Chapter 10

Analyzing Electrical Yields

An electrical circuit must be designed to the tolerances associated with the specific manufacturing process. The electrical yield refers to the number of parts that meet the electrical test specifications. Maximizing the yield is important for the overall process efficiency. Star-Hspice analyzes the yield by using statistical techniques and observing the effects of element and model parameter variation.

Specifying Analytical Model Types

You can model parametric and statistical variation in circuit behavior in Star-Hspice by using:

- The .PARAM statement. Use to investigate the performance of a circuit as specified changes in circuit parameters are made.
- Monte Carlo Analysis. The statistical standard deviations of component values are known. Use for centering a design for maximum process yield, or for determining component tolerances.
- Worst Case Corners Analysis. The component value limits are known. Automates quality assurance for basic circuit function for process extremes, quick estimation of speed and power trade-offs, and best case and worst case model selection through parameter corners library files.
- Data Driven Analysis. Use for cell characterization, response surface, or Taguchi analysis. See *Chapter 19, Performing Cell Characterization*. Automates cell characterization, including timing simulator polynomial delay coefficient calculation. There is no limit on the number of parameters simultaneously varied or number of analyses to be performed. Convenient ASCII file format for automated parameter value generation. Can replace hundreds or thousands of Star-Hspice runs.

Yield analyses are used to modify DC operating points, DC sweeps, AC sweeps, and transient analysis. They can generate scatter plots for operating point analysis and family of curves plots for DC, AC, and transient analysis.

The .MEASURE statement in Star-Hspice is used with yield analyses, allowing you to see distributions of delay times, power, or any other characteristic described with a .MEASURE statement. Often, this is more useful than viewing a family of curves generated by a Monte Carlo analysis. When the .MEASURE statement is used, a table of results is generated as an *.mt#* file, which is in readable ASCII format, and can be displayed in AvanWaves. Also, when .MEASURE statements are used in a Monte Carlo or data driven analysis, the

Star-Hspice output file includes calculations for standard statistical descriptors:

$$Mean = \frac{x_1 + x_2 + \ldots + x_n}{N}$$

$$Variance = \frac{(x_1 - Mean)^2 + \ldots(x_n - Mean)}{N - 1}$$

$$Sigma = \sqrt{Variance}$$

$$Average\ Deviation = \frac{|x_1 - Mean| + \ldots + |x_n - Mean|}{N - 1}$$

Using Algebraic Expressions

You can replace any parameter defined in the netlist by an algebraic expression with quoted strings. Then, use these expressions as output variables in the .PLOT, .PRINT, and .GRAPH statements. Using algebraic expressions can expand your options in the input netlist file. Use algebraic expressions in the following ways:

- Scaling or changing of element and model parameters
- Parameterization:
```
.PARAM x=5
```
- Algebra:
```
.PARAM x=’y+3’
```
- Functions:
```
.PARAM rho(leff,weff)=’2+*leff*weff-2u’
```
- Hierarchical subcircuit algebraic parameter passing
```
.subckt inv in out wp=10u wn=5u qbar_ic=vdd
.ic qbar=qbar_ic
...
.ends
```
- Algebra in elements:
```
R1 1 0 r=’ABS(v(1)/i(m1))+10’
```
- Algebra in .MEASURE statements:
```
.MEAS vmax MAX V(1)
.MEAS imax MAX I(q2)
.MEAS ivmax PARAM=’vmax*imax’
```
- Algebra in output statements:
```
.PRINT conductance=PAR(‘i(m1)/v(22)’)
```

Star-Hspice uses double-precision numbers (15 digits) for expressions, user-defined parameters, and sweep variables. For better precision, use parameters instead of constants in algebraic expressions, since constants are only single-precision numbers (7 digits).

Algebraic Expressions for Output

The syntax is:

```
PAR('algebraic expression')
```

In addition to using quotations, the expression must be defined inside the PAR( ) statement for output.The continuation character for quoted parameter strings is a double backslash, “\\”. (Outside of quoted strings, the single backslash, “\”, is the continuation character.)

Using Built-In Functions

In addition to simple arithmetic operations (+, -, *, /), Star-Hspice provides a number of built-in functions that you can use in expressions, for example. The Star-Hspice built-in functions are listed in Table 10-1:.

Table 10-1: Star-Hspice Built-in Functions

HSPICE Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x (radians)
cos(x)	cosine	trig	Returns the cosine of x (radians)
tan(x)	tangent	trig	Returns the tangent of x (radians)
asin(x)	arc sine	trig	Returns the inverse sine of x (radians)
acos(x)	arc cosine	trig	Returns the inverse cosine of x (radians)
atan(x)	arc tangent	trig	Returns the inverse tangent of x (radians)
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x (radians)
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic sine of x (radians)
tanh(x)	hyperbolic tangent	trig	Returns the hyperbolic sine of x (radians)
abs(x)	absolute value	math	Returns the absolute value of x: |x|
sqrt(x)	square root	math	Returns the square root of the absolute value of x: sqrt(-x) = -sqrt(|x|)
pow(x,y)	absolute power	math	Returns the value of x raised to the integer part of y: $x^{(\text{integer part of y})}$
pwr(x,y)	signed power	math	Returns the absolute value of x raised to the y power, with the sign of x: (sign of x)$\|x\|^y$
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(|x|)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)$\log_{10}$(|x|)

Table 10-1: Star-Hspice Built-in Functions

HSPICE Form	Function	Class	Description
exp(x)	exponential	math	Returns *e* raised to the power x: e^x
db(x)	decibels	math	Returns the base 10 logarithm of the absolute value of x, multiplied by 20, with the sign of x: (sign of x)$20\log_{10}(\|x\|)$
int(x)	integer	math	Returns the largest integer less than or equal to x
sgn(x)	return sign	math	Returns -1 if x is less than 0, 0 if x is equal to 0, and 1 if x is greater than 0
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y: (sign of y)|x|
min(x,y)	smaller of two args	control	Returns the numeric minimum of x and y
max(x,y)	larger of two args	control	Returns the numeric maximum of x and y
lv(<Element>) or lx(<Element>)	element templates	various	Returns various element values during simulation. See “Element Template Output” on page 4-35 for more information.
v(<Node>), i(<Element>)...	circuit output variables	various	Returns various circuit values during simulation. See “Transient Sigma Sweep Results” on page 10-53 for more information.

Star-Hspice reserves the variable names listed in Table 10-2: for use in elements such as E, G, R, C, and L. You cannot use them for any other purpose in your netlist (in .PARAM statements, for example).

Table 10-2: Star-Hspice Special Variables

HSPICE Form	Function	Class	Description
time	current simulation time	control	Parameterizes the current simulation time during transient analysis.
temper	current circuit temperature	control	Parameterizes the current simulation temperature during transient/temperature analysis.
hertz	current simulation frequency	control	Parameterizes the frequency during AC analysis.

User-Defined Functions

An expression can contain parameters that have not yet been defined. A function must have at least one argument, and not more than two. Functions can be redefined.

Syntax

```
fname1 (arg1, arg2) = expr1 (fname2 (arg1, ...) = expr2) off
```

where:

fname	specifies function name. This parameter must be distinct from array names and built-in functions. Subsequently defined functions must have all their embedded functions previously defined.
arg1, arg2	specifies variables used in the expression
off	voids all user-defined functions

Example

```
f(a,b)=POW(a,2)+a*b  g(d)=SQRT(d)  h(e)=e*f(1,2)–g(3)
```

Using Parameters in Simulation

Parameters are similar to variables found in most programming languages. They hold a value that is either assigned at design time or calculated during the simulation based on circuit solution values.

Star-Hspice supports forward referencing of parameter names. That is, you can reference a parameter that you have not yet defined.

The Star-Hspice input:

```
.PARAM Third = Second
.PARAM Second = First
.PARAM First = 1
```

is equivalent to the following, in that all three parameters are set equal to 1:

```
.PARAM First = 1
.PARAM Second = First
.PARAM Third = Second
```

A parameter definition in Star-Hspice always takes the last value found in the Star-Hspice input (subject to local versus global parameter rules). Thus, the definitions below assign the value 3 to the parameter DupParam.

```
.PARAM DupParam= 1
...
.PARAM DupParam= 3
```

The value 3 will be substituted for all instances of DupParam, including instances that occur earlier in the input than the .PARAM DupParam=3 statement.

You can use parameters to represent a variable value for any element parameter (such as the width of a transistor), or, in some cases, simulation control parameters (such as the duration of a transient simulation).

```
.PARAM VddP= 3.3v$ Define the parameter values
.PARAM VssP= 0.0v
Vdd Vdd 0 VddP$ Define supplies
Vss Vss 0 VssP
.PARAM StepSize= 1ns
```

```
.PARAM Steps= 10
.TRAN Step=StepSize Start=0ns Stop='StepSize*Steps'
```

All parameter values in Star-Hspice are IEEE double floating point numbers.

Note: The statements above show the proper way to define supplies that will be changed using an .ALTER statement. ALTER does not support redefinition of voltage sources—only their parameterized values.

Parameter resolution order is as follows:

1. Resolve all literal assignments
2. Resolve all expressions
3. Resolve all function calls

Parameter passing order is shown in Table 10-3:.

Table 10-3: – Parameter Passing Order

.OPTION PARHIER=GLOBAL	**.OPTION PARHIER=LOCAL**
Analysis sweep parameters	Analysis sweep parameters
.PARAM statement (library)	.SUBCKT call (instance)
.SUBCKT call (instance)	.SUBCKT definition (symbol)
.SUBCKT definition (symbol)	.PARAM statement (library)

Parameter Definition

Parameters in Star-Hspice are names that have associated numeric values. You can use any of the following methods to define parameters:

■ Simple parameter	.PARAM <SimpleParam> = 1e-12
■ Algebraic defintition	.PARAM <AlgebraicParam> = ’SimpleParam*8.2’
■ User-defined function	.PARAM <MyFunc(x, y)> = ’Sqrt((x*x)+(y*y))’
■ Subcircuit default	.SUBCKT <SubName> <ParamDefName> = + <Value>
■ Predefined analysis function	PARAM <mcVar> = Agauss(1.0,0.1)
■ .MEASURE statement	MEASURE <MeasType> <MeasParam> + <MeasureClause>

Simple Parameter Value Assignment

A simple parameter assignment is a constant real number. The parameter keeps this value unless there is a later definition that changes its value, or it is assigned a new value by an algebraic expression during simulation. There is no warning if a parameter is reassigned.

Syntax:

```
.PARAM <ParamName> = <RealNumber>
```

Example

```
.PARAM TermValue = 1g
rTerm Bit0 0 TermValue
rTerm Bit1 0 TermValue
rTerm Bit2 0 TermValue
 ...
```

Algebraic Parameter Definition

Algebraic parameter assignments are made by an algebraic expression of real values, predefined function, user-defined function, or circuit or model values. A complex expression must be enclosed in single quotes in order to invoke the Star-Hspice algebraic processor. A simple expression consists of a single parameter name.

Syntax

```
.PARAM <ParamName> = ’<Expression>’    $ Quotes are mandatory
```

or

```
.PARAM <ParamName1> = <ParamName2>$ Cannot be recursive!
```

Examples

```
.PARAM npRatio= 2.1
.PARAM nWidth= 3u
.PARAM pWidth= ’nWidth * npRatio’
Mp1 ... <pModelName> W=pWidth
Mn1 ... <nModelName> W=nWidth
...
```

Note: To use an algebraic expression as an output variable in a .PRINT, .PLOT, or .PROBE statement, use the PAR keyword. For example:

```
  .PRINT DC v(3) gain=PAR(‘v(3)/v(2)’) PAR(‘v(4)/v(2)’)
```

The syntax is:

```
  .PARAM <ParamName>= ’<Expression>’
```

Examples

```
 .PARAM SqrtOf2       = ’Sqrt(2)’
 .PARAM Pi            = ’355/113’
 .PARAM Pi2           = ’2*Pi’
```

Define circuit values defined by a direct algebraic evaluation:

```
 r1 n1 0 R=’1k/sqrt(HERTZ)’ $ Resistance related to frequency.
```

User-Defined Function Parameters

A user-defined function assignment is similar to the definition of an algebraic parameter definition. Star-Hspice extends the algebraic parameter definition to include function parameters that are used in the algebraic that defines the function.

User defined functions cannot be nested more than three deep.

Syntax

```
.PARAM <ParamName>(<pv1>[, <pv2>])='<Expression>'
```

Example

```
.PARAM CentToFar (c)       = '(((c*9)/5)+32)'
.PARAM F(p1,p2)            = 'Log(Cos(p1)*Sin(p2))'
.PARAM SqrdProd (a,b)      = '(a*a)*(b*b)'
```

Subcircuit Default Definitions

The specification of hierarchical subcircuits allows you to pick default values for circuit elements. This is typically used in cell definitions so the circuit can be simulated with typical values.

Syntax

```
  .SUBCKT <SubName> <PinList> [<SubDefaultsList>]
```

where <*SubDefaultsList*> is

```
  <SubParam1> = <Expression> [<SubParam1> = <Expression> ...]
```

Subcircuit Default Definitions Examples

The following example implements an inverter with a Strength parameter. By default, the inverter can drive three devices. By entering a new value for the parameter Strength in the element line, the user can select larger or smaller inverters to suit the application.

```
.SUBCKT Inv a y Strength = 3
   Mp1 <MosPinList> pMosMod L=1.2u W='Strength * 2u'
   Mn1 <MosPinList> nMosMod L=1.2u W='Strength * 1u'
.ENDS
```

```
...
xInv0 a y0 Inv              $ Default devices: p device = 6u,
+ n device=3u
xInv1 a y1 Inv Strength=5  $ p device = 10u, n device = 5u
xInv2 a y2 Inv Strength=1  $ p device =  2u, n device = 1u
...
```

Scoping and Default Overrides Example

Given the input netlist fragment:

```
  .PARAM DefPwid = 1u

.SUBCKT Inv a y DefPwid = 2u DefNwid = 1u
   Mp1 <MosPinList> pMosMod L=1.2u W=DefPwid
   Mn1 <MosPinList> nMosMod L=1.2u W=DefNwid
.ENDS
```

with the global parameter scoping option .OPTION PARHIER=GLOBAL set, and the following input statements

```
...
xInv0 a y0 Inv                          $ Xinv0.Mp1 width = 1u
xInv1 a y1 Inv DefPwid=5u               $ Xinv1.Mp1 width = 5u
.MEASURE TRAN Wid0 PARAM = ’lv2(xInv0.Mp1)’ $ lv2 is the
.MEASURE TRAN Wid1 PARAM = ’lv2(xInv1.Mp1)’ $ template for the
                                        $ channel width
                                        $‘lv2(xInv1.Mp1)’
...
```

the following results are produced in the listing file:

```
wid0 =   1.0000E-06
wid1 =   1.0000E-06
```

With the local parameter scoping option .OPTION PARHIER=LOCAL set, and the following statements

```
...
xInv0 a y0 Inv                     $ Xinv0.Mp1 width = 1u
xInv1 a y1 Inv DefPwid=5u          $ Xinv1.Mp1 width = 1u:
.Measure TRAN Wid0 PARAM = ’lv2(xInv0.Mp1)’
$ override the global .PARAM
.Measure TRAN Wid1 PARAM = ’lv2(xInv1.Mp1)’
...
```

the following results are produced in the listing file:

```
 wid0 =   2.0000E-06
 wid1 =   5.0000E-06
```

Predefined Analysis Function

Star-Hspice has specialized analysis types, primarily Optimization and Monte Carlo, that require a method of controlling the analysis. This section describes the usage of these analysis types with respect to parameter definitions.

Optimization

For Star-Hspice optimization, define parameters with the .PARAM statement. This statement allows you to specify the parameters to optimize, the optimization range, and an optional incremental value.

The syntax is:

```
.PARAM <ParamName> = <OptParamFunc> (<Init>, <LoLim>, <HiLim>,
+  <Inc>)
```

where:

OptParmFunc	Optimization parameter function (string)
Init	Initial value of <ParamName> (real)
LoLim	Lower limit for <ParamName> (real)
HiLim	Upper limit for <ParamName> (real)
Inc	Rounds to nearest <Inc> value (real)

Example

```
.PARAM nWidth = WidOpt ( 5u, 1u, 10u, 0.1u )
...
.TRAN <TranTimeSpec> Sweep Optimize = WidOpt Results
+  = <MeasParamList>
...
```

Monte Carlo Analysis

The .PARAM statement is also used to define parameters for Star-Hspice Monte Carlo analysis. See Performing Monte Carlo Analysis, -39 for more information.

The syntax is:

```
.PARAM <ParamName> = <MonteFunc> (<MonteParamList>)
```

where:

MonteFunc	name of the predefined Monte Carlo function (string). See Table 10-4:.
MonteParamList	Monte Carlo function control parameter values (real)

Table 10-4: Syntax for Parameter Lists for Monte Carlo Analysis Functions

<MonteFunc>	Description	<MonteParamList>
UNIF()	Relative variation of a uniform distribution	nominal_val, rel_variation <, multiplier>
AUNIF()	Absolute variation of a uniform distribution	nominal_val, abs_variation <, multiplier>
GAUSS()	Relative variation of a Gaussian distribution	nominal_val, rel_variation, sigma <, multiplier>
AGAUSS()	Absolute variation of a Gaussian distribution	nominal_val, abs_variation, sigma <, multiplier>
LIMIT()	Random choice of limit values	nominal_val, abs_variation <, multiplier>

Example

```
  .PARAM LithOffsetX=AGAUSS(0u,0.1u,3)
  ...
  .TRAN <tran_time_spec> SWEEP MONTE=30
  ...
```

The statements above produce 30 randomly generated Gaussian-distributed values, centered on 0 μ and with a three-sigma spread of 0.1 μ.

Measurement Parameters

.MEASURE statements in Star-Hspice produce a type of parameter called a measurement parameter. In general, the rules for measurement parameters are the same as the rules for standard parameters, with one exception: measurement parameters are not defined in a .PARAM statement, but are defined directly in a .MEASURE statement.

Measurement parameter results produced by .PARAM statements in .SUBCKT blocks cannot be used outside the subcircuit. That means measurement parameters defined in .SUBCKT statements cannot be passed as bottom-up parameters in hierarchial designs.

Measurement parameter names cannot conflict with standard parameter names. Star-Hspice issues an error message if it encounters a measurement parameter with the same name as a standard parameter definition.

Syntax

```
.MEASURE <MeasType> <ParamName> = <MeasClause>
```

where:

MeasType	type of measurement to be performed: AC, DC or TRAN
MeasClause	the style of measurement performed: TRIG/TARG, FIND/ WHEN, .PARAM, or others

Example: Measurement Parameter Definition

```
.MEASURE TRAN tSlope Trig v(In) Val = ’Vdd * 0.1’ Rise = 1
+ Targ v(In) Val = ’Vdd * 0.9’ Rise = 1
...
.TRAN <TranTimeSpec>
...
```

The statements above measure the amount of time it takes for the voltage on the node *In* to change from 10% to 90% of Vdd.

Example: Measurement Parameter Conflicts

For the input netlist fragment

```
.PARAM tSlope = 1ns
...
.MEASURE TRAN tSlope Trig v(In) Val = ’Vdd * 0.1’ Rise = 1
+ Targ v(In) Val = ’Vdd * 0.9’ Rise = 1
```

Star-Hspice produces the following error message:

```
**error** measure name tslope conflicts with existing
parameter definition, please rename measure statement
```

The measurement statement

```
.MEASURE TRAN tSlope Trig v(In) Val = ’Vdd * 0.1’ Rise = 1
+ Targ v(In) Val = ’Vdd * 0.9’ Rise = 1
...
.PARAM tSlope = 1ns
```

produces the following error message:

```
**error**  above line attempts to redefine tslope
```

Parameter Scoping and Passing

Parameterized subcircuits provide a method of reducing the number of similar cells that must be created to provide enough functionality within your library. Star-Hspice allows you to pass circuit parameters into hierarchical designs, allowing you to configure a cell at runtime.

For example, if you parameterize the initial state of a latch in its subcircuit definition, then you can override this initial default in the instance call. Only one cell needs to be created to handle both initial state versions of the latch.

You also can parameterize a cell to reflect its layout. Parameterize a MOS inverter to simulate a range of inverter sizes with only one cell definition. In addition, you can perform Monte Carlo analysis or optimization on a parameterized cell.

The way you choose to handle hierarchical parameters depends on how you construct and analyze your cells. You can choose to construct a design in which information flows from the top of the design down into the lowest hierarchical levels. Centralizing the control at the top of the design hierarchy involves setting *global* parameters. You can also choose to construct a library of small cells that are individually controlled from within by setting *local* parameters, and build upwards to the block level.

This section describes the scope of Star-Hspice parameter names, and how Star-Hspice resolves naming conflicts between levels of hierarchy.

Library Integrity

Integrity is a fundamental requirement for any symbol library. Library integrity can be as simple as a consistent, intuitive name scheme, or as complex as libraries with built-in range checking.

You can risk poor library integrity when using libraries from different vendors in a single design. Because there is no standardization between vendors on what circuit parameters are named, it is possible that two components can include the same parameter name with different functions. Suppose that the first vendor builds a library that uses the name *Tau* as a parameter to control one or more subcircuits in their library. Now suppose that the second vendor uses *Tau* to control a different aspect of their library. Setting a global parameter named *Tau* to control one library also modifies the behavior of the second library, which might not be the intent.

When the *scope* of a higher level parameter is *global* to all subcircuits at lower levels of the design hierarchy, lower level parameter values are overridden by the values from higher level definitions if the names are the same. The scope of a lower level parameter is *local* to the subcircuit in which the parameter is defined (but global to all subcircuits that are even lower in the design hierarchy). The local scoping rules in Star-Hspice solve the problem of lower level parameters being overridden by higher level parameters of the same name when that is not desired.

Reusing Cells

Problems with parameter names also occur when different groups collaborate on a design. Because Star-Hspice global parameters prevail over local parameters, all designers are required to know the names of all parameters, even those used in sections of the design for which they are not responsible. This can lead to a large investment in standardized libraries. You can avoid this situation by using local parameter scoping that encapsulates all information about a section of a design within that section.

Creating Parameters in a Library

To ensure that critical, user-supplied parameters are present in a Star-Hspice netlist at simulation time, Star-Hspice allows the use of “illegal defaults”—that is, defaults that cause the simulator to abort if there are no overrides for the defaults.

Library cells that include illegal defaults require that the user provide a value for each and every instance of those cells. Failure to do so causes the Star-Hspice simulation to abort.

An example is the use of a default MOSFET width of 0.0. This causes Star-Hspice to abort because this parameter is required by the HSPICE MOSFET models.

Consider the following example:

Example 1

```
...
* Subcircuit default definition
.SUBCKT Inv A Y Wid = 0    $ Inherit illegal values by default
   mp1 <NodeList> <Model> L=1u W=’Wid*2’
   mn1 <NodeList> <Model> L=1u W=Wid
.ENDS

* Invocation of symbols in a design
x1 A Y1 Inv                $ Bad! No widths specified
x2 A Y2 Inv Wid = 1u       $ Overrides illegal value for Wid
```

This simulation would abort on subcircuit instance *x1* because the required parameter *Wid* is never set on the subcircuit instance line. Subcircuit *x2* would simulate correctly. Additionally, the instances of the *Inv* cell are subject to accidental interference because the global parameter *Wid* is exposed outside the domain of the library. Anyone could have specified an alternative value for the parameter in another section of the library or the design, which could have prevented the simulation from catching the condition present on *x1*.

Now consider the effect of a global parameter whose name conflicts with the library internal parameter *Wid*. Such a global parameter could be specified by the user or in a different library. In this example, the user of the library has specified a different meaning for the parameter *Wid* to be used in the definition of an independent source.

Example 2

```
.Param Wid = 5u            $ Default Pulse Width for source
v1 Pulsed 0 Pulse ( 0v 5v 0u 0.1u 0.1u Wid 10u )
...
* Subcircuit default definition
.SubCkt Inv A Y Wid = 0    $ Inherit illegals by default
   mp1 <NodeList> <Model> L=1u W='Wid*2'
   mn1 <NodeList> <Model> L=1u W=Wid
.Ends
* Invocation of symbols in a design
x1 A Y1 Inv                $ Incorrect width!
x2 A Y2 Inv Wid = 1u       $ Incorrect! Both x1 and x2
                           $ simulate with mp1=10u and
                           $ mn1=5u instead of 2u and 1u.
```

Under global parameter scoping rules, the simulation succeeds, although incorrectly. There is no warning message that the inverter x1 has no widths assigned, because the global parameter definition for *Wid* overrides the subcircuit default.

Note: Similarly, sweeping with different values of Wid dynamically changes both the Wid library internal parameter value and the pulse width value to the current sweep's Wid value.

In global scoping, the highest level name prevails in name conflict resolution. In local scoping, the lowest level name is used.

The parameter inheritance method allows you to specify that local scoping rules be used. This feature can cause different results than you have obtained with Star-Hspice releases prior to 95.1 on existing circuits.

With local scoping rules, the Example 2 netlist correctly aborts in x1 for W=0 (default Wid=0 in the .SUBCKT definition has higher precedence than the .PARAM statement), and results in the correct device sizes for x2. This change might affect your simulation results if a circuit like the second one shown above is created intentionally or accidentally.

As an alternative to width testing in the Example 2 netlist, it is also possible to achieve a limited version of library integrity with the .OPTION DEFW. This option specifies the default width for all MOS devices during a simulation. Because part of the definition is still in the domain of the top-level circuit, this method still suffers from the possibility of making unwanted changes to library values without notification by the simulator.

Table 10-5: outlines and compares the three primary methods for configuring libraries to achieve required parameter checking in the case of default MOS transistor widths.

Table 10-5: Methods for Configuring Libraries

Method	Parameter Location	Pros	Cons
Local	on a .SUBCKT definition line	The library is protected from global circuit parameter definitions unless the user wishes to override.	Cannot be used with releases of Star-Hspice prior to Release 95.1.
		Single location for default values.	

Table 10-5: Methods for Configuring Libraries

Method	Parameter Location	Pros	Cons
Global	at the global level and on .SUBCKT definition lines	Works with older Star-Hspice versions	The library can be changed by indiscrete user or other vendor assignment and by the intervening hierarchy.
			Cannot override a global value at a lower level.
Special	.OPTION DEFW statement	Simple to do	Third party libraries or other sections of the design might depend on the option DEFW.

Parameter Defaults and Inheritance

Use the .OPTION parameter PARHIER to specify scoping rules.

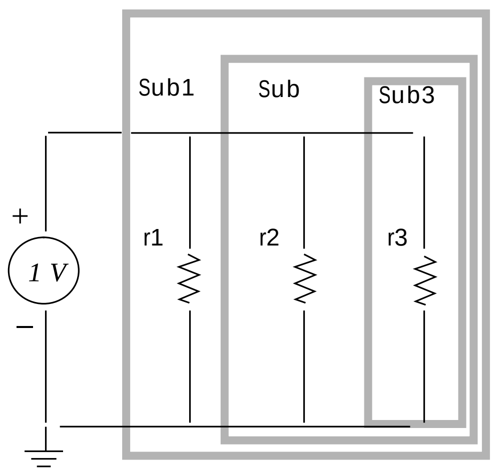
The syntax is:

```
.OPTION PARHIER = < GLOBAL | LOCAL >
```

The default setting is GLOBAL. To use the same scoping rules that HSPICE used prior to Release 95.1, set PARHIER to GLOBAL.

Figure 10-1: shows a flat representation of a hierarchical circuit that contains three resistors.

Each of the three resistors gets its simulation time resistance from the parameter named *Val*. The *Val* parameter is defined in four places in the netlist, with three different values.



```
TEST OF PARHIER
.OPTION list node post=2
ingold=2
+parhier=<Local|Global>
.PARAM Val=1
x1 n0 0 Sub1
.SubCkt Sub1 n1 n2 Val=1
   r1 n1 n2 Val
   x2 n1 n2 Sub2
.Ends Sub1
.SubCkt Sub2 n1 n2 Val=2
   r2 n1 n2 Val
   x3 n1 n2 Sub3
.Ends Sub2
.SubCkt Sub3 n1 n2 Val=3
   r3 n1 n2 Val
.Ends Sub3
```

Figure 10-1: Hierarchical Parameter Passing Problem

There are two possible solutions for the total resistance of the chain: 0.3333 Ω and 0.5455 Ω.

The PARHIER option allows you to specify which parameter value prevails when parameters with the same name are defined at different levels of the design hierarchy.

Under global scoping rules, in the case of name conflicts, the top-level assignment .PARAM Val=1 overrides the subcircuit defaults, and the total is 0.3333 Ω. Under local scoping rules, the lower level assignments prevail, and the total is 0.5455 Ω (one, two and three ohms in parallel).

The preceding example produces the results in Table 10-6:, based on the setting of the local/global PARHIER option:

Table 10-6: PARHIER=LOCAL versus PARHIER=GLOBAL Results

Element	PARHIER=Local	PARHIER=Global
r1	1.0	1.0
r2	2.0	1.0
r3	3.0	1.0

Anticipating and Detecting Parameter Passing Problems

Changes in scoping rules can cause different simulation results for designs created prior to Star-Hspice Release 95.1 from designs created after that release. Use the following checklist to determine if you will see simulation differences with the new default scoping rules. These checks are especially important if your netlists contain devices from multiple vendors' libraries.

- Check your subcircuits for parameter defaults on the .SUBCKT or .MACRO line.
- Check your subcircuits for a .PARAM statement within a .SUBCKT definition.
- Check your circuits for global parameter definitions using the .PARAM statement.
- If any of the names from the first three checks are identical, then set up two Star-Hspice jobs, one with .OPTION PARHIER = GLOBAL and one with .OPTION PARHIER = LOCAL, and look for differences in your output.

Solving Parameter Passing Problems

The easiest way to eliminate differences in your simulations between Release 95.1 and prior releases is to make sure global scoping rules are in effect (that is, make sure .OPTION PARHIER is *not* set to LOCAL). Global scoping rules are the default.

To set the default to GLOBAL for just a particular netlist, add the following line to the netlist:

```
.OPTIONS PARHIER = GLOBAL
```

To set the default to GLOBAL (which is the default if no PARHIER setting is specified) for all simulations, make the change in the Star-Hspice initialization file ($*installdir/hspice.ini*).

Note: This parameter is not recognized by releases prior to 95.1 and aborts the run if put into the hspice.ini file for previous releases.

Remember, for Star-Hspice simulation, the value of the last definition of a parameter is the final definition of the parameter.

Hierarchical Circuits, Parameters, and Models

This section describes hierarchical circuits, parameters, and models.

Subcircuits

Reusability is an important factor in circuit simulation efficiency. Subcircuits and cell macros are reusable circuits. To expand the utility of a cell macro, parameters are also used. SPICE provides the basic subcircuit, but has no provisions for the consistent naming of nodes. Star-Hspice provides a simple method for naming subcircuit nodes and elements: simply prefix the node or element with the subcircuit call name.

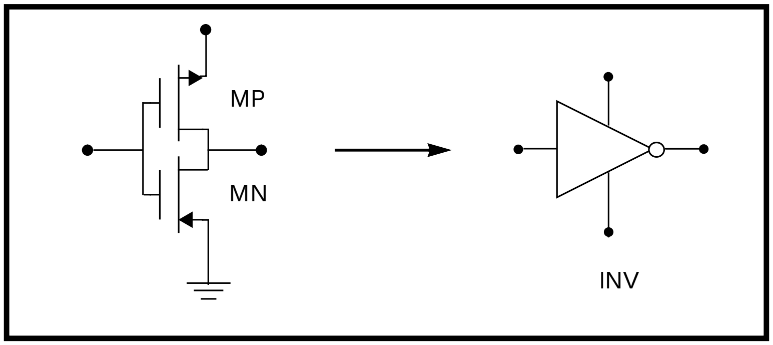


Figure 10-2: Circuit-to-Subcircuit Representation

The following Star-Hspice input creates an instance named X1 of the INV cell macro, which consists of two MOSFETs, MN and MP:

```
X1 IN OUT VD_LOCAL VS_LOCAL INV W=20
.MACRO INV IN OUT VDD VSS W=10 L=1 DJUNC=0
   MP OUT IN VDD VDD PCH W=W L=L DTEMP=DJUNC
   MN OUT IN VSS VSS NCH W='W/2' L=L DTEMP=DJUNC
.EOM
```

The MOSFETs MP and MN inside the cell macro INV called by X1 are accessed using their hierarchical names X1.MP and X1.MN. For example, to print the current through the MOSFETs:

```
.PRINT I(X1.MP)
```

The Multiply Parameter

The most basic subcircuit parameter is the M multiply parameter. This is a keyword common to all elements (except for voltage sources) and subcircuits. The multiply parameter multiplies the internal component values to give the effect of making parallel copies of the element or subcircuit. To simulate the effect of 32 output buffers switching simultaneously, only one subcircuit call needs to be placed, such as:

```
X1 in out buffer M=32
```

Multiply works hierarchically. A subcircuit within a subcircuit is multiplied by the product of the multiply parameters at both levels.

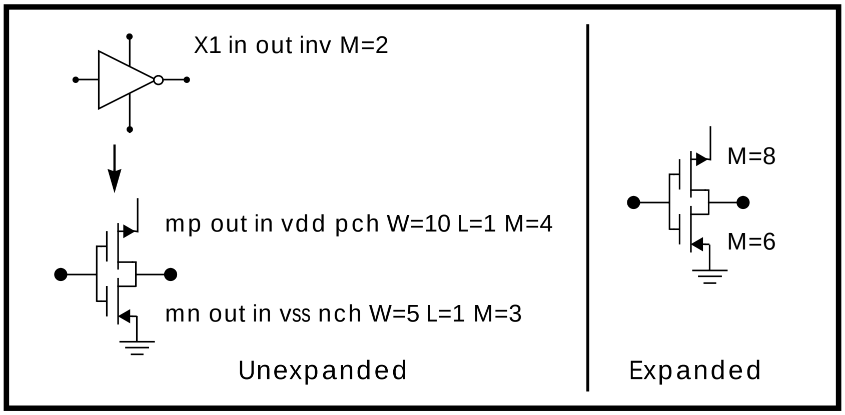


Figure 10-3: Hierarchical Parameters Simplify Flip-Flop Initialization

Example

```
X1 D Q Qbar CL CLBAR dlatch flip=0
.macro dlatch
+ D Q Qbar CL CLBAR flip=vcc
.nodeset v(din)=flip
xinv1 din qbar inv
xinv2 Qbar Q inv
m1 q CLBAR din nch w=5 l=1
m2 D CL din nch w=5 l=1
.eom
```

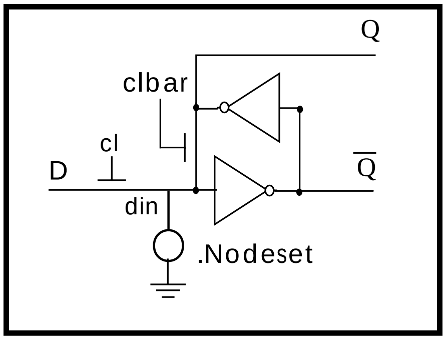
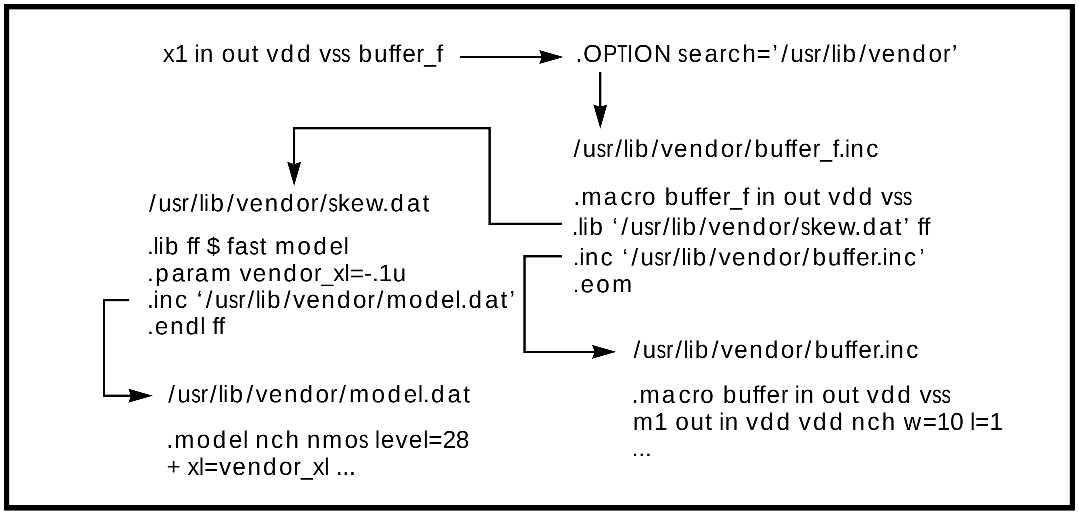


Figure 10-4: D Latch with Nodeset

Parameters in Vendor Libraries

The interface between commercial parts and circuit or system simulation is the vendor library. ASIC vendors provide comprehensive cells corresponding to inverters, gates, latches, and output buffers. Memory and microprocessor vendors generally supply input and output buffers. Interface vendors supply complete cells for simple functions and output buffers for generic family output. Analog vendors supply behavioral models. To avoid name and parameter conflicts, vendor cell libraries should keep their models within the subcircuit definitions.



```
x1 in out vdd vss buffer_f
```

```
.OPTION search='/usr/lib/vendor'
```

/usr/lib/vendor/buffer_f.inc

```
.macro buffer_f in out vdd vss
.lib '/usr/lib/vendor/skew.dat' ff
.inc '/usr/lib/vendor/buffer.inc'
.eom
```

/usr/lib/vendor/skew.dat

```
.lib ff $ fast model
.param vendor_xl=-.1u
.inc '/usr/lib/vendor/model.dat'
.endl ff
```

/usr/lib/vendor/buffer.inc

```
.macro buffer in out vdd vss
m1 out in vdd vdd nch w=10 l=1
...
```

/usr/lib/vendor/model.dat

```
.model nch nmos level=28
+ xl=vendor_xl ...
```

Figure 10-5: Vendor Library Usage

Varying Parameters and Libraries

Use the .ALTER statement to run simulations of the same basic circuit but using different library parameters.

Using the .ALTER Statement

Use the .ALTER statement to specify that a simulation be rerun using different:

- Circuit topologies
- Models
- Library components
- Elements
- Parameter values
- Options
- Test vectors
- Source stimulus (must parameterize)
- Analysis variables (must parameterize)
- Print and plot commands (must parameterize)

Altering Design Variables and Subcircuits

The following rules are used when altering design variables and subcircuits.

1. If the name of a new element, .MODEL statement, or subcircuit definition is identical to the name of an original statement of the same type, the new statement replaces the old. New statements are added to the input netlist file.
2. Element and .MODEL statements within a subcircuit definition can be changed and new element or .MODEL statements can be added to a subcircuit definition. Topology modifications to subcircuit definitions should be put into libraries and added with .LIB and deleted with .DEL LIB.
3. If a parameter name of a new .PARAM statement in the .ALTER module is identical to a previous parameter name, the new assigned value replaces the old.

4. If elements or model parameter values were parameterized when using .ALTER, these parameter values must be changed through the .PARAM statement. Do not redescribe the elements or model parameters with numerical values.
5. Options turned on by an .OPTION statement in an original input file or a .ALTER block can be turned off.
6. Only the actual altered input is printed for each .ALTER run. A special .ALTER title identifies the run.
7. .LIB statements within a file called with an .INCLUDE statement cannot be revised by .ALTER processing, but .INCLUDE statements within a file called with a .LIB statement can be accepted by .ALTER processing.

Using Multiple .ALTER Statements

For the first run, Star-Hspice reads the input file only up to the first .ALTER statement and performs the analyses up to that .ALTER statement. After the first simulation is completed, Star-Hspice reads the input between the first .ALTER and the next .ALTER or .END statement. These statements are then used to modify the input netlist file. Star-Hspice then resimulates the circuit. For each additional .ALTER statement, Star-Hspice performs the simulation preceding the first .ALTER statement, then performs another simulation using the input between the current .ALTER statement and the next .ALTER statement or the .END statement. If you do not want to rerun the simulation preceding the first .ALTER statement every time, put the statements preceding the first .ALTER statement in a library and use the .LIB statement in the main input file, and put a .DEL LIB statement in the .ALTER section to delete that library.

.DEL LIB Statement

The .DEL LIB statement is used with the .ALTER statement to remove library data from memory. The .DEL LIB statement causes the .LIB call statement with the same library number and entry name to be removed from memory the next time the simulation is run. A .LIB statement can then be used to replace the deleted library.

Syntax

```
.DEL LIB ‘<filepath>filename’ entryname
.DEL LIB libnumber entryname
```

where:

entryname	entry name used in the library call statement to be deleted
filename	name of a file for deletion from the data file. The file path plus file name can be up to 64 characters in length and can be any file name that is valid for the operating system being used. The file path and name must be enclosed in single or double quote marks.
filepath	path name of a file, if the operating system supports tree-structured directories
libnumber	library number used in the library call statement to be deleted

Performing Worst Case Analysis

Worst case analysis often is used for design and analysis of MOS and BJT IC circuits. The worst case is simulated by taking all variables to their 2-sigma or 3-sigma worst case values. Since it is unlikely that several independent variables will attain their worst case values simultaneously, this technique tends to be overly pessimistic, and can lead to over-designing the circuit. However, it is useful as a fast check.

Model Skew Parameters

Avant! has extended the models in Star-Hspice to include physically measurable model parameters. The parameter variations allow the circuit simulator to predict the actual circuit response to the extremes of the manufacturing process. The physically measurable model parameters are called “skew” parameters because they are skewed from a statistical mean to obtain the predicted performance variations.

Examples of skew parameters are the difference between the drawn and physical dimension of metal, polysilicon, or active layers of an integrated circuit.

Generally, skew parameters are chosen independent of each other, so that combinations of skew parameters can be used to represent worst cases. Typical skew parameters for CMOS technology include:

- XL – polysilicon CD (critical dimension of poly layer representing the difference between drawn and actual size)
- XW_n, XW_p – active CD (critical dimension of active layer representing the difference between drawn and actual size)
- TOX – gate oxide thickness
- RSH_n, RSH_p – active layer resistivity
- $DELVTO_n$, $DELVTO_p$– threshold voltage variation

These parameters are allowed in any level MOS model in Star-Hspice. The DELVTO parameter simply shifts the threshold value. It is added to VTO for the Level 3 model and is added to or subtracted from VFB0 for the BSIM model.

Table 10-7: shows whether deviations are added to or subtracted from the average.

Table 10-7: Sigma Deviations

Type	Param	Slow	Fast
NMOS	XL	+	-
	RSH	+	-
	DELVTO	+	-
	TOX	+	-
	XW	-	+
PMOS	XL	+	-
	RSH	+	-
	DELVTO	-	+
	TOX	+	-
	XW	-	+

Skew parameters are chosen based on the available historical data collected either during fabrication or electrical test. For example, the poly CD skew parameter XL is collected during fabrication. This parameter is usually the most important skew parameter for a MOS process. Historical records produce data as shown in Figure 10-6:.

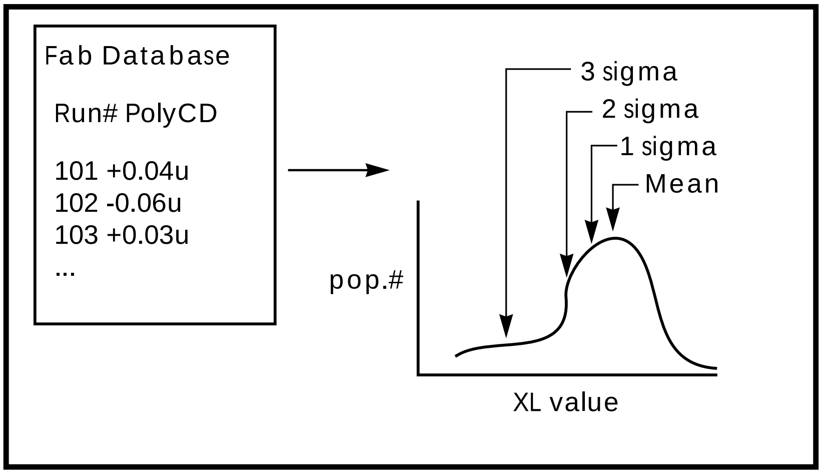


Figure 10-6: Historical Records for Skew Parameters in a MOS Process

Using Skew Parameters in Star-Hspice

The following example shows how to create a worst case corners library file for a CMOS process model. The physically measured parameter variations must be chosen so that their proper minimum and maximum values are consistent with measured current (IDS) variations. For example, a 3-sigma variation in IDS can be generated from a 2-sigma variation in the physically measured parameters.

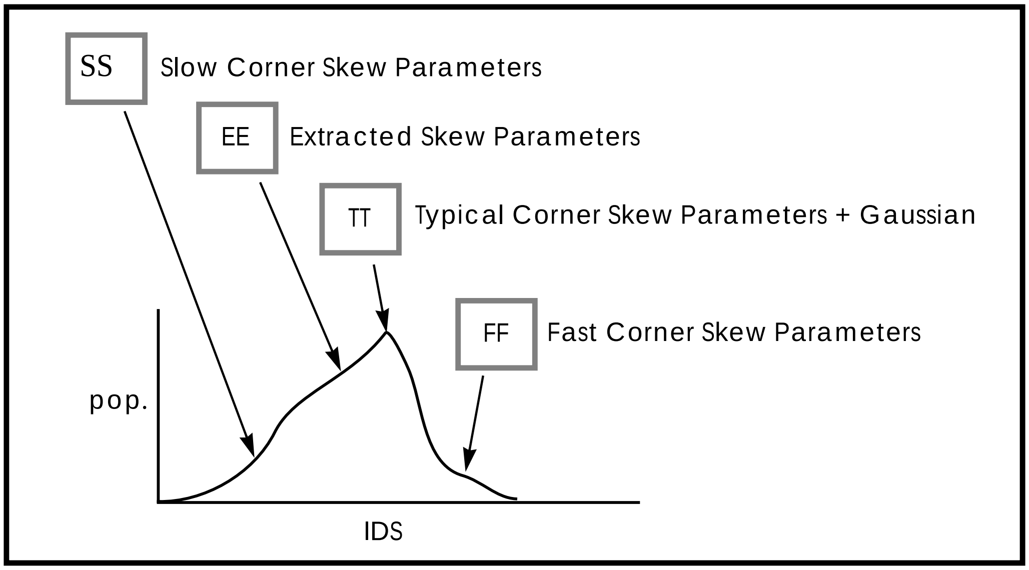

SS
Slow Corner Skew Parameters
EE
Extracted Skew Parameters
TT
Typical Corner Skew Parameters + Gaussian
FF
Fast Corner Skew Parameters
pop.
IDS

Figure 10-7: Worst Case Corners Library File for a CMOS Process Model

The simulator accesses the models and skew through the .LIB library statement and the .INCLUDE include file statement. The library contains parameters that modify .MODEL statements. The following example of .LIB of model skew parameters features both worst case and statistical distribution data. The statistical distribution median value is the default for all non-Monte Carlo analysis.

Example of Skew Parameter Overlay for a Model

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY DATE:3/4/91
$ PROCESS: 1.0U CMOS, FAB22, STATISTICS COLLECTED 3/90-2/91
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN

.PARAM
$ polysilicon Critical Dimensions
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
$ Active layer Critical Dimensions
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
$ Gate Oxide Critical Dimensions (200 angstrom +/- 10a at 1
$ sigma)
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'
$ Threshold voltage variation
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'

.INC '/usr/meta/lib/cmos1_mod.dat'$ model include file

.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES

.PARAM TOX=230 XL=-0.18u DELVTON=-.15V DELVTOP= 0.15V
.INC '/usr/meta/lib/cmos1_mod.dat'$ model include file

.ENDL FF
```

The model would be contained in the include file */usr/meta/lib/cmos1_mod.dat*.

```
.MODEL NCH NMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTON .....
.MODEL PCH PMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTOP .....
```

Note: The model keyname (left-hand side) is being equated to the skew parameter (right-hand side). Model keynames and skew parameters can have the same names.

Skew File Interface to Device Models

The skew parameters are model parameters. They are used most often for transistor models, but they also apply to passive components. A typical device model set includes:

- MOSFET models for all device sizes using automatic model selector
- RC wire models for polysilicon, metal1, and metal2 layers (These models include temperature coefficients and fringing capacitance. They apply to drawn dimension)
- Single and distributed diode models for N+,P+, and well (includes temperature, leakage, and capacitance based on drawn dimension)

- BJT models for parasitic bipolar transistors, as well as any special BJTs such as a BiCMOS for ECL BJT process (includes current and capacitance as a function of temperature)
- Metal1 and metal2 transmission line models for long metal lines
- Models must be able to accept elements with sizes based on drawn dimension. A cell might be drawn at 2 μ dimension, shrink to 1 μ, with a physical size of 0.9 μ and an effective electrical size of 0.8 μ. The following dimension levels must be accounted for:

  drawn size
  shrunken size
  physical size
  electrical size

Note: *Most simulator models go directly from drawn size to the electrical size. Star-Hspice is designed to support all four size levels for MOS models. Figure 10-8: shows the importance of the four size levels.*

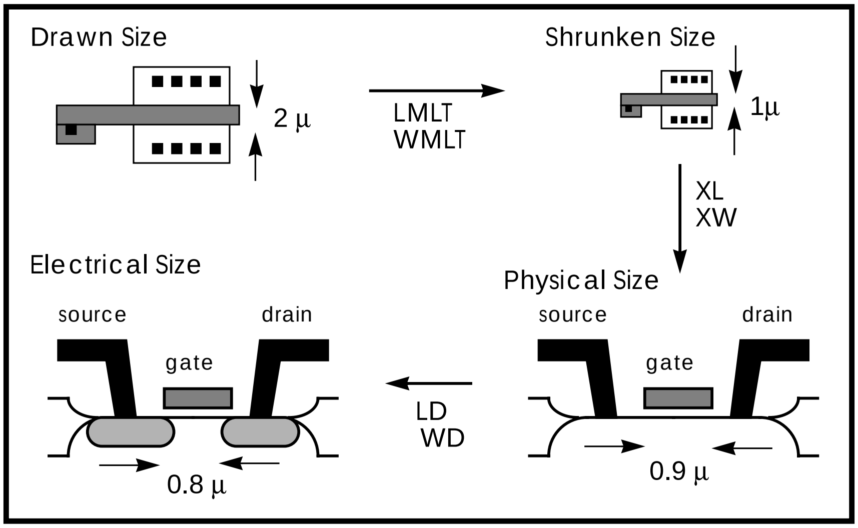


Figure 10-8: Device Model from Drawn to Electrical Size

Performing Monte Carlo Analysis

Monte Carlo analysis uses a random number generator to create the following types of functions:

Gaussian Parameter Distribution

- Relative variation – variation is a ratio of average
- Absolute variation – variation is added to average
- Bimodal – nominal parameters are statistically reduced by multiplication of distribution

Uniform Parameter Distribution

- Relative variation – variation is a ratio of average
- Absolute variation – variation is added to average
- Bimodal – nominal parameters are statistically reduced by multiplication of distribution

Random Limit Parameter Distribution

- Absolute variation – variation is added to average
- Min or max variation is randomly selected

The number of times the operating point, DC sweep, AC sweep, or transient analysis is performed is determined by the value of the analysis keyword MONTE.

Monte Carlo Setup

To set up a Monte Carlo analysis, use the following Star-Hspice statements:

- .PARAM statement – sets a model or element parameter to a Gaussian, Uniform, or Limit function distribution.
- .DC, .AC, or .TRAN analysis – enable MONTE.
- .MEASURE statement – calculates output mean, variance, sigma, and standard deviation.

Analysis Syntax

Select the type of analysis desired, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating point:

```
.DC MONTE=val
```

DC sweep:

```
.DC vin 1 5 .25 SWEEP MONTE=val
```

AC sweep:

```
.AC dec 10 100 10meg SWEEP MONTE=val
```

TRAN sweep:

```
.TRAN 1n 10n SWEEP MONTE=val
```

The value “val” represents the number of Monte Carlo iterations to be performed. A reasonable number is 30. The statistical significance of 30 iterations is quite high. If the circuit operates correctly for all 30 iterations, there is a 99% probability that over 80% of all possible component values operate correctly. The relative error of a quantity determined through Monte Carlo analysis is proportional to val$^{-1/2}$.

Monte Carlo Output

Use .MEASURE statements as the most convenient way to summarize the results.

The .PRINT statement generates tabular results and prints all Monte Carlo parameter usage values. If one iteration is out of specification, obtain the component values from the tabular listing. A detailed resimulation of that iteration might help identify the problem.

.GRAPH generates a high resolution plot for each iteration. In contrast, AvanWaves superimposes all iterations as a single plot and allows you to analyze each iteration individually.

.PARAM Distribution Function Syntax

A .PARAM parameter is assigned to the keywords of Star-Hspice elements and models. A distribution function is assigned to each .PARAM parameter. The distribution function is recalculated for each element or model keyword use of a parameter. This feature allows a parameterized schematic netlist to be used for Monte Carlo analysis with no additional modifications.

The syntax is:

```
.PARAM xx=UNIF(nominal_val, rel_variation <, multiplier>)
```

or

```
.PARAM xx=AUNIF(nominal_val, abs_variation <, multiplier>)
```

or

```
.PARAM xx=GAUSS(nominal_val, rel_variation, sigma <,
+  multiplier>)
```

or

```
.PARAM xx=AGAUSS(nominal_val, abs_variation, sigma <,
+  multiplier>)
```

or

```
.PARAM xx=LIMIT(nominal_val, abs_variation)
```

where:

xx	the parameter whose value is calculated by distribution function
UNIF	uniform distribution function using relative variation
AUNIF	uniform distribution function using absolute variation
GAUSS	Gaussian distribution function using relative variation
AGAUSS	Gaussian distribution function using absolute variation
LIMIT	random limit distribution function using absolute variation, +/- abs_variation is added to nominal_val based on whether the random outcome of a -1 to 1 distribution is greater or less than 0.

nominal_val	nominal value for Monte Carlo analysis and default value for all other analyses
abs_variation	the AUNIF and AGAUSS vary the nominal_val by +/- abs_variation
rel_variation	the UNIF and GAUSS vary the nominal_val by +/- (nominal_val · rel_variation)
sigma	the abs_variation or rel_variation is specified at the sigma level. For example, if sigma=3, then the standard deviation is abs_variation divided by 3.
multiplier	If not specified, the default is 1. The calculation is repeated this many times and the largest deviation is saved. The resulting parameter value might be greater or less than nominal_val. The resulting distribution is bimodal.

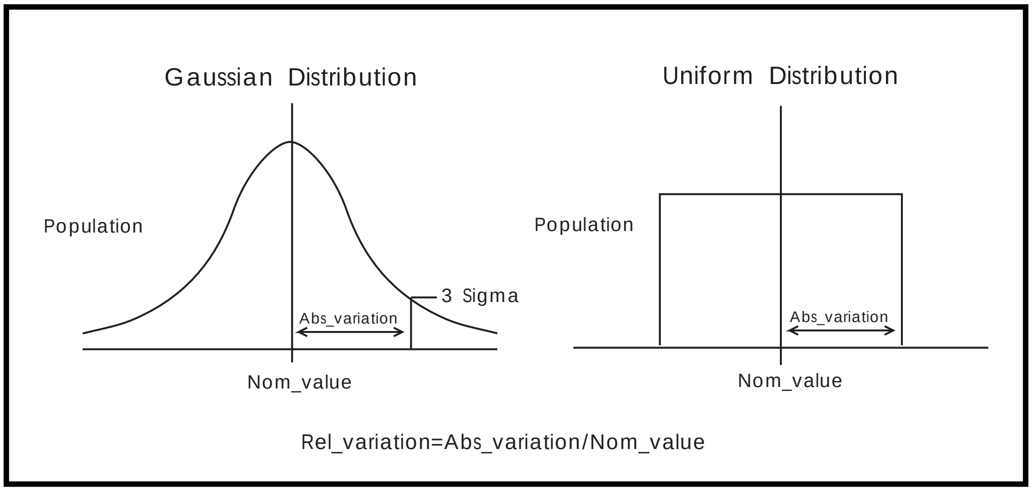


Figure 10-9: Monte Carlo Distribution

Monte Carlo Parameter Distribution Summary

A new random variable is calculated each time a parameter is used. If no Monte Carlo distribution is specified, then the nominal value is assumed. When a Monte Carlo distribution is specified for only one analysis, the nominal value is used for all other analyses.

You can assign a Monte Carlo distribution to all elements that share a common model. The actual element value will vary by the element distribution. A Monte Carlo distribution also can be assigned to a model keyword, and all elements that share that model use the same keyword value. This allows double element and model distributions to be created.

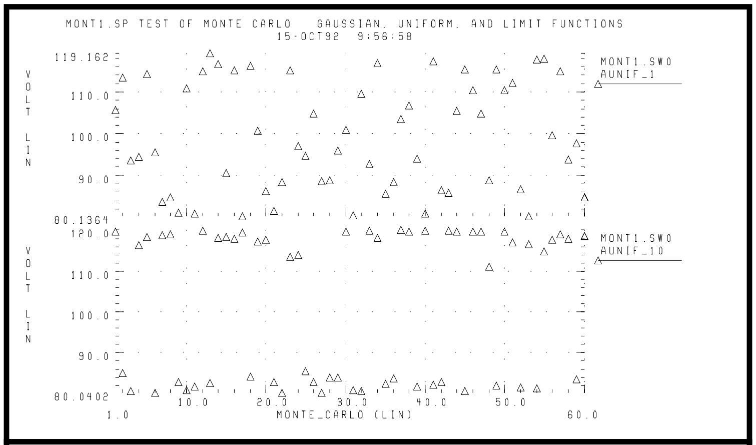
For example, the MOSFET channel length varies from transistor to transistor by a small amount corresponding to the die distribution. The die distribution is responsible for offset voltages in operational amplifiers and for the tendency of flip-flops to settle into random states. However, all transistors on a given die site will vary by the wafer or fabrication run distribution, which is much larger than the die distribution, but affects all transistors the same. The wafer distribution is assigned to the MOSFET model; it sets the speed and power dissipation characteristics.

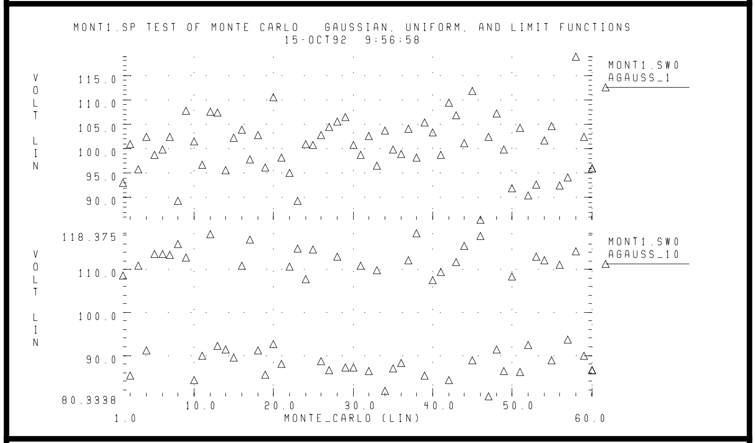
Monte Carlo Examples

Example 1: Gaussian, Uniform, and Limit Functions

```
Test of monte carlo gaussian, uniform, and limit functions

.options post
.dc monte=60

* setup plots
.model histo plot ymin=80 ymax=120 freq=1

.graph model=HISTO aunif_1=v(au1)
.graph model=HISTO aunif_10=v(au10)
.graph model=HISTO agauss_1=v(ag1)
.graph model=HISTO agauss_10=v(ag10)
.graph model=HISTO limit=v(L1)

* uniform distribution relative variation +/- .2
```

```
.param ru_1=unif(100,.2)

Iu1 u1 0 -1
ru1 u1 0 ru_1

* absolute uniform distribution absolute variation +/- 20
* single throw and 10 throw maximum
.param rau_1=aunif(100,20)
.param rau_10=aunif(100,20,10)

Iau1 au1 0 -1
rau1 au1 0 rau_1

Iau10 au10 0 -1
rau10 au10 0 rau_10

* gaussian distribution relative variation +/- .2 at 3 sigma
.param rg_1=gauss(100,.2,3)

Ig1 g1 0 -1
rg1 g1 0 rg_1

* absolute gaussian distribution absolute variation +/- .2 at
3 sigma
* single throw and 10 throw maximum
.param rag_1=agauss(100,20,3)
.param rag_10=agauss(100,20,3,10)

Iag1 ag1 0 -1
rag1 ag1 0 rag_1

Iag10 ag10 0 -1
rag10 ag10 0 rag_10

* random limit distribution absolute variation +/- 20
.param RL=limit(100,20)

IL1 L1 0 -1
rL1 L1 0 RL
.end
```


MONT1.SP TEST OF MONTE CARLO GAUSSIAN, UNIFORM, AND LIMIT FUNCTIONS
15-OCT92 9:56:58
VOLT LIN
119.162
110.0
100.0
90.0
80.1364
120.0
110.0
100.0
90.0
80.0402
MONT1.SW0
AUNIF_1
MONT1.SW0
AUNIF_10
1.0
10.0
20.0
30.0
40.0
50.0
60.0
MONTE_CARLO (LIN)


MONT1.SP TEST OF MONTE CARLO GAUSSIAN, UNIFORM, AND LIMIT FUNCTIONS
15-OCT92 9:56:58
VOLT LIN
115.0
110.0
105.0
100.0
95.0
90.0
118.375
110.0
100.0
90.0
80.3338
MONT1.SW0
AGAUSS_1
MONT1.SW0
AGAUSS_10
1.0
10.0
20.0
30.0
40.0
50.0
60.0
MONTE_CARLO (LIN)

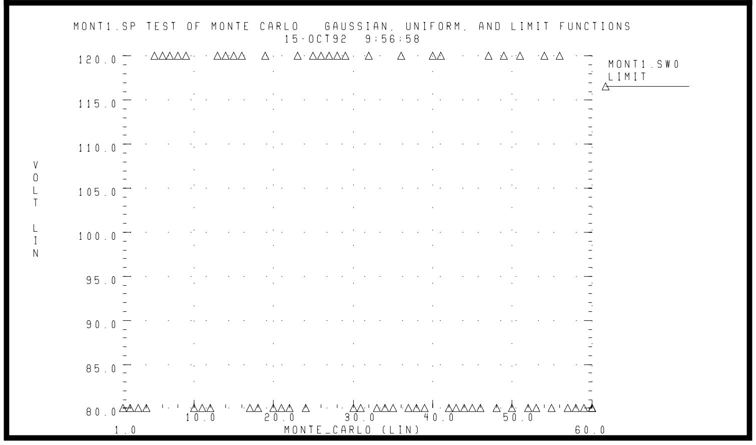

MONT1.SP TEST OF MONTE CARLO GAUSSIAN, UNIFORM, AND LIMIT FUNCTIONS
15-OCT92 9:56:58
VOLT LIN
120.0
115.0
110.0
105.0
100.0
95.0
90.0
85.0
80.0
MONT1.SW0
LIMIT
1.0
10.0
20.0
30.0
40.0
50.0
60.0
MONTE_CARLO (LIN)

Figure 10-10: Gaussian, Uniform, and Limit Functions

Example 2: Major and Minor Distribution

MOS IC processes have both a major and a minor statistical distribution of manufacturing tolerance parameters. The major distribution is the wafer-to-wafer and run-to-run variation. The minor distribution is the transistor-to-transistor process variation. The major distribution determines electrical yield. The minor distribution is responsible for critical second-order effects, such as amplifier offset voltage and flip-flop preference.

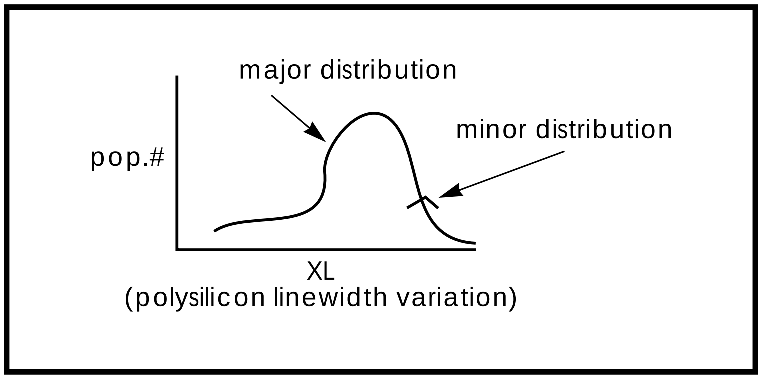


Figure 10-11: Major and Minor Distribution of Manufacturing Variations

The example below is a Monte Carlo analysis of a DC sweep of the supply voltage VDD from 4.5 volts to 5.5 volts. Transistors M1 through M4 form two inverters. The channel lengths for the MOSFETs are set by the nominal value of the parameter LENGTH, which is set to 1u. Since all of the transistors are on the same integrated circuit die, the distribution is given by the parameter LEFF, which is a ±5% distribution in the variation of the channel lengths at the ±3-sigma level. Each MOSFET gets an independent random Gaussian value.

The parameter PHOTO controls the difference between the physical gate length and drawn gate length. Because both n-channel and p-channel transistors use the same layer for the gates, the Monte Carlo distribution XPHOTO is set to the local parameter PHOTO.

PHOTO lithography for both NMOS and PMOS devices is controlled by XPHOTO, which is consistent with the physics of manufacturing.

```
File: MONDC_A.SP
.DC VDD 4.5  5.5  .1   SWEEP MONTE=30
.PARAM LENGTH=1U  LPHOTO=.1U
.PARAM LEFF=GAUSS (LENGTH, .05, 3)
+ XPHOTO=GAUSS (LPHOTO, .3, 3)
.PARAM  PHOTO=XPHOTO

M1  1  2  GND GND  NCH  W=10U L=LEFF
M2  1  2  VDD VDD  PCH  W=20U L=LEFF

M3  2  3  GND GND  NCH  W=10U L=LEFF
M4  2  3  VDD VDD  PCH  W=20U L=LEFF

.MODEL NCH NMOS  LEVEL=2 UO=500 TOX=100 GAMMA=.7 VTO=.8
+ XL=PHOTO
.MODEL PCH PMOS  LEVEL=2 UO=250 TOX=100 GAMMA=.5 VTO=-.8
+ XL=PHOTO
.INC Model.dat
.END
```

Example 3: RC Time Constant

This simple example demonstrates the uniform distribution for resistance and capacitance and the resulting transient waveforms for 10 different random values.

```
*FILE: MON1.SP WITH UNIFORM DISTRIBUTION
.OPTION LIST POST=2
.PARAM RX=UNIF(1,  .5) CX=UNIF(1,  .5)
.TRAN  .1  1  SWEEP MONTE=10
.IC 1  1
R1  1  0   RX
C1  1  0   CX

.END
```

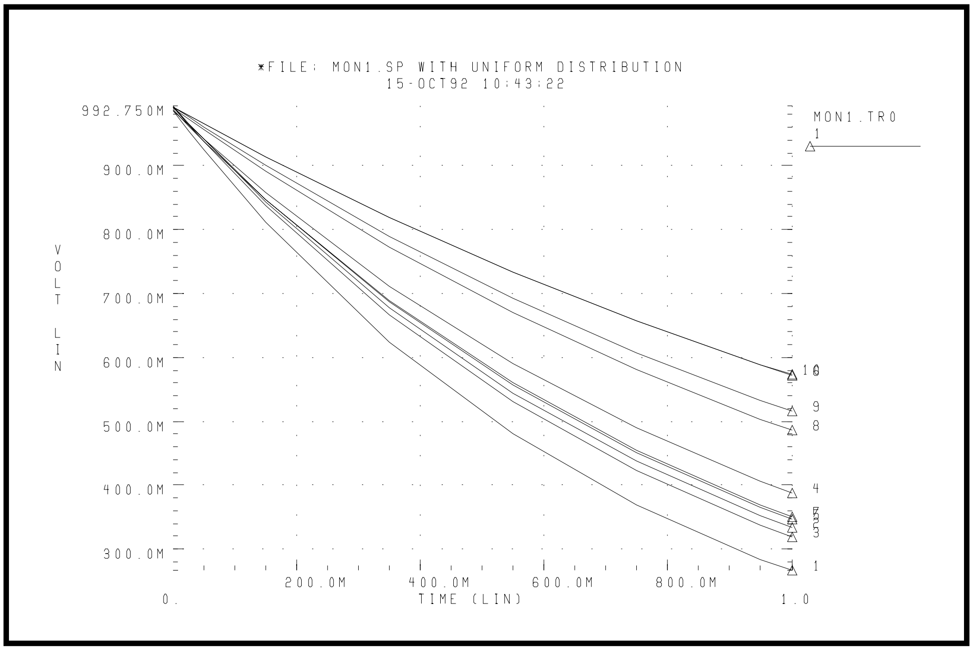


Figure 10-12: Monte Carlo Analysis of RC Time Constant

Example 4: Switched Capacitor Filter Design

The capacitors used in switched capacitor filter applications are composed of parallel connections of a basic cell. Use Monte Carlo techniques to estimate the variation in total capacitance. There are two distributions involved in the capacitance calculation:

- Minor distribution of cell capacitance from cell-to-cell on a single die
- Major distribution of the capacitance from wafer-to-wafer or manufacturing run-to-run

The minor distribution is the element distribution, and the major distribution is the model distribution.

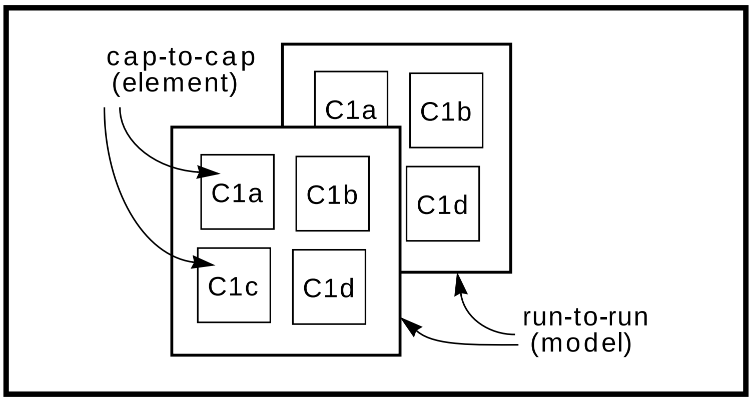


Figure 10-13: Monte Carlo Distribution

You can approach this problem from either physical or electrical levels. The physical level relies on physical distributions such as oxide thickness and polysilicon linewidth control. The electrical level relies on actual capacitor measurements.

Physical Approach

Assume that the variation in capacitance for adjacent cells is controlled by the local variation of polysilicon, since the oxide thickness control is excellent for small areas on a single wafer.

Next, define a local poly linewidth variation and a global or model level poly linewidth variation.

The local polysilicon linewidth control for a line 10 μ wide, manufactured with process A, is assumed to be ±0.02 μ for a 1-sigma distribution. The global or model level polysilicon linewidth control is much wider; use 0.1 μ for this example. The global oxide thickness is assumed to be 200 angstroms with a ±5 angstrom variation at 1 sigma.

The cap element is assumed to be square, with local poly variation in both directions. The cap model has two distributions, the poly linewidth distribution and the oxide thickness distribution. Since the effective length is

$$\text{Leff} = \text{Ldrawn} - 2 \cdot \text{DEL}$$

the model poly distribution is half the physical per-side values.

```
C1a 1 0 CMOD  W=ELPOLY  L=ELPOLY
C1b 1 0 CMOD  W=ELPOLY  L=ELPOLY
C1C 1 0 CMOD  W=ELPOLY  L=ELPOLY
C1D 1 0 CMOD  W=ELPOLY  L=ELPOLY

$ 10U POLYWIDTH,0.05U=1SIGMA
$ CAP MODEL USES 2*MODPOLY  .05u= 1 sigma
$ 5angstrom oxide thichness AT 1SIGMA
.PARAM ELPOLY=AGAUSS(10U,0.02U,1)
+      MODPOLY=AGAUSS(0,.05U,1)
+      POLYCAP=AGAUSS(200e-10,5e-10,1)

.MODEL CMOD C THICK=POLYCAP  DEL=MODPOLY
```

Electrical Approach

The electrical approach assumes no physical interpretation, but requires a local or element distribution and a global or model distribution.

Assume that the capacitors can be matched to ±1% for the 2-sigma population. The process can maintain a ±10% variation from run to run for a 2-sigma distribution.

```
C1a 1 0 CMOD  SCALE=ELCAP
C1b 1 0 CMOD  SCALE=ELCAP
C1C 1 0 CMOD  SCALE=ELCAP
C1D 1 0 CMOD  SCALE=ELCAP
.PARAM  ELCAP=Gauss(1,.01,2) $ 1% at 2 sigma
+      MODCAP=Gauss(.25p,.1,2) $10% at 2 sigma
.MODEL CMOD C CAP=MODCAP
```

Worst Case and Monte Carlo Sweep Example

The following example measures the delay of a pair of inverters. The input is buffered by an inverter, and the output is loaded by another inverter. The model was prepared according to the scheme described in the previous sections. The first .TRAN analysis statement sweeps from the worst case 3- sigma slow to 3-sigma fast. The second .TRAN does 100 Monte Carlo sweeps.

HSPICE Input File

The Star-Hspice input file can contain the following sections.

Analysis Setup Section

The simulation is accelerated by the use of the AUTOSTOP option, which automatically stops the simulation when the .MEASURE statements have achieved their target values.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma, then Monte Carlo
.option nopage nomod acct
+ autostop post=2
.tran 20p 1.0n sweep sigma -3 3 .5
.tran 20p 1.0n sweep monte=20
.option post co=132
.param vref=2.5
.meas m_delay trig v(2) val=vref fall=1
+ targ v(out) val=vref fall=1
.meas m_power rms power to=m_delay
.param sigma=0
```

Circuit Netlist Section

```
.global 1
vcc 1 0 5.0
vin in 0 pwl 0,0 0.2n,5
x1 in 2 inv
x2 2 3 inv
x3 3 out inv
x4 out 5 inv
.macro inv in out
   mn out in 0 0 nch W=10u L=1u
   mp out in 1 1 pch W=10u L=1u
.eom
```

Skew Parameter Overlay for Model Section

```
* overlay of gaussian and algebraic for best case worst case and
+  monte carlo
* +/- 3 sigma is the maximum value for parameter sweep
.param
+ mult1=1
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'
+ rshncd=agauss(50,8,1) rshn='rshncd-sigma*8'
+ rshpcd=agauss(150,20,1) rshp='rshpcd-sigma*20'
```

MOS Model for N-Channel and P-Channel Transistors Section

```
* level=28 example model for high accuracy model
.model nch nmos
+ level=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwn tox=tox delvto=delvton rsh=rshn
+ ld=0.06u wd=0.2u
+ acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0
+ js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=1.4e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=0
+ vfb0=-.5 phi0=0.65 k1=.9 k2=.1 eta0=0
+ muz=500 u00=.075
+ x3ms=15 u1=.02 x3u1=0
+ b1=.28 b2=.22 x33m=0.000000e+00
+ alpha=1.5 vcr=20
+ n0=1.6 wfac=15 wfacu=0.25
+ lvfb=0 lk1=.025 lk2=.05
+ lalpha=5
.model pch pmos
+ level=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u
+ acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp
+ js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=-1.7e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=5
+ vfb0=-.1 phi0=0.65 k1=.35 k2=0 eta0=0
+ muz=200 u00=.175
+ x3ms=8 u1=0 x3u1=0.0
+ b1=.25 b2=.25 x33m=0.0
+ alpha=0 vcr=20
+ n0=1.3 wfac=12.5 wfacu=.2
+ lvfb=0 lk1=-.05
.end
```

Transient Sigma Sweep Results

The plot in Figure 10-14: shows the family of transient analysis curves from the transient sweep of the sigma parameter from -3 to +3. Sigma is then algebraically coupled into the skew parameters and the resulting parameters modify the actual NMOS and PMOS models.

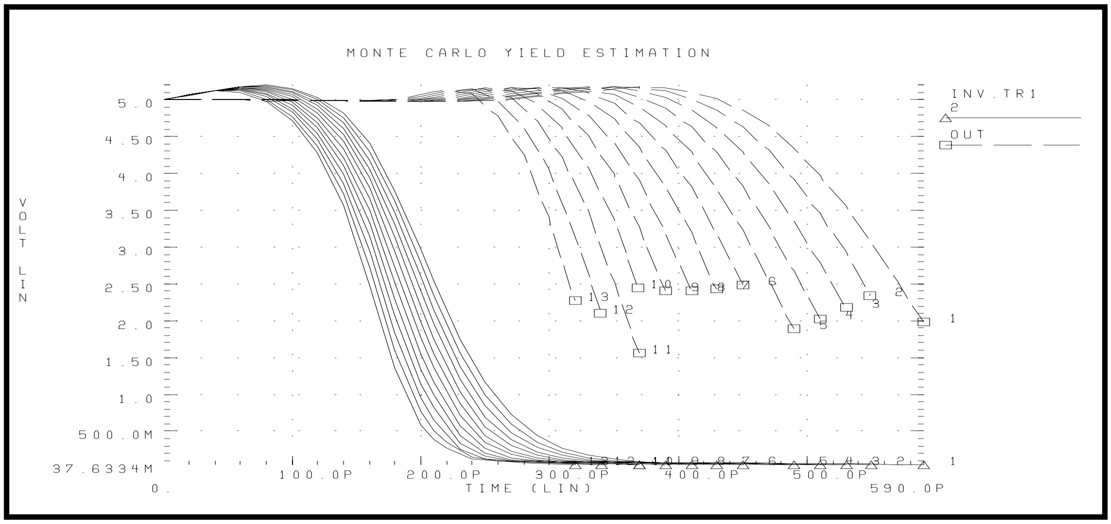


Figure 10-14: Sweep of Skew Parameters from -3 Sigma to +3 Sigma

You can view the transient family of curves by plotting the .MEASURE output file. The plot in Figure 10-15: shows the measured pair delay and the total dissipative power against the parameter SIGMA.

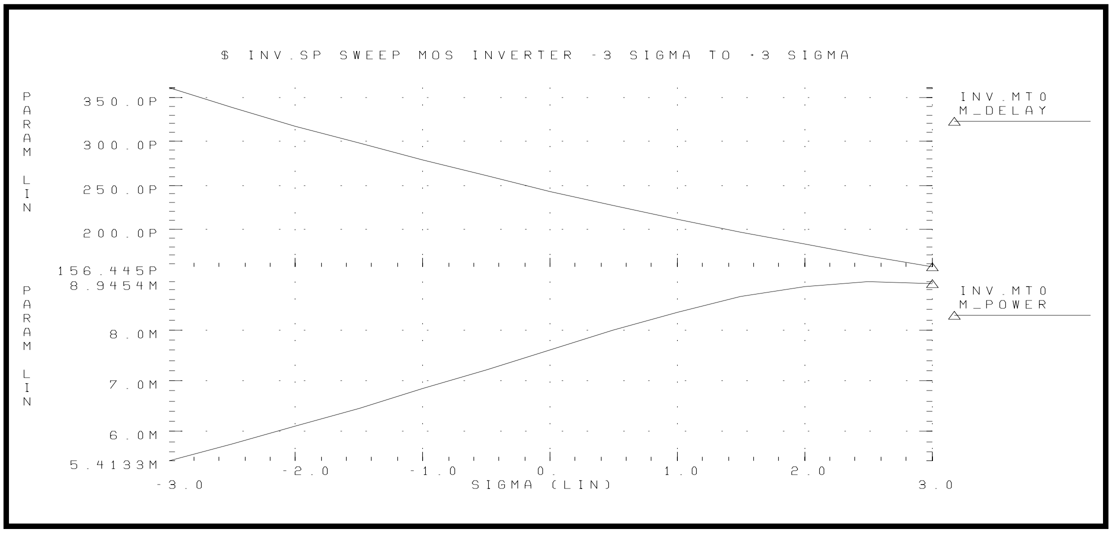


Figure 10-15: Sweep MOS Inverter, Pair Delay and Power: -3 Sigma to 3 Sigma

Monte Carlo Results

The output of the Monte Carlo analysis is evaluated in this section. The plot in Figure 10-16: is a quality control step that plots TOX against XL (polysilicon critical dimension). The cloud of points was obtained in Avant!'s graphing software by setting XL as the X-axis independent variable and plotting TOX with a symbol frequency of 1. This has the effect of showing the points without any connecting lines. The resulting graph demonstrates that the TOX model parameter is randomly independent of XL.

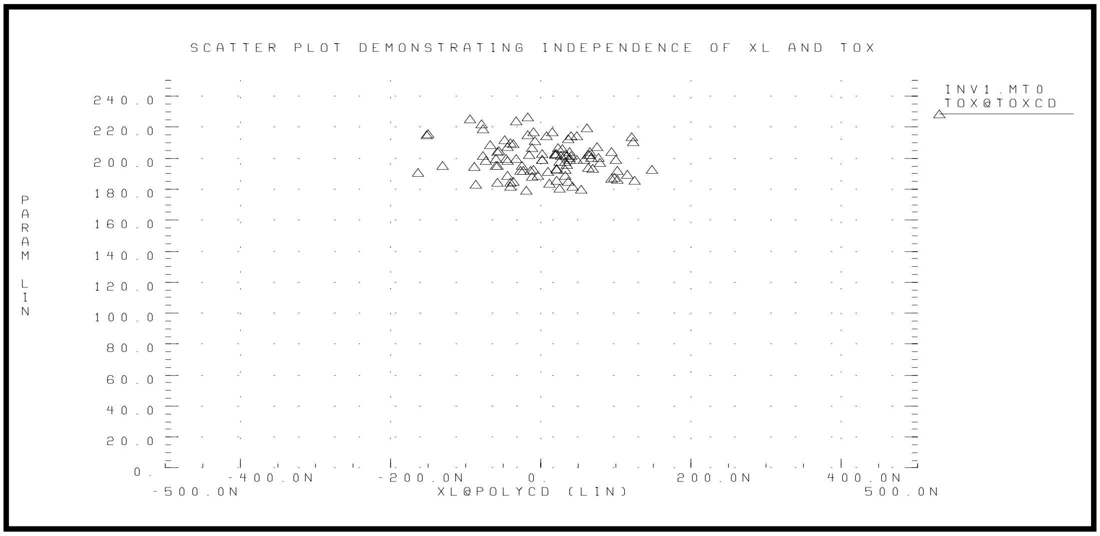


Figure 10-16: Scatter Plot, XL and TOX

The next graph (see Figure 10-17:) is a standard scatter plot of the measured inverter pair delay against the Monte Carlo index number. If a particular result looks interesting, such as if the very smallest delay was obtained in simulation 68 ("monte carlo index = 68"), you can read the output listing file and obtain the actual Monte Carlo parameters associated with that simulation.

```
*** monte carlo index = 68 ***
MONTE CARLO PARAMETER DEFINITIONS
polycd: xl = -1.6245E-07
```

```
nactcd: xwn = 3.4997E-08
pactcd: xwp = 3.6255E-08
toxcd: tox = 191.0
vtoncd: delvton = -2.2821E-02
vtopcd: delvtop = 4.1776E-02
rshncd: rshn = 45.16
rshpcd: rshp = 166.2

m_delay = 1.7946E-10 targ= 3.4746E-10 trig= 1.6799E-10
m_power = 7.7781E-03 from= 0.0000E+00 to= 1.7946E-10
```

From the preceding listing, you can see that the *m_delay* value of 1.79e-10 seconds is the fastest pair delay. In addition, the Monte Carlo parameters can be examined.

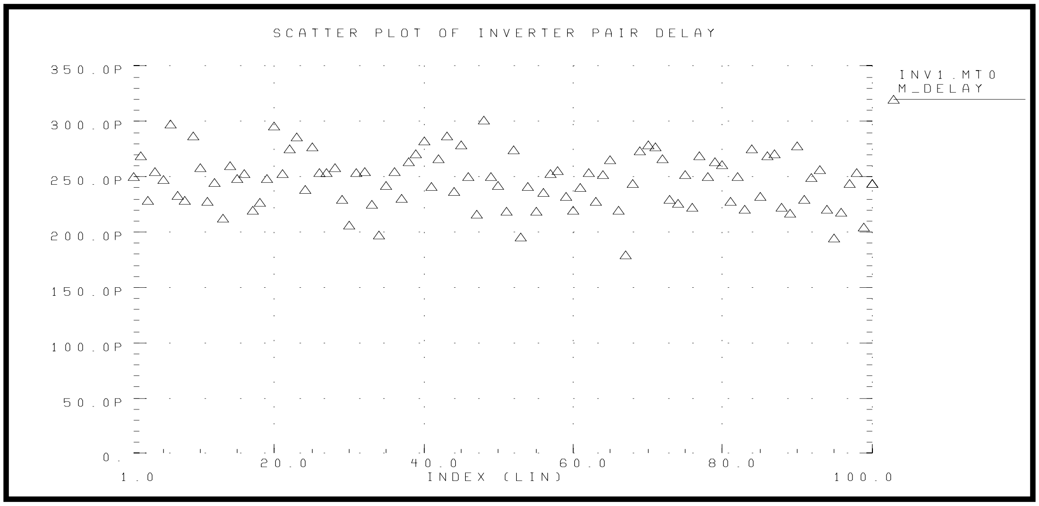


Figure 10-17: Scatter Plot of Inverter Pair Delay

Plotting against the Monte Carlo index number does not help in centering the design. The first step in centering a design is to determine the most sensitive process variables. We can do this by graphing the various process parameters against the pair delay. Select the pair delay as the X-axis independent variable, and also set the symbol frequency to 1 to obtain the scatter plot. The graph in Figure 10-18: demonstrates the expected sensitivity of output pair delay to channel length variation (polysilicon variation).

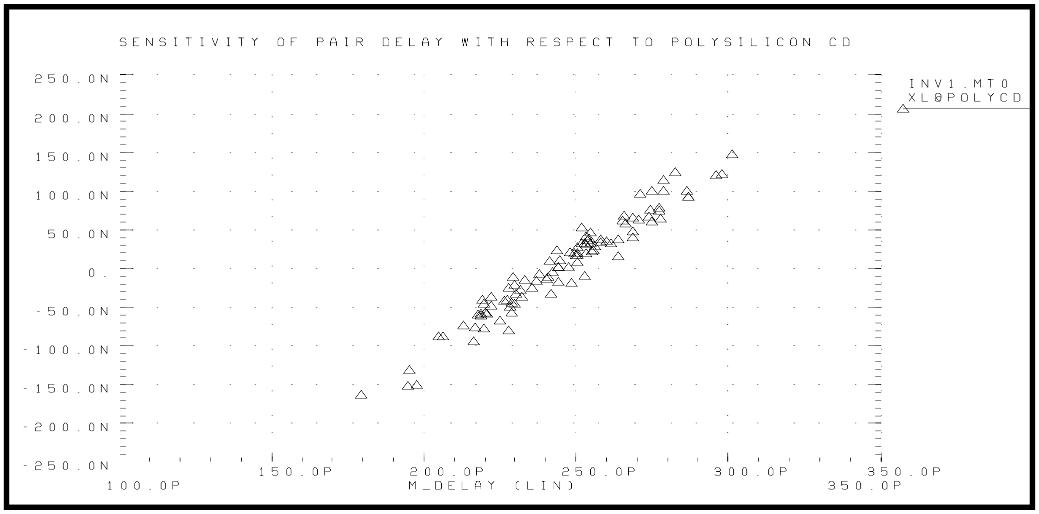


Figure 10-18: Sensitivity of Delay with Poly CD (XL)

Now, the parameter TOX is plotted against pair delay (Figure 10-19:). Note that there is no clear tilt to the scatter plot. This indicates that TOX is a secondary process parameter compared to XL. To explore this in more detail, set the skew parameter XL to a constant and simulate.

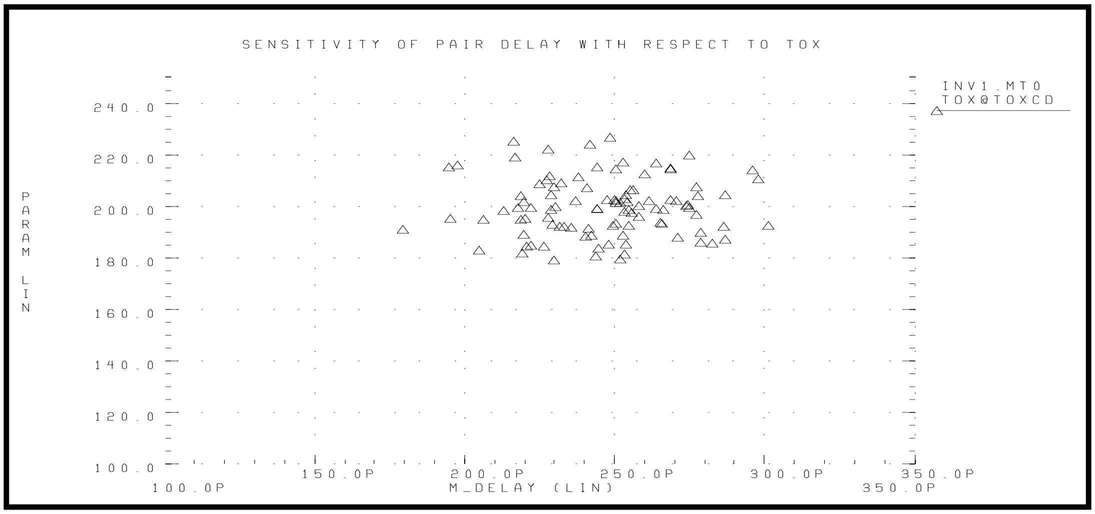


Figure 10-19: Sensitivity of Delay with TOX

The plot in Figure 10-20: shows the overlay of a 3-sigma worst case corners response and the 100 point Monte Carlo. Notice that the actual (Monte Carlo) distribution for power/delay is very different than the +3 sigma to -3 sigma plot. The worst case was simulated in 0.5 sigma steps. The actual response is closer to ± 1.5 sigma instead of ± 3 sigma. This produces a predicted delay variation of 100 ps instead of 200 ps. Therefore, the advantage of using Monte Carlo over traditional 3-sigma worst case corners is a 100% improvement in accuracy of simulated-to-actual distribution. This is an example of how the worst-case procedure is overly pessimistic.

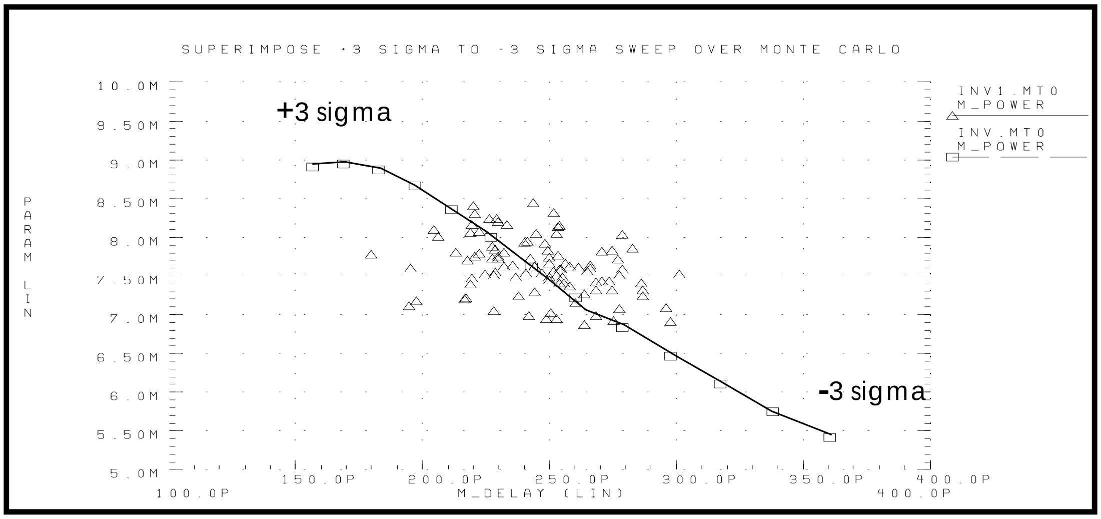


Figure 10-20: Superimpose Sigma Sweep over Monte Carlo

Now take the Monte Carlo plot and superimpose the assumed part grades from marketing studies (Figure 10-21:). In this case we have used a 250 ps delay and 7.5 mW power dissipation to determine the 4 binning grades. A manual count gives: Bin1 - 13%, Bin2 - 37%, Bin3 - 27%, Bin4 - 23%. If this circuit were representative of the entire chip, we would predict a present yield of 13% for the premium Bin 1 parts, assuming the design and process variations.

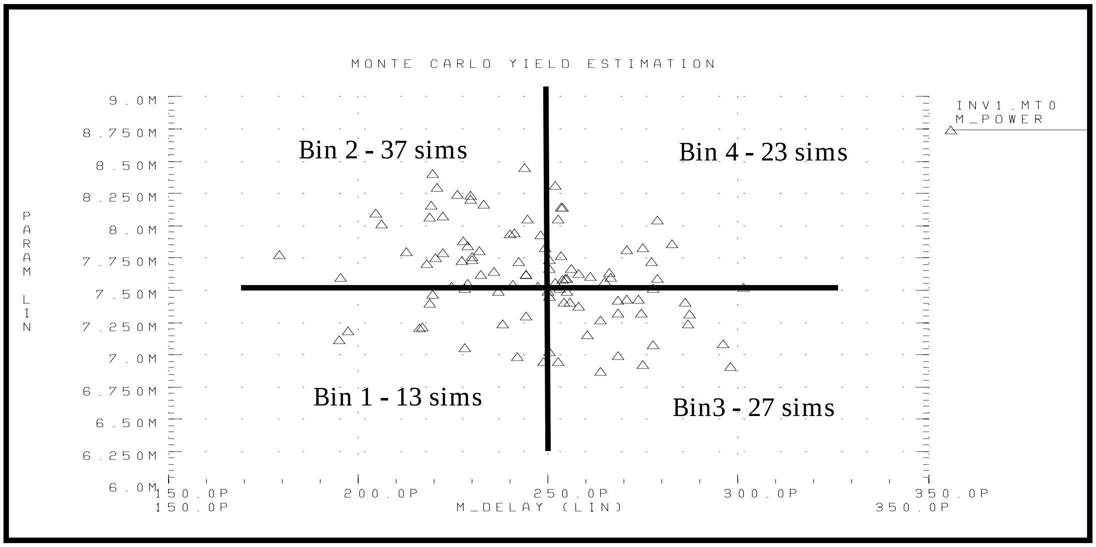


Figure 10-21: Speed/Power Yield Estimation

Simulating Circuit and Model Temperatures

Temperature affects all electrical circuits. The key temperature parameters associated with circuit simulation are (see Figure 10-22:):

- Model reference temperature – each model might be measured at a different temperature and each model has a TREF parameter
- Element junction temperature – each resistor, transistor, or other element generates heat and will be hotter than the ambient temperature.
- Part temperature – at the system level, each part has its own temperature
- System temperature – a collection of parts form a system that has a local temperature.
- Ambient temperature – the ambient temperature is the air temperature of the system.

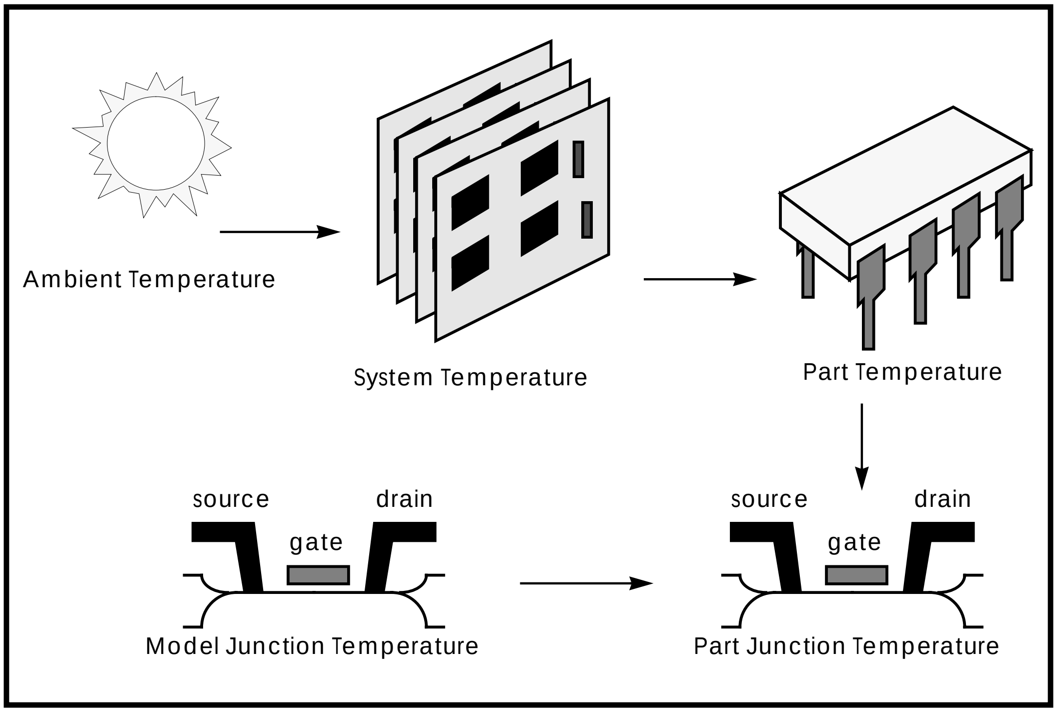


Figure 10-22: Part Junction Temperature Sets System Performance

Temperatures in Star-Hspice are calculated as differences from ambient temperature:

$$T\mathrm{ambient} + \Delta system + \Delta part + \Delta junction = Tjunction$$

$$Ids = f(Tjunction, Tmodel)$$

Every element in Star-Hspice has a keyword DTEMP. This is the difference between junction and ambient temperature. An example of using DTEMP in a MOSFET element statement is shown below.

```
M1 drain gate source bulk Model_name W=10u L=1u DTEMP=+20
```

Temperature Analysis

Star-Hspice allows you to specify three temperatures:

- Model reference temperature, specified in a .MODEL statement using the TREF parameter (or TEMP or TNOM, for some models). This is the temperature, in °C, at which the model parameters are measured and extracted. The value of TNOM can be set in a .OPTION statement. Its default value is 25 °C.
- Circuit temperature, specified using a .TEMP statement or the TEMP parameter. This is the temperature, in °C, at which all elements are simulated. To modify the temperature for a particular element, you can use the DTEMP parameter. The default circuit temperature is the value of TNOM.
- Individual element temperature, specified as the circuit temperature plus an optional amount specified using the DTEMP parameter

You can specify the temperature of a circuit for a Star-Hspice run with either the .TEMP statement or the TEMP parameter in the .DC, .AC, or .TRAN statements. The circuit simulation temperature set by any of these statements is compared against the reference temperature set by the TNOM option. TNOM defaults to 25 °C unless the option SPICE is used, in which case it defaults to 27 °C. The derating of component values and model parameters is calculated by using the difference of the circuit simulation temperature and the reference temperature, TNOM.

Since elements and models within a circuit can be operating at different temperatures (for example, a high-speed I/O buffer switching at 50 MHz will be much hotter than a low-drive NAND gate switching at 1 MHz), use an element temperature parameter, DTEMP, and a model reference parameter, TREF. Specifying DTEMP in an element statement causes the element temperature for the simulation to be:

element temperature = circuit temperature + DTEMP

Specify the DTEMP value in the element statement (resistor, capacitor, inductor, diode, BJT, JFET, or MOSFET statement). You can assign a parameter to DTEMP, then sweep the parameter using the .DC statement. The DTEMP value defaults to zero.

By specifying TREF in the model statement, the model reference temperature is changed (TREF overrides TNOM). The derating of the model parameters is based on the difference of the circuit simulator's temperature and TREF, instead of TNOM.

.TEMP Statement

The syntax is:

```
.TEMP t1 <t2 <t3 ...>>
```

t1 t2 …	The temperatures, in °C, at which the circuit is simulated

Example 1

```
.TEMP -55.0 25.0 125.0
```

The .TEMP statement sets the circuit temperatures for the entire circuit simulation. Star-Hspice uses the temperature set in the .TEMP statement along with the TNOM option setting (or the TREF model parameter) and the DTEMP element temperature, and simulates the circuit with individual elements or model temperatures.

Example 2

```
.TEMP 100
D1 N1 N2 DMOD DTEMP=30
D2 NA NC DMOD
R1 NP NN 100 DTEMP=-30
.MODEL DMOD D IS=1E-15 VJ=0.6 CJA=1.2E-13 CJP=1.3E-14
TREF=60.0
```

From the .TEMP statement, the circuit simulation temperature is given as 100°C. Since TNOM is not specified, it defaults to 25°C. The temperature of the diode is given as 30°C above the circuit temperature by the DTEMP parameter. That is, D1temp = 100°C + 30°C = 130°C. The diode, D2, is simulated at 100°C. R1 is simulated at 70°C. Since TREF is specified at 60°C in the diode model statement, the diode model parameters given are derated by 70°C (130°C - 60°C) for diode D1 and by 40°C (100°C - 60°C) for diode D2. The value of R1 is derated by 45°C (70°C - TNOM).