_misc.lib
Vin    1 0 dc 2 ac 1 pwl(0,1V 59s,60V)
X1     1 0 polycap PARAMS:C0=1u C1=0.01u C2=0.0001u

.tran/op .1s 59 0 .1s;              C = -I(Vin)/d(V(1))
*                                    = 1.0101u to 1.96u

.ac lin 10 10k 100k       ;         C = -Ii(Vin)/(2*pi*Frequency)
*                                    = 1.0204u

.probe
.end
```

Figure 87 *Circuit file for voltage-controlled capacitor test circuit*

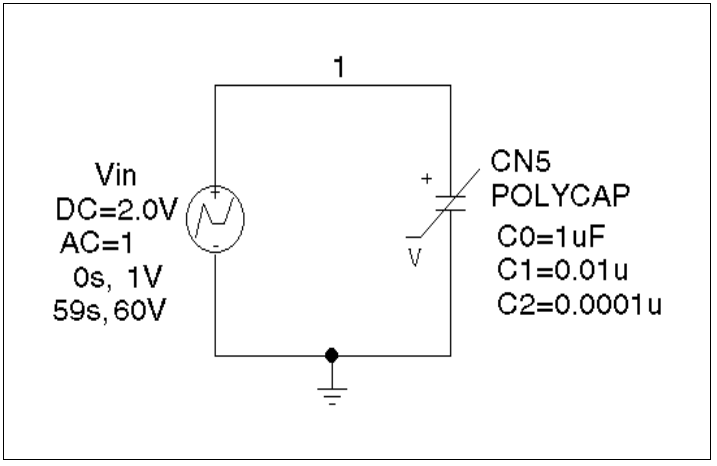


Figure 88 *Schematic for voltage-controlled capacitor test circuit*

- The transient analysis slowly varies the voltage across the capacitor from 1V to 60V. The effective capacitance can be viewed in Probe as the expression: -I(Vin)/d(V(1)), which is derived from equation (2b). The minus (-) sign is required because PSpice measures voltage source currents as flowing from the positive node to the negative. Probe will show that the capacitance varies from 1.0101u(F) to 1.96u(F). The transient analysis results are shown in Figure 89.

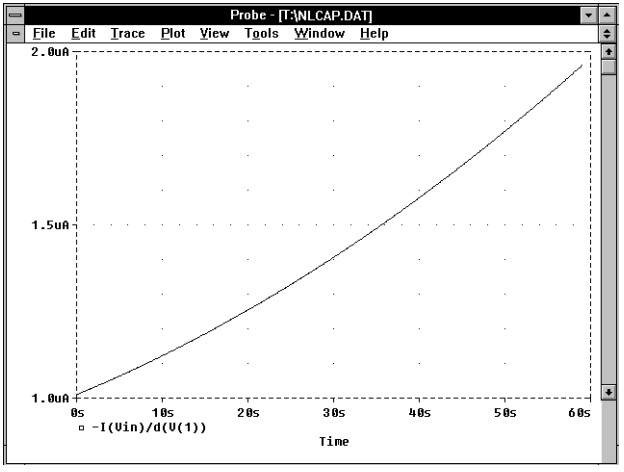


Figure 89 *Probe plot of transient analysis results*

- In the AC analysis, the capacitance is biased at 2V by the voltage source Vin, then Vin is used as a 1V AC source to stimulate the capacitor and perform the linear analysis. In this analysis, the effective admittance of the capacitor at any frequency is

  `-I(Vin)/V(1) = C(2V) * j * ω`

  whereV(1)=1, and ω=2π*frequency.

  The capacitance, therefore, can be plotted in Probe as: -Ii(Vin)/(2*3.1416*Frequency). This expression should result in a constant value of approximately 1.0204u(F). The AC analysis results are shown in Figure 90.

Note *The IC1 and IC2 setpoints (or the .IC statement) are generally the recommended method for specifying initial conditions; not the IC= optional parameter with the Use Init. Conditions option in the Analysis/Setup/Transient dialog box (or the UIC option in the .TRAN statement). Setpoints are the only way to set initial conditions for nonlinear capacitor models like this one.*

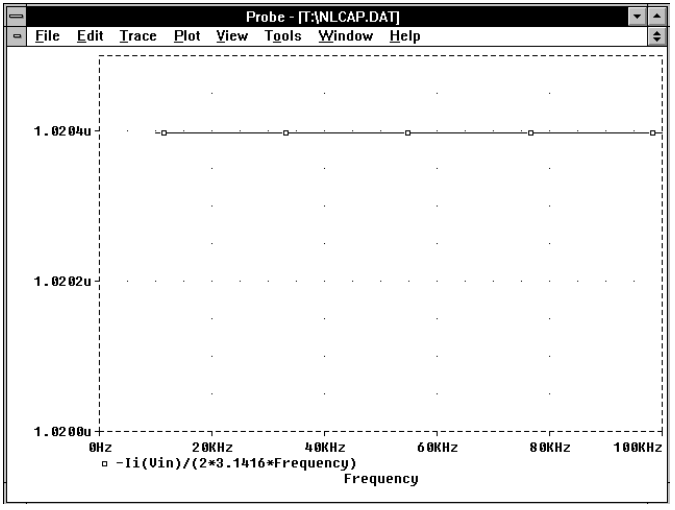


Figure 90 *Probe plot of AC analysis results*

For another example of a nonlinear capacitor model, please see the application note *Modeling Voltage-Variable Capacitors*, April 1992.

Obtain S-Parameter Data from Probe

The Design Center Source newsletter, April 1994

RF and microwave circuit designers frequently express the input and output characteristics of circuits in terms of scattering parameters (*s*-parameters). By adding two subcircuits, *s*-parameter data can be printed to the output file or displayed in Probe. The method presented here is qualified for two-port networks, but the concept can be extended for *n*-port networks.

Theory

S-parameters measure the ratio of the incident and reflected signal. The incident signals are defined as *a1* and *a2*. The reflected signals are defined as *b1* and *b2*. The incident and reflected signals are related to voltages and currents at ports 1 and 2 by

$$a1 = \frac{V1 + Z_0 I1}{2\sqrt{Z}} \quad (1)$$

$$b1 = \frac{V1 - Z_0 I1}{2\sqrt{Z}} \quad (2)$$

$$a2 = \frac{V2 + Z_0 I2}{2\sqrt{Z}} \quad (3)$$

$$b2 = \frac{V2 - Z_0 I2}{2\sqrt{Z}} \quad (4)$$

The scattered waves are related to the incident waves by the following set of linear equations:

$$b1 = S_{11}a1 + S_{12}a2 \quad (5)$$

$$b2 = S_{21}a1 + S_{22}a2 \quad (6)$$

or, in matrix form as

$$\begin{bmatrix} b1 \\ b2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} a1 \\ a2 \end{bmatrix} \quad (7)$$

The S_{ij} coefficients are dimensionless ratios; for most applications, the characteristic impedance of the system, Z_0, is 50 ohms. S_{11} is the input reflection ratio and is defined as the ratio of the input port reflected wave to the input incident wave. If the incident wave at the output, *a2*, is set to zero, then the equations reduce to $b1 = S_{11}a1$ and $b2 = S_{21}a1$. Using the defining equations, this reduces to

$$S_{11} = \frac{b1}{a1} = \frac{V1 - Z_0 I1}{V1 + Z_0 I1} = \frac{Z - Z_0}{Z + Z_0} = 2\left(\frac{Z}{Z + Z_0}\right) - 1 \quad (8)$$

where *V1/I1* is the input impedance *Z*. Similarly, S_{21} is the forward transmission ratio and is defined as the ratio *b2/a1*. If the input and output load impedances of the circuit are the same, then S_{21} is the voltage measured at the output multiplied by 2. If the incident wave at the input is set to zero, then the equations reduce to $b1 = S_{12}a2$ and $b2 = S_{22}a2$.

Defining the Subcircuits

To make all of the necessary measurements, two subcircuits are required. Both of these subcircuits, shown in Figure , and Figure 94 can be created by drawing the schematic and using the File/Symbolize command to generate a corresponding symbol. For the purpose of this article, these custom symbols are named XMITS and REFLECTS, respectively.

The XMITS circuit shown in Figure is used to measure the forward, S_{21}, and reverse, S_{12}, transmission coefficients. Since the output load matches the input load, the transmission coefficients are the output voltage multiplied by 2. The E device, E1, has a gain of 2. The interface pin labeled CKT is used to connect to the external circuit. The pin, S_TR, is a hidden pin (see Figure 92); if it is left unconnected in a schematic, Schematics will generate a unique net. Alternatively, a specific net can be named for connection by defining the ipin(S_TR) attribute value (see Figure 93) for the XMITS symbol instance;

this way, the S_TR pin will have a known label (*s21* in this case) when analyzing simulation results within Probe.

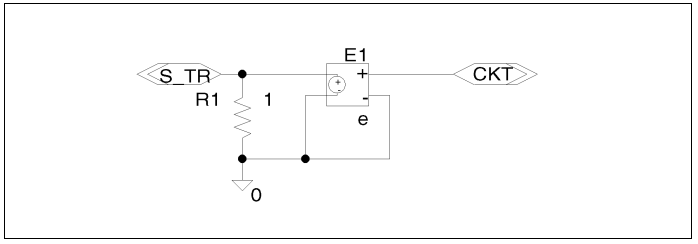


Figure 91 *Transmission coefficients measurement circuit*

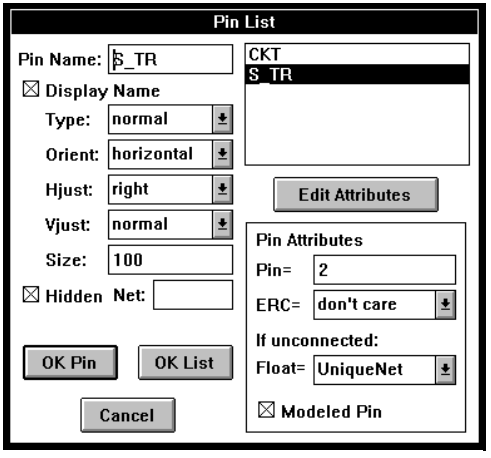


Figure 92 *Pin list for the XMITS symbol*

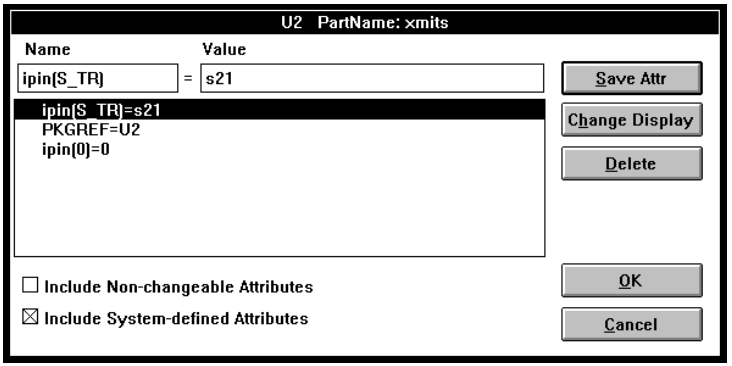


Figure 93 *Attributes for the XMITS symbol showing hidden S_TR pin*

The REFLECTS circuit shown in Figure 94 is used to measure the input, S_{11}, and output, S_{22}, reflection coefficients. The reflection coefficients are the input voltage multiplied by 2 minus AC unity. The E device, E1, has a gain of 2. As in the transmission coefficients measurement circuit, the interface pin labeled CKT is used to connect to the external circuit. The pin, S_RE, is a hidden pin like S_TR described above. The REFLECTS symbol also has a DC_BIAS attribute. On active circuits, the DC level can be set on voltage source, V1, by changing the DC_BIAS attribute value for the REFLECTS symbol instance in Schematics. By default, this attribute is set to zero.

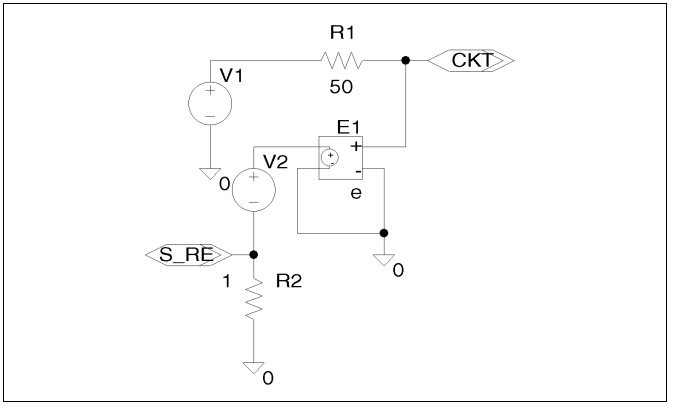


Figure 94 *Reflection coefficients measurement circuit corresponding to the REFLECTS symbol*

Using the Subcircuits

The subcircuits can be used for both passive and active circuits. The circuit shown in Figure 95 is a 4th-order Butterworth band-pass filter with a center frequency of 250 MHz. The first circuit measures S_{11} and S_{21}. The second circuit measures S_{12} and S_{22}. For simulation, the AC analysis settings are 500 linear points from 200 MHz to 350 MHz. The results of the analysis are shown in Figure 96.

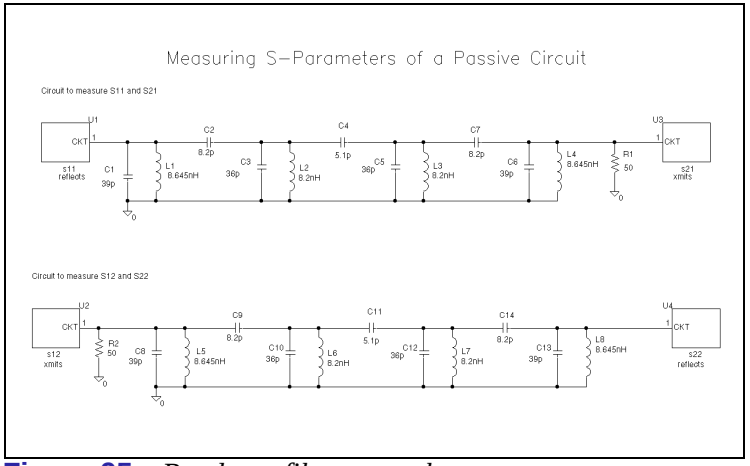


Figure 95 *Bandpass filter example*

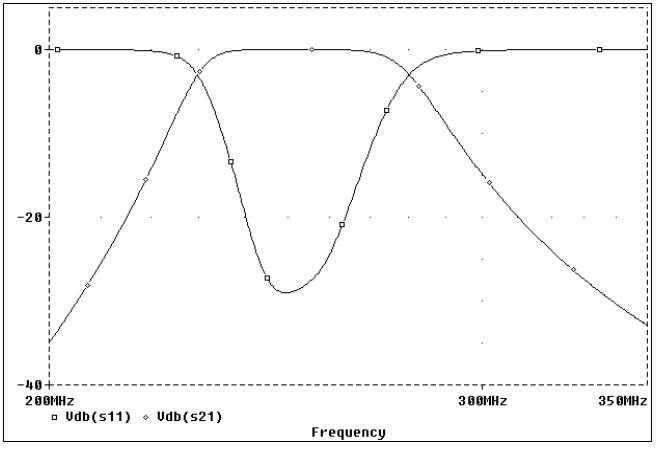


Figure 96 *S_{11} and S_{21} for Filter Example*

It is also possible to measure the *s*-parameters of an active circuit. To illustrate this, we have chosen to measure the *s*-parameters of the RF transistor, MRF571/MC. Figure 98 shows the circuits for this example. The transistor is biased for a VCE of 6.0 volts and an IC of 5.0 mA. The current is set by the current source at the emitter of the transistor. The DC bias is set by V1 in the circuit that measures S_{11} and S_{21}, and by the DC_BIAS attribute in the reflection measurement subcircuit that measures S_{12} and S_{22}. For simulation, the AC analysis settings are 100 points per decade from 200 MHz to 2 GHz. The results shown in Figure 98, Figure 99, and Figure 100 are expressed as magnitude and phase. This is typically the way most manufacturer's data sheets show the *s*-parameters.

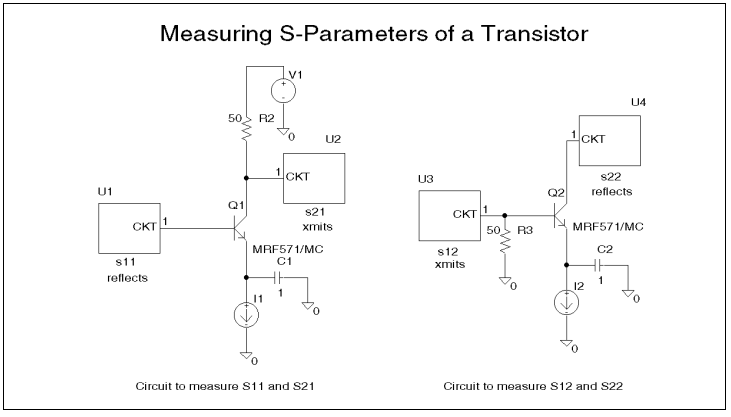


Figure 97 *Transistor example*

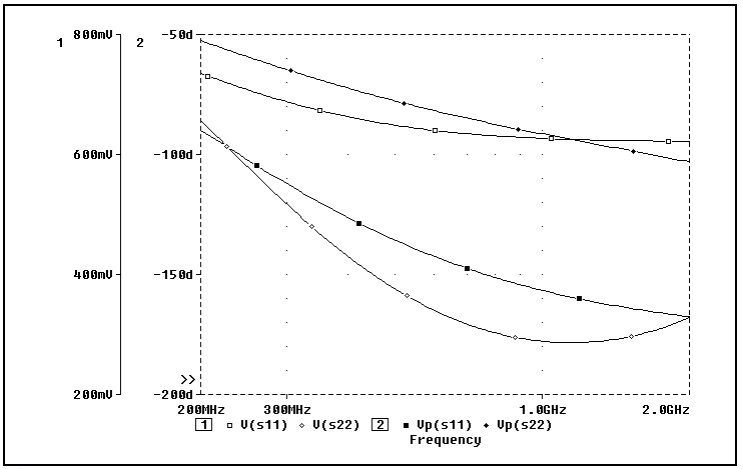


Figure 98 *S_{11} and S_{22} magnitude and phase*

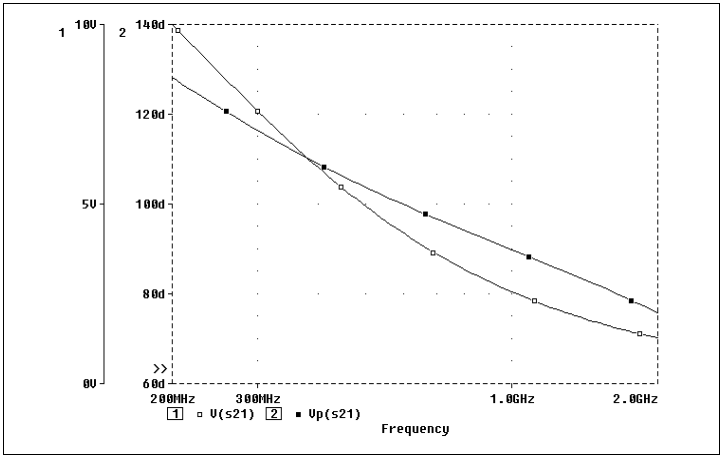


Figure 99 *S_{21} magnitude and phase*

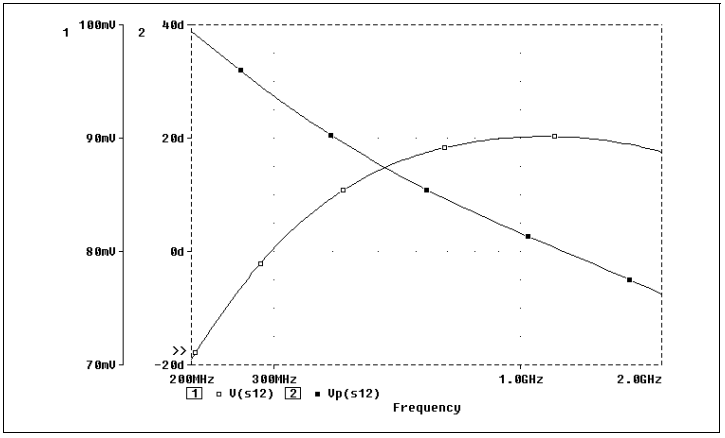


Figure 100 *S_{12} magnitude and phase*

Radiation Effect Modeling

The design of electrical systems for military and space applications requires a consideration of the effects of transient and total dose radiation on system performance. Simulation of radiation effects using PSpice helps to identify critical circuit components which could cause system failure and gives guidance for radiation hardening of the system. The following information is intended to provide a starting point for those interested in using PSpice for radiation effects analyses.

Dose-Rate Effects

When semiconductor devices such as diodes, transistors, and integrated circuits are exposed to ionizing radiation, such as gamma-rays or X-rays, hole-electron pairs are generated within the semiconductor material. These free carriers result in the generation of photocurrents as they are swept through the depletion regions of the *p-n* junctions of the device or integrated circuit. The magnitude of these currents can be orders of magnitude greater than normal signal levels and can result in temporary or permanent system failure. PSpice can be easily used to model this type of transient effect.

The magnitudes of the photocurrents are often obtained from experimental data, but can be obtained from theoretical expressions if information on the fabrication process is available. Usually, if an integrated circuit process has been developed for use in a radiation environment, dosimetry measurements have been made on each of the junction types (for example, source, drain and well-to-substrate diffusions in a CMOS technology). Therefore, the photocurrent data is directly available. If no experimental data is available, it is possible for the integrated circuit designer to use a theoretical expression for photocurrent generation. Wirth and Rogers [1] developed the following equation for a rectangular pulse of ionizing radiation.

$$Ipp(t) = gamma \cdot q \cdot g \cdot A \cdot ((W+L erf())^{1/2}) \cdot U(t) - (W+L \cdot erf())^{1/2}) \cdot U(t-tp))$$

where:

Ipp	is the photocurrent in amperes,
gamma	is the dose rate in rads(Si)/sec,
q	is the electron charge (1.6E-19 coulombs),
g	is a constant whose value is 4.2E13 carriers/(rad·cm^3),
A	is the junction area in cm^2,
W	is the depletion width in cm,
L	is the diffusion length in cm,
tau	is the minority carrier lifetime in sec,
tp	is the radiation pulse width in sec,
t	is time in sec, and
U(t)	is a unit step function of time.

For the circuit designer using discrete components, no information is usually available on the process used to fabricate the devices. Dosimetry measurements have been made, however, on a large number of commercial discrete components. This information is often available within aerospace, military and government organizations.

An examination of plots of the Wirth equation [2] or of actual experimental data shows that the photocurrent shapes are closely approximated by the EXP source function available in PSpice. In some cases, it may be desired to model the data with the piecewise linear (PWL) source function.

When setting up the circuit description for this type of analysis, photocurrent generators should be placed across all *p-n* junctions in the design. This must include all parasitic junctions as well as the junctions that make up the explicit design

components. Examples of photocurrent generator placement are shown in Figure 101.

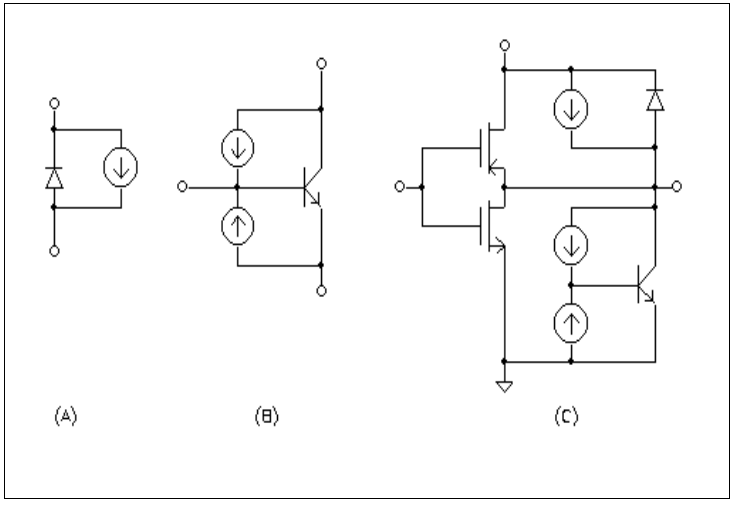


Figure 101 *Photocurrent generator placement*

Figure 101 A and B show the proper placement for a diode and a bipolar transistor respectively. These devices could represent discrete components or devices on a bipolar integrated circuit. Figure 101 C is an example of the parasitic bipolar diodes and transistors that exist in a CMOS technology. Note that the polarity is such that current flow is from the *n*-type material to the *p*-type material.

It is easiest to use a current-controlled current source for photocurrent modeling. For example, assume that you have a small circuit containing only two discrete diodes, both of type DMOD. Further assume that the rest of the circuit consists only of passive components. A portion of the PSpice circuit description is shown below.

A small controlling circuit is set up consisting only of a voltage

```
VPDMOD 1 0 EXP (0 40MV 100NS 8NS 150NS 50NS)
RPDMOD 1 0 1

D12  10 2 DMOD
FD12 10 2 VPDMOD

D20   7 6 DMOD
FD20  7 6 VPDMOD
```

source VPDMOD and a 1 ohm resistor RPDMOD. Note that the photocurrent has been defined as a voltage source such that both the voltage across the source and current through the source are equal in magnitude to the desired photocurrent. In this case the exponential source function has been used to model the photocurrent. Figure 102 shows the photocurrent waveform from this example.

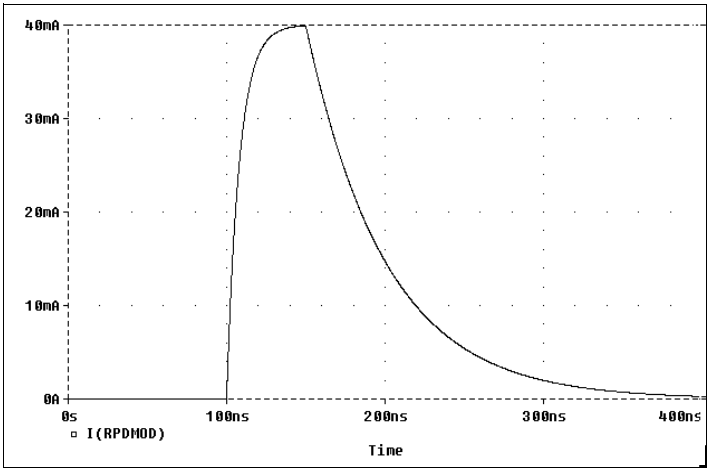


Figure 102 *Photocurrent waveform*

A controlled source is then placed across each occurrence of the diode DMOD in the circuit. If the circuit had contained other semiconductor devices, an additional controlling circuit would have to be defined for each *p-n* junction type.

For an integrated circuit design, the photocurrent modeling process is actually somewhat simpler. Since there are relatively few *p-n* junction types in a typical integrated circuit technology, the number of controlling sources required will be very small. In this case, define the magnitudes of the controlling sources as current densities. By doing this, the <*gain*> parameter in the

definition of the current-controlled current source becomes the area of the specific *p-n* junction being modeled.

The following example demonstrates how PSpice can be used to determine the dose rate sensitivity of an integrated circuit. Assume that a die contains a number of diodes all fabricated with the same diffusion and differing only in diode device area. Also assume that dosimetry measurements on this diffusion type have shown a peak current density of 4E-2 amps/cm^2 at a dose rate of 1E7 rad(Si)/sec. A small section of a typical circuit description is shown in the following text.

```
.PARAM DOSE_RATE = 1E7              ;initialization
.PARAM JP_MEAS   = 4E-2             ;peak measured current density
.PARAM DOSE_MEAS = 1E7              ;at measured dose rate

.FUNC  JP_PEAK(DOSE_RATE)  DOSE_RATE*JP_MEAS/DOSE_MEAS

* perform simulations at 1e6, 1e7, and 1e8 rad(Si)/sec
.STEP PARAM DOSE_RATE LIST 1E6 1E7 1E8

VPDIF1 1 0 EXP (0 {JP_PEAK(DOSE_RATE)} 100NS 8NS 150NS 50NS)
RPDIF1 1 0 1

D12  10 2 DIC1 AREA=2.5E-3
FD12 10 2 VPDIC1 2.5E-3

D20   7 6 DIC1 AREA=5E-3
FD20  7 6 VPDIC1 5E-3
```

For dose rates up to 1E8, there is a linear relationship between dose rate and photocurrent. This relationship has been defined using the .FUNC statement. For higher dose rates, a nonlinear relationship could be defined using the same capability. The voltage source VPDIF1 and the resistor RPDIF1 are used to model the photocurrent data obtained from a test structure. Note that the peak current is defined as a current density. As in the earlier example, controlled sources are inserted across all diodes of this diffusion type. The <*gain*> term of each current-controlled current sources is the area in cm^2 of the specific diode. Thus, diode D20 is fabricated with a diffusion area twice as large as that of diode D12. The .STEP capability is then used to perform the simulations to determine the circuit response as a function of dose rate.

Single-Event Upset

Single-Event Upset (SEU) occurs when a high-energy ionizing particle, such as heavy ions, alpha particles or protons, irradiates a circuit or passes through an integrated circuit causing a disruption in the system logic. The most common effects of SEU are logic upsets in high-density digital circuits particularly memories and the registers in microprocessors. Exact upset levels are very difficult to predict because of the complex nature of the physical mechanism involved. However, simulations can aid greatly in evaluating design and process changes to increase the hardness of the circuit to this type of event.

One analysis procedure for SEU is similar to that used for dose-rate effects discussed earlier. A current generator is inserted in the circuit description to model the charge collected on an assumed susceptible node as a result of the particle hit. Messenger [3] developed the following analytical approximation for this current

$$I(t) = Io \cdot sec(theta) \cdot (e^{-t/\alpha} - e^{-t/\beta})$$

where:

Io	is approximately the maximum current,
theta	is the angle of incidence of the particle to the surface of the circuit,
α	is a collection time constant of the junction, and
β	is the time constant of initially establishing the ion track.

Io varies according to particle type and is proportional to the energy of the particle and the doping profile of the semiconductor material. The total current increases as the angle of incidence varies from grazing to near normal.

The following partial circuit description demonstrates how to perform the analysis.

```
.PARAM IO = 1E-3     ; approximate peak current
.PARAM ANG = 1.3     ; angle of incidence (in radians)

.FUNC ISEU(IO,ANG) IO*(EXP(-TIME/150PS)-EXP(-TIME/40PS))/COS(ANG)

.STEP PARAM IO LIST 1E-3 3E-3 1E-2
GSEU  1 1 0 VALUE={ISEU(IO,ANG)}
```

The .FUNC statement is used to define the SEU current as a function of the peak current, IO, and the incidence angle, ANG. Note that this expression assumes a particle hit at TIME=0. Using the Analog Behavioral Modeling extensions, a current source GSEU is included in the circuit description to represent the charge collected on the susceptible node (in this case, node 11). The .STEP statement is then used to perform an analysis to determine the circuit response either as a function of peak current as shown, or as a function of the incidence angle.

Total Dose Effects

Ionizing radiation produces both transient photocurrents, as discussed earlier, and permanent damage in semiconductor devices. The degree of this permanent damage is proportional to the total accumulated dose and results in a general degradation of device performance.

For example, when MOS structures are exposed to ionizing radiation, they experience a net change in the charge associated with the Si-SiO_2 interface. This change in charge has two components: (i) the charge stored in the oxide, and (ii) the charge stored in the hole-electron interface traps. The net result is to shift the threshold voltages of both *p*-channel and *n*-channel devices toward enhancement mode operation. In addition to this threshold voltage shift, increases in the surface state density produce decreases in surface mobility and a variation in the turn-on characteristic of the MOS transistors. The decreased mobility is the result of increased diffusion carrier scattering at the silicon surface. The variation in turn-on characteristics is caused by the filling and emptying of the surface traps as the channel is formed.

Essentially all investigations of the effects of total dose radiation (permanent damage) on semiconductor devices have been made in terms of changes in the device electrical parameters. Analyses on MOS technologies are generally performed as follows:

> Model parameters are extracted from test devices at several total dose levels over the range of interest. You must be very careful not to project much outside the range for which

experimental data is available, since slight process variations can radically affect the model parameter versus total dose relationship.

An analysis of the model parameters is then performed to determine what parameters are most affected by the total dose and then, relationships are developed to describe the shift in the model parameters as a function of total dose.

The following example is given to show the technique. It is assumed that changes in the threshold voltage and mobility are the dominant total dose effects.

```
.PARAM DOSE = 0.0    ;initialization of total dose

.STEP PARAM DOSE LIST 0 1E5 1E6

.FUNC VTO_DOSE(DOSE)
+1.65*(1 - 4.18E-6*DOSE + 2.76E-12*DOSE*DOSE)

.FUNC UO_DOSE(DOSE) 600.0*(1 - 5.00E-7*DOSE)

.MODEL NN2 NMOS ( VTO={VTO_DOSE(DOSE)}  UO={UO_DOSE(DOSE)} )
```

In this example, the .STEP capability is used to determine the circuit performance as a function of total dose. Simulations are performed at DOSE = 0.0 (pre-rad), 1E5, and 1E6 rads(Si). Note that the functional relationship between the threshold voltage and total dose has been defined as a nonlinear function, in this case a second order polynomial fit. The variation in mobility as a function of total dose has been assumed to be linear.

References

[1] J. L. Wirth, S. C. Rogers, “The Transient Response of Transistors and Diodes to Ionizing Radiation,” *IEEE Transactions on Nuclear Science*, NS-11(5):24-38, November 1964.

[2] *Handbook of Modeling for Circuit Analysis Including Radiation Effects*, Air Force Weapons Laboratory, AFWL-TR-79-86, Albuquerque, New Mexico, May 1979.

[3] C. G. Messenger, “Collection of Charge on Junction Nodes from Ion Tracks,” *IEEE Transactions on Nuclear Science*, NS-29(6):2024, December 1982.

Signal Integrity of Stub Versus Daisy-Chain Layouts

The Design Center Source newsletter, July 1993.

Signal Integrity Analysis (Polaris)—an integrated feature of the software—is used to investigate the effects of board layout on a circuit design. This article compares the behavior of different layout topologies for a simple digital circuit. We will use Polaris to extract transmission line effects from the alternative board layouts, PSpice to simulate the designs with parasitics added, and Probe to view the resulting waveforms.

The Example

The schematic in Figure 103 illustrates two alternative design approaches for a circuit in which one TTL output feeds a total of five TTL inputs. Three of the driven devices are gates; the other two are flip-flops. A simple pulse is used as the stimulus.

The first approach shown in the upper part of the schematic will be laid out with stub connections. The second approach shown in the lower part of the schematic will be laid out as a daisy

chain. This approach also has a terminator at the farthest end of the daisy-chained signal.

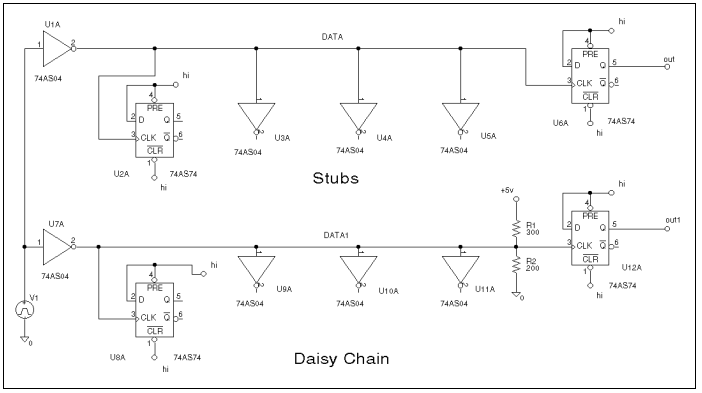


Figure 103 *Schematic showing the stub (upper) and daisy chain (lower) design approaches for the example circuit*

Signal Integrity Considerations

Transmission line effects should be considered when the interconnect delays in a physical design are greater than about one-half of the rise times of signals on those interconnects.

For this example, we are using AS-series logic, with rise and fall times around 0.6 nsec. Propagation delay for signals on typical printed circuit boards is around 1.8 nsec per foot. This guideline states that transmission line effects will be important for delays greater than 0.3 nsec. This is equivalent to a trace length of about 2 inches.

There are three major effects to consider: delay, reflections, and crosstalk. This article concentrates on the effects of reflections. Delay and crosstalk are not significant in this example.

Reflections

Reflections arise because the traces, considered as transmission lines, have a characteristic impedance (Z0). Any mismatch between the impedance of the device driving the line, the line itself, and the final termination of the line, will result in reflections. The magnitude and phase of the reflected signals depend on the complex impedance presented by the load(s) on the line.

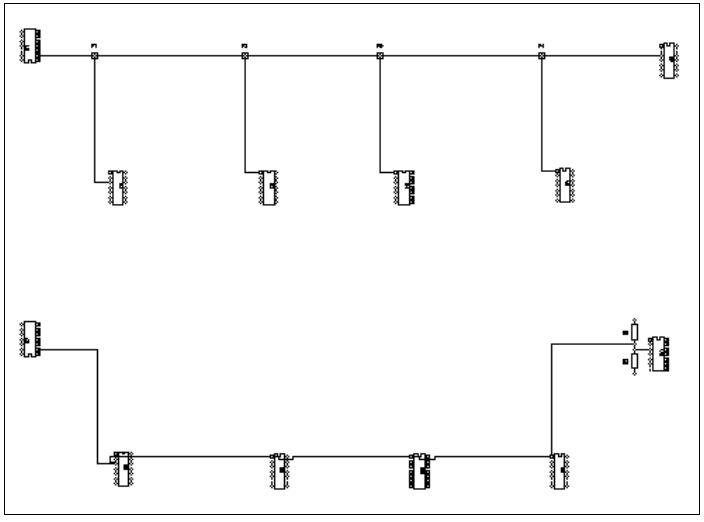


Figure 104 *Stub (upper) and daisy chain (lower) layout topologies*

Consider the two layout styles—stub and daisy-chain—from a reflection point of view. In each case, a voltage step is applied by the driver. This launches a step down the line whose amplitude depends on the impedance of the driver, the characteristic impedance of the line, and the height of the voltage step. The PADS layout in Figure 104 shows the two styles.

Stub Case

When the step reaches a T junction, it splits into two. Part of the energy travels down the stub and is then mostly reflected back when it encounters the high impedance load at the end of the stub. This introduces a *notch* of twice the delay of the stub into the wave traveling down the main signal path. The wave reflected from the stub will travel in both directions along the main signal path. If there is an impedance mismatch between the driver and the line, some of this energy will be reflected in the forward direction. With several stubs, very complex waveforms may be observed.

Daisy-Chain Case

As the step travels down the line, it encounters impedance changes at device connections, due to the resistive and capacitive load placed on the line. These changes in impedance cause some energy to be reflected back along the line. As above, if there is a mismatch between the driver and the line, some of this energy will be reflected in the forward direction. The impedance changes at the connection points are much smaller, however, than when stubs are used. This means that the waveform will remain relatively clean as it travels along the signal path.

Termination

If the line is not terminated at the far end, most of the incident energy will be reflected back along the line. This will be scattered by the stubs and loads, and eventually arrive back at the driver. If there is a significant mismatch between the driver and the line, then some of this energy will be reflected in the forward direction, and so on. (Traditionally, a Bergeron diagram is used to derive the magnitude of the various reflections in a mismatched system.)

Using Polaris to Extract Parasitics

Polaris takes the physical layout description (the geometry of the layout) and additional information describing the electrical characteristics of the board (thickness of layers, their dielectric constants, conductor resistivity, and thickness). From these inputs, Polaris produces a set of transmission line segments that model the traces on the board. Some of the segments may be coupled; these are the segments which give rise to crosstalk.

In this domain, trace lengths greater than about 2 inches should be treated as transmission lines. Minimum Transmission Line Length should be set to “2000mil” in the main Polaris Setup dialog. The other fields can be left at their default values. The Crosstalk Mode should be set to “All” in order to extract all nets present in the layout.

Schematics merges the extracted board parasitics into a PSpice netlist. PSpice then simulates the circuit including lossy transmission line effects (with coupling, if present). The user has control over selecting the nets for which parasitic information is to be included. Simulation of the entire set of parasitics is expensive in terms of both time and the amount of output data produced—and is usually unnecessary.

The extracted parasitics can be displayed by Schematics using the Select Signals button in the main Polaris dialog. The results for this example are summarized in Table 3. The row labeled DATA contains the characteristics of the main net in the design using stubs; DATA1 contains similar information for the design using the daisy chain approach.

Table 3 *Parasitics for the Example Circuit Topologies*

	Length	Distr. L	Distr. C	R	C_{via}
DATA	0.577	430 nH	22.3 pF	1.28	10 pF
DATA1	0.434	327 nH	16.8 pF	0.96	8 pF

The distributed inductance and distributed capacitance results shown in the table can be used to compute the delay and characteristic impedance for the nets as follows:

$$Z0 = \sqrt{\frac{L}{C}}$$

$$T_d = \sqrt{L \cdot C}$$

The delay and characteristic impedance for the example nets are: 3.1 nsec and 140 ohms (DATA); 2.3 nsec and 140 ohms (DATA1). This gives us some idea of the magnitude of effects to expect. The output impedance of an AS output stage is around 10 ohms; this is a significant mismatch with the 140 ohm line impedance, so there will be large reflections at this interface. An AS input stage presents a complex nonlinear load, but outside the switching region, it is equivalent to 10 Kohms or more. This means that the stubs (and the daisy chain *tap* points) will be lightly loaded.

Simulating the Design

In this design, the only relevant nets are DATA and DATA1. The Select Signals dialog is used to select just these two nets for simulation with parasitic effects included.

The pulse source has a period of 80 nsec. A 140 nsec transient analysis is performed to see the parasitic effects for three complete edges.

Once the simulation has completed, Probe can be used to view the resulting waveforms. If simulations have been performed both with and without parasitics, Schematics offers the choice of using Probe to view either set of results, or a concatenation of the two.

Adding parasitics to a previously all-digital net effectively turns all visible signals on the net into analog signals. For example, if there is an all-digital net in a design labeled CLOCK, you would enter “CLOCK” as the trace specification in Probe in order to display this digital signal. Once parasitics have been added to the net, however, there are no longer any digital signals. Now to display the trace in Probe, “V(CLOCK)” must be specified.

In this example, the “without” results are of little interest. Figure 105 shows the results of the simulation with parasitics applied.

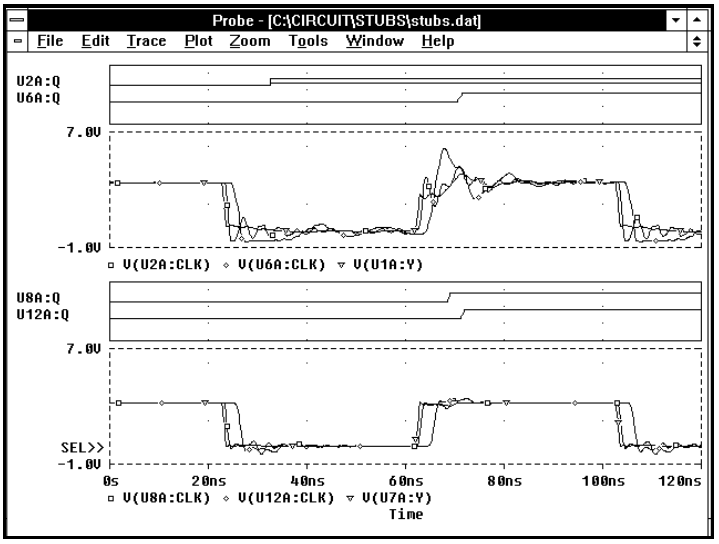


Figure 105 *Analysis of parasitics in the stub approach (top) and the daisy chain approach (bottom)*

The upper set of plots shows what happens in the stub layout case. The signal present at the clock input of the flip-flop at the end of the first stub—V(U2A:CLK)—has been so degraded that false triggering occurs about 26 nsec into the run. This can be seen by the flip-flop output—U2A:Q—becoming an X state. The simulation also reports “simulation errors”—in this case, violations of the minimum clock width for the flip-flop. The Tools/Simulation Errors command in Probe provides easy access to this information.

The lower set of plots shows what happens in the cleaned-up design, using daisy-chaining and termination. The analog signals are noticeably cleaner. The digital signals show no sign of false triggering.

Summary

High-speed PC board design requires that transmission line effects be taken into consideration when delays are comparable with edge speeds of the logic used in the design.

Board layout strategy can have a pronounced effect on the integrity of the signals in the physical implementation of a design. Poor signal integrity compromises noise margins, resulting in products which may fail intermittently under data-dependent conditions.

Signal Integrity Analysis (Polaris) allows the effects of board layout and fabrication to be incorporated into the simulation of a design. The designer can verify that the design operates as intended in the presence of these parasitic effects.

Simulating High-Q Circuits Using Open Loop Response

Introduction

The length of time it takes to perform a transient simulation on a high-Q oscillator circuit makes simulation inefficient. The dominant time constant of the circuit, due to the Q of the crystal, means that the simulation must run a minimum of Q cycles before the circuit reacts. And, because you must wait Q cycles, there is no way to force an oscillator to a steady state condition. For a crystal with moderately high-Q (20,000), it can take close to a million cycles before the oscillator reaches a steady state condition. Besides the long time required for the simulation to run, the data file created by the simulation will be extremely large.

But, it is possible to use AC analysis to simulate high-Q circuits. These results can be used to investigate whether the loop gain and phase are conducive to producing a stable oscillation.

Theory

This approach to measuring loop gain relies on the voltage and current feedback theory as described in detail in *A Guide to Circuit Simulation and Analysis Using PSpice*, references [1] and [2]. The circuit is analyzed by injecting a current to measure the current gain, and a voltage to measure the voltage gain of the circuit.

Previously, the circuit to be analyzed was treated as a subcircuit, with the signal path, broken to make measurements, “pulled out.” The subcircuit was then used twice: once so that a voltage could be injected, and again so that a current could be injected.

There is a cleaner approach to measuring the loop gain. First create a subcircuit for the device used to make the measurements, then reinsert this subcircuit into the circuit to be analyzed. The loop gain subcircuit, consisting of two voltage sources and one current source, is as follows:

```
.SUBCKT LGT A B PARAMS: VOLTS=0
V1 1 A DC 0 AC {VOLTS}
V2 1 B DC 0 AC 0
I1 1 0 DC 0 AC {1-VOLTS}
.ENDS
```

When the value of the VOLTS parameter is equal to 0, the I1 current source will have a value of 1, and voltage sources V1 and V2 will have a value of 0. This condition allows measurement of the current gain of the circuit. Sources V1 and V2 will be used to sense the input current and the output current of the circuit. The current gain of the circuit is determined with the following equation:

$$Ti = \frac{I(V1)}{I(V2)}$$

When the value of the VOLTS parameter is equal to 1, the V1 voltage source will have a value of 1, and the I1 current source will have a value of 0. This condition allows measurement of the voltage gain of the circuit. The voltage gain is the ratio of the voltages measured at the nodes of the subcircuit, or

$$Tv = \frac{V(A)}{V(B)}$$

To perform the current gain and the voltage gain portions of the analysis, it is necessary to perform a parametric analysis in addition to the AC analysis. This can be accomplished by placing a PARAM symbol on the schematic. In the example shown in Figure 108, the parameter is named ACVAL, and the default value is set to 0. The parametric sweep is set to run the AC analysis with the global parameter ACVAL set to each of the values defined in the value list. The value list for this analysis contains two values: 0 and 1.

Once the voltage gain and the current gain are calculated, the total loop gain can be determined by the following relation:

$$\frac{1}{T+1} = \frac{1}{Ti+1} + \frac{1}{Tv+1}$$

Therefore,

$$T = ((Ti+1) \parallel (Tv+1)) - 1$$

where

$$x \parallel y = \frac{1}{\frac{1}{x} + \frac{1}{y}} = \frac{x \cdot y}{x+y}$$

The symbol for this subcircuit can be a simple box with 2 pins (Figure 106), one on each side:

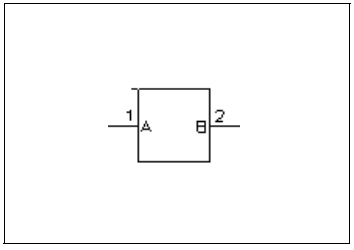


Figure 106 *Symbol Graphic*

The symbol contains the attribute VOLTS = 0. This parameter is used to pass the value of a global parameter to the subcircuit. In the example circuit, the global parameter ACVAL is defined and the attribute value of VOLTS is set to {ACVAL}. If a parameter value is not passed to the subcircuit, the default value for VOLTS is then 0. The template for the symbol is as follows (as a single line):

```
X^@REFDES %A %B @MODEL= PARAMS: ~VOLTS|VOLTS=@VOLTS|
  ~VOLTS|VOLTS=0|
```

The attributes for the symbol are show in Figure 107 below.

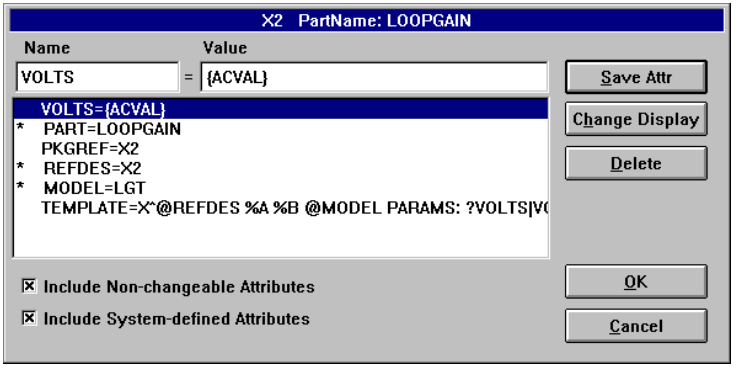


Figure 107 *Symbol Attributes*

Example Circuit

Figure 108 shows a 100 kHz crystal oscillator circuit with a Q of 20,000. To analyze this circuit, the LOOPGAIN subcircuit is inserted into the feedback path. The A node is connected to the output side of the feedback path and the B node is connected where the feedback is summed into the circuit. If the subcircuit is inserted backwards, the results will show a loss, or be upside down. The AC analysis is set up to sweep the circuit from 99.9 kHz to 100.1 kHz using a linear sweep of 100 points.

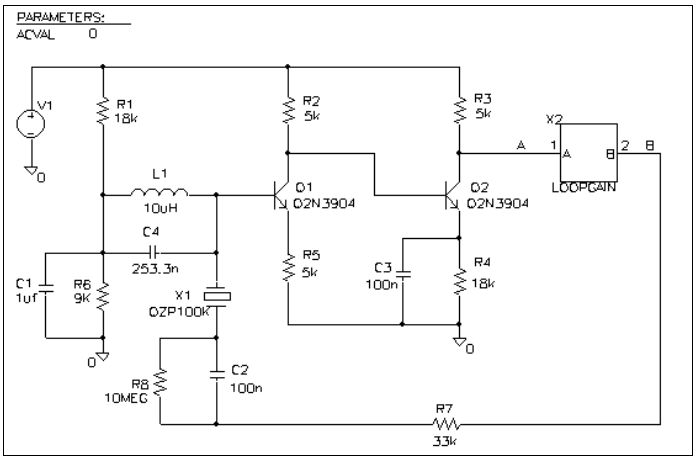


Figure 108 *100 kHz crystal oscillator circuit*

After the analysis is complete, the results may be viewed using Probe. The following Probe macros can be defined:

```
par(a,b)=(((a)*(b))/((a)+(b)))
Ti=((I(X_X2.V1)@1)/I(X_X2.V2)@1)
Tv = (V(A)@2/V(B)@2)
T=(par(Ti+1,Tv+1)-1)
```

The macro par(a,b) defines the parallel combination of the arguments a and b. The macros are entered by selecting Trace/ Macro in Probe and entering the macros in the macro dialog box. Click on the Save button to insert each new macro into the list. To exit from the dialog box, click OK.

Results

The results (Figure 109, Figure 110, and Figure 111) for this oscillator circuit show the peak gain and a 180 degree phase shift at the resonant frequency of the crystal. The tests show that this circuit will operate as an oscillator.

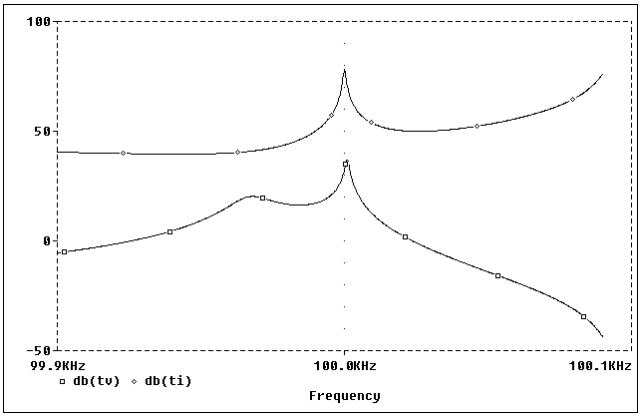


Figure 109 *Current gain Ti in db, and voltage gain Tv in db*

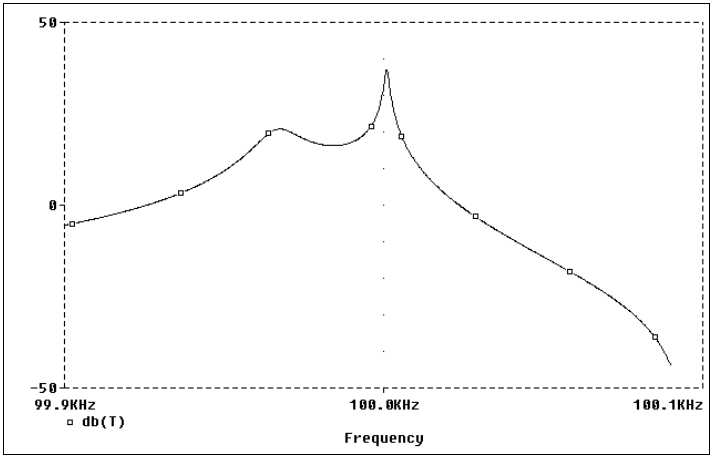


Figure 110 *Total loop gain T in db*

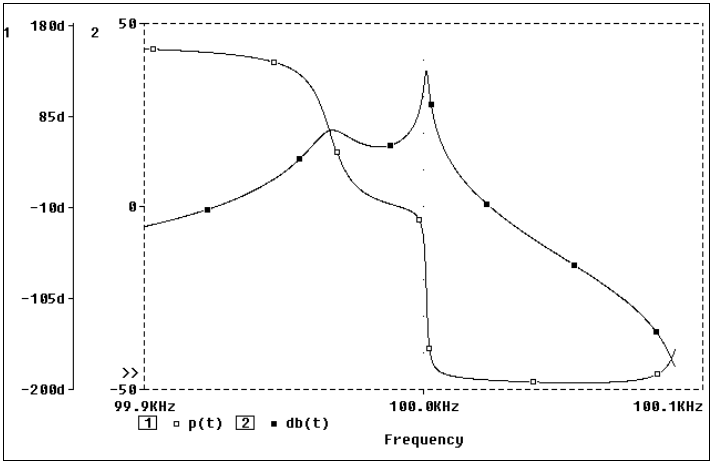


Figure 111 *Total loop gain in db and phase in degrees*

References

[1] Paul W. Tuinenga, *SPICE: A Guide to Circuit Simulation and Analysis Using PSpice*. Prentice-Hall, 1988, pages 59-65.

[2] Paul W. Tuinenga, *SPICE: A Guide to Circuit Simulation and Analysis Using PSpice*. Second Edition, Prentice-Hall, 1992, pages 81-88.

Simulating Power Circuits

MicroSim Corporation Newsletter, October 1988

U.C. Berkeley developed the SPICE program to simulate integrated circuits. In fact, the acronym SPICE stands for Simulation Program with Integrated Circuit Emphasis. Because of the "integrated circuit emphasis," the focus was on accurate simulation of circuits containing many small, fast transistors.

However, the central algorithms are not restricted to such circuits. They work equally well for discrete components, power circuits, and microwave designs. But because of the original emphasis on integrated circuits, the default values of some overall parameters are not optimal for other classes of circuitry. We would like to make some comments on power electronics in particular.

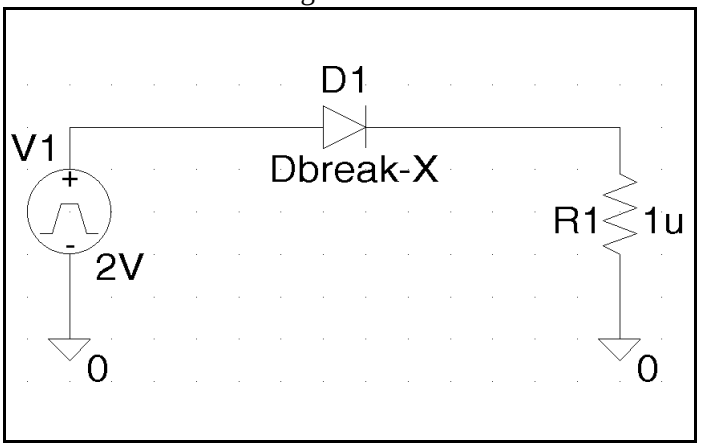
Diode and Transistor Parasitics

The default value for all parasitic resistances and capacitances in PSpice .MODEL statements is zero. This means that if you model a diode without specifying the parasitic resistance (RS) and the zero-bias *pn* capacitance (CJO), the diode will have no ohmic resistance and no junction capacitance. Not only is this unrealistic, but it can lead to numerical problems in some power circuits.

The problem with having RS equal zero is that the circuit may have nothing to limit the forward current through a diode. To take an extreme example, forward-biasing a diode with a 2-volt source can cause the diode to try to conduct megamperes of current. The forward current of an ideal diode (RS = 0) varies exponentially with voltage (please see the diode equations in the *MicroSim PSpice Reference Manual* for more details). Without an ohmic resistance to limit it, the current can easily become large enough to cause numerical problems.

An Example

Consider this schematic diagram:



Note that R1 is only a millionth of an ohm. Diode D1 is modeled after a 1N3600. Its parameters are as follows:

BV	75	CJO	2.500 E- 12	EG	1.110
FC	0.5	IBV	100.0 E -6	IKF	21.53
IS	8.845 E -18	ISR	2.668 E- 6	M	0.333
N	0.7522	NR	2	RS	1.220
TT	8.656 E -9	VJ	0.75	XTI	3

PSpice runs to completion with this model, and Probe displays a maximum current of a little more than 1 ampere. But, if we change the value of RS from 1.220 ohms to zero, then PSpice terminates with an error message referring us to the output file. In the file we find this message:

```
ERROR --
Time step = 287.9E-15 is too small in Transient
Analysis at Time = .5.
Minimum allowable step size = 1.000E-12.
The device which is changing too fast is D_D1.
```

Capacitance Problems

A different problem can arise if both CJO and transit time (TT) equal zero: the diode then has a zero switching time. Without capacitance to limit the diode's switching speed, a simulation involving transient analysis may try to make a transition in zero

time. This will cause the program to cut back the internal time step, successively making it smaller and smaller, until the program finally gives up and reports a transient convergence problem.

In both cases the conclusion is the same: if you model a diode or transistor, be sure to include parasitic resistances and capacitances.

Tolerances

The main error tolerance in PSpice defines the relative accuracy of voltages and currents (RELTOL), and a has default value of 0.001 or 0.1%. This default is not a source of concern for power circuits. However, the default values for best accuracy of voltages (VNTOL) and best accuracy of currents (ABSTOL) may need to be changed for circuits handling large voltages and currents.

The default values of VNTOL and ABSTOL are 1 μV and 1 pA respectively. The double precision arithmetic used by PSpice has about 16 digits of accuracy. Four of these are lost in the course of solving the circuit matrix, leaving a dynamic range of about 12 orders of magnitude.

If your circuit has currents in kiloamperes and an ABSTOL setting of 1 pA, the current ratio will exceed this range, possibly causing a convergence problem. We recommend setting ABSTOL equal to 1 μA in this case. If your currents are in megamperes, we recommend setting ABSTOL to 1 mA.

In general, we recommend setting VNTOL and ABSTOL about 9 orders of magnitude smaller than the typical voltages and currents in the circuit. Although the VNTOL default value of 1 μV meets this criterion for most power circuits, it's better to set ABSTOL higher than its default value of 1 pA.

Solving Differential Equations with MicroSim PSpice

by Ian Wilson, Vice President of Engineering, MicroSim Corporation

The motivation for this newsletter article was a published article (reference [1]). Thanks to the authors for their interest and for providing the data needed to produce this article.

Overview

MicroSim PSpice is well known for its ability to solve the equations which arise in circuit analysis. What is less well known is that MicroSim PSpice can also be used to solve problems in other domains which can be expressed as differential equations. This article presents some examples of using MicroSim PSpice as an “analog computer” to solve sets of differential equations describing the kinetics of a chemical reaction, and shows MicroSim PSpice Optimizer being used to fit parameters to this system from measured data.

Solving Simple Differential Equations

Consider a familiar example: the voltage across a parallel capacitor/resistor combination as a function of time. The circuit equation for this example is:

$$\frac{\mathbf{V}}{\mathbf{R}} + \mathrm{C}\frac{\mathbf{dV}}{\mathbf{dt}} = \mathbf{0}$$

rearranging a little,

$$\frac{dV}{dt} = \frac{-V}{\tau}$$

where:

$$\tau = RC$$

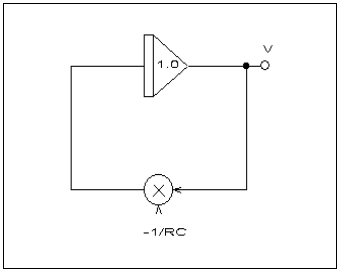
Because V is a function of the single variable, t, this is an Ordinary Differential Equation (ODE). Its solution is the equation of exponential decay, which is:

$$V_0 e^{-t/\tau}$$

where V_0 is the initial voltage on the capacitor.

To see how MicroSim PSpice can be used to solve the equation above, consider an ideal integrator. Suppose its output is the voltage we want: V. Now the input to the ideal integrator is evidently dV/dt.

A circuit which represents the equation is shown in the following schematic. The schematic was drawn using MicroSim Schematics. The parts shown come from the Analog Behavioral Modeling (ABM) symbol library, "abm.slb," supplied with the program. The symbols INTEG (integrator) and MULT



(multiplier) are used.

The symbol containing "1.0" is an integrator with gain 1. Its output is obtained by integrating the voltage at its input. This voltage is constrained to be -1/RC times the output voltage, V, by closing the feedback loop.

This way of setting up differential equations for solution is the way that analog computers were used. In this example, an integrator block would be used; the constant 1/RC would be supplied by a gain block and using the inverting input to the integrator would provide the -1.

The initial condition for this problem is that the initial voltage is V_0. On an analog computer this voltage would be derived from a reference and patched to the *initial condition* input of the integrator. Using the ABM integrator symbol, INTEG, the initial voltage is specified by setting the value of the "IC=" attribute on the symbol.

Running a Transient Analysis on the ABM representation of the problem shows the expected exponential decay of voltage with time.

Coupled Differential Equations

Systems of interest usually contain more than one variable. There may be several interacting voltages in a circuit. In a chemical reaction, the rate of production of a component may depend on the concentrations of several other components.

For example, if we have three components x_1, x_2, and x_3, the equations controlling their rates of decay and production might be:

$$\frac{dx_1}{dt} = -k_1 x_1 \quad \text{(decay)}$$

$$\frac{dx_2}{dt} = -k_2 x_2 \quad \text{(decay)}$$

$$\frac{dx_3}{dt} = k_{31} x_1 + k_{32} x_2 \quad \text{(creation)}$$

Sets of equations like this can be solved using similar techniques to the first problem. In this example, we would use three

integrators with three feedback loops and three node voltages to solve for x_1, x_2, and x_3.

Let's look at a real example (from reference [1]). This is a chemical system which contains four components, x_1, x_2, x_3, and x_4. They are related by four equations containing "rate constants" (the Ks) and "physical constants" (R and Q):

$$-\frac{dx_1}{dt} = (K_1 R x_2 + K_2 x_1)\sqrt{K_3 Q x_3} + 2K_3 Q x_3 + K_4 R x_1 x_2$$

$$-\frac{dx_2}{dt} = K_1 x_2 \sqrt{K_3 Q x_3} + \frac{2K_3 Q x_3}{R} + K_4 x_1 x_2$$

$$\frac{dx_3}{dt} = K_3 x_3$$

$$\frac{dx_4}{dt} = K_4 x_1 x_2$$

A schematic containing ABM components for integration, summing, and taking square roots, etc., is shown in Figure 114. It also contains definitions for Q and R (the physical constants), and the Ks (which are defined separately so that they can be modified by MicroSim PSpice Optimizer).

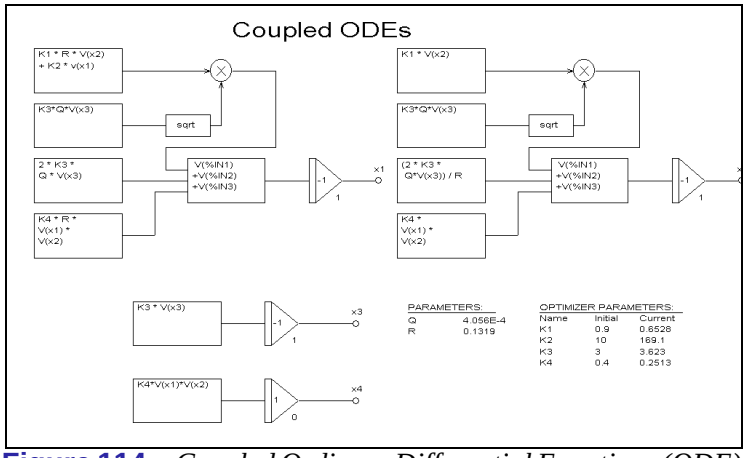


Figure 114 *Coupled Ordinary Differential Equations (ODE) implemented using MicroSim Schematics ABM components for integration, summing, square roots, etc. The physical constants Q and R are shown, as well as the rate constant, K, to demonstrate a chemical system.*

Fitting Model Parameters

The original article reference [1] was about finding values for the Ks. This ensures that the behavior of the system of equations best fits a set of experimental data. The equations should be best fit in a least squares sense.

MicroSim PSpice Optimizer is well-suited for this task. Each of the four K parameters is specified as an optimizer parameter. In other words, the optimizer may vary these in search of a fit with the data. The experimental data is set up as shown in Table 4. The alpha, beta, and gamma columns represent measured concentrations.

Table 4 *Experimental Data*

Time	Alpha	Beta	Gamma
0.1258	0.472	0.972	0.022
0.2516	0.262	0.957	0.034
0.0533	0.116	0.941	0.043
0.7549	0.069	0.935	0.050
1.2582	0.039	0.926	0.056
1.7615	0.030	0.920	0.059
2.2648	0.026	0.916	0.062

Three *external specifications* are set up in MicroSim PSpice Optimizer to act as the optimization targets. This causes the optimizer to minimize the sum of squares of differences between the tabular values at specified measurement times. It also minimizes the results of solving the set of equations with particular parameter values for each of the parameters alpha, beta, and gamma.

After a number of iterations, the optimizer finds a set of parameter values which provides a significantly better fit to the measured data. These parameters, although somewhat different from the results shown in the reference article, produce a good fit to the measured data. Figure 115 shows the reduction in RMS error as a function of the number of iterations performed by the optimizer.

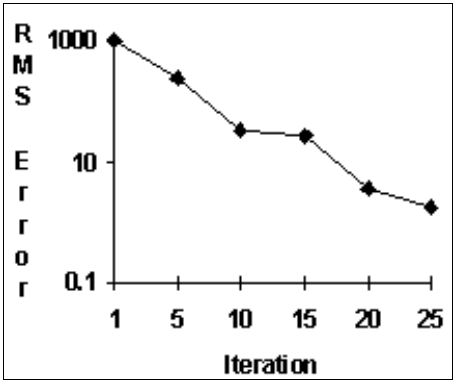


Figure 115 *Normalized RMS Error*

Summary

MicroSim PSpice can be used to solve single and coupled ordinary differential equations by setting up feedback loops around integrator blocks. This is the same technique used by analog computers to solve differential equations.

MicroSim PSpice Optimizer can be used to fit model parameters to measured data in this type of system as well as for more traditional applications such as fitting diode model parameters.

The various files used in the examples shown in this article can be downloaded from our bulletin board system. To do this, select [**T**]ech Support from the main menu, [**6**] File Transfer, [**1**] Download User Requested Files, and select “diffeq.exe” for download. Our BBS number is (714) 830-1550 (14.4k-1200, N-8-1).

References

“Estimating Model Parameters with HiQ,” G. Huvard and E. Eller, Scientific Computing & Automation, February 1995.

Snubbing Resistors

MicroSim Corporation Newsletter, October 1989

In simulating circuits containing inductors, you can run into spurious ringing between the inductors and parasitic capacitances elsewhere in the circuit. Consider, for example, the rectifier bridge circuit shown in Figure 116.

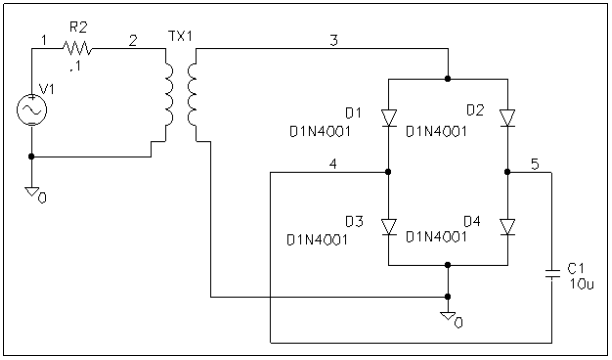


Figure 116 *Bridge schematic*

```
D_D1 3 4 D1N4001
D_D2 3 5 D1N4001
D_D3 4 0 D1N4001
D_D4 5 0 D1N4001
V_V1 1 0
+SIN 0 120 60 0 0 0
R_R2 1 2 .1
C_C1 5 4 10u
K_TX1 L1_TX1 L2_TX1 .99
L1_TX1 2 0 100mH
L2_TX1 3 0 100mH
```

The parasitic capacitance of the bridge can ring against the inductor. During the transient simulation, this shows up in PSpice taking unnecessarily small internal time steps and, of course, in ringing on the resulting waveforms. Realistically, an inductor as large as one hundred millihenrys cannot actually support high frequency oscillations. Its Q falls off well before that.

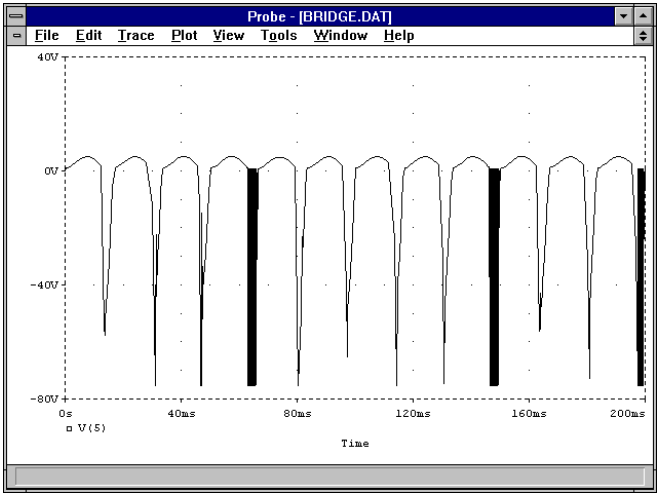


Figure 117 *Probe output*

The solution to modeling this correctly is to add a snubbing resistor across the inductor. The modified rectifier circuit is shown in Figure 118.

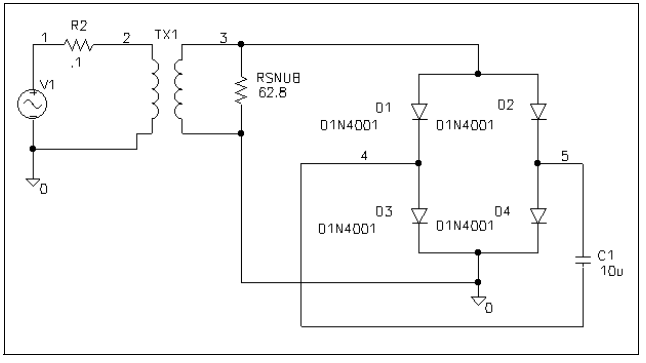


Figure 118 *Bridge with Snubbing Resistor*

```
D_D1 3 4 D1N4001
D_D2 3 5 D1N4001
D_D3 4 0 D1N4001
D_D4 5 0 D1N4001
V_V1 1 0
+SIN 0 120 60 0 0 0
R_R2 1 2 .1
C_C1 5 4 10u
K_TX1 L1_TX1 L2_TX1 .99
L1_TX1 2 0 100mH
L2_TX1 3 0 100mH
R_RSNUB 3 0 62.8
```

The value of RSNUB is chosen to match the impedance of the inductor at the corner frequency at which the Q is to begin falling off. In this example, we chose the roll-off frequency of the inductor to be 100 kHz giving an impedance of

$$2\pi \cdot f \cdot L = 6.28 \cdot 100 \cdot 100e\text{-}3 = 62.8$$

At low frequencies (like 60 Hz), the impedance of LSECONDARY is low and RSNUB has little effect on the circuit's behavior. At higher frequencies, RSNUB shunts LSECONDARY and prevents it from supporting the ringing. The action of RSNUB parallels the physical mechanisms (primarily eddy current losses) which limit the frequency response of an inductor.

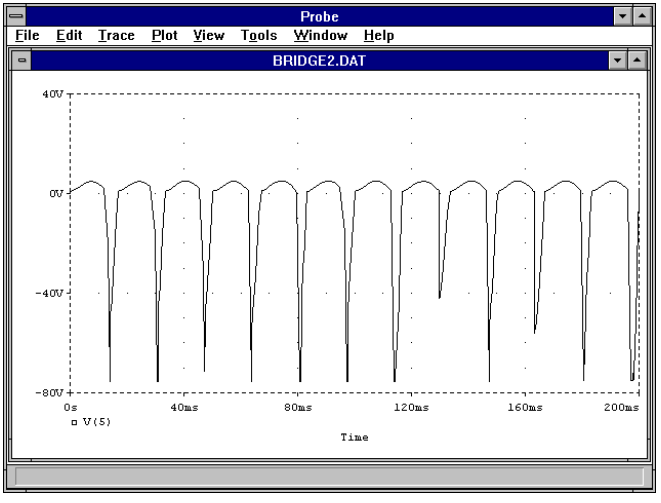


Figure 119 *With Snubbing Resistor*

A fringe benefit of using snubbing resistors is that they limit spikes on inductors. If components in series with an inductor switch off while current is still flowing in the inductor, the *d*i/*d*t can be very high. The basic inductors in PSpice have nothing which limits the size of such spikes and, if large enough, they may cause convergence problems. A snubbing resistor will keep such spikes to a large but tractable size and, thereby, eliminate such convergence problems.

Temperature Effects on Monte Carlo Analysis

The Design Center Source newsletter, July 1993

The MicroSim PSpice user's guide states the following:

"The statistical analyses perform multiple runs, as does the .TEMP command. Conceptually, the .MC [Monte Carlo] or .WCASE [sensitivity/worst-case] loops are inside the .TEMP loop. However, since both temperature and tolerances affect the model parameters, one quickly gets into some rather detailed questions of how the two interact. Therefore, we recommend not using .TEMP when using .MC or .WCASE in the same circuit."

The purpose of this article is to elucidate the interaction of statistical—Monte Carlo or worst-case—and temperature analyses. Monte Carlo and temperature analyses are discussed in the ensuing examples. The same principles apply to the interaction of worst-case and temperature analyses.

To illustrate how the two interact, consider the circuit file setups in Figure 120-and Figure 122. Because the MOSFET parameter, KP, is the only parameter that is affected by both temperature and Monte Carlo, we will limit this discussion to how Monte Carlo and temperature affect this parameter.

```
SIMPLE MOSFET CIRCUIT (NMOS)
*
VGG   2 0 DC 3.0
VDD   1 0 DC 10.0
M1    1 2 0 0 MOSMOD
.MODEL MOSMOD NMOS (KP=2M, DEV=10%, GAMMA=0, VTO=2)
.DC LIN VDD 0 10 0.1
.OP
.END
```

Figure 120 *Nominal run*

```
SIMPLE MOSFET CIRCUIT (NMOS)   ; TEMP=27, 50, 75, 100
*
VGG   2 0 DC 3.0
VDD   1 0 DC 10.0
M1    1 2 0 0 MOSMOD
.MODEL MOSMOD NMOS (KP=2M, DEV=10%, GAMMA=0, VTO=2)
.DC LIN VDD 0 10 0.1
.TEMP 27 50 75 100
.OP
.END
```

Figure 121 *Temperature run*

```
SIMPLE MOSFET CIRCUIT (NMOS) WITH MONTE CARLO AND TEMP
*
VGG   2 0 DC 3.0
VDD   1 0 DC 10.0
M1    1 2 0 0 MOSMOD
.MODEL MOSMOD NMOS (KP=2M, DEV=10%, GAMMA=0, VTO=2)
.DC LIN VDD 0 10 0.1
.TEMP 27 50 75 100
.MC 3 DC I(VDD) MIN LIST OUTPUT ALL
.OP
.END
```

Figure 122 *Monte Carlo with temperature run*

In the nominal run (Figure 120), the value of KP remains unchanged. In the temperature run (Figure 121), the value of KP changes according to the equation:

$$KP(T) = KP \cdot \left(\frac{T}{TNOM}\right)^{\frac{-3}{2}}$$

The values obtained from the simulation are listed in Table . These values were verified using the equation above.

In the Monte Carlo run (Figure 123), the value of KP is calculated by:

$$KP' = KP \cdot (1 + rand \cdot DEV)$$

where

KP' = Monte Carlo adjusted value,

rand = random number, where -1 ≤ rand ≤ 1,

and

DEV = tolerance assigned to parameter.

KP' is only calculated for the second and third passes. For three passes with a uniform distribution and a 10% tolerance, *rand* assumes the values of -0.8657 for pass 2 and -0.6119 for pass 3. The results for this simulation are listed in Table 5. These results were verified using the equation above.

The Monte Carlo run with temperature (Figure 122) illustrates the effect temperature changes have on the Monte Carlo passes. At 27°C, the results should match those obtained from the Monte Carlo run without temperature changes (the previous run based on Figure 123). At 50°C, 75°C, and 100°C, the nominal Monte Carlo results should match the results obtained in the run where only temperature was varied (compare to Table). The results of this analysis are summarized in Table . Upon inspection of the analysis results, it can be seen that the Monte Carlo adjustment is made first, followed by compensation for temperature. This may seem odd at first, but when considering the real device, this makes sense. Consider three MOSFETs that are of the same type. The transconductance of the three will vary at the same temperature, but the temperature dependence of the transconductance is approximately the same for all three devices.

Table 5 *KP Values as Temperature Varies*

Temperature (°C)	KP_{TEMP} (*10^{-3})
27	2.00000
50	1.79032
75	1.60099
100	1.44282

Table 6 *KP Values for successive Monte Carlo Runs*

Monte Carlo Pass	KP_{MC} (*10^{-3})
1 (nominal)	2.00000
2	1.82686
3	1.87762

Table 7 *KP Value Comparison for Monte Carlo with and without Temperature Effect*

Pass	KP_{MC} ($*10^{-3}$)	$KP_{MC/TEMP}$ ($*10^{-3}$) at 27°×C
1 (nominal)	2.00000	2.00000
2	1.82686	1.82686
3	1.87762	1.87762
Pass	**KP_{MC} ($*10^{-3}$)**	**$KP_{MC/TEMP}$ ($*10^{-3}$) at 50°×C**
1 (nominal)	2.00000	1.79032
2	1.82686	1.63533
3	1.87762	1.68077
Pass	**KP_{MC} ($*10^{-3}$)**	**$KP_{MC/TEMP}$ ($*10^{-3}$) at 75°×C**
1 (nominal)	2.00000	1.60099
2	1.82686	1.46239
3	1.87762	1.50302
Pass	**KP_{MC} ($*10^{-3}$)**	**$KP_{MC/TEMP}$ ($*10^{-3}$)at 100°×C**
1 (nominal)	2.00000	1.44282
2	1.82686	1.31792
3	1.87762	1.35453

When using temperature with Monte Carlo, PSpice first adjusts the transconductance to account for geometry, doping, and other process deviations (device tolerance). PSpice then adjusts the temperature. This is illustrated in Figure 123. One may question the accuracy of this interplay between the two analyses. At first, it appears that the overall parameter adjustment may be greater than the defined tolerance, but it can be shown that this is not the case. Consider the percentage of nominal of KP for the Monte Carlo only runs. This is found by:

$$\%\,\text{of nominal} = \left[\frac{KP_{MC}}{KP}\right] \cdot 100$$

This results in:

Pass 2 = 91.343% of nominal
Pass 3 = 93.881% of nominal

Now consider the percentage of nominal at temperature. The same analysis yields the same percentage of nominal for all Monte Carlo with temperature runs. To conclude, using temperature with Monte Carlo (or worst-case) is valid, and the interaction is straightforward and predictable.

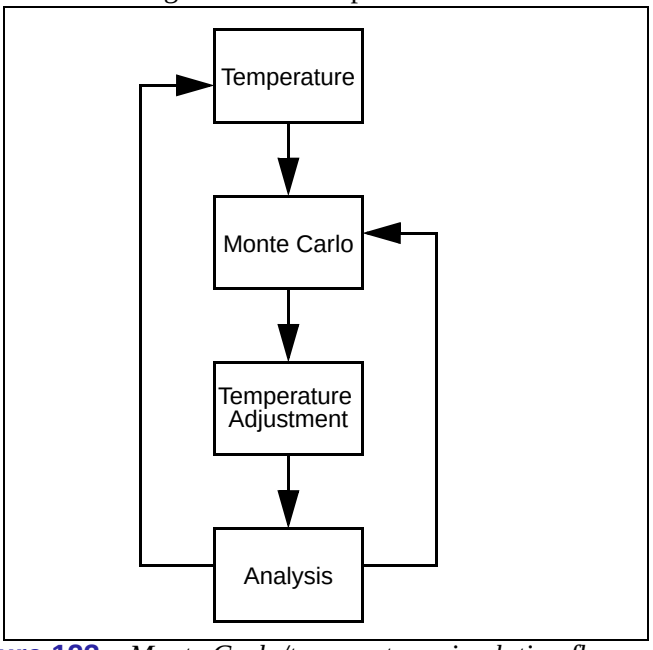


Figure 123 *Monte Carlo/temperature simulation flow*

Acknowledgment: We would like to thank the engineers at Texas Instruments for providing the material presented in this article.

Test Analog Circuits with Random Digital Data

by Mike Wyatt, Honeywell

An efficient method for analyzing analog circuits with digital data is to import random bit patterns into a MicroSim™ PSpice® simulation using a file-based piecewise linear (PWL) independent voltage or current source. This verification technique is effective for mixed analog/digital circuits used, for example, in high-speed logic, digital RF, and MODEM systems. By using software programs to generate the random data, it is easy to create comprehensive test vector sets which fully exercise the circuit.

This article describes several BASIC software programs which are used to generate MicroSim PSpice-compatible random digital test data for bipolar, unipolar, TTL, CMOS, ECL, and CML technologies. For demonstration, a portion of an RF MODEM type ASIC is modeled and simulated using input data created by the bipolar-compatible random digital data generator.

Generating Random Digital Data Files

Figure 124 lists a BASIC program (“BDATA.BAS”) that produces a bipolar-compatible random digital data file (“BDATA1.PWL”) for use with the MicroSim PSpice file-based PWL voltage source. The data is generated in a time-amplitude sequence comprised of NPOINTS of random bipolar bits limited in amplitude to ±1 and separated by one second. (During simulation, time and amplitude values can be scaled as described in the next section.) RISE controls the bit’s rise and fall times, with a default setting of 5% (0.05) of the bit period. The bit patterns are random and different each time the program is run. This occurs because the random number generator used

to generate the bit pattern samples the computer's real-time clock for a seed.

A comparable program which produces data files compatible with unipolar (0,1) technology (call it "UDATA.BAS") can be derived from the "BDATA.BAS" program with only simple changes. The two PRINT statements:

```
PRINT #1, USING "##.#"; -1
```

change to

```
PRINT #1, USING "##.#"; 0
```

Also, the two statements referencing "BDATA1.PWL" should be changed to "UDATA1.PWL."

Figure 125 lists a BASIC program ("TTLDATA.BAS") which produces a digital data file that is compatible with TTL (0.8,2) technology. The output file is named "TTLDATA1.PWL."

```
* PWL BIPOLAR-COMPATIBLE DATA SOURCE GENERATOR.
*     NPOINTS ARE CREATED.
*     BIT PATTERN IS RANDOM AND SCALES -1 TO 1
*     BIT PERIOD IS 1 SECOND WITH 5% RISE AND FALL TIME

DEFSNG A-H, O-Z
CLS
PRINT "      SPICE DATA PATTERN GENERATION PROGRAM"
PRINT "              CREATES BDATA#.PWL"
PRINT
PRINT
PRINT

RISE = .05
NPOINTS = 1000
TSPAN = 1
SFACT = 1
RANDOMIZE TIMER

OPEN "BDATA1.PWL" FOR OUTPUT AS #1
PRINT #1, "* TIME,VOLTAGE "
PRINT #1, USING "##.####^^^^"; 0;
PRINT #1, USING " ##.#"; -1
PRINT #1, USING "##.####^^^^"; SFACT;
PRINT #1, USING " ##.#"; -1
PRINT #1, " "

N = 1
FOR I = 1 TO NPOINTS
  RANDG(N) = RND
  IF RANDG(N) > .5 THEN RANDG(N) = 1 ELSE RANDG(N) = -1
  IF N < 2 AND I < NPOINTS THEN 100
  FOR J = 1 TO N
    PRINT #1, USING "##.####^^^^"; (I - N + J + RISE) * SFACT;
    PRINT #1, USING " ##.#"; RANDG(J)
    PRINT #1, USING "##.####^^^^"; (I - N + J + 1) * SFACT;
    PRINT #1, USING " ##.#"; RANDG(J)
  NEXT J
  PRINT #1, " "
  N = 0
100 N = N + 1
NEXT I

PRINT #1, "* END OF DATA FILE"
PRINT "     FINISHED CREATING BDATA1.PWL FILE
END
```

Figure 124 *Basic program used to generate random digital data for bipolar technology. A comparable Basic program for unipolar technology can be derived from this file with minimal changes (see article).*

```
*PWL TTL-COMPATIBLE DATA SOURCE GENERATOR.
*     NPOINTS ARE CREATED.
*     BIT PATTERN IS RANDOM.
*     BIT PERIOD IS 1 SECOND WITH 5% RISE AND FALL TIME.

DEFSNG A-H, O-Z
CLS
PRINT "      SPICE DATA PATTERN GENERATION PROGRAM"
PRINT "              CREATES TTLDATA#.PWL"
PRINT
PRINT
PRINT

RISE = .05
TTLHIGH = 2
TTLLOW = .8
NPOINTS = 1000
TSPAN = 1
SFACT = 1
RANDOMIZE TIMER

OPEN "TTLDATA1.PWL" FOR OUTPUT AS #1
PRINT #1, "* TIME,VOLTAGE "
PRINT #1, USING "##.####^^^^"; 0;
PRINT #1, USING " ##.#"; TTLLOW
PRINT #1, USING "##.####^^^^"; SFACT;
PRINT #1, USING " ##.#"; TTLLOW
PRINT #1, " "

N = 1
FOR I = 1 TO NPOINTS
  RANDG(N) = RND
  IF RANDG(N) > .5 THEN RANDG(N) = TTLHIGH ELSE RANDG(N) = TTLLOW
  IF N < 2 AND I < NPOINTS THEN 100
  FOR J = 1 TO N
    PRINT #1, USING "##.####^^^^"; (I - N + J + RISE) * SFACT;
    PRINT #1, USING " ##.#"; RANDG(J)
    PRINT #1, USING "##.####^^^^"; (I - N + J + 1) * SFACT;
    PRINT #1, USING " ##.#"; RANDG(J)
  NEXT J
  PRINT #1, " "
  N = 0
100 N = N + 1
NEXT I

PRINT #1, "* END OF DATA FILE"
PRINT "     FINISHED CREATING TTLDATA1.PWL FILE"
END
```

Figure 125 *Basic program used to generate random digital data for TTL technology. With simple changes, this program can also be used to generate data files that are ECL- and CML-compatible (see article).*

Digital data files compatible with ECL technology can be created by altering the TTLHIGH and TTLLOW variables in the "TTLDATA.BAS" program. To create files for CML technology, these same variables can be altered to minimum and maximum logic current levels. In the latter case, you must use the file-based PWL current source rather than the usual voltage source used in the previous cases.

Tying Digital Data Files to PWL Sources

Within a schematic, place a VPWL_file symbol instance to define the voltage source to be used with a bipolar, unipolar, TTL, CMOS, or ECL-compatible digital data file; use IPWL_file for a current source to be used with a CML-compatible data file. Double-click on the symbol instance to bring up the Edit/Attributes dialog, and assign the name of the digital data file to the FILE attribute, time-scale-factor to the TSF attribute, and value-scale-factor to the VSF attribute. Note that the file name is displayed on the schematic.

Within a circuit file, specify a file-based PWL voltage or current source using the syntax:

<V | I><*name*>
+ <+ *node*> <-*node*>
+ [*DC input*] [*AC input*]
+ PWL FILE <*file name*>
+ TIME_SCALE_FACTOR=<*value*>
+ VALUE_SCALE_FACTOR=<*value*>

where V<*name*> is used for a voltage source and I<*name*> is used for a current source. See the ***MicroSim PSpice*** *Circuit Analysis Reference Manual* for details.

The time-value pairs contained in the data file can be scaled during simulation to suit your specific application; simply assign values to the time-scale-factor and value-scale-factor parameters described above. This method is also used to derive CMOS-compatible digital test data from unipolar data files by setting the value-scale-factor to the same value as the Vdd supply.

Example: Transient Analysis of an RF MODEM

Figure 126 shows the hierarchical schematic for the data amplifier and comparator portion, U1, of a custom RF MODEM type ASIC. U1 is subjected to a random digital test data signal from Vdata. Vdata is a MicroSim PSpice VPWL_file voltage source. The FILE attribute for Vdata is set to the name of the bipolar-compatible digital data file, “BDATA1.PWL” (generated by “BDATA.BAS”).

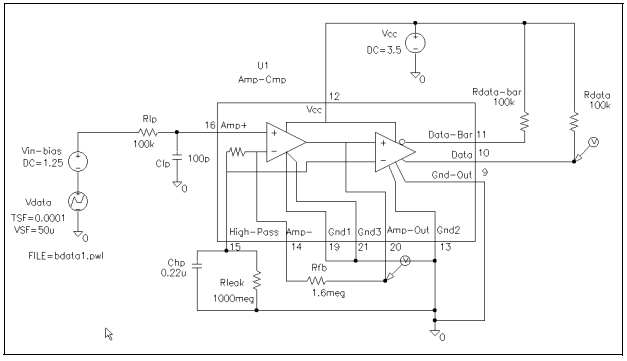


Figure 126 *(left) Test circuit for the data amplifier and comparator (U1) portion of an RF MODEM design.*

The VSF (voltage-scale-factor) attribute for Vdata is set to 50 uV—the smallest amplitude to which the circuit must respond. The TSF (time-scale-factor) attribute for Vdata is set to 0.0001—the period (10 kbaud) of the desired random data stream.

Rlp and Clp form a low pass filter to limit the bandwidth of the incoming signal and Chp sets the circuits’ low frequency response. Rleak is included to model Chp’s leakage effects.

Plots of U1’s comparator and amplifier outputs for different values of Chp are shown in Figure 127. Notice the missing bit decisions in U1’s comparator output when Chp is too small.

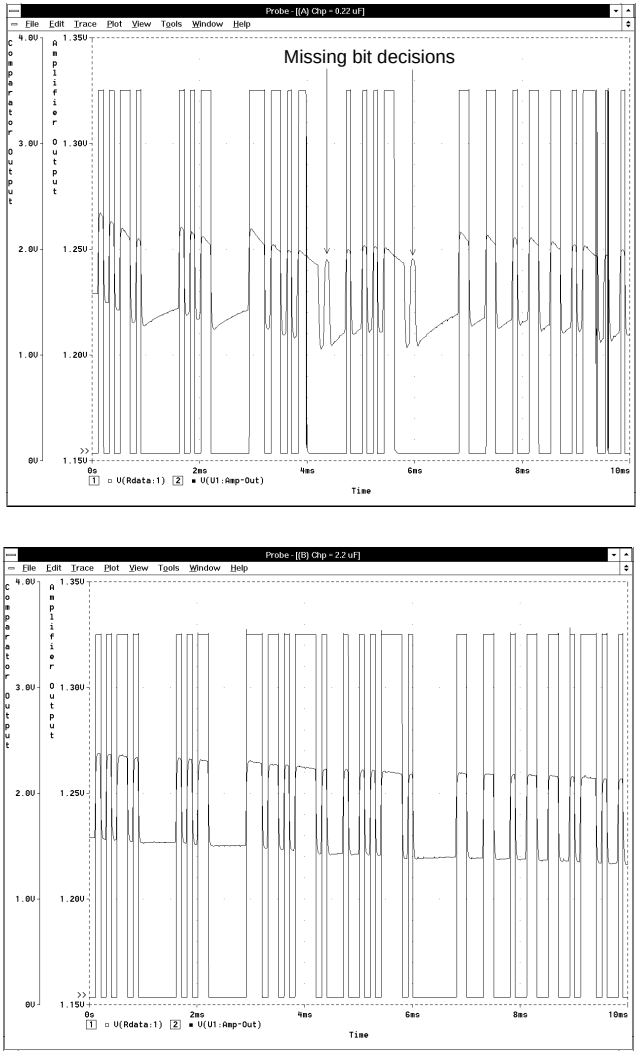


Figure 127 *Simulation results for Chp values of 0.22 uF (above) and 2.2 uF (below). Chp values that are too small result in missing bit decisions in the U1 comparator.*

To Download Files from the BBS

The uncompiled versions of the BASIC programs shown in this article, as well as that for the unipolar (0,1) technology—"BDATA.BAS" (bipolar), "UDATA.BAS" (unipolar), and TTLDATA.BAS" (TTL)—are available on the MicroSim bulletin board in one self-extracting file named "pwlgen.exe." To download this file, select [**T**]ech Support from the main menu, [**6**] File Transfer, [**1**] Download User Requested Files, and select "pwlgen.exe" for download. The BBS number in the U.S. is (714) 830-1550 (14.4k-1200, N-8-1).

Biography: Michael A. Wyatt is a senior engineering fellow with Honeywell, specializing in analog circuits, systems, RF, and ASIC designs. He is frequently published in the "design ideas" section of *EDN* and *Electronic Design*, and can be reached by voice at (813) 539-5653 or FAX at (813) 539-2558.

Use Constrained Optimization to Improve Circuit Performance

Many applications of optimization to electronic design are naturally expressed as “minimize an objective while meeting a set of requirements.” For example, the objective might be power consumption; the requirements might be minimum allowable gain and 3 dB bandwidth. In order to solve this type of problem, a constrained optimization algorithm is required. When the constraints are nonlinear functions of the parameters, the problem is one of optimization with nonlinear constraints.

Paragon, the most recent addition to MicroSim’s Design Center family, is designed to tackle this type of problem. It will handle both constrained and unconstrained problems, with either a single objective function or the sum of squares of a set of objective functions being minimized. Special techniques are required to solve constrained optimization problems. Attempting to use unconstrained optimizers is inaccurate (the result may not be the constrained optimum) and inefficient (many wasted simulations are required to solve unconstrained subproblems).

The Example Circuit

The schematic shown in Figure 128 is an idealized CMOS amplifier cell. The amplifier consists of a common source stage (M1) with active load (M3 and M4), and a source follower (M2).

For this example, we would like to reduce the power consumption of the design. The improved design must still meet its specifications—in this case, gain of 20 and a 3 dB bandwidth of at least 1 MHz. The initial design consumes 2.2 mW. It has both excess gain (23.8) and bandwidth (2.2 MHz), so a reduction in power consumption appears feasible.

In this design, power, gain, and bandwidth depend nonlinearly on circuit parameters such as transistor dimensions. This makes it impractical to optimize the design analytically.

Selecting Parameters for Optimization

The first step in the optimization process is to identify parameters in the design which may be varied by the optimizer. In this case, three circuit values have been parameterized. These are the length and width of MOSFET M1 (L1 and W1), and the bias current for the active load (Iref).

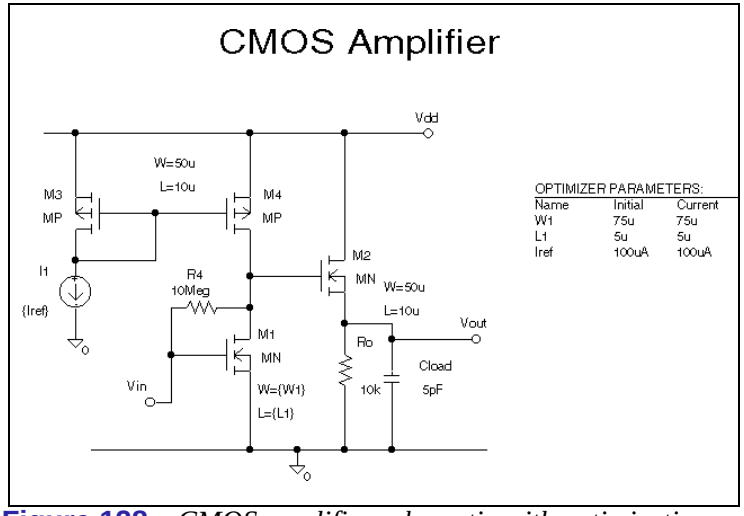


Figure 128 *CMOS amplifier schematic with optimization design parameters. Parameters W1 and L1 are used in expressions assigned to the width and length attributes, respectively, of MOSFET M1. Parameter Iref is the bias current value for the active load.*

Initial values for the parameters are visible in the OPTPARAM symbol on the schematic (entitled "OPTIMIZER PARAMETERS:"). Paragon enforces upper and lower limits on the parameters; so for each parameter, you need to determine its starting value and bounds. For instance, W1 has been defined

with an initial value of 75 µ and range of 10 µ-150 µ (see Figure 129).

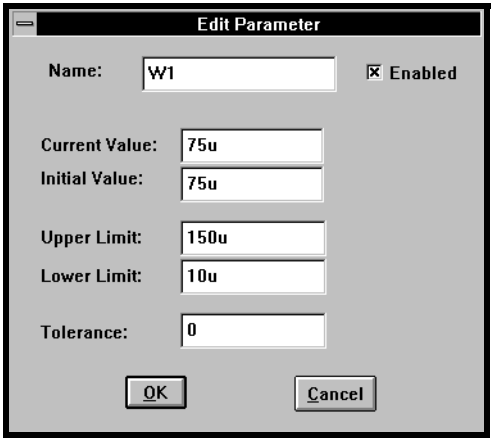


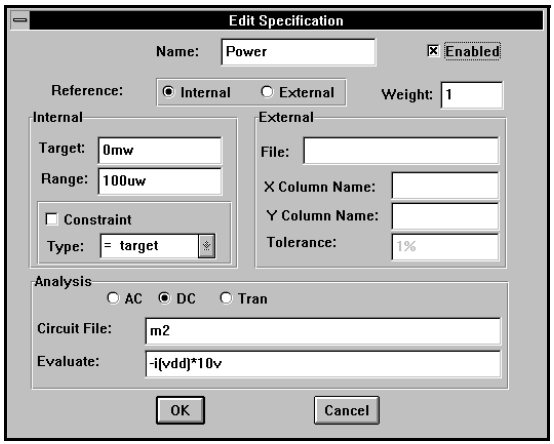
Figure 129 *Design parameters can be defined from within Schematics by specifying the attributes of an OPTPARAM symbol instance (see Figure 128), or from within Paragon using the Edit Parameter dialog (shown here). Either method allows for the specification of the parameter's name, initial value, valid range, and tolerance.*

Identifying Goals and Constraints

The example has a single goal—minimize power consumption (see Figure 130). This can be measured by performing a 1-point DC analysis in PSpice followed by using Probe to compute the power according to the expression

```
-I(Vdd) * 10V
```

“Goal” Specification



“Constraint” Specification

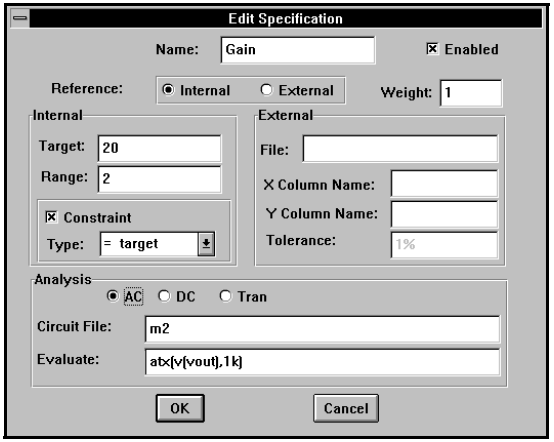


Figure 130 *Performance specifications are divided into “goals” and “constraints” which are defined as shown in these dialogs. Goals define the performance objectives, e.g., minimize power consumption. Constraints define the conditions to which the circuit’s performance should adhere while attempting to meet the objective, e.g., gain should measure between 19 and 21. The means for measuring these performance characteristics must also be defined, i.e., a Probe trace or Probe goal function applied to the results of a specified analysis type.*

This example has two constraints: gain and 3 dB bandwidth (see Figure 130). These can both be measured by performing an AC analysis. The gain is then determined by measuring the spot gain at 1 kHz (assuming that the bandwidth is much greater than this). The 3 dB bandwidth is measured by finding the frequency where the output has fallen by 3 dB from its low-frequency value. Two Probe goal functions are used to make the measurements:

```
AtX(V(Vout), 1k)
LPBW(Vdb(Vout), 3)
```

Their definitions are shown in Figure 131.

```
* Value at Given X
AtX(1,where) = y1
{ 1 | sf xvalue(where) !1; }

* Bandwidth of Lowpass Response
LPBW(1,dblev) = y1
{ 1 | sf level(max-dblev) !1; }
```

Figure 131 *Probe goal functions used to measure performance can be formulated and tested using Probe's Performance Analysis feature.*

Specifying the Optimization Type

In this example, we have a single objective—power. It is a positive number whose value is to be minimized. We set this up within Paragon by:

1. Selecting Options/Defaults,
2. Clicking on Advanced Options,
3. Clicking on the Minimize button in the One Goal box (see Figure 132). (The default, Least Squares, would be appropriate if it were required to minimize the square of some function.)

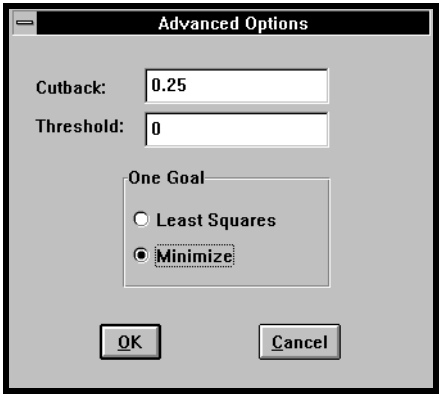


Figure 132 *When the problem is to optimize for a single goal, the reduction algorithm must be specified. "Least-squares" minimizes the square of the deviation between the performance measure and the target value. "Minimization" attempts to reduce the performance measure.*

Setting Up the Analyses

The goals and constraints in this example require both a DC and an AC analysis. We set up a 1-point DC sweep of the supply voltage at its nominal value of 10 V. We also set up an AC analysis to do a log sweep from 1 kHz to 10 MHz.

Performing the Optimization

To start the optimization process, we select Tune/Auto/Start within Paragon. Paragon determines the current performance of the circuit using the initial parameter values. It then determines the partial derivative of each goal and constraint with respect to each parameter using a finite difference technique. (You can inspect the derivatives by selecting Tune/Update Derivative**s** followed by Tune/Show Derivatives.)

Paragon then enters an iterative loop where it computes trial values for the parameters, simulates the circuit using these

values, and measures the performance. New derivatives are then determined, and the process repeats until either the specifications are met, the user interrupts the process, or no satisfactory progress can be made.

As the optimization proceeds, Paragon displays the values for all of the goals and constraints; a small ‘thermometer’ indicates whether each one is within the specified range. A larger indicator shows the combined progress on the goal(s).

In this example, the optimization is run for 15 iterations (see Figure 133). At this point, the power consumption has been reduced to 852 μW. The gain is 20 and the bandwidth has been driven down to its lower limit of 1 MHz (both constraints are said to be active).

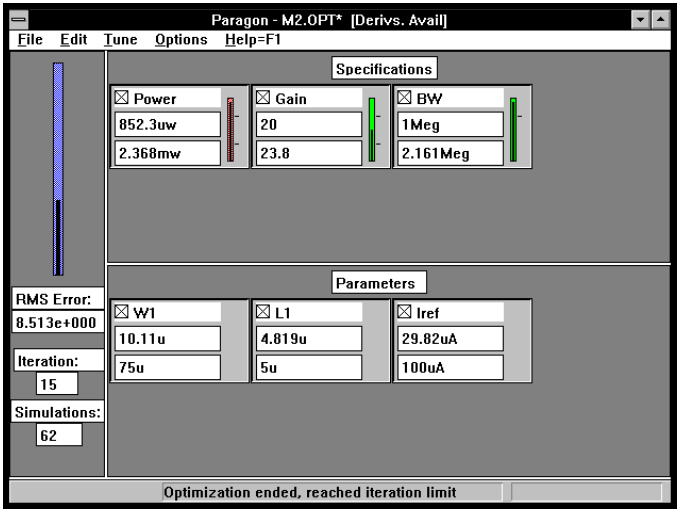


Figure 133 *As optimization proceeds, Paragon displays the working values for performance measures and design parameters. ‘Thermometer’ indicators show how closely target specifications are met; when within range, the specification indicators turn from red to green. In this case, power has been reduced by a factor of 2.5 while adhering to gain and bandwidth (BW) constraints.*

Tradeoffs in the Design

A set of numbers called Lagrange Multipliers are associated with the result of a constrained optimization. (After running the optimization, the Lagrange Multipliers can be viewed by selecting File/Report within Paragon.) These correspond to the incremental *cost* of perturbing each of the active constraints at the optimum. In this example there are two constraints, so there are two Lagrange Multipliers. The first of these (-2.2×10^{-5}) tells us that if the gain specification is increased by 1%, the minimum power will decrease by approximately 4.4 μW. The second (7.5×10^{-10}) tells us that if the bandwidth is increased by 1%, the minimum power will increase by approximately 7.5 μW. These numbers are of considerable interest, as they reveal the tradeoffs which are available in the design at the optimized performance point found.

Summary

Many problems in analog performance optimization naturally involve nonlinear constraints. Paragon has been developed to tackle this type of problem, which is difficult for the unconstrained optimizers typically found in the EDA industry. Paragon can solve optimization problems efficiently and accurately, and reveals much useful information about the tradeoffs present in the design at the optimum point.

Use Ferrite Bead Models to Analyze EMI Suppression

by Steve Hageman Applied DC

The use of ferrite cores can help ensure that high performance systems pass EMI/EMC regulations. Traditionally, ferrite beads were used to make high frequency low pass filters and to keep transistors from oscillating—a quick EMI fix. Though still the preferred design solution in some EMI prevention cases, ferrite beads are not the quick fix they once were.

With today's high performance systems, it is now important to consider the bead's real properties and how to analyze the resulting bead circuit. The models presented here allow inclusion of two of the more popular ferrite EMI bead materials into your PSpice simulations. These are the Fair-Rite #43 and #73 materials [1].

What To Model?

The most important factors in ferrite bead-based design are:

- Impedance vs. bead size
- Impedance vs. frequency
- Impedance vs. DC current bias
- Impedance vs. number of turns on the bead

By modeling these parameters, a wide range of practical circuits can be realistically simulated.

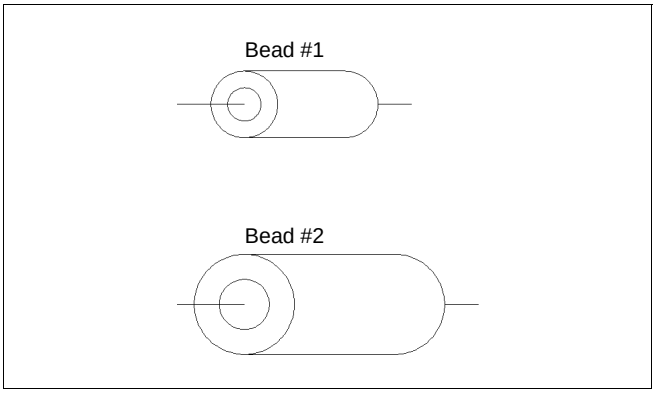
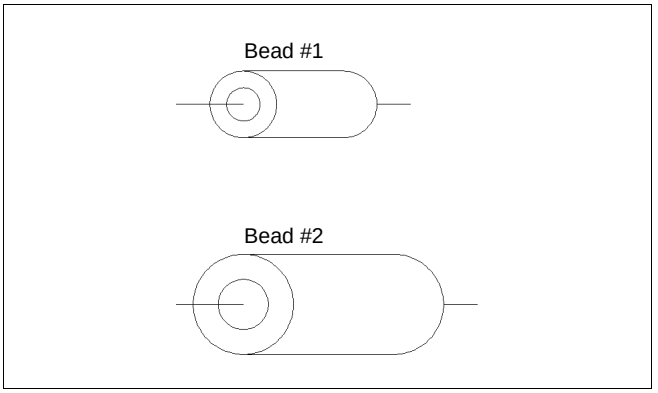
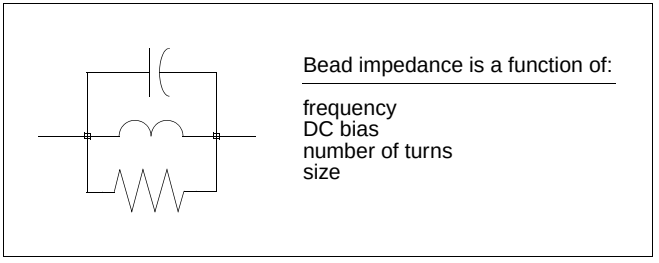


Figure 134 *After eight iterations, Paragon shows a 40% reduction in silicon area occupied by the switches, while maintaining the differential and integral nonlinearities within the specified constraints.*

Theory states that the impedance of a bead depends on its constituent material and the dimensions of the bead. For example, it is intuitive that bead #2 in Figure 135 on page -242 has more inductance than bead #1. The increase in inductance or impedance can be related to the physical core properties by the empirical equation





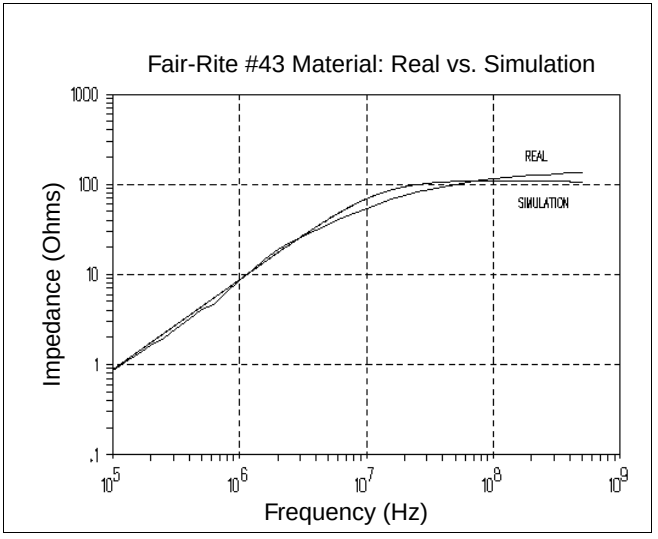


Figure 135 *Ferrite beads come in all shapes and sizes. While it may seem intuitive that bead #2 has greater inductance than bead #1, just how much more inductive is it? This question and many more can be answered by using the PSpice ferrite bead models .*

$$K = (N)^2 \times \left(0.417 + 3.41 \times L \times \log\left(\frac{OD}{ID}\right)\right) \quad (1)$$

where, *K* is the impedance scaling factor, *L* is the bead length, *OD* is the outside diameter of the bead, and *ID* is the inside diameter of the bead (all in inches). *N* is the number of turns of wire on the bead. A wire passing straight through the center of the core counts as one turn; hence the minimum number of turns on a bead is 1.

The constants were derived by measurement of actual core parameters* with an HP4195A Impedance Analyzer and using curve fitting software on a PC to get a good fit to the measurements. These measurements were made correctly, based on actual performance. Note that some manufacturers' curves are generated by the *theoretical physics* of the material; in these cases, the measurements may not entirely correlate with manufacturers' values.

The bead's electrical properties can be modeled by a simple parallel L-R-C circuit (see Figure 136). By properly picking the L, R, and C components, a good fit can be made to the actual bead's performance (see Figure 137). These are generally the only components that need to be changed in the model to simulate different bead materials.

*. A bead measuring 0.43" x 0.2" x 0.08" (Len, OD, ID) was used as the *standard bead* to which all others were scaled.

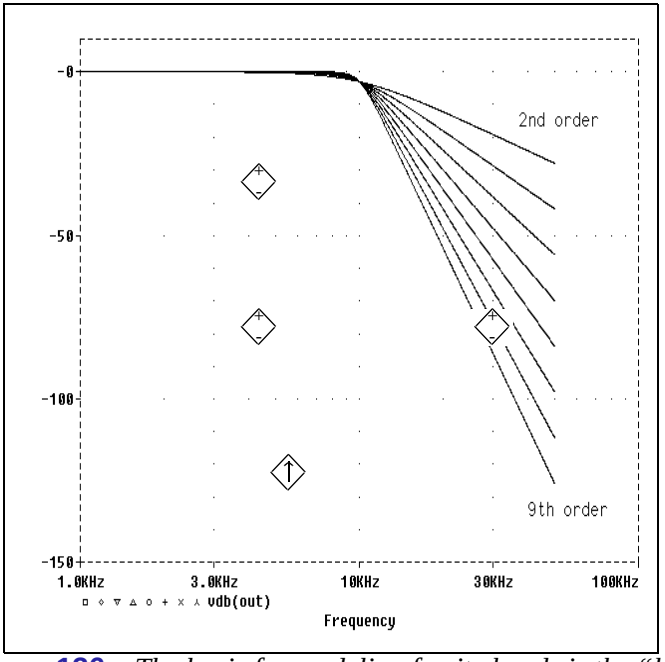


Figure 136 *The basis for modeling ferrite beads is the "low Q" parallel R-L-C circuit. A single R-L-C network is modified by adding the effects of DC bias, size, and number of turns in the models. This allows one model to be used in any situation.*

```
; Complete model of Fair-Rite's #43 EMI bead material
; The model is used as follows:
;
; X1 IN OUT BEAD_43 PARAMS:ID=0.1, OD=0.3, L=0.3, N=1
;
; where, IN and OUT are the nodes that the bead is connected
; to. BEAD_43 is the subcircuit name and the parameters: ID, OD,
; and L are the dimensions of the bead in inches. N is the number
; of turns on the bead.
;

.SUBCKT BEAD_43 Z1 Z2 PARAMS:ID=0.08, OD=0.2, L=0.43, N=1

* * Impedance multiplier * *
EOUT          Z1      10      VALUE={V(GEOM)*V(BIAS)*V(ZREF)}
VSENSE        10      Z2      DC 0
GCOPY         0       ZREF    VALUE={I(VSENSE)}

* * Impedance correction for bead size and number of turns * *
EGEOM         GEOM    0
+ VALUE={N*N*(0.417+3.41*L*LOG10(OD/ID)) }
RGEOM         GEOM    0  1MEG

* * Impedance correction for DC current bias, turns and diameter * *
EBIAS         BIAS    0       TABLE {I(VSENSE)*(N/(OD/0.2))}=
+ (-14,0.06) (-10.0,0.09) (-8.0,0.12) (-6.0,0.17) (-4.0,0.254) (-2.0,0.463)
+ (-1.0,0.70) (0.0,1.0) (1.0, 0.70) (2.0,0.463) (4.0,0.254) (6.0,0.17)
+ (8.0,0.12) (10.0,0.09) (14,0.06)
RBIAS         BIAS    0       1MEG

* * Model of 43 material for standard bead * *
LBEAD         ZREF    0       1.4U
RBEAD         ZREF    0       110
CBEAD         ZREF    0       {1.0P * N}

.ENDS
```

Figure 137 *The simulated versus actual impedance curves are quite accurate at low DC Bias and with one turn. The accuracy of the model generally degrades with increasing turns and when operated at high DC bias.*

The bead is modeled in PSpice by using a basic impedance multiplier. The electrical schematic of Figure 138 shows the circuit used. (See also the section entitled “Editor’s Note: An Equivalent Schematics Circuit.”) The circuit operates by sensing the AC current flowing through the bead and producing the voltage that relates to the correct impedance for that AC current and frequency. The equation for this is

$$Vout = Z \times Iin \qquad (2)$$

To make this equation applicable to all circuit configurations of a bead, EOUT in Figure 139 takes inputs from the geometry of

the bead, DC bias level, number of turns, and the reference bead impedance to produce the correct equivalent impedance.

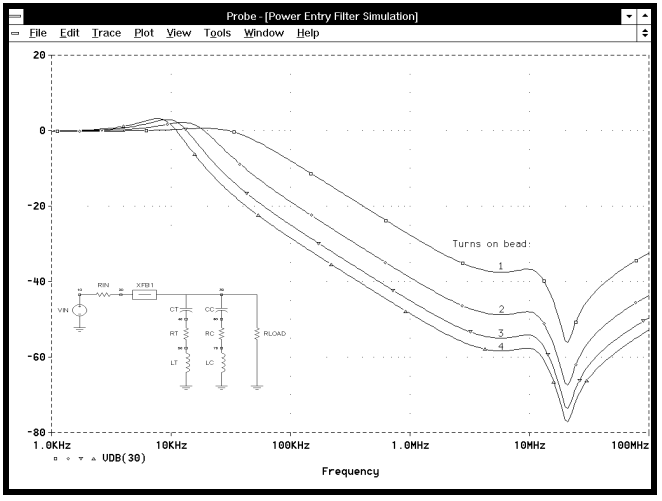


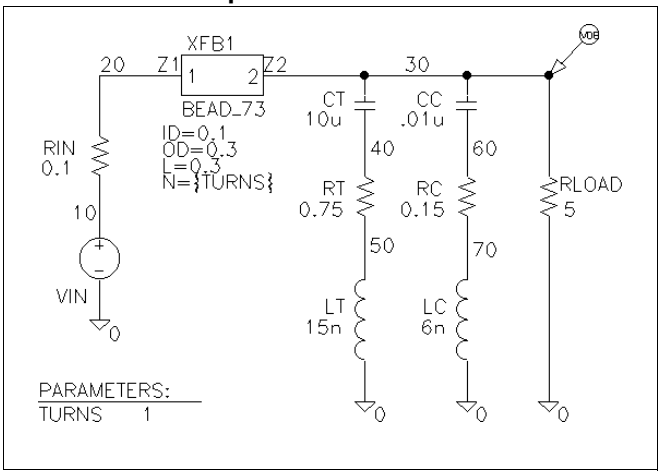
Figure 138 *The simulated versus actual impedance curves are quite accurate at low DC Bias and with one turn. The accuracy of the model generally degrades with increasing turns and when operated at high DC bias.*

```
<< BEAD_73 TEST CIRCUIT >>
.INC BEAD_73.INC

.AC DEC 50 1k 100MEG
.PROBE

.PARAM TURNS=1
.STEP PARAM TURNS LIST 1,2,3,4

VIN 10 0 AC 1 DC 5
RIN 10 20 0.1

XFB1 20 30 BEAD_73 PARAMS: OD=0.3, ID=0.1, L=0.3, N={TURNS}

CT 30 40 10U
RT 40 50 0.75
LT 50 0 15N

CC 30 60 0.01U
RC 60 70 0.15
LC 70 0 6N

RLOAD 30 0 5

.END
```

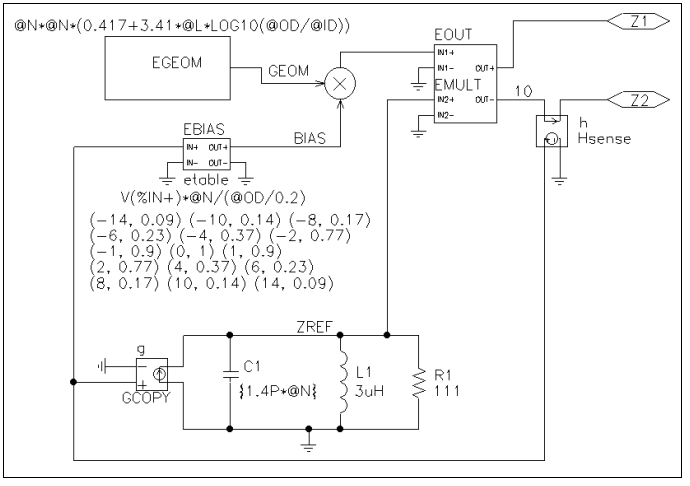
Figure 139 *The electrical circuit of the model is based on an impedance multiplier circuit. The basic impedance of the R-L-C*

The impedance correction for bead size is exactly like equation (1) (EGEOM in Figure 136 and Figure 137). The correction for DC bias does not fit well to a reasonable polynomial curve, so a lookup table is used to reduce the effective bead impedance with DC bias current. This method is usually preferable anyway. A lookup table has very well-defined values for points outside the bounds of the table, whereas a high-order polynomial may yield surprising outputs when inputs are well beyond the data used to fit the polynomial.

Top-Level Test Schematic



Bead_73 Sub-Schematic



The lookup table for DC bias correction is further scaled by a factor that relates to the outside diameter of the bead. The larger the bead diameter, the more DC bias current it can accept before saturation. Conversely, if the number of turns is increased on the bead, the current it can support before saturation is reduced proportionally to *N*.

The last non-ideal correction for the bead is the CBEAD factor that relates to the turns on the bead. The inductance of the bead increases by N^2, but the effect of the extra windings on the core

increases the high frequency capacitance proportionally to *N*. So the bead capacitance is first order corrected by multiplying by the number of turns, *N*.

All of these inputs are multiplied together at EOUT to produce the proper impedance for the bead. The model is valid for any analysis type and has exhibited no numerical instability in any of the dozen or so test circuits tried.

Figure 139 shows the subcircuit listing for the Fair-Rite #43 bead model. The subcircuit listing for the #73 bead model is similar; however, the following statements must be substituted for EBIAS, LBEAD, RBEAD, and CBEAD:

```
EBIAS BIAS 0 TABLE {I(VSENSE) * N / (OD/0.2)} =
+ (-14,0.09)(-10.0,0.14)(-8.0,0.17)(-6.0,0.23)(-4.0,0.37)
+ (-2.0,0.77) (-1.0, 0.9) (0.0,1.0) (1.0,0.9) (2.0,0.77)
+ (4.0,0.37) (6.0,0.23) (8.0,0.17) (10.0,0.14) (14,0.09)

LBEADZREF    0  3U
RBEADZREF    0  111
CBEADZREF    0  {1.4P * N}
```

Using the Bead Models

Beads are used in many situations today to control EMI. One of the more useful configurations is to provide power supply decoupling on a per board basis. A typical circuit is shown inset in Figure 6 on page 248. This is a standard circuit that may be used on a digital or analog PCB power entry. The bead used is a 0.3 OD x 0.1 ID x 0.3 L #73 core; the capacitors are modeled for series resistance and inductance as described in reference [2].

Figure 5 on page 248 shows the circuit file used to run the simulation. The resulting filter response is plotted versus the number of turns on the bead (Figure 4 on page 248). An interesting thing happens with this circuit. As the number of turns is increased, the filter attenuation increases at high frequencies as expected; but at 10 kHz, the Q of the first pole also increases. This results in more peaking in the filter response which may or may not cause a problem for the circuit. It is, however, nice to know the full ramifications of increasing turns on the bead.

It can also be seen that increasing the turns on the bead above two does not significantly increase the filter rejection. This is due to the effects of the DC bias increasing proportionally to *N*. The core can only support so many *amp-turns* before saturation—hence, the *diminishing returns* of attenuation as the turns are increased above a certain point.

What the Models Don't Do

As with all behavioral models, some caution should be used interrupting the results of the simulation. While the trends are always in the right direction, the absolute simulation numbers are subject to errors of up to 50% for worst-case bead/circuit configurations. The models are most accurate for the one turn, low DC bias case. The accuracy degrades as more turns are added or when the core is operated with high DC bias.

This is not too serious when you consider that the beads themselves may be subject to 40% variation between samples. The models are most useful for finding what to optimize, i.e., turns on the bead, bead size, external circuit elements, etc. If you are dealing with critical parts of your circuit, always trust careful measurements on your actual prototype.

Editor's Note: An Equivalent Schematics Circuit

Figure 6 shows how the bead model depicted in Figure 139 can be drawn in Schematics for Fair-Rite's #73 bead. Notice that the parameters passed to the original subcircuit model of Figure 5 are implemented in the top-level schematic as attributes of the hierarchical Bead_73 symbol. Each occurrence of one of these parameters in Bead_73's sub-schematic is preceded by an '@' character to indicate parameter *value* substitution.

To Download Files from the BBS

The Fair-Rite #43 and #73 bead models presented in this article can be downloaded from the MicroSim bulletin board in one self-extracting zip file named “beads.exe.” This file also contains the equivalent schematics and symbol files for use in Schematics-based Design Center systems. To download the self-extracting file, select [**T**]ech Support from the main menu, [**6**] File Transfer, [**1**] Download User Requested Files, and download “beads.exe.” The BBS number in the U.S. is (714) 830-1550 (14.4k-1200, N-8-1).

References Cited

Fair-Rite Soft Ferrites, 11th edition catalog, POB J, One Commercial Row, Wallkill, NY 12589

Hageman, S. “Improve Simulation Accuracy When Using Passive Components,” *The Design Center Source*, April 1994, MicroSim Corporation, Irvine, CA.

Acknowledgments: Special thanks go to William Kimmel of Kimmel Gerke Associates for laying the groundwork for modeling EMI ferrites in his article, “Wide Frequency Impedance Modeling of EMI Ferrites,” published in the *IEEE 1994 Symposium on EMC*. Mr. Kimmel is a well known EMI/EMC consultant and lecturer. He may be reached at 1544 North Pascal, St. Paul, Minnesota, 55108, (612) 330-3728.

Biography: Steve Hageman is an analog designer specializing in power conversion. He owns the consulting firm, Applied DC, and may be reached by voice/FAX at (510) 687-0483.

Using the Inductor Coupling Symbols

The new inductor coupling symbols may be used to couple up to six independent inductors on a schematic. The Symbol Library file, “magnetic.slb”, contains one symbol for each nonlinear magnetic core model in the model library file “magnetic.lib”.

To Use the Symbols in “magnetic.slb”

Draw the schematic and assign the desired names (reference designators) to all of the symbols. (Reference designators are assigned automatically, but may be changed by double-clicking on them.)

Select the coupling symbol for the desired CORE model from “magnetic.slb”, and place one coupling symbol, anywhere on the schematic, for each group of coupled inductors. These symbols have no pins; they are represented by the letter K enclosed in a box.

Double-click on each coupling symbol (on the K-in-a-box, not the attributes) and enter the reference designators for the coupled inductors as the values for L_i (i=1,2,...,6).

Set the value of the COUPLING attribute to the value of the coupling factor, K.

To Use the Kbreak and K_Linear Symbols

A generic symbol, Kbreak, is provided in “breakout.slb” for specifying arbitrary nonlinear magnetic core models. Kbreak has a preassigned model attribute, but its corresponding model in “breakout.lib” has no parameters. The K_Linear symbol in

“analog.slb” is provided for specifying linear coupling, between inductors.

Draw the schematic and assign the desired names (reference designators) to all of the symbols. (Reference designators are assigned automatically, but may be changed by double-clicking on them.)

Place one coupling symbol, Kbreak or K_Linear, anywhere on the schematic, for each group of coupled inductors. These symbols have no pins; they are represented by the letter K enclosed in a box.

Double-click on each coupling symbol (on the K-in-a-box, not the attributes) and enter the reference designators for the coupled inductors as the values for L_i (i=1,2,...,6).

Set the value of the COUPLING attribute to the value of the coupling factor, K.

Referencing the CORE Model for Kbreak

A CORE model may be referenced by the Kbreak symbol by following the steps below depending on whether you have a CORE model that is defined in a configured model library file or if you are defining a CORE that is not in any library file.

To Reference a CORE Model in a Configured Model Library File (i.e., a library whose name appears in the Analysis/Library & Include Files dialog box):

1. Single click on the symbol (K-in-a-box) to select it.
2. Select the Edit/Model/Edit Model Reference command.
3. Specify the desired model name.

To define your own CORE model parameters:

1. Change the reference to KBREAK.
2. Select Edit/Model/Edit Instance Model (Text).
3. Enter the model parameters and a new model name, if desired. The default model name will be kbreak-x. The

model will be saved to a local library named "<*schematic_name*>.lib"

Note *Substitute "Parts" for "Text" in step 2 in order to invoke the Parts program for modeling the B-H loop of the magnetic core.*

Important Notes

The "dot" convention for the coupling is related to the direction in which the inductors are connected. The dot is always next to the first pin to be netlisted. When the inductor symbol, L, which is shipped with Schematics is placed without rotation, the "dotted" pin is the left one. Edit/Rotate (<Ctrl R>) rotates the inductor +90deg, which makes this pin the one at the bottom, etc.

Certain rules must be followed when setting the attributes for coupling symbols and the inductors they affect.

Nonlinear CORE models may be applied to one or more inductors, so:

1. The L_1 attribute must have a value (name); the other L_i may be blank.
2. The MODEL attribute must reference a CORE model.
3. The VALUE attributes of the affected inductor symbol(s) must be set to the number of windings (turns).

Linear coupling must be applied two or more inductors (K_Linear only):

1. The L_1 and at least one other L_i attribute must have values (names); the rest may be left blank.
2. The MODEL attribute must be left blank.
3. The values assigned to the inductor symbols must be in Henries.

Using Multipliers for Signal Processing

Multipliers are often used for signal processing applications. In this note, two examples are presented to illustrate the use of a multiplier to make an amplitude modulator, and to make a frequency doubler.

Amplitude and Balanced Modulation

Amplitude modulation is a technique which uses a low-frequency signal to control the amplitude of a high-frequency signal. A simple modulator can be constructed using a multiplier as shown in Figure 140.

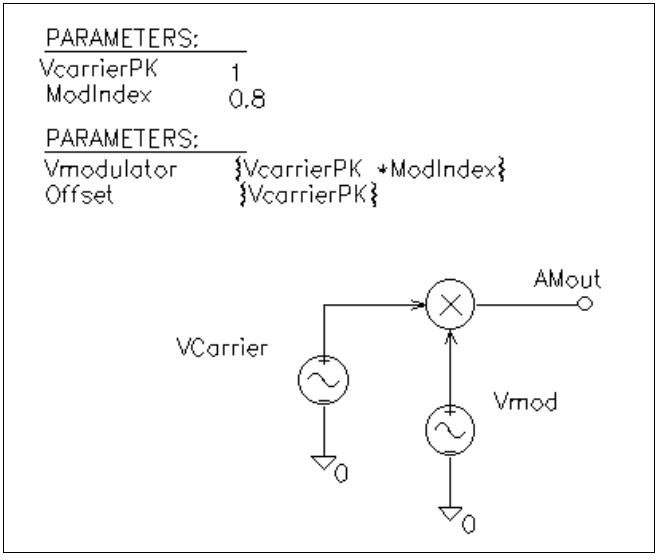


Figure 140 *A simple amplitude modulator circuit*

One input is the high-frequency or carrier signal, and the other input is the modulating signal.

A sinusoidal source at a frequency of 10 kHz is used to represent the carrier signal and a second source at a frequency of 1 kHz is

used to represent the modulating signal. Notice that the peak amplitude for the carrier is set to 1 volt using the parameter VcarrierPK. The modulating index is the ratio of the peak of the modulating signal to the peak of the carrier. Here, the index is set to 0.8 or 80% modulation. A typical broadcast AM signal includes the carrier as well as the sidebands in the transmission. To get such a double sideband transmitted carrier signal (DSB-TC) we must bias or offset the modulating signal by a value equal to the carrier's peak voltage.

```
.PARAM VcarrierPK=1
.PARAM ModIndex=0.8 ; 80% modulation
.PARAM Vmodulator={VcarrierPK * ModIndex}
.PARAM Offset={VcarrierPK} ; for transmitted carrier

EMULT1 AMout 0 VALUE {V(1)*V(2)}
VCarrier 2 0 SIN (0 {VcarrierPK} 10k)
Vmod 1 0 SIN ({Offset} {Vmodulator} 1k)

.TRAN 100uS 2mS
.PROBE
.END
```

The amplitude modulated signal and the modulating signal from this simulation are shown in Figure 141.

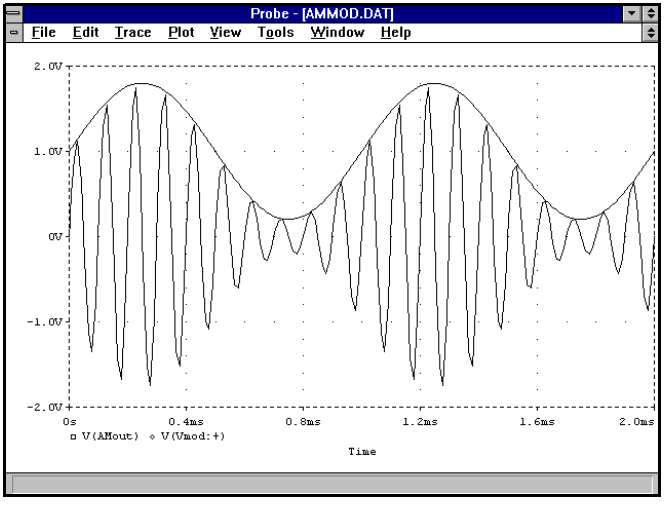


Figure 141 *An amplitude modulator output signal*

A balanced modulator produces a double sideband suppressed carrier signal (DSB-SC). By setting the offset of the modulating signal to be zero in the above circuit, we will suppress the carrier. Notice, the output of this modulator shown in Figure ; the shape of its upper A balanced modulator output signal

envelope resembles a full-wave rectified AC source.

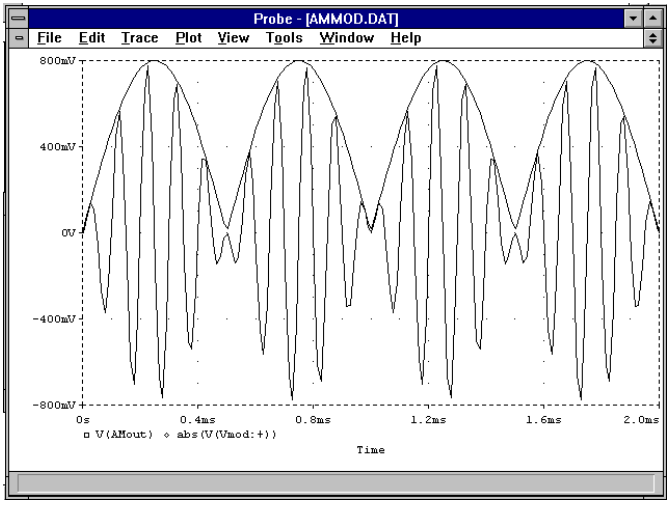


Figure 142 *A balanced modulator output signal*

Frequency Doubling

Another application for a multiplier is as a frequency doubler (see Figure 143). Connecting a sinusoidal source simultaneously to both inputs of a multiplier will yield a signal with double the input frequency. The first multiplier, Xmul, produces a waveform that has one-half the amplitude of the original input signal with a DC offset of one-half the input waveform’s peak value. The DC offset is removed with a voltage source called Voffset. The amplitude of the original

signal is restored with a second multiplier which doubles the signal.

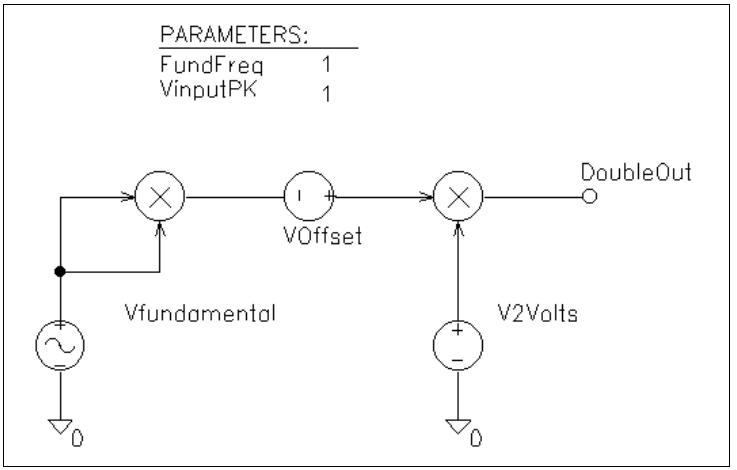


Figure 143 *A simple frequency doubler circuit*

```
.PARAM FundFreq=1
.PARAM VinputPK=1
EMULT1 1 0 VALUE {V(2)*V(2)}
EMULT2 DoubleOut 0 VALUE {V(3)*V(4)}
Vfundamental 2 0 SIN (0 {VinputPk} {FundFreq})
VOffset 4 1 DC {-VinputPK/2} ; remove DC offset
V2Volts 3 0 DC 2 ; restore the amplitude
.tran 1ms 5s
.probe
.END
```

Figure 144 shows the original input signal, as well as the frequency doubled output signal.

These examples have illustrated how a multiplier implemented using the E device in PSpice can be used in signal processing applications such as amplitude modulation and frequency doubling.

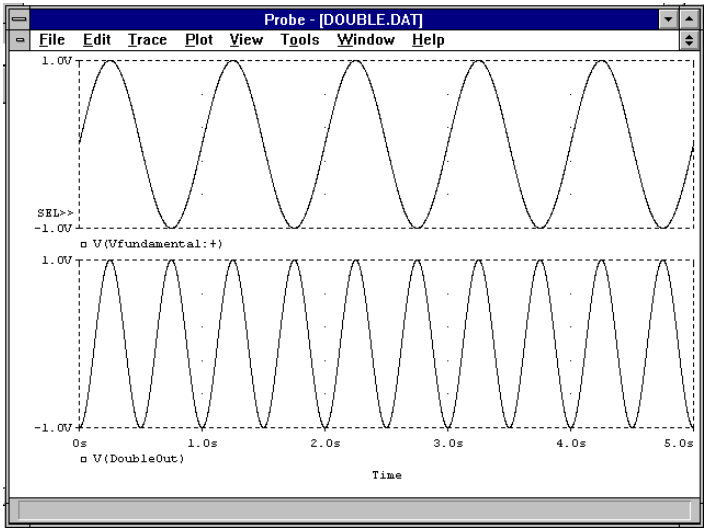


Figure 144 *Output results for frequency doubler*

Using PSpice to Simulate the Discharge Behavior of Common Batteries

By Steven C. Hageman APPLIED DC

Note: An article based on this manuscript appeared in *EDN Magazine*, October 28, 1993

As the use of battery-operated electronic devices become more widespread, so too does the need for simulation models used to analyze the operating characteristics of batteries. The most common batteries in use today are: non-rechargeable Alkaline cells, rechargeable Nickel-Cadmium (NICD) cells, Nickel-Metal-Hydride (NIMH) cells, and sealed Lead-Acid cells.* This article presents PSpice behavioral models for simulating the four battery types mentioned above.

Battery Variables

All of the battery types modeled here share some common characteristics and deviations from ideal during discharge.

- The *capacity* of any group of cells may vary from +/- 20% up to +/- 50% when shelf time, number of recharge cycles, and manufacturing variances are taken into account. For this reason, parameters that change less than 15% are not considered in these models.
- The *capacity* of a cell decays with time after a complete charge. For Alkaline cells, this decay takes years to affect the usable capacity. For NICD and Lead-Acid batteries, the decay is 10 to 30% per month. This effect may be simulated by specifying a reduced state of charge at the start.

*. The term *cell* is used to indicate a single energy source. The term *battery* is used to distinguish a power source composed of a single cell or several cells.

- The major deviation from ideal is that the *usable capacity* of a cell varies depending on the discharge rate. At very low discharge rates (<) 100 hours), all batteries are very efficient. At very fast discharge rates (< 10 hours), the batteries are not as efficient and usable capacity is lost.
- For *pulsed loads* with cycle times greater than 10 seconds, the cell gives more total capacity than under a constant load. The rest portion of the pulsed load allows the battery chemistry to recover some of the lost capacity. But, as the pulsed load cycle time becomes less than 1 second, the cell does not have enough time to recover and usable capacity is not increased. In these cases, the RMS value of the pulsed discharge current should be used in the simulation.
- *Cell temperature* affects both the cell resistance and usable capacity. Low cell temperatures reduce the usable capacity; only a slight decrease is noted at high temperatures. For the battery types modeled here, the change in resistance versus temperature falls below the 15% change threshold, so these effects are not modeled. These changes may be accounted for by adjusting the parameters passed to the various cell subcircuits.
- *Cell resistance* is a function of the cell's state of charge and, although there is a negligible effect on Lead-Acid and NICD types, Alkaline cells show a 2:1 to 4:1 increase in cell resistance from full charge to full discharge. Still, cell resistance is fairly flat and constant until 80% discharged, then the resistance increases sharply. The sharp fall in cell voltage during discharge can be looked upon as a large increase in cell resistance.
- *Open circuit cell voltage* varies with discharge temperature. But, this variation, even over a 0 to 60°C range, is much less than the difference in actual cell discharge voltage. Therefore, it is not useful to simulate. NICD batteries are the exception; these are used in high-rate discharge applications where the cells may increase in temperature by 25°C during discharge. Cell discharge voltage versus temperature is modeled in the NICD subcircuit.

Behavioral Modeling

Figure 145 shows the results of discharging seven identically rated NICD cells to see how well their capacity track. These cells were in weekly use for 1 to 2 years and exhibit a 2:1 spread in measured capacity. Alkaline and Lead-Acid batteries have similar variations even between new cells.

This indicates that there is little practical value in overly accurate models. Therefore, only those battery characteristics that present a 10 to 15% or greater change during discharge are modeled.

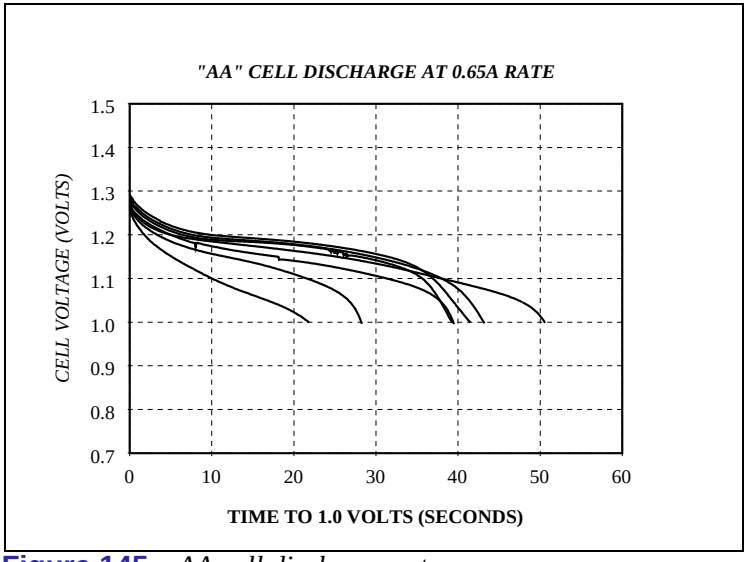


Figure 145 *AA cell discharge rate*

The batteries are modeled using these functional blocks (refer to Figure 146 and Figure 147):

1. Capacitor representing the A-H capacity of the cell.
2. Discharge rate normalizer to determine the lost capacity at high discharge rates.
3. A circuit to discharge the A-H capacity of the cell.
4. Cell voltage versus state-of-charge lookup table.
5. Cell resistance.

6 For NICD batteries, the thermal effects of the cell under high discharge rates.

To start modeling a cell, several actual discharge curves should be measured on a computerized constant-current load analyzer [1] at a low rate (20 to 200 hours) to get an actual voltage versus capacity curve. A single curve is then made by averaging several curves, or picking a *typical* curve from the data. This data is then converted into a parameterized PSpice lookup table Voltage-Controlled Voltage Source (VCVS). This models the cell's output voltage versus the state-of-charge at low discharge rates. A simplified VCVS definition is

```
E_Cell+OUT -OUTTABLE {V(x)} = (0,1.5) (0.5,1.3) (1.0,0.0)
```

where:

E_Cell	signifies the PSpice call to a VCVS named E_Cell
+OUT and -OUT	are the output nodes of the VCVS
TABLE	is the PSpice behavioral modeling TABLE directive
{V(x)}	is the controlling voltage for the table

(0,1.5) (0.5,1.3) (1.0,0.0) are the table pairs that are output to +OUT and -OUT based on the value of V(x). If V(x) is 0, signifying 0% discharge, then E_Cell will have a value of 1.5 Volts (table pair 1). If the cell is 50% discharged then the second table pair will be used and so on. For in-between discharge values, PSpice uses linear interpolation between the table pairs.

Note *The actual lookup tables are composed of 30 or more pairs of data to provide finer granularity of the resulting discharge voltage curve.*

To model the discharge current sense and the cell resistance, a zero-valued voltage source is added in series with the output voltage. The cell resistance is modeled as a simple resistor for NICD or Lead-Acid cells and as a more complex variable resistance that depends on the cell's state of charge for Alkaline cells.

To model the state-of-charge, a simple, appropriately sized capacitor is used as the charge storage element that simulates the available charge of the cell. This capacitor is sized so that it has a value of 1 Volt at 100% cell capacity and 0.5 Volts at 50% cell

capacity. This capacitor is given the following value at the start of the simulation by PSpice's "Parameterization" function:

```
C_CellCapacity 50 0 {3600*CAPACITY*FudgeFactor}
```

The capacitor, C_CellCapacity, is connected between nodes 50 and 0 and is given a value of the Amp-hour capacity of the cell times a conversion from hours to seconds (3,600 seconds = 1 hour) times a *fudge factor* (FudgeFactor). If a cell has a 10 Amp-hour capacity, C_CellCapacity equals 10 * 3,600 or 36,000 Farads; this is a big capacitor, but a workable value that is easy to understand.

FudgeFactor adjusts for the difference in the manufacturer's listed Amp-hour capacity (i.e., some *cutoff voltage* with some capacity remaining at the cutoff) and the simulated capacity of 0 Volts output at 0% remaining capacity. To correct for this, and still allow the model user to use the manufacturer's listed capacity, a FudgeFactor value of 1.01 to 1.1 is included.

The actual usable capacity of a cell depends on the rate at which it is being discharged. Most manufacturers list the capacity at the most favorable rate—usually at greater than 20 hours discharge. At any faster rate, the cell is less efficient and results in a nonlinear function of the discharge rate. This must be characterized as a lookup table at many discharge rates. This inefficiency is modeled as a VCVS in series with the output voltage of the battery state-of-charge node (the voltage on C_CellCapacity). This VCVS subtracts a given amount of *capacity* from the cell during discharge. The amount subtracted depends on the rate at which the cell is being discharged.

To determine the rate at which the cell is being discharged, it is convenient to normalize the discharge rate in Amps to a more conventional cell rate called the *C rate*. The C rate is defined as the capacity of the cell in Amp-hours when it is discharged completely in one hour. This normalization makes it easy to determine the cell inefficiency at different rates, and between different cell sizes, because it converts discharge in Amps to discharge in "C" units of the battery capacity at one hour. This conversion is done in the model by the VCVS, E_Rate, as follows.

```
E_Rate RATE 0 VALUE = {I(V_Sense) / CAPACITY}
```

E_Rate is the sensed discharge current in Amps divided by the Amp-hour capacity of the cell. The node, RATE, is the instantaneous rate at which the cell is being discharged (see Figure 146).

This instantaneous rate information can almost be fed directly to E_Lost_Rate to determine the actual available capacity. But, when the discharge is a low duty cycle, high value pulsed load, the cell supplies a large initial current which decays in seconds to a lower value. For pulsed loads, the cell recovers between pulses and delivers a higher proportion of its capacity than a cell under constant discharge. The delayed rate is modeled by an RC lowpass filter (R1 and C1 of Figure 146). The exact value of the RC time constant depends on the type and size of cell being simulated. E_Lost_Rate is built like the E_Cell table as follows.

```
E_Lost_Rate 50 SOC TABLE { V(x) } = (0.0,0.0) (1.5,0.5)
```

The table entries indicate the capacity unavailable from the cell at high discharge rates. The table entry shows that at a discharge rate of 0, the cell loses 0% of its capacity (first entry). If the discharge rate is 1.5 times the rated capacity of the cell (1.5 C), the cell loses 50% of its capacity (second entry in the table).

In Figure 146, the *State-Of-Charge* (SOC) node is the subtraction of the voltage on the capacitor C_CellCapacity and E_Lost_Rate. The SOC node represents the capacity in the cell for a given discharge rate during the simulation. G_Discharge discharges C_CellCapacity at the cell rate. The voltage on node 50 relates to the capacity remaining in the cell if the discharge rate is low enough to actually run the cell dry. At low discharge rates, these two nodes are the same; at high discharge rates, node SOC is at a lower potential than node 50. If, at the end of a high discharge rate the cell reverts to a low discharge, nearly the entire rated capacity can be recovered from the cell. At the high discharge rate, approximately 60% of the cell's rated capacity can be used.

All that needs to be done now is to link the state of charge with the cell voltage to get an output. The state of charge is 1 Volt for 100%, while the cell voltage table is just the opposite. To make

the cell voltage correct, the state-of-charge voltage must be inverted as shown in Figure 146.

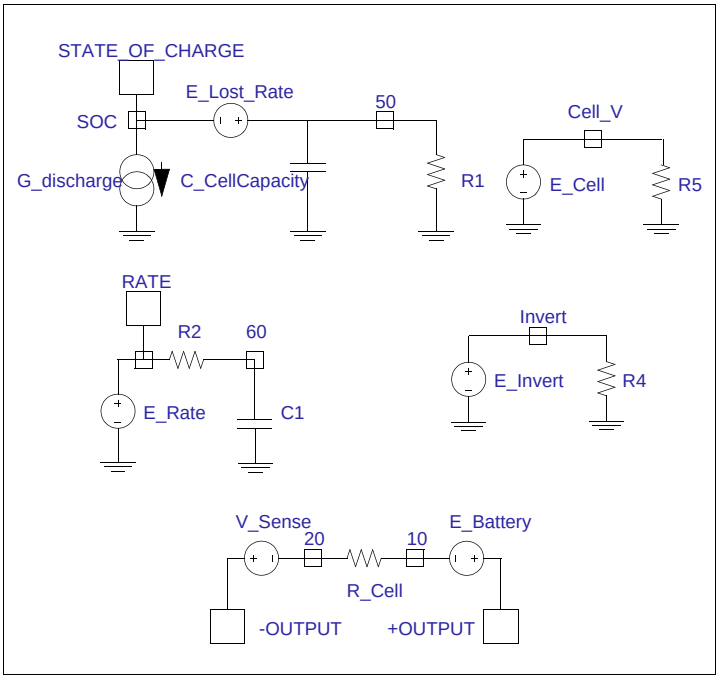


Figure 146 *Functional schematic developed for all of the modeled cell types; only minor changes are required to complete each detailed model type*

Model Differences for Different Battery Types

Alkaline cells (see listings in Figure 149 and Figure 151)

Alkaline cell resistance is not fixed throughout the discharge. The resistance model is developed by determining the relationship of the output's current and voltage, then linking this to the battery's state of charge. The cell has a small resistance increase from 100% to 20% cell capacity, then increases to twice its initial value at 0% capacity.

When the discharge current approaches 100 mA, the discharge capacity versus discharge rate produces a kink in the discharge curve. Below 100 mA, the cell loses capacity gradually; above 100 mA, the rate of lost capacity increases significantly. Despite the 100 mA discharge rate being the same for all of the cell sizes, the C rating is not. Because it isn't possible to relate this kink to a specific C discharge rate, a separate E_Lost_Rate table must be developed for each cell size.

A separate subcircuit model is used to model the 9 Volt Alkaline cell. The cell resistance change versus discharge state is more pronounced in this type of battery, and is modified accordingly in the model shown in Figure 151.

Nickel-cadmium cells (see Figure 147 and listing in Figure 152)

These cells are often used at very high discharge rates up to 20 C. Discharging a fully charged cell in 5 or 6 minutes (10 C rate) releases significant amounts of heat. To account for this, a thermal model is included in the cell subcircuit.

The thermal model depends on two fundamental relationships:

- The thermal temperature rise of a cell per watt dissipated in free air is approximately $\Theta_{cell} = 13.41 * V^{-0.61}$ where V is the cell volume in cubic inches, and Θ_{cell} is the thermal rise of the cell in °C per watt dissipated.
- The thermal time constant for material of the density used in making NICD batteries is approximately 20 minutes per pound, or, expressed in more convenient terms, 2.65 seconds per gram.

These empirical relationships are used with the calculated cell power dissipation (*Cell Discharge Current*^{2} * *Cell Resistance*) to get a temperature rise and time constant model for the cell temperature. The cell temperature rise above ambient temperature is available at node CELL_TEMP in the NICD model. The temperature information is also used to add or subtract from the cell discharge voltage to account for the cell temperature E_Temp in the NICD model.

Another small modification was made to the NICD model to facilitate the direct entry of manufacturers' rated capacity data. Most NICD batteries are not rated at their maximum capacity for low discharge rates. The norm is to rate them at the C to C/5 rate, leaving 30% more than the rated capacity if the cell is used at low discharge rates. To account for this difference a Voltage-Controlled Current Source (VCCS), G_LowRate, is used to add a small amount of current to C_CellCapacity during discharge at rates less than 1 C.

Nickel-Metal-Hydride cells (see listing in Figure 156)

These cells are modeled like the NICD cells, but without the fast discharge thermal effects.

Lead-acid cells (see listing in Figure 154)

Since these cells are almost universally used in batteries composed of 3, 6, or more cells (6 or 12 Volt batteries), the model is changed slightly. The single cell voltage is multiplied by the number of cells to get the total battery voltage.

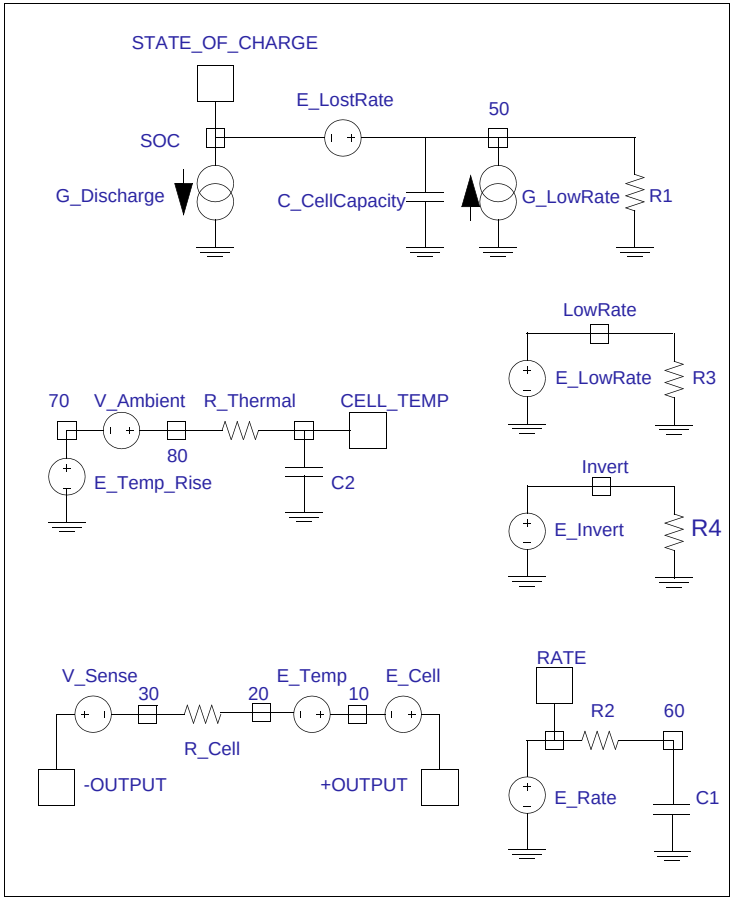


Figure 147 *NICD battery discharge models*

Using the Discharge Models

To use the models in a simulation, add the required number of cells to the circuit file, and pass the appropriate parameters to the subcircuit model. The manufacturers' data sheets may be consulted for information on the cell parameters shown in Table 8.

Table 8 *Cell Parameters*

Cell Model	Parameter	Meaning
ALKALINE	CAPACITY	Capacity of the cell measured at greater than 100 hour discharge time
	RESISTANCE	Initial cell resistance in ohms
9V ALKALINE		No parameters are needed for a generic 9 Volt Alkaline battery
NICKEL-CADMIUM	CAPACITY	Amp-hour capacity of cell measured at the C/5 rate
	RESISTANCE	Mid-discharge cell resistance in ohms
	CELLTEMP*	Initial cell temperature at start of simulation
	VOLUME*	Volume of cell in cubic inches
	WT*	The weight of the cell in grams
NICKEL-METAL-HYDRIDE	CAPACITY	Amp-hour capacity of cell measured at the C/5 rate
	RESISTANCE	Mid-discharge cell resistance in ohms
LEAD-ACID	CAPACITY	Capacity of the battery measured at the 20 hour rate
	RESISTANCE	Mid-discharge battery resistance in ohms
	CELLS	The number of cells that make up the battery (6V = 3 cells, 12V = 6, etc.)

***** These parameters are only needed at a discharge rate greater than 5 C to account for cell temperature rise.

Since little standardization exists for rating methods between manufacturers, the popular *typical* parameters are summarized in Table 9 for the most common consumer batteries.

Table 9 *Subcircuit Parameters for the Cell Models*

Cell Type	Capacity (Amp-hour)	Resistance (Ohms)	Volume (in3)	Weight (gm)	Cell Number
Alkaline Cells Typical subcircuit call for a single Alkaline N cell: X1 +node -node SOC RATE ALKALINE PARAMS: CAPACITY=0.9, RESISTANCE=5 The correct E_Lost_Rate lookup table for an N cell must not be commented in the circuit file shown in Figure 149.					
N	0.9	0.8	—	—	—
AAA	1.2	0.6	—	—	—
AA	2.5	0.3	—	—	—
C	7.5	0.2	—	—	—
D	16.4	0.07	—	—	—
Nickel-Cadmium Cells (Standard)* Typical subcircuit call for a single NICD N cell: X1 +node -node SOC RATE CELL_TEMP NICD + PARAMS: CAPACITY=0.15, RESISTANCE=5, VOLUME=0.2, WT=9					
N	0.15	0.027	0.2	9	—
AAA	0.18	0.021	0.24	10	—
AA	0.55	0.012	0.48	24	—
SUB C	1.2	0.005	1.1	50	—
C**	1.8	0.0045	1.6	80	—
D**	4.0	0.0035	3.4	160	—
Nickel-Metal-Hydride Cells Typical subcircuit call for a single NIMH AA cell: X1 +node -node SOC RATE NIMH PARAMS: CAPACITY=1.1, RESISTANCE=0.03					
AA	1.1	0.03	—	—	—
4/5A	1.5	estimated 0.02	—	—	—

Table 9 *Subcircuit Parameters for the Cell Models*

Cell Type	Capacity (Amp-hour)	Resistance (Ohms)	Volume (in3)	Weight (gm)	Cell Number
Lead-Acid Cells Typical subcircuit call for a 6 Volt, 1.3 Amp-hour Lead-Acid battery: X1 +node -node SOC RATE LEADACID PARAMS: CAPACITY=1.3, RESISTANCE=0.06, CELLS=3					
6 V - 1.3 A-hr	1.3	0.06	—	—	3
6 V - 4.0 A-hr	4.0	0.025	—	—	3
6 V - 6.5 A-hr	6.5	0.02	—	—	3
6 V - 10 A-hr	10	0.015	—	—	3
12 V - 1.3 A-hr	1.3	0.12	—	—	6
12 V - 4.0 A-hr	4.0	0.05	—	—	6
12 V - 6.5 A-hr	6.5	0.04	—	—	6
12 V - 10 A-hr	10	0.03	—	—	6

***** The Volume and Weight parameters applicable to NICD cells need only be used when simulating a discharge rate grater than approximately 5 C and when the temperature profile of the cell is desired.

****** Although real NICD C and D cells can be purchased, most consumer C and D batteries are actually SUB C cells in a big empty can.

Temperature Effects

During a fast discharge, the cell temperature of a NICD can change by 25°C or more. The effect of cell temperature on voltage is accounted for in the NICD model by changing the cell voltage based on the calculated temperature. The other models do not incorporate temperature effects directly into the simulation.

The major temperature influence on the cell is capacity. This may be accounted for by adjusting this parameter at the start of a simulation. The following equations give the new capacity for each cell type at any discharge temperature from 0 to 60°C based on the initial capacity at 25°C.

- Alkaline Cells:

  ```
  NewCapacity = OldCapacity * (0.85 + 8.64E-
  3 * T - 1.05E-4 * T^2)
  ```

- Nickel-Cadmium Cells:

  ```
  If T > 25°C
  NewCapacity=OldCapacity
  If T < 25°C
  NewCapacity=OldCapacity*(0.815+7.5E-3*T)
  ```

- Nickel-Metal-Hydride Cells:

  ```
  NewCapacity=OldCapacity*(0.913+1.1E-2*T-3.0E-4*T^2)
  ```

- Lead-Acid Cells:

  ```
  NewCapacity=OldCapacity*(0.84 + 7.96E-3*T-6.07E-5*T^2)
  ```

To use the equations, simply plug in the 25°C capacity for *OldCapacity* and the new discharge temperature for *T* into the proper equation. If desired, these equations can also be built into the subcircuit models.

Example Circuit—AA NICD 2 Ohm Discharge Test

To demonstrate the use of a cell in an actual circuit, a real AA NICD cell was discharged into a constant resistance of 2 Ohms. Then the following circuit was used with PSpice to simulate the 2 Ohm discharge. This circuit, run with the capacity of the NICD model normalized to the actual capacity of the measured cell, shows results which compare very favorably with the real behavior (see Figure 148).

```
.INC "NICD.CIR"        ; Include the NICD subcircuit
.TRAN 30 3000          ; Simulate for 50 Minutes
.PROBE                 ; Write a Probe data file

RLoad 10      0        2        ; Load resistor - 2 Ohms

.IC V(X1.50)=1 V(X1.60)=0      ; Set 100% charged capacity

      * * SUBCIRCUIT CALL FOR AA NICD CELL * *

X1    10 0 SOC RATE CELL_TEMP NICD
+ PARAMS: CAPACITY=0.46, RESISTANCE=0.012, CELLTEMP=25
+     VOLUME=48 , WT=24

.END
```

The .IC statement sets the initial conditions, and must be set for every subcircuit used. V(X1.50) sets the initial charge on node 50 of the X1 subcircuit. This is the voltage on the battery Amp-hour capacity model which simulates the initial state of charge. Setting this node to 1 Volt equals an initial state of charge of 100%. Likewise, 0.8 Volts would represent an 80% initial state of charge.

The next initial condition (V(X1.60)=0) sets the voltage on the delayed lost rate calculator to zero. This allows the voltage on capacitor C1 (internal to the subcircuit) to start at 0 Volts as it would if the discharge current was zero before the simulation started. Another way to achieve this result is to switch *on* the discharge currents just after the simulation starts. This automatically sets the delayed lost rate voltage to zero at the start of the simulation.

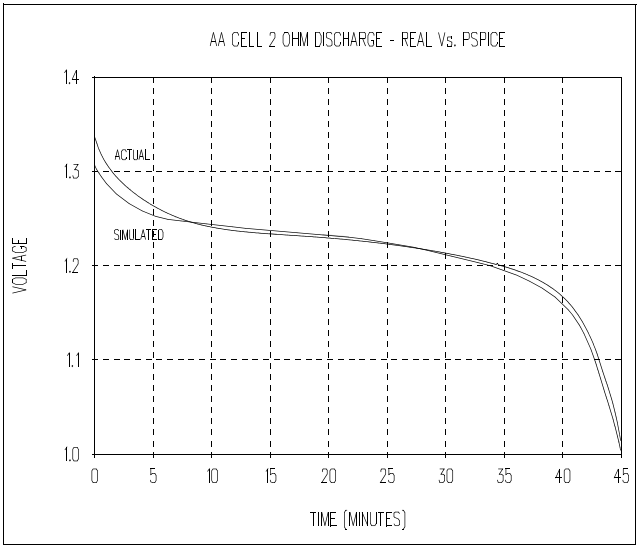


Figure 148 *The 2 Ohm discharge test results using the NICD simulation model closely match discharge data for an actual AA cell*

Limitations of the Models

The circuits presented here trade off accuracy with simplicity and simulation time. There are several cases where the simulated and actual results vary significantly.

In practice, Alkaline cells are sometimes used in 2 to 8 hour shifts, and rest the remainder of the day. This rest time allows the cell to recover part of its discharge capacity. This phenomenon is not specifically modeled, and when simulated with the alkaline subcircuit (see Figure 149), the observed capacity may be up to 25% short over a cell's actual performance. Using the Alkaline models in these patterns will give a conservative estimate of capacity.

When a battery is discharged to a low terminal voltage level, then disconnected from the load, the battery voltage will recover to some higher level in an hour or so. This phenomenon is accounted for in these models when they are discharged at high

current levels then left to rest. When the models are used at low discharge rates to low terminal voltages however, they do not show this voltage recovery. The battery chemistry tries to make a voltage potential difference even if only a few molecules of unused material remain. In this state of discharge, the internal resistance of the cell can be an order of magnitude, or more, than its initial value. If any load is reconnected, the terminal voltage will quickly collapse again to zero.

The models were designed to be used with the standard cutoff voltages as specified by the battery makers. For NICD batteries, this is 0.8 to 1.1 Volts per cell. For Alkaline cells, the cutoff is 0.8 to 1.2 Volts per cell. For Lead-Acid cells, the cutoff voltage is typically 1.5 to 1.7 Volts per cell. Usage beyond these limits should be studied carefully because they were not specifically examined in the modeling process.

These models were not designed to be connected in parallel. This is not acceptable in consumer design anyway, because there is no way to guard against the end user putting an Alkaline cell in parallel with a NICD or Carbon cell. If these different types of cells are connected together, the charge most likely won’t equalize. This results in overcharging and leakage of the weak cell, thus causing damage.

Simulation Speed

The goal of simulation is to obtain results faster than can be achieved with the hardware, or to measure behaviors which cannot easily be accessed in the hardware. The following notes should help when making speed/accuracy trade-offs.

- Don’t go overboard on the models that you attach to these batteries during a simulation. Simulate the power drain from your circuit, not the transistor-level circuit itself. These models have been tried and verified with many different discharge regimens, and are believed to be accurate enough to allow finding maximum or minimum battery life. The trends these models simulate are believed to be basically accurate, even though the absolute capacity simulated may be 20% or so off.
- Don’t simulate pulsed current loads with cycle times less than 5 seconds or so. Using short cycle time pulsed currents may make the simulation run slower than real time. To speed up the simulation with fast pulsed loads, use the RMS average of the pulsed current. This will provide you with a ballpark answer.
- Use a minimum of semiconductor models hooked up to these models. Semiconductors contain many internal nonlinear equations that must be solved for each time point, thus slowing simulation time.
- To prevent convergence problems, ABSTOL and VNTOL should be set to values about nine orders of magnitude less than the maximum currents and voltages in your circuit.
- RELTOL may be relaxed to 1% from its 0.1% default value to speed up the simulation.
- If you experience convergence problems, use the .IC directive to set the initial voltages on critical nodes in your circuit.

- If you still experience convergence problems, use a voltage-controlled switch model (S device type in PSpice) to connect the battery to the load after the simulation starts. Use as slow of a connect transition time as possible to avoid stalling the simulator.

```
* * AMP-HOUR CAPACITY OF BATTERY * *
C_CellCapacity 50 0 { 3600 * CAPACITY * 1.01 }
R1 50 0 1G
* * CELL RESISTANCE * *
E_Resistance 20 10 VALUE = {I(V_Sense) * RESISTANCE * V(Cell_Res)}
* * CELL RESISTANCE Vs. REMAINING CHARGE MULTIPLIER FACTOR * *
E_Cell_R Cell_Res 0 TABLE { V(50) } = (0,2) (0.2,1) (1,1)
R3 Cell_Res 0 1G
* * CELL OUTPUT CURRENT SENSE * *
V_Sense -OUTPUT 20 0
* * CELL OUTPUT VOLTAGE VS STATE OF CHARGE * *
E_Invert Invert 0 TABLE { V(SOC) } = (0,1) (1,0)
R4 Invert 0 1G
E_Cell +OUTPUT 10 TABLE { V(Invert) } =
+ (0.000E+00 1.528E+00) (2.320E-03 1.511E+00) (4.640E-03 1.500E+00)
+ (9.280E-03 1.481E+00) (1.392E-02 1.468E+00) (1.856E-02 1.457E+00)
+ (2.552E-02 1.442E+00) (3.248E-02 1.430E+00) (3.944E-02 1.419E+00)
+ (4.872E-02 1.406E+00) (5.800E-02 1.394E+00) (6.728E-02 1.380E+00)
+ (7.656E-02 1.370E+00) (1.206E-01 1.326E+00) (2.691E-01 1.230E+00)
+ (5.522E-01 1.126E+00) (8.213E-01 1.021E+00) (9.025E-01 9.901E-01)
+ (9.257E-01 9.792E-01) (9.443E-01 9.676E-01) (9.559E-01 9.564E-01)
+ (9.628E-01 9.445E-01) (9.698E-01 9.299E-01) (9.744E-01 9.181E-01)
+ (9.791E-01 9.043E-01) (9.814E-01 8.937E-01) (9.837E-01 8.800E-01)
+ (9.860E-01 8.654E-01) (9.884E-01 8.470E-01) (9.907E-01 8.040E-01)
+ (9.930E-01 6.417E-01) (9.953E-01 3.795E-01) (9.976E-01 3.354E-01)
+ (1.0 0.0)
.ENDS
```

Figure 149 *Alkaline cell model covering all of the popular consumer-type cells; valid for one to one-thousand hour discharge (continued on the next page).*

```
PSpice Alkaline battery discharge model
* Optimized for N through D Cells, and discharge rates from 1 to 1,000 hours.
*--- Nodes
* +OUTPUT, -OUTPUT = +/- cell connections (floating)
* SOC = state-of-charge output node, (1V=100%, 0V=0%)
* RATE=instantaneous discharge rate, (1V=C,10V=10C) referred to 50 hour rate
*--- Parameters
* CAPACITY = battery capacity in Amp-hours, 1=1A-hr, 0.5=0.5A-hr
* measured at 100 hour or greater rate
* RESISTANCE = total battery resistance in ohms
.SUBCKT ALKALINE
+ +OUTPUT -OUTPUT SOC RATE
+ PARAMS: CAPACITY=1, RESISTANCE=1
* * DISCHARGE RATE CALCULATION * *
E_Rate RATE 0 VALUE = { I(V_Sense)/CAPACITY }
R2 RATE 60 10 ; R2-C1 -> 10 Second time constant
C1 60 0 1
* * DISCHARGE AND STATE OF CHARGE * *
G_Discharge SOC 0 VALUE = { I(V_Sense) } ; Discharge Current
* * LOST CAPACITY DURING FAST DISCHARGE DELAYED BY R2-C1 * *
_Lost_Rate 50 SOC TABLE { V(60) } =
* * Use one of the following tables!!! * *
;----- Use this table for N cells -----
;+ (0.0,0.0) (0.019,0.056) (0.043,0.13) (0.072,0.28)
;+ (0.12,0.39) (0.21,0.58) (0.31,0.69)
;----- Use this table for AAA and AA cells -----
;+ (0.0,0.0) (0.018,0.08) (0.043,0.14) (0.08,0.2)
;+ (0.14,0.3) (0.26,0.48) (0.4,0.6)
;----- Use this table for C cells -----
;+ (0.0,0.0) (0.17,0.13) (0.035,0.31) (0.055,0.45)
;+ (0.093,0.53) (0.17,0.65) (0.27,0.73)
;----- Use this table for D cells -----
;+ (0.0,0.0) (0.0091,0.091) (0.017,0.15) (0.032,0.36)
;+ (0.058,0.42) (0.079,0.61) (0.13,0.73) (0.18,0.82)
```

Figure 150 *Alkaline cell model covering all of the popular consumer-type cells; valid for one to one-thousand hour discharges (Continued)*

```
PSpice 9 Volt Alkaline battery discharge model
* Tested for discharge rates from 0 to 400 mA to 4.8 Volt cutoff voltage.
*--- Nodes
* +OUTPUT, -OUTPUT = +/- cell connections (floating)
* SOC = state-of-charge output node, (1V=100%, 0V=0%)
* RATE = instantaneous discharge rate, (1V=C,10V=10C)
*          referred to 120 hour rate
*--- Parameters
* CAPACITY = mA-hr capacity of 9 Volt Alkaline battery
.SUBCKT ALK_9V
+ +OUTPUT -OUTPUT SOC RATE
+ PARAMS: CAPACITY=0.565
* * DISCHARGE RATE CALCULATION * *
E_Rate RATE 0 VALUE = { I(V_Sense)/CAPACITY }
R2 RATE 60 10 ; R2-C1 -> 10 Second time constant
C1 60 0 1
* * DISCHARGE AND STATE OF CHARGE * *
G_Discharge SOC 0 VALUE = { I(V_Sense) } ; Discharge Current
* * LOST CAPACITY DURING FAST DISCHARGE DELAYED BY R2-C1 *
E_Lost_Rate 50 SOC TABLE { V(60) } =
+ (0.0,0.0) (0.025,0.009) (0.046,0.080) (0.088,0.14) (0.18,0.21) (0.71,0.45)
* * AMP-HOUR CAPACITY OF BATTERY * *
C_CellCapacity 50 0 { 3600 * CAPACITY * 1.06 }
R1 50 0 1G
* * CELL RESISTANCE * *
E_Resistance 20 10 VALUE = { I(V_Sense) * 2.0 * V(Cell_Res) }
* * CELL RESISTANCE Vs. REMAINING CHARGE MULTIPLIER FACTOR * *
E_Cell_R Cell_Res 0 TABLE { V(50) } = (0,4) (0.2,2) (1,1)
R3 Cell_Res 0 1G
* * CELL OUTPUT CURRENT SENSE * *
V_Sense -OUTPUT 20 0
* * CELL OUTPUT VOLTAGE VS STATE OF CHARGE * *
E_Invert Invert 0 TABLE { V(SOC) } = (0,1) (1,0)
R4 Invert 0 1G
E_Cell +OUTPUT 10 TABLE { V(Invert) } =
+ (0.00 9.18) (0.05 8.82) (0.10 8.62)
+ (0.15 8.41) (0.20 8.30) (0.25 8.21)
+ (0.30 8.09) (0.35 7.99) (0.40 7.95)
+ (0.45 7.89) (0.50 7.79) (0.55 7.66)
+ (0.60 7.55) (0.70 7.18) (0.75 6.96)
+ (0.80 6.58) (0.85 6.12) (0.90 5.42)
+ (0.95 4.51) (1.00 0.00)
.ENDS
```

Figure 151 *9 Volt alkaline cell model for discharge rates from 0 to 400 mA*

```
PSpice Nickel-Cadmium battery discharge model
* Optimized for N through SUB C Standard Cells, Discharge rates
from 0C to 10C.
*--- Nodes
* +OUTPUT, -OUTPUT = +/- cell connections (floating)
* SOC = state-of-charge output node, (1V=100%, 0V=0%)
* RATE = instantaneous discharge rate, (1V=C,10V=10C)
* CELL_TEMP = cell temperature output node
*--- Parameters
* CAPACITY = cell capacity in Amp-hours, 1=1A-hr, 0.5=0.5A-hr
* measured at 1 hour or greater rate
* RESISTANCE = cell resistance in ohms
* CELLTEMP = ambient temperature in degrees C
* VOLUME = cell volume in cubic inches
* WT = cell weight in grams
.SUBCKT NICD
+ +OUTPUT -OUTPUT SOC RATE CELL_TEMP
+ PARAMS: CAPACITY=1, RESISTANCE=1, CELLTEMP=25, VOLUME=1, WT=1
* * DISCHARGE RATE CALCULATION * *
E_Rate RATE 0 VALUE = { I(V_Sense)/CAPACITY }
R2 RATE 60 1 ;R2-C2 provide 3 second delayed time constant
C1 60 0 3
* * LOW RATE ADDITIONAL ENERGY LOOK-UP TABLE AND TRANSFER * *
E_LowRate LowRate 0 TABLE {V(RATE)} = (0,0) (0.001,0.2) (0.1,0.2) (1,0)
R3 LowRate 0 1G
G_LowRate 0 50 VALUE = { V(LowRate)*I(V_Sense) }
* * DISCHARGE AND STATE OF CHARGE * *
G_Discharge SOC 0 VALUE = { I(V_Sense) } ; Discharge Current
* * LOST CAPACITY DURING FAST DISCHARGE DELAYED BY R2-C1 * *
E_LostRate 50 SOC TABLE { V(60) } = (1.0,0) (10,0.25)
* * AMP-HOUR CAPACITY OF BATTERY * *
C_CellCapacity 50 0 { 3600 * CAPACITY * 1.03 }
R1 50 0 1G
* * POWER DISSIPATION AND CELL TEMPERATURE * *
E_Temp_Rise 70 0 VALUE = {PWR(I(V_Sense),2)*RESISTANCE*(13.4 PWR(VOLUME,-
0.6065))}
V_Ambient 80 70 { CELLTEMP }
R_Thermal CELL_TEMP 80 { 2.65 * WT }
C2 CELL_TEMP 0 1
```

Figure 152 *Nickel-Cadmium cell model valid for discharge rates from 0 to 10 C, where C is the one hour rated capacity in Amps (continued on the next page).*

```
     * * CELL RESISTANCE * *
     R_Cell 20 30 { RESISTANCE }
     * * CELL VOLTAGE TEMPERATURE COEFFICIENT * *
     E_Temp 10 20 TABLE { V(CELL_TEMP) } = (0,-0.025) (25,0) (60,-0.100)
     * * CELL OUTPUT CURRENT SENSE * *
     V_Sense -OUTPUT 30 0
     * * CELL OUTPUT VOLTAGE VS STATE OF CHARGE * *
     E_Invert Invert 0 TABLE { V(SOC) } = (0,1) (1,0)
     R4 Invert 0 1G
E_Cell +OUTPUT 10 TABLE { V(Invert) } =
+ (0.0000000000E+00 1.3148600000E+00) (1.7391197842E-03 1.3114600000E+00)
+ (8.6956352158E-03 1.3008400000E+00) (1.7391252284E-02 1.2910200000E+00)
+ (3.1304265000E-02 1.2794000000E+00) (4.8695517284E-02 1.2685600000E+00)
+ (6.4347649784E-02 1.2608600000E+00) (9.2173675215E-02 1.2504800000E+00)
+ (1.3739093478E-01 1.2401000000E+00) (2.9217314000E-01 1.2300000000E+00)
+ (4.9738998228E-01 1.2199000000E+00) (6.0347665207E-01 1.2099000000E+00)
+ (7.3738938358E-01 1.1909200000E+00) (7.8782396620E-01 1.1801600000E+00)
+ (8.2086740543E-01 1.1705000000E+00) (8.4695429293E-01 1.1602800000E+00)
+ (8.6608462870E-01 1.1501000000E+00) (8.7999764142E-01 1.1403000000E+00)
+ (8.9043241457E-01 1.1309800000E+00) (8.9912806793E-01 1.1212400000E+00)
+ (9.0782372129E-01 1.1090600000E+00) (9.1304106250E-01 1.1001000000E+00)
+ (9.1825840370E-01 1.0899000000E+00) (9.2347583564E-01 1.0784600000E+00)
+ (9.2695405706E-01 1.0705000000E+00) (9.3217141642E-01 1.0586000000E+00)
+ (9.3564971043E-01 1.0508000000E+00) (9.3912795000E-01 1.0430000000E+00)
+ (9.4434529120E-01 1.0303200000E+00) (9.4782360336E-01 1.0207400000E+00)
+ (9.5130184293E-01 1.0102600000E+00) (9.5304094457E-01 1.0046200000E+00)
+ (9.5478006435E-01 9.9866000000E-01) (9.5651918413E-01 9.9248000000E-01)
+ (9.5825830392E-01 9.8596000000E-01) (9.5999749629E-01 9.7910000000E-01)
+ (9.6173659793E-01 9.7178000000E-01) (9.6347571771E-01 9.6386000000E-01)
+ (9.6521483750E-01 9.5520000000E-01) (9.6695395728E-01 9.4556000000E-01)
+ (9.6869307707E-01 9.3460000000E-01) (9.7043217870E-01 9.2192000000E-01)
+ (9.7217137108E-01 9.0686000000E-01) (9.7391049086E-01 8.8908000000E-01)
+ (9.7564961064E-01 8.6722000000E-01) (9.7738873043E-01 8.3990000000E-01)
+ (9.7912783206E-01 8.0636000000E-01) (9.8086695185E-01 7.6520000000E-01)
+ (9.8260607163E-01 7.1436000000E-01) (9.8434519142E-01 6.6000000000E-01)
+ (9.8608438379E-01 6.0778000000E-01) (9.8782348543E-01 5.5698000000E-01)
+ (9.8956260521E-01 5.0776000000E-01) (9.9130172500E-01 4.5810000000E-01)
+ (9.9304084478E-01 4.0860000000E-01) (9.9477996457E-01 3.5850000000E-01)
+ (9.9651906620E-01 3.0526000000E-01) (9.9825825858E-01 2.4604600000E-01)
+ (9.9999737836E-01 1.8616600000E-01) (1.0000000000E+00 0.0000000000E+00)

.ENDS
```

Figure 153 *Nickel-Cadmium cell model valid for discharge rates from 0 to 10 C, where C is the one hour rated capacity in Amps (Continued)*

```
PSpice Lead-Acid battery discharge model
* Optimized for 6 and 12 Volt Lead-Acid batteries with capacities from
* 1.3 to 10 Amp-hours; discharge rates to 1 hour rate
*--- Nodes
* +OUTPUT, -OUTPUT = +/- cell connections (floating)
* SOC = state-of-charge output node, (1V=100%, 0V=0%)
* RATE=instantaneous discharge rate, (1V=C,10V=10C) referred to 20 hour rate
*--- Parameters
* CAPACITY = battery capacity in Amp-hours, 1=1A-hr, 0.5=0.5A-hr
* measured at 20 hour or greater rate
* RESISTANCE = total battery resistance in ohms
* CELLS = number of cells in battery (3 for 6V, 6 for 12V)
.SUBCKT LEADACID
+ +OUTPUT -OUTPUT SOC RATE
+ PARAMS: CAPACITY=1, RESISTANCE=1, CELLS=3
* * DISCHARGE RATE CALCULATION * *
E_Rate RATE 0 VALUE = { I(V_Sense)/CAPACITY }
R2 RATE 60 60 ;R2, C1 = 60 SECOND DELAY
C1 60 0 1
* * DISCHARGE AND STATE OF CHARGE * *
G_Discharge SOC 0 VALUE = { I(V_Sense) } ; Discharge Current
* * LOST CAPACITY DURING FAST DISCHARGE DELAYED BY R2-C1 * *
E_Lost_Rate 50 SOC TABLE { V(60) } =
+ (0.05,0.0) (0.089,0.11)(0.16,0.20)(0.62,0.39)(0.8,0.47) (1.6,0.44)
* * AMP-HOUR CAPACITY OF BATTERY * *
C_CellCapacity 50 0 { 3600 * CAPACITY * 1.15 }
R1 50 0 1G
* * CELL RESISTANCE * *
R_Cell 10 20 { RESISTANCE }
* * BATTERY OUTPUT VOLTAGE * *
E_Battery +OUTPUT 10 VALUE = { V(Cell_V) * CELLS }
* * CELL OUTPUT CURRENT SENSE * *
V_Sense -OUTPUT 20 0
```

Figure 154 *Lead-Acid model which actually models the more common 6 and 12 Volt batteries by multiplying the number of cells by the single cell voltage to get the total battery voltage (continued on next page).*

```
* * SINGLE Lead-Acid CELL OUTPUT VOLTAGE VS STATE OF CHARGE * *
E_Invert Invert 0 TABLE { V(SOC) } = (0,1) (1,0)
R4 Invert 0 1G
R5 Cell_V 0 1G
E_Cell Cell_V 0 TABLE { V(Invert) } =
+ (0.000E+00 2.171E+00) (5.222E-04 2.149E+00) (1.828E-03 2.128E+00)
+ (1.263E-01 2.101E+00) (4.908E-01 2.001E+00) (6.385E-01 1.949E+00)
+ (7.459E-01 1.900E+00) (7.834E-01 1.875E+00) (8.117E-01 1.850E+00)
+ (8.313E-01 1.826E+00) (8.436E-01 1.801E+00) (8.517E-01 1.773E+00)
+ (8.556E-01 1.750E+00) (8.591E-01 1.724E+00) (8.616E-01 1.702E+00)
+ (8.646E-01 1.676E+00) (8.677E-01 1.648E+00) (8.707E-01 1.623E+00)
+ (8.732E-01 1.600E+00) (8.850E-01 1.499E+00) (8.965E-01 1.401E+00)
+ (9.000E-01 1.333E+00) (1.000E+00 0.000E+00)
.ENDS
```

Figure 155 *Lead-Acid model which actually models the more common 6 and 12 Volt batteries by multiplying the number of cells by the single cell voltage to get the total battery voltage continued).*

```
PSpice Nickel-Metal-Hydride battery discharge mode
* Optimized for 4/5A and AA standard cells discharge rates from 0C to 5C.
* Note: This technology is new as of late 1993. The acutal performance of
* NIMH cells is likely to change quickly as the production bugs are worked out
* Use with care.
*--- Nodes
* +OUTPUT, -OUTPUT = +/- cell connections (floating)
* SOC = state-of-charge output node, (1V=100%, 0V=0%)
* RATE = instantaneous discharge rate, (1V=C,10V=10C)
*--- Parameters
* CAPACITY = cell capacity in Amp-hours, 1=1A-hr, 0.5=0.5A-hr
* measured at 5 hour rate
* RESISTANCE = total cell resistance in ohms
.SUBCKT NIMH
+ +OUTPUT -OUTPUT SOC RATE
+ PARAMS: CAPACITY=1, RESISTANCE=1
```

Figure 156 *Nickel-Metal-Hydride cell model; based on limited actual data since there are few commonly available cells to test; therefore, use with caution*

```
* * DISCHARGE RATE CALCULATION * *
E_Rate RATE 0 VALUE = { I(V_Sense)/CAPACITY }
R2 RATE 60 1 ; R2-C2 provide 3 second delayed time constant
C1 60 0 3
* * LOW RATE ADDITIONAL ENERGY LOOK-UP TABLE AND TRANSFER * *
E_LowRate LowRate 0 TABLE {V(RATE)} = (0,0) (0.001,0.15) (0.1,0.1) (0.2,0)
R3 LowRate 0 1G
G_LowRate 0 50 VALUE = { V(LowRate)*I(V_Sense) }
* * DISCHARGE AND STATE OF CHARGE * *
G_Discharge SOC 0 VALUE = { I(V_Sense) } ; Discharge Current
* * LOST CAPACITY DURING FAST DISCHARGE DELAYED BY R2-C1 * *
E_LostRate 50 SOC TABLE { V(60) } = (0.2,0.0) (1.0,0.15) (5,0.2)
* * AMP-HOUR CAPACITY OF BATTERY * *
C_CellCapacity 50 0 { 3600 * CAPACITY * 1.01 }
R1 50 0 1G


* * CELL RESISTANCE * *
R_Cell 20 30 { RESISTANCE }
* * CELL OUTPUT CURRENT SENSE * *
V_Sense -OUTPUT 30 0
* * CELL OUTPUT VOLTAGE VS STATE OF CHARGE * *
E_Invert Invert 0 TABLE { V(SOC) } = (0,1) (1,0)
R4 Invert 0 1G
E_Cell +OUTPUT 20 TABLE { V(Invert) } =
+(0.0000E+00 1.3346E+00) (7.0989E-03 1.3244E+00) (1.6327E-02 1.3144E+00)
+(2.9283E-02 1.3042E+00) (4.2593E-02 1.2942E+00) (6.8859E-02 1.2841E+00)
+(1.3008E-01 1.2733E+00) (4.3605E-01 1.2633E+00) (5.1165E-01 1.2532E+00)
+(5.8033E-01 1.2432E+00) (6.4635E-01 1.2331E+00) (7.0190E-01 1.2231E+00)
+(7.5834E-01 1.2130E+00) (8.0324E-01 1.2030E+00) (8.3075E-01 1.1929E+00)
+(8.5116E-01 1.1828E+00) (8.6820E-01 1.1727E+00) (8.8310E-01 1.1627E+00)
+(8.9641E-01 1.1527E+00) (9.0848E-01 1.1425E+00) (9.1860E-01 1.1324E+00)
+(9.2730E-01 1.1223E+00) (9.3475E-01 1.1122E+00) (9.4167E-01 1.1021E+00)
+(9.4841E-01 1.0919E+00) (9.5480E-01 1.0817E+00) (9.6013E-01 1.0716E+00)
+(9.6439E-01 1.0615E+00) (9.6776E-01 1.0515E+00) (9.7060E-01 1.0407E+00)
+(9.7291E-01 1.0299E+00) (9.7486E-01 1.0190E+00) (9.7663E-01 1.0080E+00)
+(9.7823E-01 9.9782E-01) (9.8001E-01 9.8706E-01) (9.8196E-01 9.7630E-01)
+(9.8391E-01 9.6612E-01) (9.8586E-01 9.5606E-01) (9.8799E-01 9.4542E-01)
+(9.9012E-01 9.3524E-01) (9.9225E-01 9.2518E-01) (9.9420E-01 9.1498E-01)
+(9.9580E-01 9.0400E-01) (9.9687E-01 8.9186E-01) (9.9740E-01 8.7990E-01)
+(9.9775E-01 8.6280E-01) (9.9793E-01 8.4818E-01) (9.9811E-01 8.2718E-01)
+(9.9828E-01 7.9518E-01) (9.9846E-01 7.4066E-01) (9.9864E-01 6.4712E-01)
+(9.9882E-01 5.1380E-01) (9.9899E-01 3.3476E-01) (1.0000E+00 0.0000E+00)
.ENDS
```

Figure 157 *Nickel-Metal-Hydride cell model; based on limited actual data since there are few commonly available cells to test; therefore, use with caution (continued)*

References

[1] Hageman, S.C. “PC-based power supply tester slashes setup time to minutes,” *Personal Engineering & Instrumentation News*, December 1990, Pages 65-66.

MicroSim, *Circuit Analysis Reference Manual, Version 5.3*, 1993, MicroSim Corporation, Irvine, CA.

DURACELL, *Alkaline Dioxide Batteries*, DURACELL INC., Bethel, CT.

SANYO, Various literature on Nickel-Cadmium batteries, Sanyo Electric Inc., Little Ferry, NJ.

Panasonic, *Sealed Lead-Acid Batteries - Technical Handbook*, Panasonic Industrial Co. Secaucus, NJ.

Panasonic, *Batteries*, Panasonic Industrial Co. Secaucus, NJ.

GATES, *Rechargeable Batteries - Applications Handbook*, 1992, ISBN 0-7506-9228-6, Butterworth-Heinemann, Stoneham, MA

General Electric, *Nickel-Cadmium Battery Application Engineering Handbook*, 1975, General Electric Company. (The GATES book above contains most if not all of the information from this discontinued publication)

Power-Sonic, *Battery* Selector Guide, Power-Sonic Corp., Redwood City, CA.

What Will Digital Worst-Case Timing Simulation Do For You?

The Design Center Source newsletter, January 1992

Now, with the "digital worst-case timing simulation" feature, you will be able to use PSpice version 5.1 (or later) to evaluate the timing behavior of your digital and mixed analog/digital designs as a function of component propagation delay tolerances.

Earlier versions of PSpice allow specification of component delay values (MINimum, TYPical, and MAXimum), and simulation using any one of these delays. With the introduction of the digital worst-case timing capability, you can now run your design in true "worst-case" mode, which simulates all devices with the full range of MIN through MAX delays.

Component propagation delays are expressed in the .MODEL parameters associated with component types, with -MN, -TY, and -MX suffixes (e.g., TPLHMN) representing MINimum, TYPical and MAXimum delay values. Usual practice is to obtain these values from the manufacturer's specification sheets for the components used in your design. In cases where some of these parameters are unspecified, PSpice can establish the missing values by extrapolation. Provided below is an example of MIN and MAX propagation delay specifications for a BUF primitive.

```
.MODEL T_BUF UGATE ( ; BUF timing model
+ TPLHMN=5ns TPLHTY=8ns TPLHMX=10ns
+ TPHLMN=9ns TPHLTY=10ns TPHLMX=15ns
+ )
```

Digital worst-case timing simulation is perhaps best thought of as a tool that can tell you whether or not your digital design will operate as expected, under the worst possible combination of component delay tolerances. In this regard, worst-case is superior to separate MIN and MAX simulations, which rely on observing circuit behavior only at the extremes of specified

tolerances. Consider the example in Figure 158: first using only d = MIN propagation delays, then only the MAX delays.

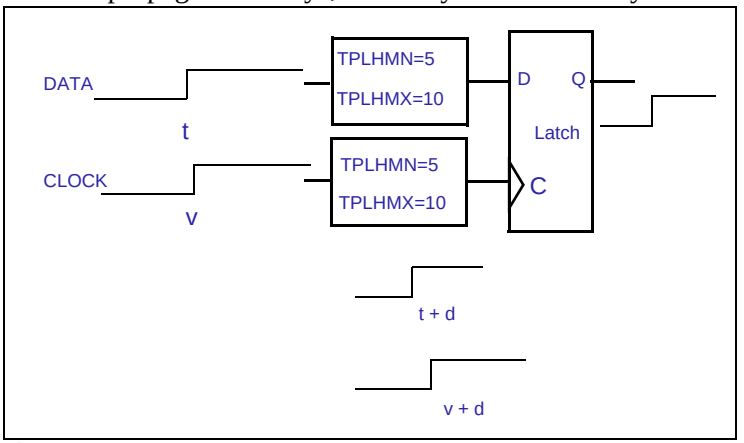


Figure 158 *Example of MIN or MAX propagation delay for BUF primitive*

Now, seeing that correct behavior is indeed observed at both extremes of the propagation delay range, consider the effects of having one of the components operating “slow” and the other “fast” as shown in Figure 159.

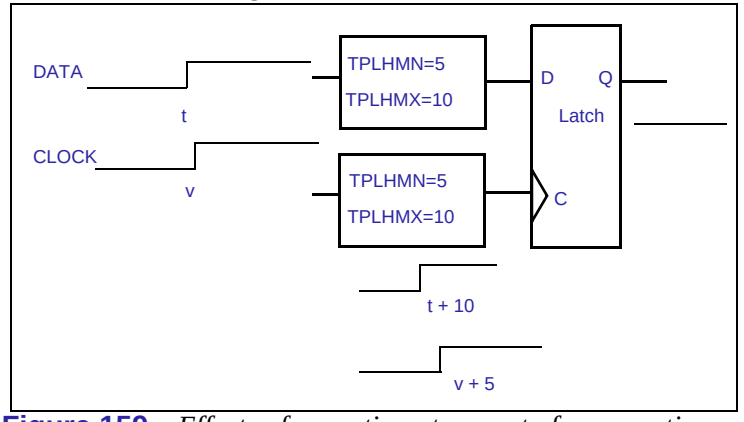


Figure 159 *Effects of operation at one set of propagation delay extremes*

Digital worst-case timing simulation will help you identify situations such as this, where the timing of signals is critical to the proper operation of the design. In a simple timing simulation (using one of MIN, TYP, or MAX delays), signal propagation through digital devices is normally represented as “instantaneous” transitions, such as those in the examples above. But during worst-case operation, the effects of individual

component delay ranges are propagated throughout the circuit. The “transitions” take both the MIN as well as the MAX delay characteristics of their propagation paths; therefore, transitions may be thought of as “regions of signal ambiguity.” This is due to the uncertainty of which delay value (MIN, MAX, or somewhere in between) actually applies to each component used in the design. PSpice represents this type of signal ambiguity with “Rising” (R), and “Falling” (F) logic levels.

The worst-case operation of the previous example is illustrated in Figure 160.

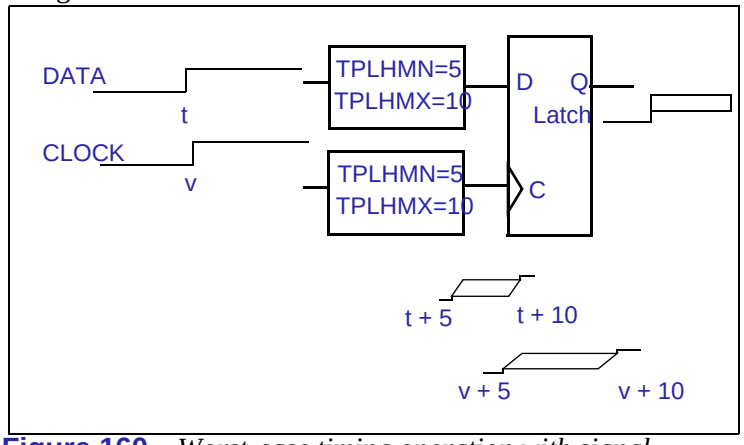


Figure 160 *Worst-case timing operation with signal ambiguities*

Note that, due to the uncertainty of the arrival time of both the data and the clock signals, the latch output is set to “X” (unknown).

Other tools called “timing verifiers” are sometimes used in the design process to identify problems that are indigenous to circuit definition. They yield analyses that are inherently pattern-*independent*, and very often pessimistic, in the sense that they tend to find more problems than will truly exist. This is due to the fact that they do not consider the actual usage of the circuit under applied stimuli. PSpice does not provide this type of “static” timing verification. Digital worst-case timing simulation, as provided by PSpice, is a pattern-*dependent* mechanism that allows a designer to locate timing problems subject to the constraints of specific applied stimuli.

The PSpice user’s guide, provided with the software package, contains a detailed description of this new digital worst-case

timing simulation mode, as well as a discussion of application methodology. By using digital worst-case timing simulation as an integral part of your design methodology, you can dramatically improve your chances of producing robust designs that have a high degree of immunity from the effects of varying combinations of individual component tolerances.

Worst-Case Analysis in PSpice

The Design Center Source newsletter, October 1993

Introduction

This article discusses the analog worst-case analysis feature of PSpice. Operation of this feature, underlying assumptions on which it is based, and any limitations are discussed. Simple examples are provided to illustrate the operation of this feature. It is hoped that the information provided here will give you a better understanding of this feature, helping you to apply it properly and with realistic expectations.

The examples are presented in circuit file syntax where the PSpice command, .WCASE (or .WC), is used to specify a worst-case analysis. For users of the MicroSim package with Schematic Capture, this type of analysis is set up using the Monte Carlo/Worst Case selection in the Analysis/Setup dialog. Please refer to the *Circuit Analysis Reference Manual* for detailed descriptions of worst-case analysis setup parameters that apply to your application system.

Analysis Description

Worst-case analysis is used to find the worst probable output of a circuit or system, given the restricted variance of its parameters. For instance, if the values of R1, R2, and R3 can vary by ±10%, then the worst-case analysis will attempt to find the combination of possible resistor values which result in the worst simulated output. As with any other analysis, there are three important parts: inputs, procedure, and outputs.

Inputs

Besides the circuit description, two forms of information are required from the user: (a) parameter tolerances, and (b) a definition of what *worst* means. PSpice allows tolerances to be set on any number of the parameters that characterize a model. Models can be defined for nearly all primitive analog circuit components—resistors, capacitors, inductors, semiconductor devices, etc. PSpice reads the standard model parameter tolerance syntax specified in the .MODEL statement. For each model parameter, PSpice uses the nominal, minimum, and maximum probable values, and the DEV and/or LOT specifiers; the probability distribution type (e.g., UNIFORM or GAUSS) is ignored.

The criterion for determining the *worst* values for the relevant model parameters is defined in the .WC statement as a function of any standard output variable in a specified range of the sweep. In a given range, the measurement must be reduced to a single value by one of these five collating functions:

- MAX
  Maximum output variable value
- MIN
  Minimum output variable value
- YMAX
  Output variable value at the point where it differs the most with the nominal run
- RISE_EDGE(value)
  Sweep value where the output variable value crosses *above* a given threshold value
- FALL_EDGE(value)
  Sweep value where the output variable value crosses *below* a given threshold value

Worst is user-defined as the highest (HI) or lowest (LO) possible collating function relative to the nominal run.

Note *Analog behavioral models can be used to measure waveform characteristics other than those detected by the available collating functions—e.g., rise time or slope. Analog behavioral models can also be used to incorporate several voltages and currents into one output variable to which a collating function may be applied.*

Procedure

To establish the initial value of the collating function, worst-case analysis begins with a nominal run with all model parameters at their nominal values. Next, multiple sensitivity analyses determine the individual effect of each model parameter on the collating function. This is accomplished by varying model parameters, one at a time, in consecutive simulations. The direction (*better* or *worse*) in which the collating function changes with a small increase in each model parameter is recorded. Finally, for the worst-case run, each parameter value is taken as far from its nominal as allowed by its tolerance, in the direction which should cause the collating function to be its worst (given by the HI or LO specification).

This procedure saves time by performing the minimum number of simulations required to make an educated guess at the parameter values which produce the worst results. It also has some limitations, which will be discussed in the following sections.

Outputs

A summary of the sensitivity analysis is printed in the PSpice output file (“.out”). This summary shows the percent change in the collating function corresponding to a small change in each model parameter. If a .PROBE statement is included in the circuit file, then the results of the nominal and worst-case runs are saved for viewing with the Probe graphical waveform analyzer.

An Important Condition for Correct Worst-Case Analysis

Worst-case analysis is not an optimization process. That is, it does not *search* for the set of parameter values which result in the worst result. It assumes that the worst case will occur when each parameter has been either pushed to one of its limits or left at its nominal value as indicated by the sensitivity analysis. It will show the true worst-case results when the collating function is *monotonic* within all tolerance combinations. Otherwise, there is no guarantee. Usually you cannot be certain if this condition is true, but insight into the operation of the circuit may alert you to possible anomalies.

The schematic shown in Figure 161 is for an amplifier circuit which is simply a biased BJT. This circuit will be used to demonstrate how a simple worst-case analysis works. It will also show how *non-monotonic* dependence of the output on a single parameter can adversely affect the worst-case analysis. Since an AC (small-signal) analysis is being performed, setting the input to unity means that the output, Vm([OUT]), is the magnitude of the gain of the amplifier. The only variable declared in this circuit is the resistance of *Rb2*. Since the value of *Rb2*

determines the bias on the BJT, it also affects the amplifier's gain.

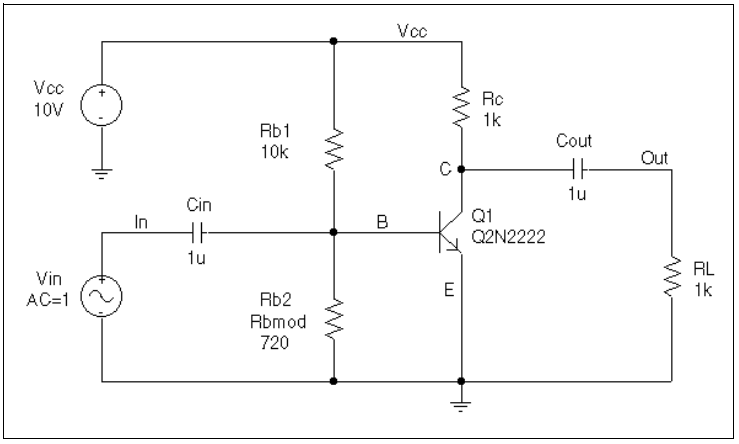


Figure 161 *Simple biased BJT amplifier*

Figure 162 is the circuit file used to run *either* a parametric analysis (.STEP; shown enabled in the circuit file) that sets the value of resistor *Rb2* by stepping model parameter R through values spanning the specified DEV tolerance range, *or* a worst-case analysis (shown disabled, i.e., commented out in the circuit file) that allows PSpice to determine the worst-case value for parameter R based upon a sensitivity analysis. PSpice will allow only one of these analyses to be run in any given simulation. Note that the AC and worst-case analysis specifications (.AC and .WC statements) are written so that the worst-case analysis tries to minimize Vm([OUT]) at 100 kHz.

```
* Worst-case analysis comparing monotonic and non-monotonic
* output with a variable parameter

.lib

***** Input signal and blocking capacitor *****
Vin     In        0         ac        1
Cin     In        B         1u

***** "Amplifier" *****
* gain increases with small increase in Rb2, but
* device saturates if Rb2 is maximized.
Vcc     Vcc       0         10
Rc      Vcc       C         1k
Q1      C         B         0         Q2N2222
Rb1     Vcc       B         10k
Rb2     B         0         Rbmod     720
.model Rbmod res(R=1 dev 5%)                    ; WC analysis results
                                                ;are correct
* .model Rbmod res(R=1.1 dev 5%)                 ; WC analysis misled
                                                ; by sensitivity

***** Load and blocking capacitor *****
Cout    C         Out       1u
Rl      Out       0         1k

* Run with either the .STEP or the .WC, but not both.
* This circuit file is currently set up to run the .STEP
* (.WC is commented out)

**** Parametric Sweep—providing plot of Vm([OUT]) vs. Rb2 ****
.STEP Res Rbmod(R) 0.8 1.2 10m

***** Worst-case analysis *****
* run once for each of the .model definitions stated above)
* WC AC Vm([Out]) min range 99k 101k list output all

.AC Lin 3 90k 110k
.probe
.end
```

Figure 162 *Amplifier netlist and circuit file set up to run either a parametric (.STEP) or worst-case (.WC) analysis of the specified AC analysis; these simulations demonstrate the conditions under which worst-case analysis works well and those that can produce misleading results when output is not monotonic with a variable parameter (see Figure 164 and Figure 165)*

For demonstration, the parametric analysis is run first, generating the curve shown in Figure 164 and Figure 165. This curve, derived using the YatX goal function shown in Figure 163, illustrates the non-monotonic dependence of gain on *Rb2*. To do this yourself, place the goal function definition in a “probe.gf” file in the circuit directory. Then run Probe, load all

of the AC sweeps, set up the X axis for performance analysis, and add the trace YatX(Vm([OUT]),100k).

```
YatX(1, X_value)=y1
{
 1|sfxv(X_value)!1;
}
```

Figure 163 *YatX goal function used on the simulation results for the parametric sweep (.STEP) defined in Figure 162; resulting curve shown in Figure 164 and Figure 165*

Next, the parametric analysis is commented out and the worst-case analysis is enabled. Two runs are made using the two versions of the *Rbmod* .MODEL statement shown in the circuit file. The model parameter, R, is a multiplier which is used to scale the nominal value of any resistor referencing the *Rbmod* model (*Rb2* in this case).

The first .MODEL statement leaves the nominal value of *Rb2* at 720 ohms. The sensitivity analysis increments R by a small amount and checks its effect on Vm([OUT]). This slight increase in R causes an increase in the base bias voltage of the BJT, and increases the amplifier's gain, Vm([OUT]). The worst-case analysis correctly sets R to its minimum value for the lowest possible Vm([OUT]) (see Figure 164).

The second .MODEL statement scales the nominal value of *Rb2* by 1.1 to approximately 800 ohms. The gain will still increase with a small increase in R, but a larger increase in R will increase the base voltage so much that it will drive the BJT into saturation and nearly eliminate the gain. The worst-case analysis will be fooled by the sensitivity analysis into assuming that *Rb2* must be minimized to degrade the gain, but maximizing *Rb2* is much worse (see Figure 165). Note that even an optimizer, which checks the local gradients to determine how the parameters should be varied, would be fooled by this circuit.

Consider a slightly different scenario: *Rb2* is 720 ohms, so that maximizing it is not enough to saturate the BJT; but *Rb1* is variable also. The true worst case occurs when *Rb2* is maximized and *Rb1* is minimized. Checking their individual

effects is not sufficient, even if the circuit were simulated four times with each resistor, in turn, set to its extreme values.

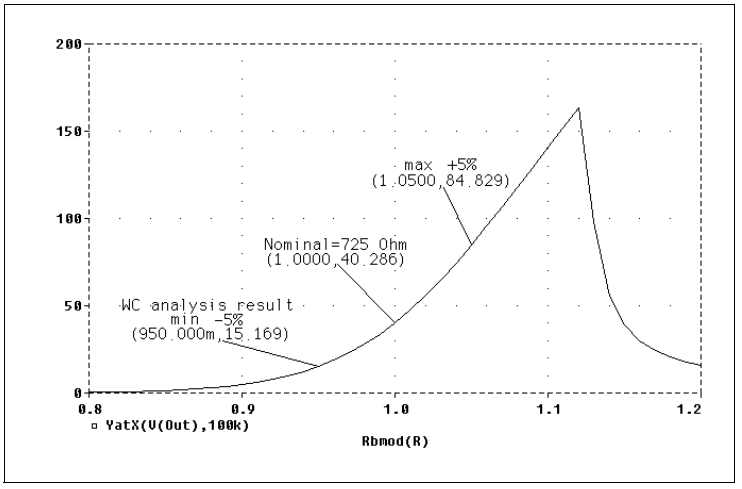


Figure 164 *Output is monotonic within the tolerance range; sensitivity analysis correctly points to the minimum value*

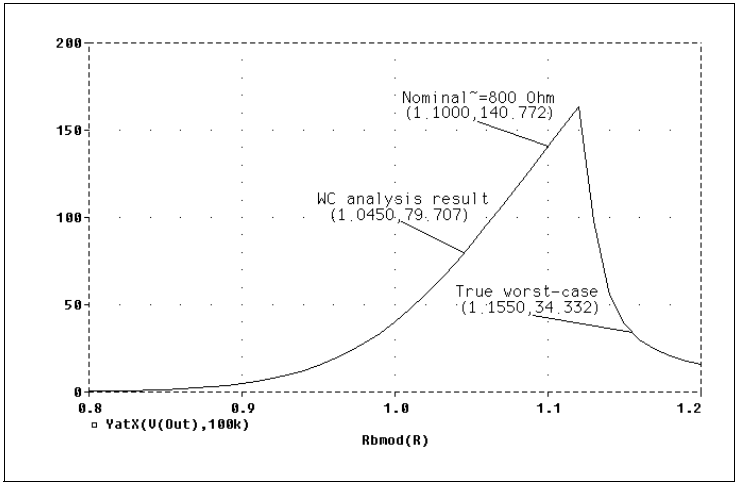
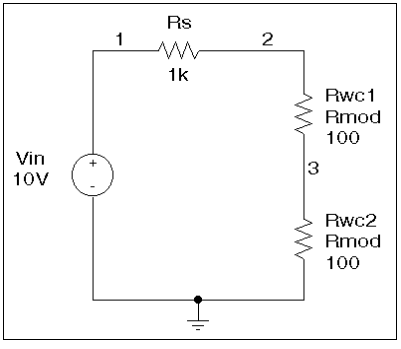


Figure 165 *Output is **non-monotonic** within the tolerance range thus producing incorrect worst-case results*

Hints and Other Useful Information

VARY BOTH, VARY DEV, and VARY LOT

When VARY BOTH is specified in the .WC statement and a model parameter is specified with both DEV and LOT tolerances defined, the worst-case analysis may produce unexpected results. The sensitivity of the collating function is only tested with respect to LOT variations of such a parameter; i.e., during the sensitivity analysis, the parameter is varied once affecting all devices referring to it and its effect on the collating function is recorded. For the worst-case analysis, the parameter is changed for all devices by LOT + DEV in the determined direction. Consider the example schematic and circuit file shown in Figure 166.




```
WCASE VARY BOTH Test

Vin   1       0       10V
Rs    1       2       1K
Rwc1  2       3       Rmod    100
Rwc2  3       0       Rmod    100
.MODEL Rmod RES(R=1 LOT 10% DEV 5%)
.DC   Vin     LIST    10
.WC   DC      V(3)    MAX     VARY BOTH      LIST
OUTPUT ALL
.ENDS
```

Figure 166 *Schematic and circuit file demonstrating use of VARY BOTH*

In this case, V(3) is maximized if:

1. *Rwc1* and *Rwc2* are both increased by 10% per the LOT tolerance specification, and then
2. *Rwc1* is decreased by 5% and *Rwc2* is increased by 5% per the DEV tolerance specification.

The final values for *Rwc1* and *Rwc2* should be 105 and 115, respectively. However, because *Rwc1* and *Rwc2* are varied together during the sensitivity analysis, it is assumed that both must be increased to their maximum for a maximum V(3). Therefore, both will be increased by 15%.

Here again, the purpose of the technique is to reduce the number of simulations. For a more accurate worst-case analysis, you should first perform a worst-case analysis with VARY LOT, manually adjust the nominal model parameter values according to the results, then perform another analysis with VARY DEV specified.

Gaussian Distributions

Parameters using Gaussian distributions are changed by 3σ (three times sigma) for the worst-case analysis.

YMAX Collating Function

The purpose of the YMAX collating function is often misunderstood. This function does not try to maximize the *deviation* of the output variable value from nominal. Depending on whether HI or LO is specified, it tries to maximize or minimize the *output variable value* itself at the point where maximum deviation occurred during sensitivity analysis. This *may* result in maximizing or minimizing the output variable value over the entire range of the sweep. This collating function is usually useful when you know the direction in which the maximum deviation will occur.

RELTOL

During the sensitivity analysis, each parameter is varied (multiplied) by 1+RELTOL where RELTOL is specified in a .OPTIONS statement, or defaults to 0.001.

Sensitivity Analysis

The sensitivity analysis results are printed in the output file (".out"). For each varied parameter, the percent change in the collating function and the sweep variable value at which the collating function was measured are given. The parameters are listed in *worst output* order; i.e., the collating function was its worst when the first parameter printed in the list was varied.

When the YMAX collating function is used, the output file will also list mean deviation and sigma values. These are based on the changes in the output variable from nominal at every sweep point in every sensitivity run.

Manual Optimization

Worst-case analysis can be used to perform *manual optimization* with PSpice. The monotonicity condition is usually met if the parameters have a very limited range. Performing worst-case analysis with tight tolerances on the parameters yields sensitivity and worst-case results (in the output file) which can be used to decide how the parameters should be varied to achieve the desired response. You can then make adjustments to the nominal values in the circuit file, and perform the worst-case analysis again for a new set of gradients. Parametric sweeps (.STEP), like the one performed in the circuit file shown in Figure 162, can be used to augment this procedure.

Monte Carlo Analysis

Monte Carlo (.MC) analysis may be helpful when worst-case analysis cannot be used. Monte Carlo analysis can often be used to verify or improve on worst-case analysis results. Monte Carlo analysis randomly selects possible parameter values, which can be thought of as randomly selecting points in the *parameter space*. The worst-case analysis assumes that the worst results will occur somewhere on the surface of this space, where parameters (to which the output is sensitive) are at one of their extreme values. If this is not true, the Monte Carlo analysis may find a point at which the results are worse. To try this, simply replace .WC in the circuit file with .MC *<#runs>*, where *<#runs>* is the number of simulations you are willing to perform. More runs will provide higher confidence results. To save disk space, do not specify any OUTPUT options. The Monte Carlo summary in the output file will list the runs in decreasing order of collating function value.

Now add the option

```
OUTPUT LIST RUNS <worst_run#>
```

to the .MC statement, and simulate again. This will perform only two simulations, the nominal and the worst Monte Carlo run. The parameter values used during the worst run will be written to the output file, and the results of both simulations will be saved.

Using Monte Carlo analysis with YMAX is a good way to obtain a conservative guess at the maximum possible deviation from nominal, since worst-case analysis usually cannot provide this information.

Voltage-Controlled Oscillators

The Design Center Source newsletter, July 1990

In this discussion, let's take a look at modeling Voltage Controlled Oscillators (VCOs), and see how several different VCOs can be modeled using PSpice. Most of the examples use PSpice's Analog Behavioral Modeling capabilities.

Sine Function VCO

A simple form of VCO is obtained by starting with the time domain function for a sinusoidal source:

```
esin  out  0  value {sin( (twopi * fc * time) + phi)}
rsin  out  0  1G
```

In this example, twopi, fc and phi are all (constant) global parameters defined with a .PARAM statement.

In order to run a simulation using the above example you will need to set the time steps taken by the simulator. A simple approach is to set the time step ceiling on the .TRAN command. For example, to view 10 cycles of a 1 MHz source, with 20 samples per cycle:

```
.tran  1us  10us  0  50ns
```

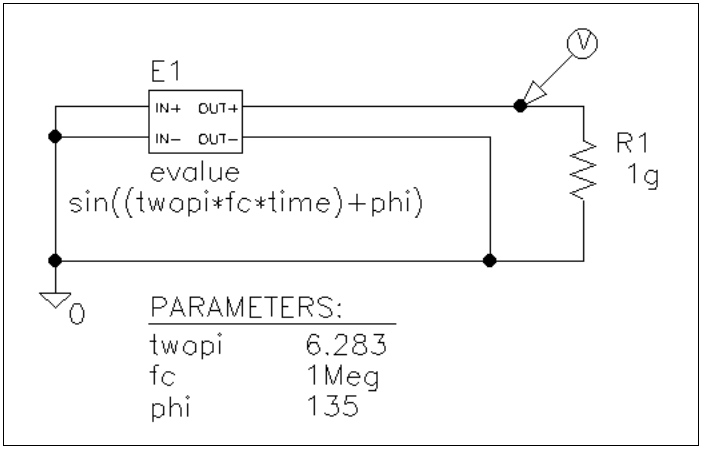


Figure 167 *Simple VCO circuit*

The single frequency source can be turned into a VCO by making phi a function of a controlling voltage instead of a constant:

$$y(t) = \sin(2\pi f_c t + \phi(t))$$

The instantaneous frequency is given by the time derivative of total phase:

$$2\pi f_{inst} = 2\pi f_c + \phi'(t)$$

The relationship between the frequency deviation, $f_d = f_{inst} - f_c$ and ϕ is given by:

$$\phi(t) = \int 2\pi f_d(t)dt$$

For a linear VCO we want f_d to be proportional to the controlling voltage v_{ctrl}, so:

$$\phi(t) = 2\pi k_1 \int v_{ctrl}(t)dt$$

where k_1 is in Hertz/volt.

Using PSpice, the integrator can be modeled as a controlled current source plus a capacitor. The varying phase term is added into the controlled voltage (sine) source:

```
evco1  out1  0  value {sin( twopi * (fc * time + v(int)))}
rvco1  out1  0  1G

gint   0     int  value {k1 * v(ctrl) * 1e-6}
cint   int   0    1u
rint   int   0    1G
.ic    v(int) = 0
```

To complete the example, a controlling voltage is required. Here is a stimulus that starts at 0 v, remains at this level for 5 µs, then steps to 1 v and stays there:

```
vstim  ctrl  0  pwl(0,0v  5us,0v   5.01us,1v)
rstim  ctrl  0  1G
```

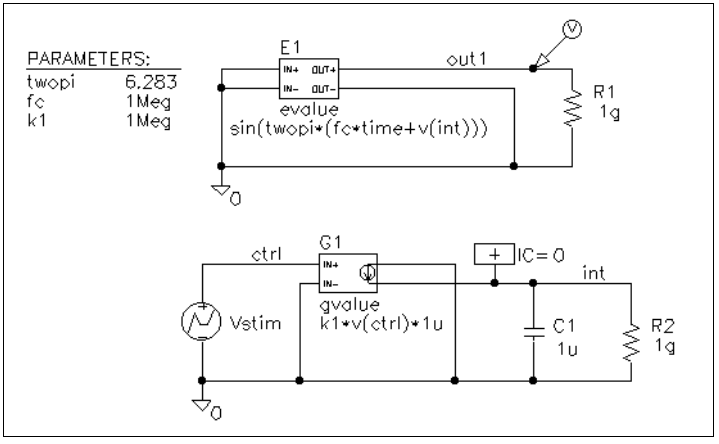


Figure 168 *VCO with variable phase*

Used with the VCO above and setting fc to 1 MHz and k1 to 1 MHz/volt gives an output signal that is 1 MHz for the first 5 µs and 2 MHz for the next 5 µs.

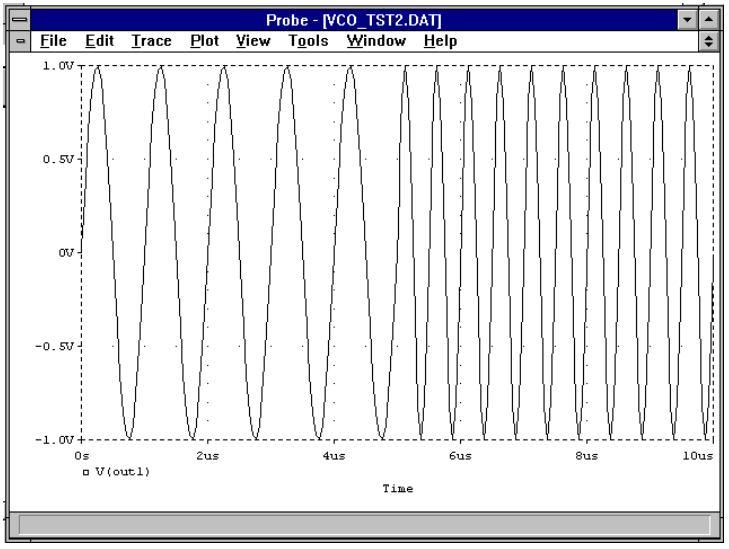
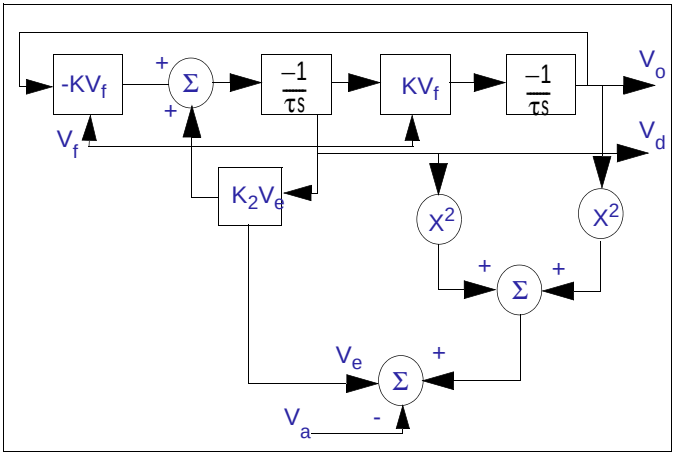


Figure 169 *Output of variable phase VCO*

Dual Integrator VCO

An alternate approach to a behavioral model VCO is to start from a 2-integrator loop. Changing the time constant of one or both integrators allows the frequency of oscillation to be controlled. Some form of limiting is required in order to produce output of bounded amplitude.

A particularly elegant example can be found in Reference[1]. This sinusoidal VCO has independent control of both frequency and amplitude. Its black-box representation reduces to:



Outputs vd and vo are in quadrature. The amplitude error term is obtained by squaring and summing the quadrature outputs, and subtracting the amplitude-setting voltage Va.

With some sleight-of-hand and use of the .FUNC command, the entire VCO can be reduced to two integrators with controlled current sources whose expressions incorporate the multiplications, additions and subtractions needed

```
* Filanovksy VCO

* .funcs for squarer and error term computation
.func ve(x,y,z) k*(pwr(x,2)+pwr(y,2)-pwr(z,2)
.param k=0.1, cv=10u

* first multiplier, summer, first integrator & error
g1 vd 0 value {k*((-
v(vo)*v(vf))+(ve(v(vo),v(vd),v(va))*v(vd)))}
c1 vd 0 {cv}
r1 vd 0 1G

* second multiplier and integrator
g2 vo 0 value {k*v(vd)*v(vf)}
c2 vo 0 {cv}
r2 vo 0 1G

* initial conditions
.ic  v(vd) = 0.1, v(vo) = 0

* controlling voltages (modify to suit)
va  va  0  1v                  ; amplitude
vf  vf  0  1v                  ; frequency

* analysis controls
.tran 10u 20m
```

Parameter values of 0.1 for k, and 10 u for cv, give a VCO with frequency of 1.5 kHz per volt at vf and amplitude of 3.2 v peak at va = 1 v.

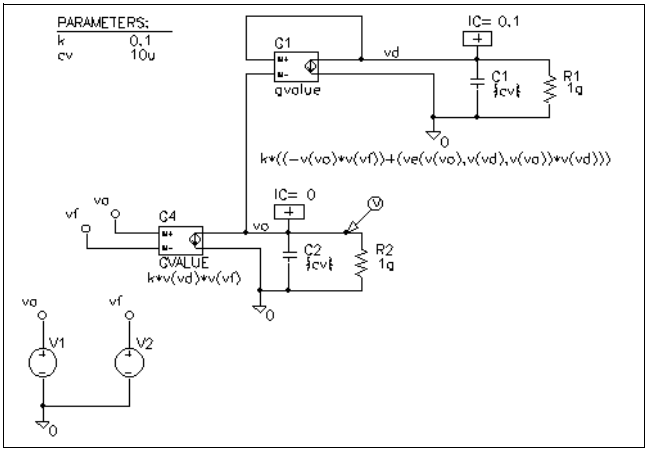


Figure 170 *Filanovsky VCO*

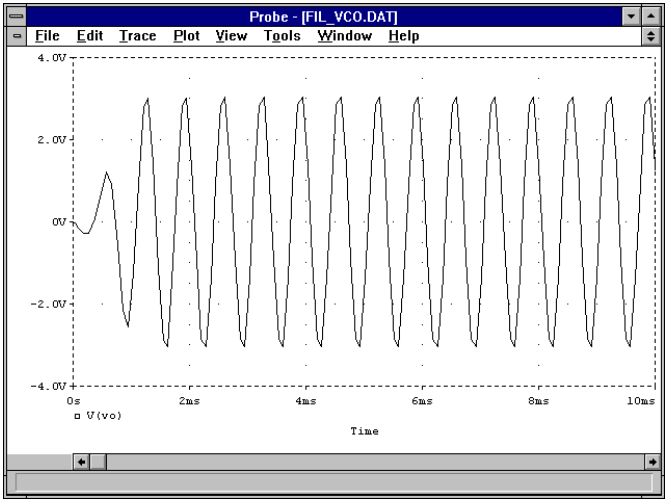


Figure 171 *Output of Filanovsky VCO*

Controlled Reactance VCO

A more circuit oriented approach to building a VCO is to take a simple tank oscillator and use a voltage-dependent reactance for one of the tank components. The example below is a Colpitts oscillator. The zx device (from MISC.LIB) is used to implement a voltage-controlled lossy inductor:

```
* Colpitts with variable tank element
xvi   ctrl   0      ref   vcc   c   zx
lref  ref    ref1   15u
rref  ref1   0      2.0
c1    c      e      20n
c2    e      0      200n
q1    c      0      e     q2n2222
re    e      vee    2k
vcc   vcc    0      10v
vee   vee    0      -10v
vctrl ctrl    0     pwl(0,1.5v   124u,1.5v   125u,2.5v)
.ic   v(c) = 10     v(e) = -0.7
.tran 10u    250u 0 1us
```

This simple VCO suffers from a number of drawbacks: variable output amplitude, square-root variation of frequency with control voltage, and slow start-up.

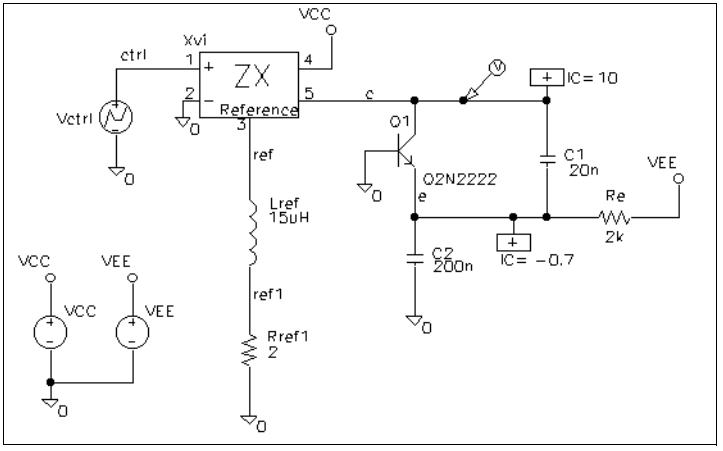


Figure 172 *Controlled Reactance VCO*

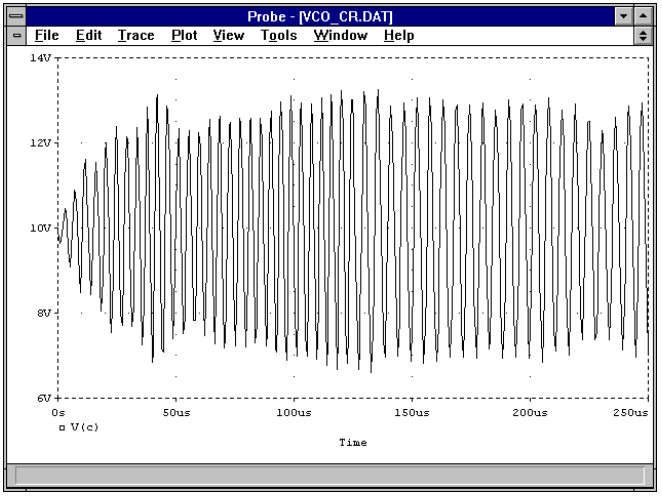


Figure 173 *Output of Controlled Reactance VCO*

Square Wave/Triangle Wave VCO

The previous examples have all been sine wave sources. If a square wave output is required, either a sine wave oscillator can be used and the output passed through a limiter (easily implemented in one line with a TABLE device), or a switching type oscillator can be used. The following example shows a current-steering VCO using ideal switches and current sources. Switches s1-s4 route the current flow around timing capacitor ct. A Schmitt trigger looks at the voltage across ct and reconfigures the switches for the next cycle.

```
* current-steering VCO
s1  2  1  4  0  mod1
s2  3  2  4  0  mod2
s3  1  0  4  0  mod2
s4  3  0  4  0  mod1
gt  5  3  value = {v(ctrl) * 1ma}
gd  1  0  value = {v(ctrl) * 1ma}
vs  5  0  20v
ct  2  0  500p
* Schmitt, behavioral model:
et  7  0  table {1e3*(0.91 + 0.55*v(4) -v(2))}  (0 0) (4 4)
ro  7  4  100
co  4  0  10p
vctrl  ctrl  0  pwl(0,1v  9u,1v  10u,2v)
rctrl  ctrl  0  1G
* output buffers:
esq    sqr   0   value = {v(4)}
rsq    sqr   0   1G
etri   tri 0   value = {v(2)}
rtri   tri 0   1G
.model mod1  vswitch(von=2.5 voff=2.6)
.model mod2  vswitch(von=2.6 voff=2.5)
.tran  1u  20u
.ic    v(2)=1.5,  v(4)=4
```

There are many more ways of modeling VCOs using PSpice. Hopefully, the methods outlined above will be a useful starting point.

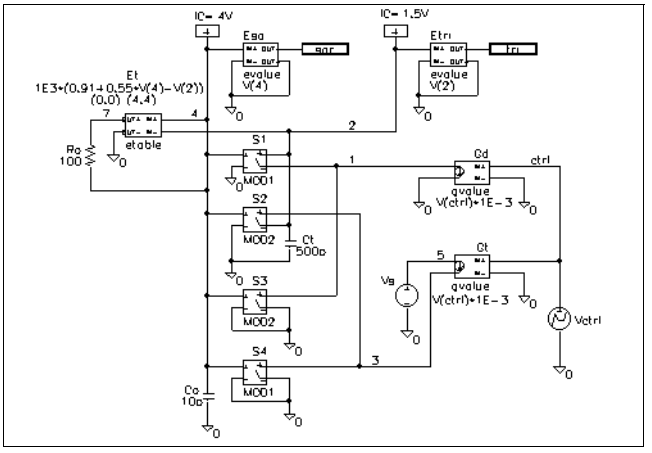


Figure 174 *Current Steering VCO circuit*

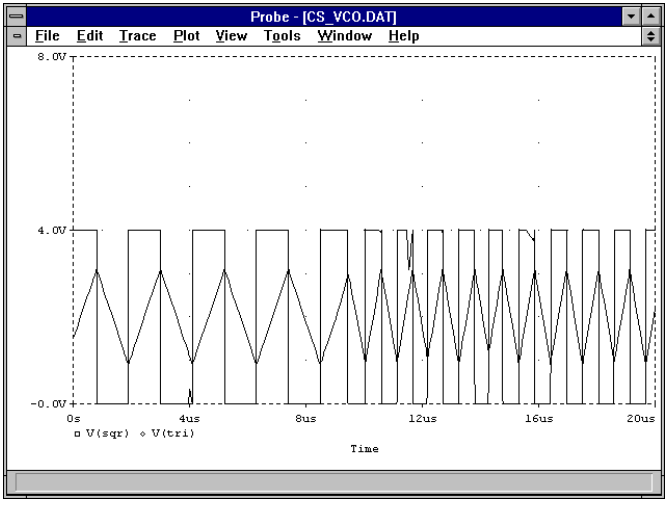


Figure 175 *Output of Current Steering VCO*

Reference

[1] I. M. Filanovsky, “Sinusoidal VCO with Control of Frequency and Amplitude,” *Proceedings of the 32nd Symposium on Circuits and Systems*, IEEE, Vol I, 446-449 (1989).