A Brief User's Guide to Hspice

by
Sameer Sonkusale
sameers@ee.upenn.edu

Introduction

Hspice is a spice simulation software, available on Sun/Unix platforms on eniac/pender machines (for e.g. DSL 100 Moore Bldg.). The syntax for writing the hspice files is same as for the most commonly used PSpice, except that you would be manually writing the spice netlist file for simulation. You don't have the schematic editor and the probe as in Pspice. The basic tutorial on spice can be found at Prof. Jan Van der Spiegels Spice tutorial webpage. In this tutorial we will cover some of the extra features provided by hspice, which are most commonly used.

A complete manual can be found at /pkg/hspice/98.2/docs/hspice.pdf on seas/ee machines. (Warning: Please don't print this reference manual on any of the ee/seas printers, since it is a very very huge file). You can also follow this link at an external site for a complete reference manual.

Selected Features:

This section will only talk about some of the advanced features provided by hspice. The basic spice tutorial should be your first visit before continuing with this section. Some of these advanced features may also be available in Pspice. Please consult the manual for more information about each of the following features.

1. Node names
Instead of using numbers to represent nodes, character strings can now be used. This will enable labeling the nodes with a name which is more appropriate for that node (e.g. input, output) rather than using numbers, making it easier to read. Also, it will enable you to locate the signals in your circuit with a name when viewing your output using awaves.

2. .PRINT Statement
The syntax is same as before, but now you could have mathematical expressions involving signals to be printed as well.

Example 1: .PRINT TRAN V(3) I(VIN) PAR('V(OUT)/V(IN) ')
This example prints out the results of a transient analysis for the nodal voltage named 3 and the current through the voltage source named VIN. The ratio of the nodal voltage at node "OUT" and node "IN" is also printed. Note that the nodes have been represented by names "OUT" and "IN" instead of numbers.

Example 2: .print varname=PAR('sqrt(v3)')
This instructs HSPICE to print the square root of the voltage "v3" and assign it the variable name varname. The results can be found in the output file as well as in the awaves graphical interface options. Apart from square root, other useful functions such as log(), sin() and tan() are supported. Consult the HSPICE manual (/pkg/hspice/98.2/docs/hspice.pdf) for a complete listing. Awaves can directly be used to compute the mathematical expressions as well.

3. .INCLUDEStatement
Syntax : .INCLUDE '<filepath> filename' *or*
.INC '<filepath> filename'
where,
filename: is the name of the file to be included in the present file which makes use of the above statement.
filepath : (is optional) provides a tree structure representation of the directory in which the file exists.

Example: .inc '/home4/sameers/Spice/Models/mos.lib'

4. .LIB Statement
Syntax: .lib '<filepath> filename' entryname
where,
filename: is the name of the library file to be included
filepath: is the path to the file
entryname: is the entry name for the section of the library file to included. The first name of the entryname cannot be an integer.

Every .lib statement should be followed by a .ENDL statement

Example: Say you have a file named mos.lib in /home4/sameers/Spice/Models/ directory which look ssomething like this

```
.LIB NMOS
.MODEL CMOSN NMOS LEVEL=3 PHI=0.700000 TOX=3.0400E-08
XJ=0.200000U TPG=1
+ VTO=0.6081 DELTA=1.3700E+00 LD=9.0910E-10 KP=7.4209E-05
+ UO=653.3 THETA=9.5780E-02 RSH=5.1480E+01 GAMMA=0.6166
+ NSUB=1.4780E+16 NFS=5.9090E+11 VMAX=1.8150E+05 ETA=7.7500E-
02
```

+ KAPPA=2.4680E-01 CGDO=5.0000E-11 CGSO=5.0000E-11
+ CGBO=3.4138E-10 CJ=2.8065E-04 MJ=5.3057E-01 CJSW=1.4649E-10
+ MJSW=1.0000E-01 PB=9.7858E-01
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 8.9440E-07
.ENDL NMOS

.LIB PMOS
.MODEL CMOSP PMOS LEVEL=3 PHI=0.700000 TOX=3.0400E-08
XJ=0.200000U TPG=-1
+ VTO=-0.8311 DELTA=3.1900E+00 LD=9.0910E-10 KP=1.9344E-05
+ UO=170.3 THETA=1.0050E-01 RSH=3.3060E+01 GAMMA=0.3046
+ NSUB=3.6060E+15 NFS=5.9090E+11 VMAX=1.6930E+05 ETA=8.5870E-
02
+ KAPPA=9.9940E+00 CGDO=5.0000E-11 CGSO=5.0000E-11
+ CGBO=3.2737E-10 CJ=2.9597E-04 MJ=4.4146E-01 CJSW=1.4645E-10
+ MJSW=1.0000E-01 PB=7.4913E-01
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 8.2900E-07
.ENDL PMOS

Then if you needed to include a PMOS model file in your spice netlist, you could say
.lib '/home4/sameers/Spice/Models/mos.lib' PMOS
or if you needed just the NMOS models you could say
.lib '/home4/sameers/Spice/Models/mos.lib' NMOS

The difference between .INC and .LIB is that .inc includes everything from a file, whereas .LIB can include only a section of the file defined by the entryname.

5. .PARAM statement

The .Param statement lets you create a parameter and assign algebraic mathematical expressions to it.

Syntax: .PARAM <parametername> = '<Expression>'

Example:
.PARAM width = 20u
.PARAM length = 'sqrt(width)*1.65'

This is useful , if you want to say, make the length and the width of the transistor related to each other in a certain fashion.

You would then make a MOS statement call as
M1 3 2 0 0 NMOS width length

instead of fixed numbers for W and L . This makes the W and the L of the transistors as parameters.

Another example of a frequency dependent resistor would be
.PARAM HERTZ = 100
.PARAM R = '1K/log(HERTZ)'
r1 3 2 R

The above declaration creates a resistor r1 whose value is a parameter R which depends on the parameter HERTZ. The ability to sweep through the different parameters is discussed in the following section on .DATA statement.

6. .DATA statement

Data-driven analysis allows the user to modify any number of parameters, then perform an operating , DC, AC, or transient analysis using the new parameter values.

The .DATA statement specifies the parameters for which values are to be changed and gives the sets of values that are to be assigned during each simulation. Lets look at an example:

Example: Say you want to perform a transient analysis and an AC analysis of your circuit for different set of values for width, length and load resitance RL. Then your spice netlist may look something like this:

```
.
.
.PARAM width = 10u
.PARAM length = 6u
.
M1 3 2 0 0 NMOS width length
Rload 3 5 RL
.
.
.TRAN        1n     1000n              SWEEP DATA=D1
.AC DEC    10     1hz      1GHz     SWEEP DATA=D1
.DATA D1  width  length  RL
+                 50u      20u      1K
+                 60u      10u      10K
+                 100u     25u      1K
.ENDDATA
.
.
```

The simulation program would then run the transient analyis and the AC analysis first for the values of width=50u, length=20u and RL=1Kohms. Then during the

next run it would perform the transient and AC analyis for the values of width=60u, length=10u and RL=10K..and finally it would do the analyses for the 3rd set of data values. Note that width, length and RL have been declared using a .PARAM statement.

7. .OPTIONS statement

This helps in specifying simulation input and output control options for the hspice. There are lots of different keywords associated with the .options statement. Please visit the reference manual for complete entries.

Example

.options post -> to store the simulation results. This option is needed to view the results using awaves.
.options probe -> to save the output variables only.

8. .PROBE statement

The .PROBE statement saves output variables into the interface and graph data files. Hspice usually saves all voltages and supply currents in addition to the output variables. Set the .OPTION PROBE to save the output variables only. Use the .PROBE statement to specify which quantities are to be printed in the output listing. This is very useful when simulating huge circuits where hspice simulations may run for hours if we didn't use the .PROBE statement. Using such a statement would save time and memory since it outputs only a few signals of interest in the whole circuit.

Syntax: .PROBE <analysis type> var1 ..... <var32>

analysis type the type of analysis for the specified plots. Analysis types are DC, AC, TRAN, NOISE and DISTO
var1 output variables to be plotted/stored. These cane be voltages. currents or element variables from a
DC, AC, TRAN, NOISE or DISTO analysis. The limit for the number of output variables in a single
.PROBE statements is 32. Additional .PROBE statements may be used to deal with more output
variables.

Example:
.PROBE DC V(4) V(5) I(Vin) beta = PAR('I1(Q1)/I2(Q2) ')

9. .MEASURE statement

The .MEASURE statement has several different formats, depending on the application. You can use it for either DC, AC or transient analysis.

Fundamental measurement modes are:

- Rise, Fall, Delay
  syntax: .MEASURE <DC | AC | TRAN> result TRIG ... TARG ...
  where *result* is the name given to the measured output
  trig .. targ ... identifies the beginning of the trigger and target specs.
  There are few other options that come with .MEASURE which not have been discussed.

  TRIG (trigger) syntax: TRIG trig_var VAL=trig_val <TD=time delay> <CROSS=c> <RISE=r><FALL=f>
  or
  TRIG AT=val
  TARG (target) syntax: TARG targ_var VAL=targ_val <TD=time delay> <CROSS = c | LAST> <RISE = r | LAST>
  + <FALL=f | LAST>
  where *TRIG* and *TARG* specifies the beginning of the trigger and of the target signal specs.
  trig_var is the name of the output variable which determines the beginning of the measurement
  trig_val is the value of the trig_var at which the counter for crossing, rises or falls is incremented by one
  targ_var is the name of the output variable whose propagation delay is determined with respect to trig_var
  targ_val specifies the value of the targ_var at which the counter for crossing, rise or falls is incremented by one
  time_delay amount of delay before the measurement should start
  cross=c, rise=r, fall=f indicates which occurences of CROSS, FALL or RISE event causes a measurement to be performed
  last indicates that the measurement is performed when the last CROSS, FALL or RISE event occurs
  at=val is a case where val indicates the time for TRAN, frequency for AC and parameter for DC, when measurement should start
  example:
  .MEASURE TRAN tdlay TRIG V(1) VAL=2.5 TD = 10n RISE=2
  + TARG V(2) VAL=2.5 FALL=2
  This example specifies that a propagation delay measurement is taken between nodes 1 and 2 for transient analysis.
  The measurement is done between the second rising edge of the voltage at node 1 and the second falling edge of the voltage
  at node 2. The VAL = 2.5 specifies the exact voltage when the measurement should begin and end.

- Average, RMS, min, max, peak-to=peak and integral
  syntax: .MEASURE <DC | AC | TRAN> result func out_var <FROM=val> <TO=val>
  where *result* is the name given to the meaurement

from specifies the initial value for the 'func' calculation. For transient it is in units of time.
to specifies the end of the func calculation
func indicates the type of measurement
like, avg -> average, max -> maximum, min-> minimum, pp -> peak to peak, rms-> root mean square, integ->integral
out_var is the name of the output variable whose function is to be measureed in the simulation.
example:
.MEAS TRAN avgval AVG V(10) FROM=10ns TO=55ns

- Find-when
This allows any indep. variables (time,freq,parameter), any dependent variables (voltage or current)
or the derivative of any dependant variables to be measured when specific event occurs. These measure
statements are useful in the unity gain frequency or phase measurements.

syntax:
.MEASURE <DC|TRAN| AC> result WHEN out_var = val <TD = val>
+ < RISE=r | LAST > < FALL=f | LAST > < CROSS=c | LAST >
or
.MEASURE <DC|TRAN|AC> result WHEN out_var1=out_var2 < TD=val >
+ < RISE=r | LAST > < FALL=f | LAST > < CROSS=c| LAST >
or
.MEASURE <DC|TRAN|AC> result FIND out_var1 WHEN out_var2=val < TD=val >
+ < RISE=r | LAST > < FALL=f | LAST >
or
.MEASURE <DC|TRAN|AC> result FIND out_var1 WHEN out_var2 = out_var3
+ <TD=val > < RISE=r | LAST > < FALL=f | LAST >
+ <CROSS=c | LAST>
or
.MEASURE <DC|TRAN|AC> result FIND out_var1 AT=val
Parameter Definitions
*cross=c,rise=r, fall=f*The numbers indicate which occurrence of a CROSS, FALL,
or RISE event causes a measurement to be performed.
find selects the FIND function
last Measurement is performed when the last CROSS, FALL, or
RISE event occurs.
result is the name of the measurement
out_var is the name of the variable that is measured

td identifies the time at which measurement is to start
when selects the WHEN function

- Equation evaluation
  Use this statement to evaluate an equation that is a function of the results of the previous .MEASURE statements
  syntax: .MEASURE <DC |AC | TRAN> result PARAM='equation'
- Derivative evaluation
  The DERIVATIVE function provides the derivative of an output variable at a
  given time or frequency or for any sweep variable, depending on the type of

  analysis. It also provides the derivative of a specified output variable when some
  specific event occurs.
  syntax:
  .MEASURE <DC|AC|TRAN> result DERIVATIVE out_var AT=val
  or
  .MEASURE <DC|AC|TRAN> result DERIVATIVE out_var WHEN var2=val
  + <RISE=r | LAST> <FALL=f | LAST> <CROSS=c | LAST>
  + <TD=tdval>
  or
  .MEASURE <DC|AC|TRAN> result DERIVATIVE out_var WHEN var2=var3
  + <RISE=r | LAST> <FALL=f | LAST> <CROSS=c | LAST>
  + <TD=tdval>
  where:
  at=val the value of out_var at which the derivative is to be found
  cross=c,rise=r,fall=f The numbers indicate which occurrence of a CROSS, FALL,
  or RISE event causes a measurement to be performed.
  derivative selects the derivative function.
  last Measurement is performed when the last CROSS, FALL, or
  RISE event occurs.
  out_var is the variable for which the derivative is to be found
  result is the name given to the measurement
  td identifies the time at which measurement is to start
  var(2,3) variables used to establish conditions at which measurement is to take place
  when selects the WHEN function
  example: .MEAS TRAN slewrate DERIV V(out) AT=25ns

The output of the .measure statement is stored in the .mt# file where the # can be any number starting from 0.

Sameer Sonkusale <sameers@ee.upenn.edu>
Last updated 22nd March 2001