

1st Tutorial on PSpice

Introduction

The following information concerns the text-edited version of MicroSim PSpice, version 8. It is offered here solely for the purpose of helping undergraduate students complete their classroom assignments in computer-aided circuit analysis at the University of Texas at Arlington. No other use of these notes is supported by the University of Texas at Arlington.

File Types Used and Created by PSpice

The basic input file for PSpice is a text (ASCII) file that has the file type "CIR." In the beginning, this will be created by hand as the primary method of getting the circuit we want modeled into the PSpice program. Later, when we use the schematic capture program, it will create the *.CIR file for us, along with several auxiliary file types. Do not use a word processor to create these *.CIR files unless you "Save as" text or as ASCII. You can use Notepad to edit these files, but the best editor for this purpose is the one that is provided by MicroSim, called "TextEdit."

The output file always generated by PSpice is a text (ASCII) file that has the file type "OUT." I.e., if you submit a data file to PSpice named "MYCIRKUT.CIR," it will create an output file named "MYCIRKUT.OUT." This output file is created even if your run is unsuccessful due to input errors. The cause for failure is reported in the *.OUT file, so this is a good place to start looking when you need to debug your simulation model. You examine the *.OUT file with the TextEdit or Notepad programs. When everything works properly, you will find the output results in this file if you are running a DC analysis. If you are running a transient analysis or a frequency sweep analysis, there will be too much data for the *.OUT file. In these cases, we add a command to the *.CIR file that tells PSpice to save the numerical data in a *.DAT file.

The aforementioned *.DAT file is by default a binary (i.e., non-ASCII) file that requires a MicroSim application called PROBE for you to see the data. PROBE is installed with PSpice from the CD-ROM. If you want, you can change the default storage format to ASCII. This is not recommended because it requires more disk space to store the data in ASCII code. Later, we will describe the procedure for invoking PROBE and creating the *.DAT file. A companion file to the *.DAT file is the *.PRB file which holds initializing information for the PROBE program.

Another common method used by experienced PSpice users is the use of *.INC (include) files. These enable us to store frequently used subcircuits that have not yet been added to a library. Then we access these *.INC files with a single command line in the *.CIR file. It is very convenient.

Other files used with PSpice are *.LIB files where the details of complex parts are saved; we may discuss this later, but it is unlikely that we will engage in LIB file alterations until you are taking advanced courses.

When we begin using the schematic capture program that is bundled with PSpice, we will encounter some additional file types. These are the *.SCH (the schematic data, itself), *.ALS (alias files) and *.NET (network connection files).

Some Facts and Rules about PSpice

- PSpice is not case sensitive. This means that names such as *Vbus*, *VBUS*, *vbus* and even *vBUS* are equivalent in the program.
- All element names must be unique. Therefore, you can't have two resistors that are both named "Rbias," for example.
- The first line in the data file is used as a title. It is printed at the top of each page of output. You should use this line to store your name, the assignment, the class and any other information appropriate for a title page. PSpice will ignore this line as circuit data. Do not place any actual circuit information in the first line.
- There must be a node designated "0." (Zero) This is the reference node against which all voltages are calculated.
- Each node must have at least two elements attached to it.
- The last line in any data file must be ".END" (a period followed by the word "end.")
- All lines that are not blank (except for the title line) must have a character in column 1, the leftmost position on the line.
 - Use "*" (an asterisk) in column 1 in order to create a comment line.
 - Use "+" (plus sign) in column 1 in order to continue the previous line (for better readability of very long lines).
 - Use "." (period) in column 1 followed by the rest of the "dot command" to pass special instructions to the program.
 - Use the designated letter for a part in column 1 followed by the rest of the name for that part (no spaces in the part name).
- Use "whitespace" (spaces or tabs) to separate data fields on a line.
- Use ";" (semicolon) to terminate data on a line if you wish to add commentary information on that same line.

The above basic information is essential to using PSpice. Learn and understand these issues now to facilitate your use of the program.

References

1. *Spice: A Guide to Circuit Simulation and Analysis Using PSpice*; Tuinenga, Paul W.; © 1992, 1988 by Prentice-Hall, Inc.; ISBN: 0-13-747270; (my favorite)
2. *Computer-Aided Analysis Using SPICE*; Banzhaf, Walter; © 1989 by Prentice-Hall, Inc.; ISBN: 0-13-162579-9; (another good reference)
3. *SPICE for Circuits and Electronics Using PSpice*; Rashid, Muhammad H.; © 1990 by Prentice-Hall, Inc.; ISBN: 0-13-834672; (supports electronics well)
4. *SPICE for Power Electronics and Electric Power*; Rashid, Muhammad H.; © 1993 by Prentice-Hall, Inc.; ISBN: 0-13-030420; (best for power electronics)

Node Designations in PSpice

The original SPICE program developed decades ago at U. C. Berkeley, accepted data only on BCD punch cards. That's why it was not case sensitive; developers have preserved this lack of case sensitivity for backward compatibility. In the original SPICE program, users were

expected to designate nodes by number. Most users used small integers, and the numbers did not have to be contiguous. Today, most spice programs accept ordinary text for node designations. If you want to declare a node as "Pbus," you can. The only restriction seems to be that you can't embed spaces in a node name. Use the underscore ("_") character to simulate spaces.

Out of habit, most users of PSpice still use small integers as node designations. This often improves the readability of a PSpice source file or output file. In general, you should avoid extremely long textual names for node designations. Naming a node "Arlington_Junior_Chamber_of_Commerce" makes your files look choppy and hard to read. Also, you will sometimes have to *type* that long cumbersome name when you are performing analysis on the output data file. My suggestion is to use small integers as node designators for most cases. However, use short descriptive names for nodes whenever clarity is improved. "T1_col," when used to designate the collector node of transistor, T1, carries a lot more meaning than "37."

Large and Small Numbers in PSpice

PSpice is a computer program used mostly by engineers and scientists. Accordingly, it was created with the ability to recognize the typical metric units for numbers. Unfortunately, PSpice cannot recognize Greek fonts or even upper vs. lower case. Thus our usual understanding and use of the standard metric prefixes has to be modified. For example, in everyday usage, "M" indicates "mega" (10^6) and "m" stands for milli (10^{-3}). Clearly, this would be ambiguous in PSpice, since it is not case sensitive. Thus, in PSpice, a factor of 10^6 is indicated by "MEG" or "meg." "M" or "m" is reserved for 10^{-3} . Another quirk of PSpice is the designation for 10^{-6} . In most publications, the Greek letter, μ , is used for this multiple. Since there can be no Greek fonts (or any other special font designations) in PSpice, the early developers of PSpice borrowed a trick from those who used typewriters. Before the IBM Selectric typewriter was introduced, most writers of technical papers had to improvise for Greek letters. Since the Latin letter "u" (at least in lower case) sort of resembled the lower case Greek μ , it was widely used as a substitute for μ . Hence, either "U" or "u" stands for 10^{-6} in PSpice. Without further background explanations, these are the metric prefix designations used in PSpice:

Number Prefix	Common Name
• 10^{12} - "T" or "t"	<i>tera</i>
• 10^9 - "G" or "g"	<i>giga</i>
• 10^6 - "MEG" or "meg"	<i>mega</i>
• 10^3 - "K" or "k"	<i>kilo</i>
• 10^{-3} - "M" or "m"	<i>milli</i>
• 10^{-6} - "U" or "u"	<i>micro</i>
• 10^{-9} - "N" or "n"	<i>nano</i>
• 10^{-12} - "P" or "p"	<i>pico</i>
• 10^{-15} - "F" or "f"	<i>femto</i>

An alternative to this type of notation, which is in fact, the default for PSpice output data, is "textual scientific notation." This notation is written by typing an "E" followed by a signed or unsigned integer indicating the power of ten. Some examples of this notation are shown below:

- 656,000 = 6.56E5
- -0.0000135 = -1.35E-5
- 8,460,000 = 8.46E6

The Most Basic Parts

Here, we present the simplest circuit elements. Knowing how to model these ideal, linear circuit elements is an essential start to modeling more complex circuits. In each case, we will only present the most fundamental version of the part at this time. Later we will show you more sophisticated uses of the part models.

Ideal Independent Voltage Sources

We begin with the DC version of the ideal independent voltage source. This is the default form of this class of part. The beginning letter of the part name for all versions of the ideal independent voltage source is "V." This is the character that must be placed in column 1 of the line in the text file that is used to enter this part. The name is followed by the positive node designation, then the negative node designation, then an optional tag: "DC" followed by the value of the voltage. The tag "DC" (or "dc" if you prefer) is optional because it is the default. Later, when we begin modeling AC circuits and voltage sources that produce pulses and other interesting waveforms, we will be required to designate the type of source or it will default back to DC.

One of the interesting uses of ideal independent voltage sources is that of an *ammeter*. We can take advantage of the fact that PSpice saves and reports the value of current entering the positive terminal of an independent voltage source. If we do not actually require a voltage source to be in the branch where we want to measure the current, we simply set the voltage source to a zero value. It still calculates the current in the branch. In fact, we *require* an independent voltage source in a branch where that branch's current is the controlling current for a current-controlled dependent source.

Examples:

```
*name +node -node type value comment
Va 4 2 DC 16.0V; "V" after "16.0" is optional
vs qe qc dc 24m ; "QE" is +node & "qc" is -node
VWX 23 14 18k ; "dc" not really needed
vwx 14 23 DC -1.8E4 ; same as above
Vdep 15 27 DC 0V ; V-source used as ammeter
```

Resistors

Although PSpice allows for sophisticated temperature-dependent resistor models, we will begin with the simple, constant-value resistor. **The first letter of the name for a resistor must be "R."** The name is followed by the positive node, then the negative node and then the value in ohms or some multiple of ohms. The value of resistance will normally be positive. Negative values are allowed in order to permit an alternative model of an energy source. A value of zero, however, will produce an error. Later, we will introduce special resistor models that will permit additional analysis methods to be used.

The resistor is not an active device, so the polarity of its connection has no effect on the values of the voltages and currents reported in the solution. However, the current through a resistor is reported as that which flows from the node on the left to the node on the right in the source code line in which the resistor is entered. Thus .PRINT statements and PROBE queries that report resistor current may show negative values of current depending on the order in which you list the resistor's two nodes in the *.CIR file. If you want to see the resistor's current as a positive value, reverse the order of the nodes on the resistor's line in the *.CIR file and re-run the analysis. Nothing else will be affected and both solutions can be correct.

Examples:

```
*name +node -node value comment
Rabc 31 0 14k ; reported current from 31 to 0
Rabc 0 31 14k ; reported current changes sign
rshnt 12 15 99m ; 0.099 ohm resistor
Rbig 19 41 10MEG ; 10 meg-ohm resistor
```

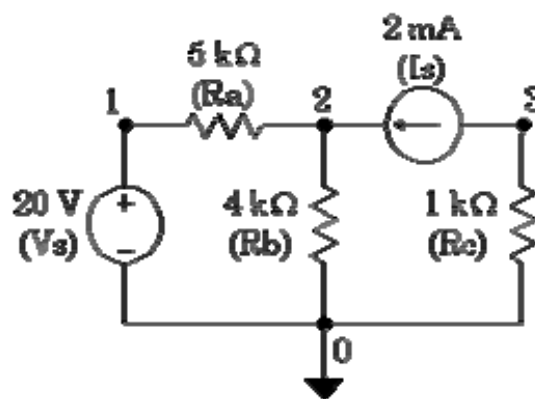
Ideal Independent Current Sources

The name of an ideal independent current source begins with the letter "I" in column 1 of the data file. As with the independent voltage source, we begin by introducing only the DC form of this part, but several other forms exist. Since the current source, is an active element, it matters greatly how it is connected. Designated current flows into the node written on the left, through the current source, out the node written on the right. As with the independent voltage source, the default type is DC. Remember that the so-called +node on a current source may have a negative voltage with respect to the so-called -node. This is due to the fact that the circuit external to the current source determines its voltage.

Examples:

```
*name +node -node type value comment
Icap 11 0 DC 35m ; 35mA flows from node 11 to 0
ix 79 24 1.7 ; "DC" not needed
I12 43 29 DC 1.5E-4 ;
I12 29 43 dc -150uA ; same as above
```

Circuit Example 1



```
Example_1 EXMPL01.CIR
Vs 1 0 DC 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 DC 2.0mA ; note the node placements
.END
```

The output file EXMPL01.OUT is below. This has been edited to remove extra lines.

```
Example_1 EXMPL01.CIR <== Title Line
Vs 1 0 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 2.0mA ; note the node placements
```

```
Example_1 EXMPL01.CIR <== Title Line
```

```
NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE
( 1) 20.0000 ( 2) 13.3330 ( 3) -2.0000 <==Results
```

```
VOLTAGE SOURCE CURRENTS
```

```
NAME CURRENT
```

```
Vs -1.333E-03 <== Current entering node 1 of Vs
```

```
TOTAL POWER DISSIPATION 2.67E-02 WATTS
```

```
JOB CONCLUDED
```

```
TOTAL JOB TIME .26
```

This was the bare-bones minimum problem we could ask of PSpice. Note that we obtained the node voltages which are sufficient information to calculate the resistor currents. However, there is another command that we can use to get even that done by PSpice.

Use of the .PRINT Command

One of the many "dot commands" in PSpice is the .PRINT command. It has many uses, but we will concentrate here on using it for printing DC voltages and currents. The .PRINT command can be repeated as often as necessary in an analysis. You can list as many items on a line as you wish.

However, we must keep in mind that the .PRINT command was designed to work with a DC or an AC sweep. This is a method of varying a parameter over a range of values so that we get a batch of cases solved all at once. Often, we do not actually want to run a sweep over many values of a parameter. We can circumvent the sweep by setting its range so that it can only run one value. Usually, a DC sweep is made by changing the values of a source; although we will later learn to sweep over other circuit parameters. For now, let's look at the syntax for a DC sweep command with the default linear type range.

```
.DC Sweep_Variable Starting_Value Stopping_Value Increment
```

For our example problem, we choose the voltage source and set the sweep variable range so that it cannot run more than one value:

```
.DC Vs 20.0 20.0 1.0
```

Since the starting value equals the stopping value, the analysis will only run for one case, i.e., for V_s at 20 volts. Remember that the only reason we are running the DC sweep statement is to *enable* the .PRINT command. The .PRINT command will not work unless there is a sweep going on. Note: What you enter in the .DC statement *overrides* any voltage value you may have placed in the part listing for the source.

Printing DC Voltages

In addition to printing the node voltages in which you type the letter "V" with the node number in parentheses, you can print the voltage between any pair of nodes; ergo, V(m,n) prints the voltage from node "m" to node "n."

```
.PRINT DC V(1) V(2) V(3) ; prints the node voltages
.PRINT DC V(1,2)          ; prints the voltage across Ra
.PRINT DC V(3,2)          ; prints the voltage across Is
```

Printing DC Currents

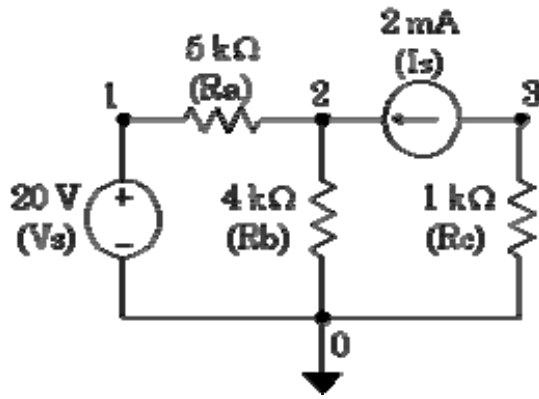
To print currents, you type the letter "I" with the element name in parentheses. Note that the reported current is that which flows into the element from the node listed on the left in the *.CIR file, through the element, and out the node listed on the right in the *.CIR file. If you want to change the sign of the reported current in a resistor, then swap the two nodes for that resistor.

```
.PRINT DC I(Ra)           ; prints the currents from + to - of
Ra
.PRINT DC I(Rb) I(Rc)     ; prints the currents through Rb and
Rc
```

Print Commands can be Combined

```
.PRINT DC V(1,2) I(Ra)    ; voltage and current for Ra
.PRINT DC V(2,0) I(Rb)    ; V(2,0) same as V(2)
.PRINT DC V(3,0) I(Rc)    ; V(3,0) same as V(3)
```

Use .PRINT with Previous Example



```
Example_2 EXMPL02.CIR
Vs 1 0 DC 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 DC 2.0mA ; note the node placements
.DC Vs 20 20 1 ; this enables the .print commands
.PRINT DC V(1,2) I(Ra)
.PRINT DC V(2) I(Rb)
.PRINT DC V(3) I(Rc)
.END
```

The output file EXMPL02.OUT is below. This has been edited to remove extra lines.

```
Example_2 EXMPL02.CIR
Vs 1 0 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 2.0mA ; note the node placements
.DC Vs 20 20 1 ; this enables the Print commands
.PRINT DC V(1,2) I(Ra)
.PRINT DC V(2) I(Rb)
.PRINT DC V(3) I(Rc)

Example_2 EXMPL02.CIR
**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C
Vs V(1,2) I(Ra)
2.000E+01 6.667E+00 1.333E-03 <== data for Ra
Example_2 EXMPL02.CIR
**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C
Vs V(2) I(Rb)
2.000E+01 1.333E+01 3.333E-03 <== data for Rb
Example_2 EXMPL02.CIR
**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C
Vs V(3) I(Rc)
<2.000E+01 -2.000E+00 -2.000E-03 <== data for Rc
JOB CONCLUDED
TOTAL JOB TIME .13
```

With a little bit of effort, we can get PSpice to do most of the work, most of the time. Note that using .PRINT has suppressed the default printing of all the node voltages. This is not a

problem in our case because we printed all three node voltages anyway. Be sure that you include everything you need in the .PRINT statements.

[Back to Main Page](#)

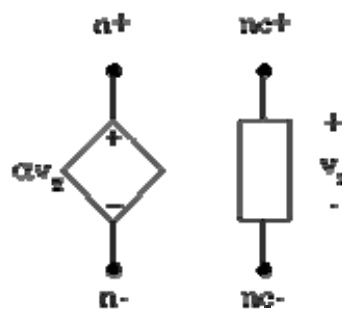
Last Modified: 04/08/2008 02:19:52

2nd Tutorial on PSpice

Simple Dependent Sources

We now extend our circuit parts list by adding the most basic dependent sources. The four dependent sources we now encounter are simple multiples of the controlling voltage or current. It is possible to model dependent sources that are complex nonlinear functions of several controlling voltages and/or currents. However, we will now concentrate on the basic linear dependent sources.

Voltage Controlled Dependent Voltage Source

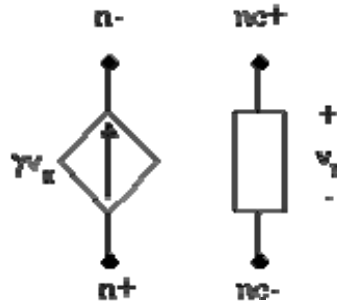


In the above figure, we find the dependent source whose positive terminal is designated as "n+" and whose negative terminal is designated as "n-." The controlling voltage is a branch voltage at some other circuit location. In this case, the positive terminal of the controlling branch is designated as "nc+" while the negative terminal is designated as "nc-." The "gain" of the dependent voltage source is α , a dimensionless quantity. For example, if v_x happened to be 16.0 volts while $\alpha = 4$, then node "n+" would be at 64.0 volts higher potential than node "n-."

The first letter of the part name for the voltage-controlled dependent voltage source is "E." This is the letter that must appear in column 1 of the *.CIR file describing the circuit. Some examples of the voltage-controlled dependent voltage source PSpice entries follow.

```
*Name  n+  n-  nc+  nc-  gain
Ebar   17  8   42   18   24.0; gain is 24
efix   3   1   11    0   20.0
efix   3   1    0   11  -20.0; same as above
efix   1   3   11    0  -20.0; same as above
efix   1   3    0   11   20.0; same as above
Ellen  12   0   20   41   16.0
```

Voltage Controlled Dependent Current Source

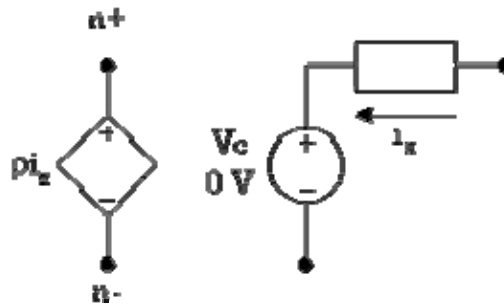


In the above voltage-controlled dependent current source a current equal to γ times v_x flows from node "n+" through the source and out node "n-." γ is called the transconductance and has the dimensions of siemens (inverse ohms). For example, if the controlling branch voltage, v_x , equals 6.0 volts and the transconductance, γ , is 0.25 siemens, the current produced by the dependent source is 1.5 amps.

The first letter of the part name for the voltage-controlled dependent current source is "G." Some examples of how this part is coded into the *.CIR file are shown below.

<i>*Name</i>	<i>n+</i>	<i>n-</i>	<i>NC+</i>	<i>NC</i>	<i>transconductance</i>
Glab	23	17	8	3	2.5
G1	12	9	1	0	4E-2
Grad	19	40	6	99	0.65
Grad	19	40	99	6	-0.65 ; same as above
Grad	40	19	99	6	0.65 ; etc.

Current Controlled Dependent Voltage Source



The current-controlled dependent voltage source as shown above, produces a voltage proportional to the current, i_x , in a different branch of the network. The transresistance, ρ , in ohms is multiplied by i_x in amps to produce the dependent source voltage in volts. Unlike the two previous examples, we cannot simply designate the controlling branch by its nodes. Since there could be multiple branches carrying very different currents between any pair of nodes, we must explicitly identify the branch of the controlling current. Eventually, we will be able to do this with any type of element. However, the only reliable method of doing this at present is to use an independent voltage source as an ammeter to report the current of the controlling branch to the dependent source. Usually, this means you must insert a zero-valued independent voltage source in series with the branch containing the controlling current so that the controlling current enters the positive terminal of the independent voltage source.

However, if there happens to be an independent voltage source that monitors the controlling current you can use it. If necessary, use a minus sign to get the right polarity.

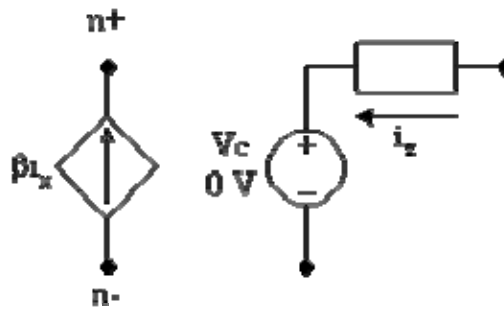
The first letter of the part name for the current-controlled dependent voltage source is "H." Some examples follow for this device.

```
*Name n+ n- Vmonitor transresistance
Hvx 20 12 Vhx 50.0
Vhx 80 76 DC 0V ; controls Hvx

Hab 10 0 V20 75.0
V20 15 5 DC 0V ; controls Hab

HAL 20 99 Vuse 10.0
Vuse 3 5 DC 20V ; actual voltage source
```

Current Controlled Dependent Current Source



The current-controlled dependent current source produces a current proportional to the controlling current, i_x , flowing in a different branch. The current gain, β , is dimensionless. Designating the control scheme is similar to setting up the current-controlled dependent voltage source previously discussed. We must use a voltage source connected in series with the controlling element so that the controlling current enters the positive terminal of the independent voltage source used as an ammeter. If no voltage source is needed for its voltage, we use a zero-valued voltage source as shown in the figure.

The first letter in the part name for this dependent source is "F." The syntax for entering this part in *.CIR files is shown in several examples below.

```
*Name n- n+ Vmonitor Gain
Ftrn 81 19 Vctl 50.0
Vctl 23 12 DC 0V ; controls Ftrn

Fcur 63 48 Vx 20.0
Vx 33 71 DC 0V ; controls Fcur

F3 2 0 V1 15.0
V1 3 1 DC 0V ; controls F3
```

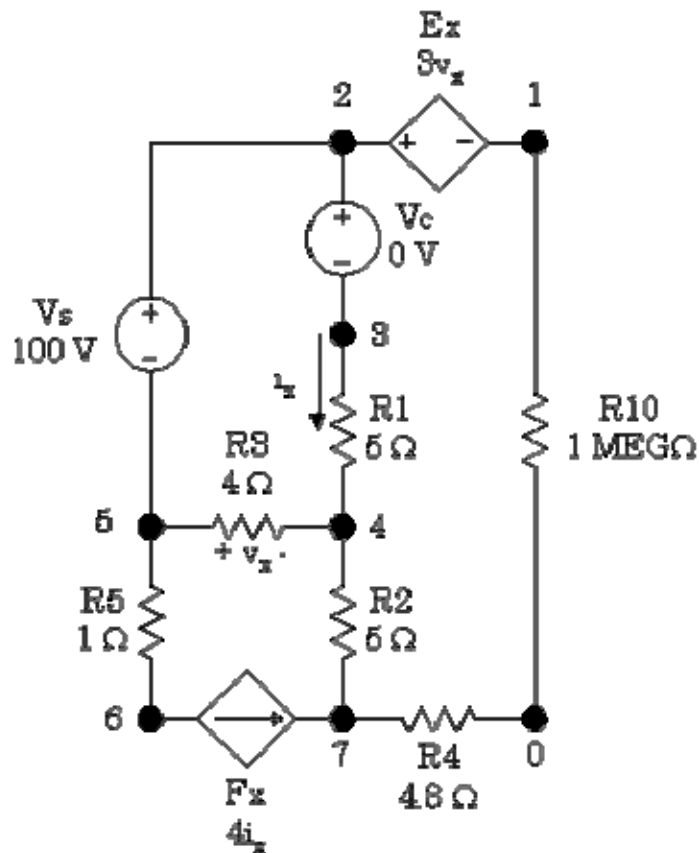
Using PSpice to find Thévenin Equivalent Circuit

In addition to performing general purpose circuit analysis, PSpice can be used to determine the Thévenin resistance and open circuit voltage of a circuit. This can be of great advantage if the circuit is complex, with several dependent sources, or if the circuit cannot be reduced by successive source transformations. The PSpice "dot command" that makes this easy, is ".TF," where "TF" indicates "transfer function." The transfer function is intended to find the ratio between a source voltage or current, and a resulting voltage difference or branch current.

This is useful in characterizing circuits. In addition to reporting the calculated transfer function ratio and input resistance at the source, PSpice reports the *output resistance* at the terminal pair of interest. The voltage across the terminal pair of interest is the Thévenin voltage and the output resistance is the Thévenin resistance. At this point we will ignore the transfer function ratio and the input resistance at the source. In fact, we do not care which source is chosen as long as we only want the Thévenin equivalent circuit parameters. An example of the syntax for the .TF command is shown below.

```
*command output_variable input_source  
.TF          V(4)          Vs
```

The above command will report the ratio between source Vs and node voltage V(4). If we wanted the Thévenin circuit from nodes 4 to 0, the output resistance reported would be our Thévenin resistance, and the voltage V(4) would be the Thévenin (open circuit) voltage. The input source can be a voltage or a current source, and the output variable can be a node voltage, branch voltage or a device current. Now we examine a specific example.



In this example, we want the Thévenin equivalent circuit from nodes 1 to 0. The 1 Megohm resistor is placed in the circuit because PSpice requires at least two connections to each node. This resistor is large enough that it will not have an effect on the calculations. Note the use of voltage source Vc which has the purpose of monitoring the control current, i_x , used for the current-controlled dependent current source, Fx. The input lines in the *.CIR file are shown below.

```
Thevenin Example No. 1
Vs 2 5 DC 100V
Vc 2 3 DC 0V; controls Fx
Fx 6 7 Vc 4.0; gain = 4
* n+ n- NC+ NC gain
Ex 2 1 5 4 3.0; gain = 3
R1 3 4 5.0
R2 4 7 5.0
R3 5 4 4.0
R4 7 0 4.8
R5 5 6 1.0
R10 1 0 1MEG; satisfies PSpice
* out_var input_source
.TF V(1,0) Vs
.END
```

Portions of the output file produced by this case will now be listed.

```
Thevenin Example No. 1
**** CIRCUIT DESCRIPTION
Vs 2 5 DC 100V
Vc 2 3 DC 0V; controls Fx
Fx 6 7 Vc 4.0; gain = 4.0
Ex 2 1 5 4 3.0; gain = 3.0
R1 3 4 5.0
R2 4 7 5.0
R3 5 4 4.0
R4 7 0 4.8
R5 5 6 1.0
Rab 1 0 1MEG
.TF V(1,0) Vs
```

```
Thevenin Example No. 1
**** SMALL SIGNAL BIAS SOLUTION TEMPERATURE = 27.000 DEG C
NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE
( 1) 180.0000 ( 2) -60.0010 ( 3) -60.0010 ( 4) -80.0010
( 5) -160.0000 ( 6) -176.0000 ( 7) -864.0E-06
VOLTAGE SOURCE CURRENTS
NAME CURRENT
Vs -4.000E+00
Vc 4.000E+00
TOTAL POWER DISSIPATION 4.00E+02 WATTS
**** SMALL-SIGNAL CHARACTERISTICS
V(1,0)/Vs = 1.800E+00 <== Transfer function
INPUT RESISTANCE AT Vs = 2.500E+01
OUTPUT RESISTANCE AT V(1,0) = 5.000E+00 <== Thévenin
resistance
JOB CONCLUDED
TOTAL JOB TIME .01
```

We conclude that the Thévenin resistance is 5 ohms and the open circuit voltage is 180 volts. Use of the .TF function allows us to get the answers in a single job. The alternative to using the .TF function would be to run one case with a large resistor across the terminal pair of interest (if necessary) to get the open circuit voltage; and then run a second case with a zero-valued voltage source across the terminal pair to get the short circuit current. Then divide the short circuit current into the open circuit voltage to get the Thévenin resistance. We prefer the ".TF" method for obtaining Thévenin equivalent circuits.

[Back to Main Page](#)

Last Modified: 04/08/2008 02:19:53

3rd Tutorial on PSpice

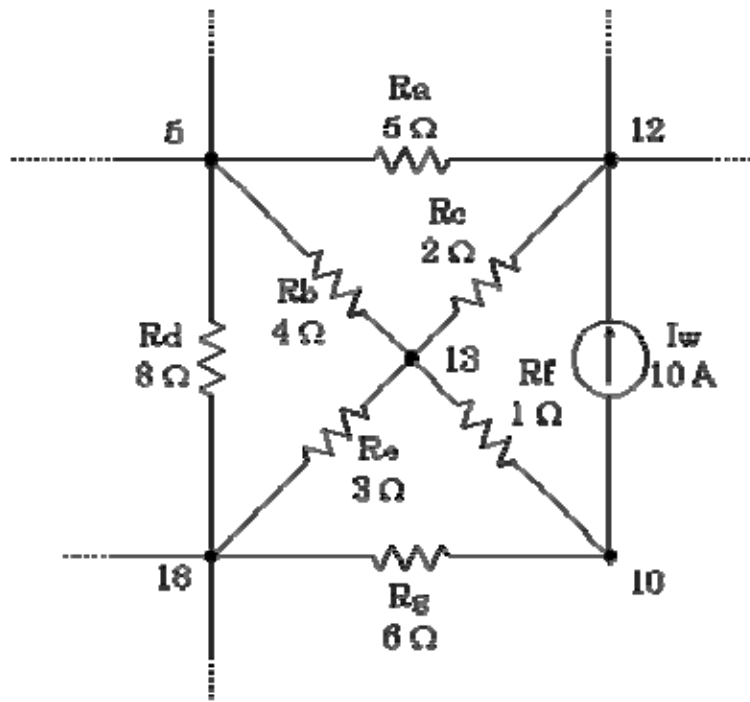
Simple Subcircuits in PSpice

One of the more useful concepts in PSpice is the use of *subcircuits* to group elements into clusters in order to replicate the clusters without having to re-enter all the elements each time. This is very useful for several reasons. First is the labor savings of replacing many lines of circuit data with a single subcircuit call. Second, the use of a subcircuit usually improves clarity by removing confusing clutter. The user can suppress printing unwanted details internal to a subcircuit, thus making the output easier to understand. If desired, the user can place often-used subcircuits into an *include* file so that the main source file for the problem is kept simple. Then the definition of the subcircuit is out of sight entirely.

Coding a Subcircuit

Each subcircuit used in a study must have a unique name. This is true of any other circuit element. Also, there must be a list of at least two nodes that can be connected to elements external to the subcircuit. A subcircuit can have many external node connections, if needed. Later, we will find that parameters can be passed to a subcircuit in order to allow unique behavior and responses from an instance of a subcircuit.

The initial line of a subcircuit section must begin with ".SUBCKT," followed by the name and then the external node list. After that, optional features (not to be discussed yet) can be added. The best method of understanding the use of a subcircuit is by example. Below, we find a cluster of components that can be combined into a subcircuit.



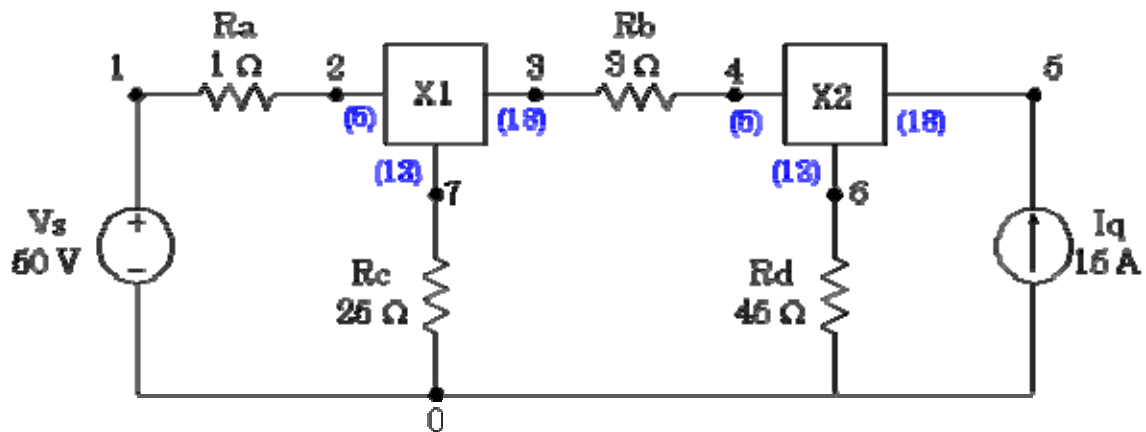
Note that nodes 5, 12 and 18 have external connections. Therefore, they must be included in the node list in the subcircuit definition. Nodes 10 and 13 do not have external connections and need not be (indeed *should* not be) included in this node list. They are internal nodes and will be used to help define the subcircuit. Now, we can code the above subcircuit as follows. Note that the code could be embedded into the rest of the code for the main circuit or could be placed in a separate *include* file.

```
*          name          nodelist
.SUBCKT Example_1      5 12 18
Iw  10 12 DC 10A
Ra   5 12 5.0
Rb   5 13 4.0
Rc  12 13 2.0
Rd   5 18 8.0
Re  13 18 3.0
Rf  10 13 1.0
Rg  10 18 6.0
.ENDS
```

Note that the subcircuit section must be terminated with a ".ENDS" command.

Invoking a Subcircuit

All subcircuit calls are made by declaring a part with a unique name beginning with "X," followed by the node list and then the subcircuit name. The node list in the calling statement must have the same number of nodes as the node list in the subcircuit definition. To demonstrate the use of the calling statement, we present the following main circuit which contains two instances of the above subcircuit. X1 and X2 are the two instances of the subcircuit "Example_1." For added clarity, the subcircuit's defined external nodes are shown in parentheses. Note that these nodes are mapped into the main circuit by *different names*.



The code for the above circuit with the subcircuit included follows:

```
Subcircuit Example No. 1
*          name          nodelist
.SUBCKT Example_1      5 12 18
Iw  10 12 DC 10A
Ra   5 12 2.0
Rb   5 13 5.0
```

```
Rc  12  13  2.0
Rd   5  18  8.0
Re  13  18  3.0
Rf  10  13  1.0
Rg  10  18  6.0
.ENDS
Vs   1   0  DC  50V
Ra   1   2  1.0 ; different from Ra above
Rb   3   4  3.0 ; different from Rb above
Rc   7   0 25.0 ; different from Rc above
Rd   6   0 45.0 ; different from Rd above
*   nodelist      name
X1   2   7   3   Example_1
X2   4   6   5   Example_1
.END
```

Scope of Element Names and Nodes in a Subcircuit

Scope of names and nodes is local to a subcircuit. In the main circuit of which the above subcircuit is a part, there is a node 5 and there are resistors with the names of "Ra," "Rb," "Rc," and "Rd," and PSpice can keep these apparent duplications separated. If the subcircuit were invoked as "X1," for example, PSpice would consider the subcircuit parts as "X1.Ra," "X1.Rb" and so on. Additionally, the internal node numbers would be treated as "X1.5," "X2.5," "X2.13" and so forth. Thus PSpice maintains uniqueness of element names and node numbers.

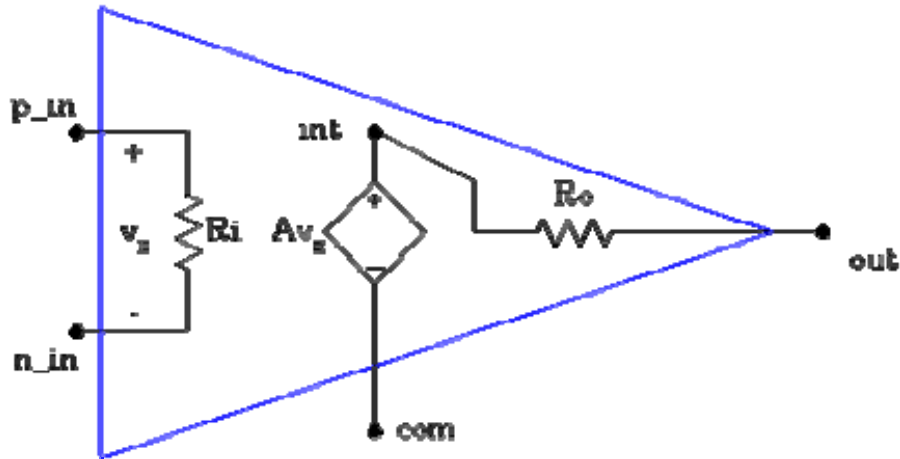
Nesting of Subcircuits

Subcircuit *calls* may be nested as long as they are not circular. In other words, you can have a part name starting with "X" within a .SUBCKT/.ENDS block provided that the "X" part definition does not call on that block for its own definition.

However, subcircuit *definitions* may *not* be nested. I.e., you can't have one .SUBCKT/.ENDS block nested within another.

An Op-Amp Example

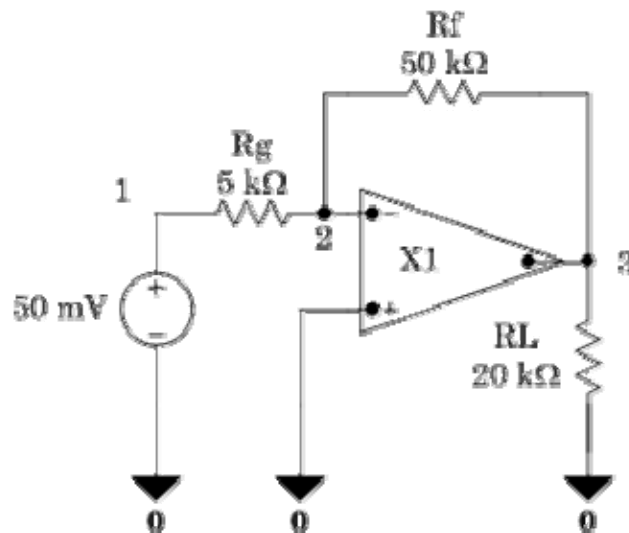
At this stage of knowledge about PSpice, we can model a simple op-amp as a subcircuit. Alas, we will not be able to show its saturation characteristics until we explain the use of the "TABLE" feature of PSpice. However, we can do a credible job of modeling an op-amp as long as it isn't allowed to saturate. The figure below illustrates this simple model of an op-amp.



In an ideal op-amp, R_i , the input resistance, and A , the open-loop gain, are infinite. Also, R_o , the output resistance, is zero. Here, we will use "typical" values of a practical op-amp. Let $A = 100,000$, $R_i = 500\text{ k}\Omega$, and $R_o = 50\ \Omega$. After all, PSpice doesn't accept infinity as a number and resistors cannot be set to zero. We will also use *text* for node designations here. Also, note the internal node "int" that is not included in the node list. The key part to the op-amp is the voltage-controlled dependent voltage source designated as part "Ex" in the listing. For a more complete description of this part, see [Tutorial No. 2](#), where dependent sources are introduced. Code to define the subcircuit follows.

```
.SUBCKT OpAmp p_in n_in com out
Ex int com p_in n_in 1e5
Ri p_in n_in 500k
Ro int out 50.0
.ENDS
```

The main circuit with which we will test this op-amp subcircuit follows. We will use a simple inverting amplifier circuit for which we can verify the results by inspection.



Subcircuit Example No. 2 - Inverting OpAmp

```
.SUBCKT OpAmp p_in n_in com out
Ex int com p_in n_in 1e5
Ri p_in n_in 500k
```

```
Ro  int  out  50.0
.ENDS
Vg  1    0    DC    50mV
Rg  1    2    5k
Rf  2    3    50k
RL  3    0    20k
X1  0    2    0     3    OpAmp
.END
```

The output file (edited to remove excess lines, etc.) is as follows:

```
Subcircuit Example No. 2 - Inverting OpAmp
**** CIRCUIT DESCRIPTION
.SUBCKT OpAmp p_in n_in com out
Ex int com p_in n_in 1e5
RI p_in n_in 500k
Ro int out 50.0
.ENDS
Vg 1 0 DC 50mV
Rg 1 2 5k
Rf 2 3 50k
RL 3 0 20k
X1 0 2 0 3 OpAmp
Subcircuit Example No. 2 - Inverting OpAmp
**** SMALL SIGNAL BIAS SOLUTION TEMPERATURE = 27.000 DEG C
NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE
( 1) .0500 ( 2) 5.017E-06 ( 3) -.4999 (X1.int) -.5017
VOLTAGE SOURCE CURRENTS
NAME CURRENT
Vg -9.999E-06
TOTAL POWER DISSIPATION 5.00E-07 WATTS
JOB CONCLUDED
TOTAL JOB TIME .19
```

Discussion of Results

Had this op-amp been ideal, the closed loop gain would have been $-10 \cdot -(R_f / R_g)$. Then the output voltage would have been $-0.5V$. Instead, we calculated $-0.4999V$. Also, an ideal op-amp would have produced a voltage of zero at the negative input. Instead, we see about 5 microvolts. These discrepancies are due to the more realistic model of the op-amp. Yet the difference is small. It seems that the concept of using the ideal op-amp model does not lead to excessive error in this case.

The advantage of using the subcircuit in PSpice is now apparent. We could have replicated the subcircuit many times, using only one line of code per replication. Later, we will add to our knowledge of the features of subcircuits.

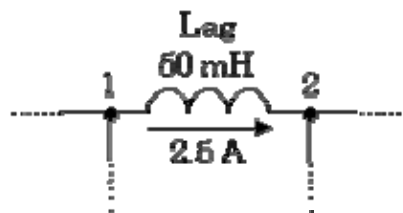
[Back to Main Page](#)

4th Tutorial on PSpice

Linear Inductors in PSpice

The next passive element we add to our parts list is the linear inductor. This part name begins with the letter, L, in column 1 of the source listing. Be aware that PSpice enables this part to access a nonlinear model description. That will be explained in a later tutorial. For now, our inductor model is a linear device incapable of saturation.

The inductor stores energy in its magnetic field. This makes it necessary to be able to specify its initial current in a simulation. Although we can include inductors in DC circuit simulations, there is usually little advantage in doing so because the inductor behaves as a short circuit under steady-state DC excitation. In steady-state AC simulations the inductor behaves as an imaginary impedance. We do not specify initial current in an inductor in either of those steady-state conditions. However, when simulating transient operations, we often need to specify this initial current.



The figure shown above shows the circuit symbol for an inductor with node designations of "1" and "2," an initial current of 2.5 A, and a value of 50 mH. An appropriate code listing for entering this element into a PSpice circuit file is:

```
*name nodelist L_val
Lag 1 2 50m IC=2.5
```

Note that the initial current is assumed to flow from the first node in the node list through the inductor towards the second node in the node list. If there is a need to change the direction of this initial current, either reverse the order of the nodes in the node list or place a minus sign in front of the value of the initial current. For better readability, the above line could be written as:

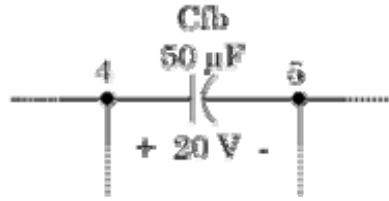
```
Lag 1 2 50mH IC=2.5A
```

The "H" for henrys and the "A" for amps will be ignored by PSpice.

Linear Capacitors in PSpice

The capacitor is the second energy storing circuit component we add to our parts list. We will assume that the capacitor is ideal in the sense of being linear and lossless. Since it can store energy, PSpice provides a method for specifying the initial voltage across the capacitor. This is useful for simulations of transient behavior of circuits with capacitors. The figure

shown below illustrates a capacitor with node designations and an initial voltage of 20 from node 4 to node 5. The part name for a capacitor must start with the letter, C.



An appropriate code listing to represent this capacitor in a PSpice listing is:

```
*name nodelist C_val
Cfb 4 5 50u IC=20
```

The capacitance of the above element is 50 μ F. This can be represented as "50u" in PSpice. (See [PSpice Tutorial No. 1](#) for a description of the system for metric prefixes in PSpice.) Note that the polarity of the initial voltage (as shown) is such that the positive side is the first node in the list with the negative side on the second node in the list. To reverse the polarity of the initial voltage for the simulation, either reverse the order of the nodes in the node list or place a minus sign in front of the value in the "IC=" phrase. For better clarity, the above capacitor could be coded as:

```
Cfb 4 5 50uF IC=20V
```

PSpice would ignore the "F" for farads and the "V" for volts.

Transient Analysis Using PSpice

One of the most interesting aspects of circuit analysis is the study of natural and step responses of circuits and the responses of circuits to time-varying sources. To perform these analyses we introduce another group of "dot" commands.

Use of the .TRAN command

This is the command that passes the user's parameters for performing the transient analysis on a circuit to the PSpice program. There are four time parameters and an instruction to use the initial conditions rather than calculated bias point values for starting conditions. First, we show a sample .TRAN statement and then we will describe its parameters.

```
* prt_stp t_max prt_dly max_stp
.TRAN 20us 20ms 8ms 10us UIC
```

In the above statement, the "20us" value labeled "prt_stp" (*print step*) is the frequency with which data is saved. In this case, the system variables are stored each 20 μ s of simulation time. The actual time steps used by PSpice may be different from this. The second parameter, "20ms," labeled as "t_max" (*final time*) is the value of time at which the simulation will be ended. Since PSpice starts at $t = 0$, there will be a total of 20ms time span of simulation for the circuit. The third parameter, "8ms," labeled as "prt_dly" (*print delay*) is the print delay time. In some cases, we do not want to store the data for the entire time span of the simulation. In our sample statement shown above, we ignore the data from the first

8ms of simulation and then store the data for the last 12ms. Most of the time, this parameter is set to zero or not used. The fourth parameter, "10us," labeled as "max_stp" (*max step*) is the maximum time step size PSpice is allowed to take during the simulation. Since PSpice automatically adjusts its time step size during the simulation, it may increase the step size to a value greater than desirable for displaying the data. When the variables are changing rapidly, PSpice shortens the step size, and when the variables change more slowly, it increases the step size. Use of this parameter is optional. The last parameter in our list is "UIC." It is an acronym for "Use Initial Conditions." Unless you include this parameter, PSpice will ignore the initial conditions you set for your inductors and capacitors and will use its own calculated bias point information instead. Note that the use of the letter "s" after the numbers in the .TRAN statement is optional. PSpice assumes these values are seconds and actually ignores the "s." However, it is recommended that you use units until you are extremely familiar with all of these commands and definitions.

Now, we will examine some more .TRAN examples.

```
.TRAN 10ns 500us
```

In the above example, PSpice will save data at each 10ns interval of the simulation starting at $t = 0$ until the final time of 500 μ s. I.e., there is no print delay and the user has given full control of the calculation step size to PSpice. In addition, PSpice will calculate its own initial conditions for any inductors and capacitors, ignoring any initial conditions set by the user.

```
.TRAN 50m 2.5 0 10m UIC
```

In the above statement, PSpice collects the data at each 50ms time interval starting from zero up to 2.5s. A zero was required as a placeholder for the print delay parameter since the maximum step size of 10ms was specified. PSpice will use the designated initial conditions of capacitor voltage and inductor current. Notice that the units were left off the numbers in this statement. Only the prefixes which size the values are needed.

Use of the .PROBE command

In addition to specifying the time parameters for a transient solution of a circuit problem, we need to specify how the data is to be saved. In most cases, this simply means that we include a line in the *.CIR file consisting of ".PROBE." This instructs PSpice to create a data file and store the data it calculates. If we create a circuit listing named "CIRCUIT1.CIR" containing a ".TRAN" statement and a ".PROBE" statement, PSpice will create a file named "CIRCUIT1.DAT" holding the data as well as the usual "CIRCUIT1.OUT" file with basic information about the circuit. By default, the data file created by PSpice is a binary data file; i.e., you can't read it with a text editor. This is the most efficient way of saving the data. However, there is an optional parameter (/CSDF) for the .PROBE statement that causes PSpice to save the data in a Common Simulation Data Format which is a text format that allows you to look at the raw data with a text editor. However, it will take up more space and PROBE doesn't load it for graphing. You will need to make a second run without the /CSDF parameter if you want to plot the data.

Also by default, .PROBE causes *all* the circuit variables to be saved, including all the variables inside each instance of each subcircuit. In some cases, this can amount to a lot of data. If you simulate a large complex circuit with many parts and need to save data at short

time intervals over a very long time span, you can easily create gigabyte-size "DAT" files. To avoid this, you can specify the values you want to save. If the ".PROBE" command is issued without any parameters, everything is saved. If you specify the quantities you want saved, *only* those quantities will be saved. We will now examine some .PROBE statements.

`.PROBE`

All the above statement does is the enable PSpice to save everything in a binary DAT file.

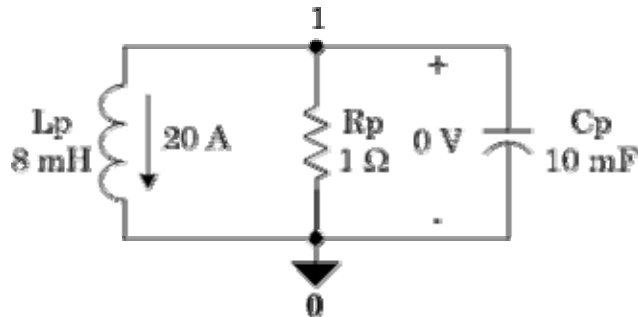
`.PROBE/CSDF`

The above statement enables PSpice to save everything in a CSDF file that can be opened (and edited) with a text editor. You can also read and understand the values. Unfortunately, PROBE cannot make plots from the data in this form.

`.PROBE V(5,23) I(Rx) I(L4)`

The above statement tells PSpice to save only the voltage drop between nodes 5 and 23, the current through resistor, Rx, and the current through inductor, L4, all in binary format. No other data will be saved.

Example of Transient Circuit Analysis



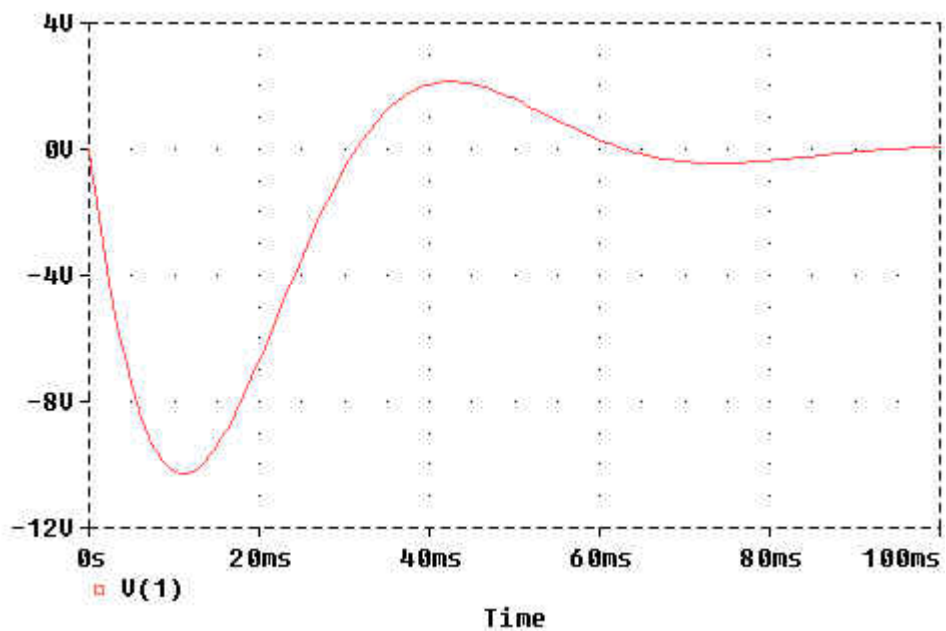
The complete listing for the "RLCNAT01.CIR" file is as follows:

```
Natural Response of a parallel RLC circuit
Rp  0  1  1.0
Lp  1  0  8mH  IC=20A
Cp  1  0  10mF  IC=0V
.TRAN 500us 100ms 0s 500us UIC
.PROBE
.END
```

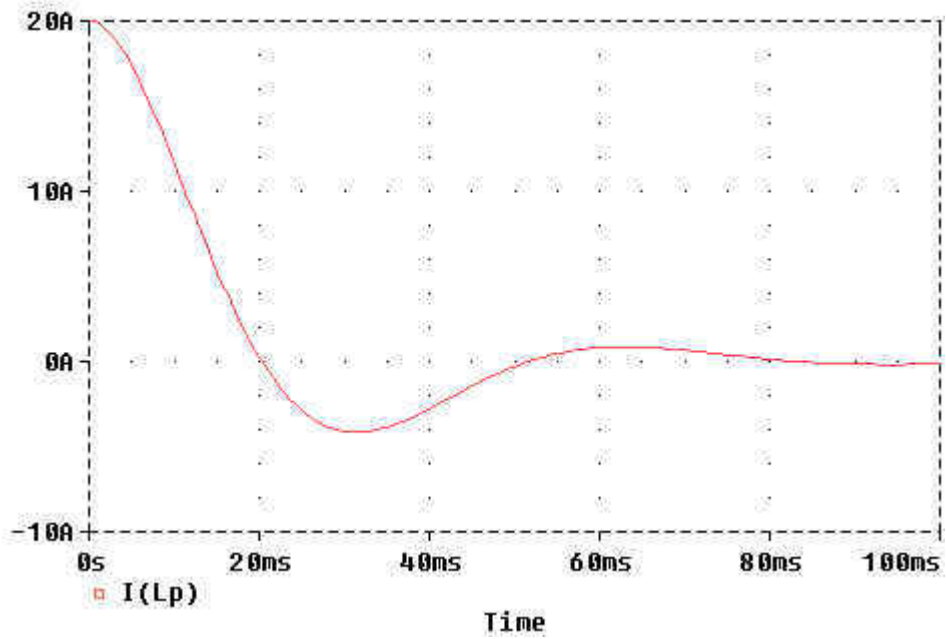
In the above example, the eight millihenry inductor, Lp, has an initial current of 20 amps flowing from node 1 through the inductor to node 0. The 10 millifarad capacitor, Cp, has an initial voltage of 0 volts. Both the print step size and the maximum step size are set to 500μs and the final time is 100ms. There is no print delay, and PSpice is instructed to use the initial conditions provided. The "RLCNAT01.OUT" file is listed below. There is little information in it because the text file can show very little of the transient behavior of the circuit.

```
**** 07/17/98 18:47:40 ***** NT PSpice 8.0 (July 1997)
Natural response of a parallel RLC circuit
RP 0 1 1.0
LP 1 0 8mH IC=20A
Cp 1 0 10mF IC=0V
.TRAN 500us 100ms 0s 500us UIC
.PROBE
JOB CONCLUDED
TOTAL JOB TIME .16
```

For meaningful information about the transient response we need to use another program that is bundled with PSpice. This program is named PROBE. The Probe program graphs the data that was saved in the "RLCNAT01.DAT" file. To invoke this program you left-click on "Run Probe" in the PSpice *File* menu. Probe will automatically open the DAT file you have just created. You can also launch Probe from the Start menu of Windows, but you will then need to go to Probe's File menu and open the DAT file you want to see. After you have Probe running with the proper DAT file open, choose "Add" in the Probe *Trace* menu. You will see a list of circuit variables that can be displayed. Choose V(1), the voltage at node 1, and then click "OK." You should see the following trace in Probe.



In Probe, click on the V(1) at the lower left corner (not here, you need to be running *Probe*) and then hit the "Delete" key. Then go back to the *Trace* menu in Probe and choose "Add" again. This time choose I(LP) and click "OK." You should see the following trace of the inductor current:



Actually, you will see a negative of the above traces. In order to get the white background as you see above, you will need to modify the "INI" file for PSpice. That will be the topic of [another tutorial](#).

[Back to Main Page](#)

5th Tutorial on PSpice

Steady-State AC Analysis in PSpice

In addition to DC circuit analysis and transient analysis, PSpice can be used to work steady-state phasor problems. To see the results of this analysis in the .OUT file, we will want to use a new form of the .PRINT command. In the first tutorial, we learned that the .PRINT DC command would not work unless we enabled it with the .DC command. This was the DC sweep command although we only allowed it to sweep a single value of voltage. We have a somewhat similar situation when we need to print AC values; i.e., we will use the .AC command to *enable* the .PRINT AC command to print our phasor voltages and currents.

AC Voltage and Current Sources`

Up to now, all our voltage and current sources were DC. We learned the syntax of the DC source in the first tutorial. The syntax for an AC source is very similar. The AC source is assumed to be a *cosine* waveform at a specified phase angle. Its frequency must be defined in a separate ".AC" command that defines the frequency for *all* the sources in the circuit. The unique information for the individual source is: the name, which must start with "V" or "I," the node numbers, the magnitude of the source, and its phase angle. Some examples follow.

```
*name nodelist type value phase(deg)
Vac 4 1 AC 120V 30
Vba 2 5 AC 240 ; phase angle 0 degrees
Ix 3 6 AC 10.0A -45 ; phase angle -45
degrees
Isv 12 9 AC 25mA ; 25 milliamps @ 0
degrees
```

Notice that the type, AC, *must* be specified, because the default is DC. If the phase angle is not specified it will be assumed as zero degrees. The units of the phase angle will be in degrees. As before, the "V" after the voltage value is optional, as is the "A" after the current value in a current source. The polarity of the AC voltage source is determined as if the voltage were a cosine function of ωt at $t = 0$. Then the node on the left is the positive node and the node on the right is the negative node. Similarly, the polarity of the AC current source is determined as if the current were a cosine function of ωt at $t = 0$. Then positive current flows into the source from the node on the left, passes through the source, and leaves the source from the node on the right.

A note of caution is needed here. By now, some of you may have discovered the "SIN" type of source by reading some of the supplementary material. The SIN is one of several useful source types (also EXP, PULSE, PWL & SFFM to name a few) that are used for *transient* analysis. Do not attempt to use *SIN* for steady-state (phasor) AC analysis nor for frequency sweeps. The SIN type is a time-based function for time-based analysis, whereas the AC type is used in frequency-based modeling. Since phasor analysis uses frequency-based models of circuit elements, always use the AC type as described in this tutorial for phasor analysis of circuits.

Use of the .PRINT AC Command

Before the .PRINT command will work, it must be enabled by the .AC command. The .AC command was designed to make a sweep of many frequencies for a given circuit. This is called a *frequency response* and will be discussed in a later tutorial. Three types of ranges are possible for the frequency sweep: LIN, DEC and OCT. At this time we only want a single frequency to be used so it does not matter which one we choose. We will pick the LIN (linear) range to designate our single frequency. Some examples of the .AC statement follow.

```
*      type #points  start stop
.AC   LIN   1        60Hz  60Hz;  <== what we want now.
.AC   LIN  11        100   200;  <== a linear range sweep
.AC   DEC  20        1Hz   10kHz; <== a logarithmic range sweep
```

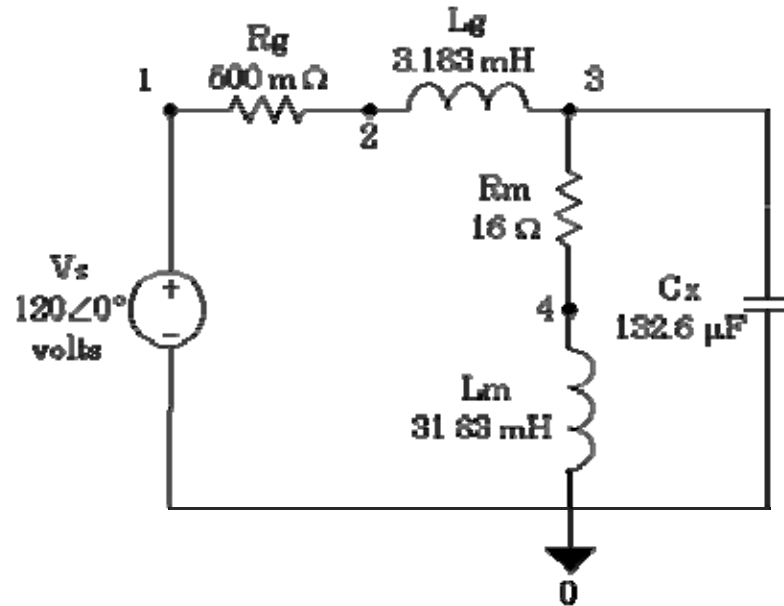
The first statement above performs a single analysis using the frequency of 60 Hz. Placing the units "Hz" after the value is optional. The second statement would perform a frequency sweep using frequencies of 100Hz, 110Hz, 120Hz, 130Hz, 140Hz, 150Hz, 160Hz, 170Hz, 180Hz, 190Hz and 200Hz. This will not be used here. The third statement performs a logarithmic range sweep using 20 points per decade over a range of four decades. This will be useful later for studying frequency response of circuits.

Finally, we can discuss the actual .PRINT AC command. Printing the components of phasor values (complex numbers) requires some options. There are four expressions needed for this: magnitude, phase (angle), real part, and imaginary part. In addition, we can print voltages or currents. For instance, to print the magnitude of a voltage between nodes 2 and 3, we would specify "VM(2,3)." The phase angle of this same voltage would be "VP(2,3)" and would be printed in degrees. If we need the current magnitude through resistor Rload, we would specify "IM(Rload)." The real part of the voltage on node 7 would be specified "VR(7)" and its imaginary part, "VI(7)." As with the .PRINT DC command, there is no limit on the number of times it can be used in a listing; nor is there a limit on how many print requests can be on a single line. Some complete examples follow:

```
.PRINT AC VM(30,9) VP(30,9); magnitude & angle of voltage
.PRINT AC IR(Rx) II(Rx); real & imag. parts Rx current
.PRINT AC VM(17) VP(17) VR(17) VI(17); the whole works on node
17
```

Example Circuit

We will analyze the following circuit at a frequency of 60 Hz.



```
60 Hz AC Circuit
Vs 1 0 AC 120V 0
Rg 1 2 0.5
Lg 2 3 3.183mH
Rm 3 4 16.0
Lm 4 0 31.83mH
Cx 3 0 132.8uF
.AC LIN 1 60 60
.PRINT AC VM(3) VP(3) IM(Rm) IP(Rm) IM(Cx) IP(Cx)
.END
```

In the above listing, the .AC command sets up the analysis for a single solution at 60 Hz. The .PRINT AC command tells PSpice to report on the voltage magnitude and phase angle at node 3, and the current magnitude and phase angle for the current through resistor Rm and the current magnitude and phase angle through capacitor Cx. The resulting output file (edited to delete clutter) follows:

```
60 Hz AC Circuit
**** CIRCUIT DESCRIPTION
Vs 1 0 AC 120 0
Rg 1 2 0.5
Lg 2 3 3.183mH
Rm 3 4 16.0
Lm 4 0 31.83mH
Cx 3 0 132.6uF
.AC LIN 1 60 60
.PRINT AC VM(3) VP(3) IM(Rm) IP(Rm) IM(Cx) IP(Cx)
```

```
60 Hz AC Circuit

**** SMALL SIGNAL BIAS SOLUTION

TEMPERATURE = 27.000 DEG C

NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE
( 1) 0.0000 ( 2) 0.0000 ( 3) 0.0000
```

```
NODE VOLTAGE ( 4) 0.0000
VOLTAGE SOURCE CURRENTS
```

```
NAME CURRENT
Vs 0.000E+00
```

```
TOTAL POWER DISSIPATION 0.00E+00 WATTS
```

```
60 Hz AC Circuit
```

```
**** AC ANALYSIS TEMPERATURE = 27.000 DEG C
```

```
FREQ      VM(3)      VP(3)      IM(Rm)      IP(Rm)
6.000E+01 1.203E+02 -3.332E+00 6.014E+00 -4.020E+01
```

```
FREQ      IM(Cx)      IP(Cx)
6.000E+01 6.013E+00 8.667E+01
```

```
JOB CONCLUDED TOTAL JOB TIME .26
```

Notice that the small signal bias solution yields zero for the voltages. This is the DC part of the solution which is zero in this case because there was no DC excitation. The AC analysis has been printed in blue. The voltage at node 3 is $120.3 \angle -3.332^\circ$ volts and the current through the capacitor is $6.014 \angle 86.67^\circ$ amps. Theory predicts that the current through a capacitor leads the voltage across the capacitor by 90° , which it does.

Summary of AC Phasor Circuit Analysis Using PSpice

- Use *AC* as the type for all independent sources
- Specify phase angle of sources if other than zero degrees
- There must be a ".AC" command to specify the frequency to be used for all sources
- Use a *.PRINT AC* command to specify which voltages and currents are to be listed in the output file
- **M** indicates *magnitude*, **P** indicates *phase angle*, **R** indicates *real part* and **I** indicates *imaginary part*, when these letters follow **V** (for voltage) or **I** (for current).