

SANDIA REPORT

SAND2013-7100
Unlimited Release
Printed August 2013

Application Note: Using Open Source Schematic Capture Tools With **Xyce**™

Thomas V. Russo

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161


Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



Application Note: Using Open Source Schematic Capture Tools With

Thomas V. Russo
Electrical Models and Simulation
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1177

Abstract

The development of the  Parallel Electronic Simulator has focused entirely on the creation of a fast, scalable simulation tool, and has not included any schematic capture or data visualization tools. This application note will describe how to use the open source schematic capture tool *gschem* and its associated netlist creation tool *gnetlist* to create basic circuit designs for **Xyce**, and how to access advanced features of **Xyce** that are not directly supported by either *gschem* or *gnetlist*.

Contents

1. Introduction	9
Target Audience	9
Prerequisites	10
Obtaining gEDA and friends	10
Simplified installation on Ubuntu, Debian, Mac OS X and FreeBSD	10
gEDA on CentOS and Red Hat Enterprise Linux	11
gEDA on Windows	12
Useful start-up options	12
Starting gschem	12
2. A first circuit with gEDA and Xyce	13
Overview	13
Drawing the circuit	14
Adding voltage sources and editing attributes	14
Adding devices	17
Connecting the devices	19
Completing the schematic	20
Adding a Model Card	22
Generating the netlist	23

Adding analysis and output statements	24
Simulation with Xyce	25
Plotting the results	25
Changing the analysis	26
Summary	27
3. Creating hierarchical designs	28
The Common Emitter Amplifier — flat design	29
Common Emitter Amplifier — hierarchical netlist version	32
Overview	32
The Common Emitter Amplifier subcircuit	33
Creating a symbol for the subcircuit	35
Drawing the higher level schematic	38
Summary	41
4. Accessing Xyce-specific Netlist Features	43
The Nonlinear Resistor Netlist	43
The nonlinear resistor schematic — workaround version	45
Patching spice-sdb, and an improved Xyce nonlinear resistor schematic	51

Appendix

A Installing gEDA on Windows	55
---------------------------------------	----

List of Figures

2.1	Diode Clipper circuit schematic from Xyce Users' Guide.	13
2.2	Select Component... dialog box.....	15
2.3	Single Attribute Editor dialog box.....	15
2.4	Edit Attributes dialog box.	16
2.5	Sources for the Diode Clipper circuit.	17
2.6	Placed components for Diode Clipper circuit.	18
2.7	First two nets added to Diode Clipper circuit.	19
2.8	Remaining nets added to Diode Clipper circuit.	20
2.9	Complete Diode Clipper schematic.	21
2.10	Model component attributes for diode clipper.	22
2.11	DC Sweep plot for diode clipper circuit.	26
3.1	Schematic for flattened common emitter amplifier circuit	29
3.2	Complete schematic for flattened common emitter amplifier circuit	31
3.3	Schematic for the common emitter amplifier subcircuit.	33
3.4	First pin added to symbol	37
3.5	Complete Amplifier symbol	39
3.6	Common Emitter Amplifier driver and load	40
4.1	Nonlinear Resistor Netlist	44

4.2	Nonlinear Resistor Top Level Schematic	46
4.3	Nonlinear Resistor Subcircuit Schematic	47
4.4	Nonlinear Resistor Subcircuit Schematic	52
4.5	Nonlinear Resistor Top-Level Schematic	53
A.1	Selecting Computer Properties	56
A.2	Advanced System Settings	57
A.3	Copying the full path	58
A.4	Navigate to the "lib\gdk-pixbuf-2.0\2.10.0" directory	59

1. Introduction

The **Xyce**[™] Parallel Electronic Simulator was written at Sandia National Laboratories as an in-house, custom, high-performance circuit simulator. Development of **Xyce**[™] has been limited to creation of the simulation engine itself, and **Xyce** does not include any schematic capture tool or graphical display software.

This note is intended as a basic tutorial for getting started the gEDA tools (<http://www.geda-project.org>) to create input for **Xyce**. We will demonstrate how to use the open source software gEDA, its schematic capture tool `gschem` and its associated netlist generation tool `gnetlist` to create circuit designs for simulation in **Xyce**, and how to use these tools to access **Xyce** features that are extensions of SPICE generally unsupported by the tools.

With gEDA's schematic capture and netlist generation tools, **Xyce** can become part of a complete design and simulation workflow that consists entirely of free, open-source tools.

Target Audience

This application note is primarily intended for existing users of **Xyce** who wish to use it in conjunction with a schematic capture tool, and who don't already have a preferred schematic capture tool.

Users with existing workflows that include another schematic capture tool might prefer to continue using their familiar tools, which may also be able to create netlists suitable for use with **Xyce**. This note does not address the peculiarities of other tools, but some of the techniques described here may be applicable to other workflows.

Current users of gEDA who are new to **Xyce** will find most of the material in the early part of this note of little use, as it is focused on introducing the basic mechanics of schematic capture with `gschem` and netlisting with `gnetlist`. The section on **Xyce**-specific netlisting issues will likely be of the most use to such users.

Prerequisites

This application note assumes that you have already downloaded and compiled **Xyce**[™] according to its documentation, that you have installed it in a manner that allows you to run it directly by typing “Xyce”¹ in the command line, and that you are able to run a basic netlist using that installed copy of **Xyce**.

We also assume that you have installed the `gnuplot` plotting software, as all of our plotting examples below will use that software. This software is available for all platforms that are supported by gEDA.

Obtaining gEDA and friends

Building and installing “gEDA” and its component software (sometimes referred to as “gEDA and friends” or “gaf”) from source is beyond the scope of this note. The gEDA project has a web site (<http://www.geda-project.org>) that provides access to gEDA documentation and downloads, and this should be consulted if you must build gEDA from source. But many operating systems provide a means of installing gEDA tools through their package management systems. See <http://wiki.geda-project.org/geda:download> for basic download instructions should you require more information than the next sections provide.

Simplified installation of gEDA and friends is known to be available on Debian, Ubuntu, and Fedora Linux; Mac OS X; and FreeBSD. A slightly more involved—yet still simple—install process allows Red Hat Enterprise Linux and CentOS installs. A brief summary of these systems’ installation process follows. Consult the gEDA download site or your system’s documentation for details.

Simplified installation on Ubuntu, Debian, Mac OS X and FreeBSD

On Ubuntu and Debian Linux systems, gEDA may be installed through the system package manager with:

```
sudo apt-get install geda
```

On Fedora Linux systems, gEDA may be installed with:

¹Or perhaps “runxyce” if you are a Sandia user who has installed one of our precompiled binary installations.

```
sudo yum install geda-gaf
```

On Macintosh OS X, the simplest install of gEDA requires a package manager such as Fink (<http://fink.thetis.ig42.org/>) or MacPorts (<http://www.macports.org/>). Installation of these packages is beyond the scope of this document, see their respective web sites for details. Once the Fink or MacPorts system is installed, gEDA may generally be installed with a single command. In Fink:

```
fink install geda-gaf
```

In MacPorts:

```
sudo port install geda-gaf
```

On FreeBSD, gEDA and friends are installed through the ports system. As root:

```
cd /usr/ports/cad/geda  
make install clean
```

gEDA on CentOS and Red Hat Enterprise Linux

While gEDA and friends are available for CentOS and Red Hat Enterprise Linux systems, the process involves a few extra steps, as the packages are included in non-default repositories.

In most cases there are only three steps. The first step is to download an “epel-release” rpm file that adds repositories, the second is to install that file using “rpm” and the third is to “yum install geda-gaf”. But since the specific file you need to download depends on which version of Red Hat Enterprise Linux or CentOS you are running and which processor you have (32-bit x86 or 64-bit x86_64), we will not enumerate the precise commands here.

Please see <http://pkgs.org/download/geda-gaf> for distribution- and platform-specific installation instructions for these packages. Under each specific platform link, there is an “Install Howto” section that gives the exact steps for that platform. These Howto instructions have been confirmed to work on Sandia Common Operating Environment Red Hat EL systems, if your system is properly set up.²

²If you receive an error from yum about not being able to download a “repomd.xml” for “epel”, check your proxy settings. If you require a proxy (as Sandia users do), both `http_proxy` and `https_proxy` environment variables need to be defined. Sandia RHEL users also need to assure that `coe-ssl-interception` is installed. Sandia RHEL users should consult their CSU support if further assistance is needed installing this software.

gEDA on Windows

Like **Xyce**, gEDA and friends are designed on and targeted at Unix-like operating systems, and as such are not as easily installed on Windows. If you require gEDA on Windows, see Appendix A, which documents the entire process.

Useful start-up options

By default, `gschem` will not automatically number components as you add them to your circuit. I have found that it is very helpful to enable autonumbering, but to do so you must create a start-up file for `gschem` to read, and add a few options to it. To make this option apply to all of your `gschem` runs automatically, put the following text into the file `.gEDA/gschemrc` in your home directory.

```
(load-from-path "auto-uref.scm") ; load the autonumbering script
(add-hook! add-component-hook auto-uref) ; autonumber when adding a component
(add-hook! copy-component-hook auto-uref) ; autonumber when copying a component
```

If you chose not to create this start-up file, you will have to number each component you add to a schematic manually, or use the “Autonumber Text” menu entry under the “Attributes” menu to number them for you after you place them. The remainder of this document will assume that you have created this start-up file.

Starting `gschem`

On most systems, running `gschem` (gEDA’s schematic capture tool) is as simple as typing “`gschem`” in a terminal window.

2. A first circuit with gEDA and Xyce

In this chapter we'll simulate the diode clipper circuit from the **Xyce**[™] Users' Guide[1] by drawing the circuit in `gschem`, generating a netlist using `gnetlist`, running a simulation in **Xyce**, and finally plotting the results in `gnuplot`.

For reference, we reproduce the diode clipper image from the users guide below in Figure 2.1.

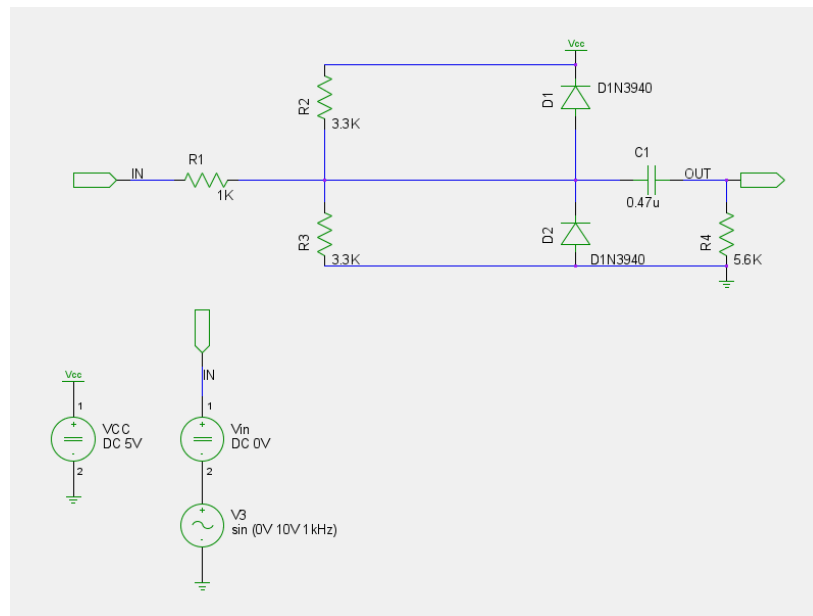


Figure 2.1. Diode Clipper circuit schematic from **Xyce** Users' Guide.

Overview

The remainder of this chapter is intended to be used as a step-by-step guide to creation of a simple test circuit. You should have `gschem` installed on your system and be able to

follow along with the text. At the conclusion of the chapter, you will have created your first schematic, generated a netlist from it, and simulated the circuit in **Xyce**.

In the following sections, we will construct a simple circuit and simulate it by taking the following general steps:

- Place components and assign attributes to them. These attributes can be component values or model names, and may also be “reference designators” (refdes) that will be used by the netlister to name the component in the netlist.
- Connect components according to the circuit design.
- Add simulation directives such as analysis statements or output control.
- Generate a netlist using `gnetlist`.
- Simulate the circuit in **Xyce**.
- View the waveforms in a plotting program.

Drawing the circuit

We will start by launching `gschem`, which will open a window with a blank design and a title block. The title block can be used to annotate your design, but for the time being we'll just ignore it and use it as a frame to enclose our circuit.

Adding voltage sources and editing attributes

Our first elements added to the design will be the various voltage sources that appear in the diode clipper of Figure 2.1. To create these:

- Click the “Add” menu and the “Component” menu entry, or click the symbol of an AND gate in the `gschem` tool bar. This will open a “Select Component...” dialog box such as the one in Figure 2.2.
- Click the “Libraries” tab and scroll down to “SPICE simulation elements.”
- Select the “vdc-1.sym” symbol (DC voltage source), and add two instances of this device side-by-side in the bottom left of the title block.
- Select the “vsin-1.sym” symbol in the “Select Component...” dialog, and insert one instance of this symbol. In this case, we will place the “vsin” symbol directly below the second DC voltage source, so that the positive terminal of the new “vsin” symbol is connected to the negative terminal of the DC source above it.

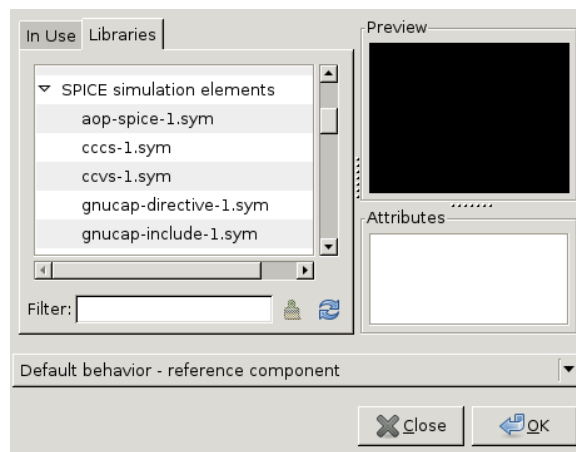


Figure 2.2. Select Component... dialog box.

- Close the “Select Component...” dialog box.

When this is complete, `gschem` will have labeled our three new voltage sources V1, V2, and V3, and your schematic will be arranged as in Figure 2.5. We will rename these to match the names that appear in the netlist in the Users Guide. Rename V1 by double-clicking the text “V1” in the schematic. A dialog box called “Single Attribute Editor” should appear (Figure 2.3) with “refdes” in the “Name” box and “V1” in the “Value” box. Change V1 to VCC, then click OK.

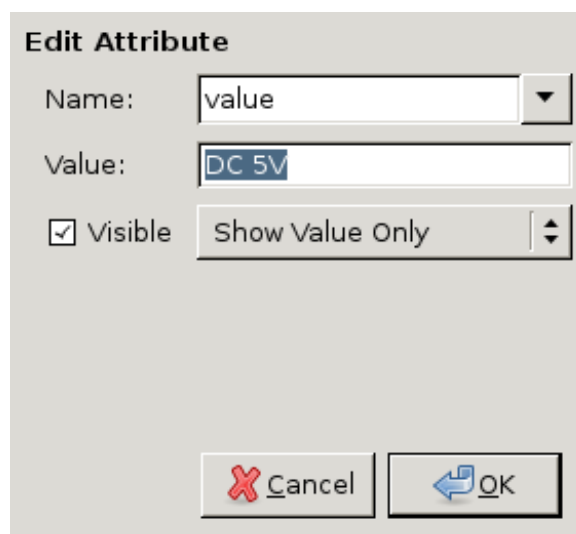


Figure 2.3. Single Attribute Editor dialog box.

Next, we will change the values of the VCC voltage source. Double-click the “DC 1V” near the VCC source, and when the “Single Attribute Editor” appears, change “DC 1V” to “DC 5V”.

We will also change the name and DC value of source V2. But this time, just to explore *glschem*’s capabilities, rather than double-clicking the “DC 1V” to open the Single Attribute Editor, we’ll do it a different way.

Click the V2 voltage source symbol itself (it will change color) and then right-click to bring up the context menu. In this menu, choose “Edit...”. This will bring up an “Edit Attributes” dialog (Figure 2.4) that allows you to edit all of the device’s attributes at once. Next to “refdes” you will see “V2”. Click this name and change it to “Vin”. Then, next to “value” click the “DC 1V” and change it to “DC 0V”. Click the “Close” button to exit the attribute editor.

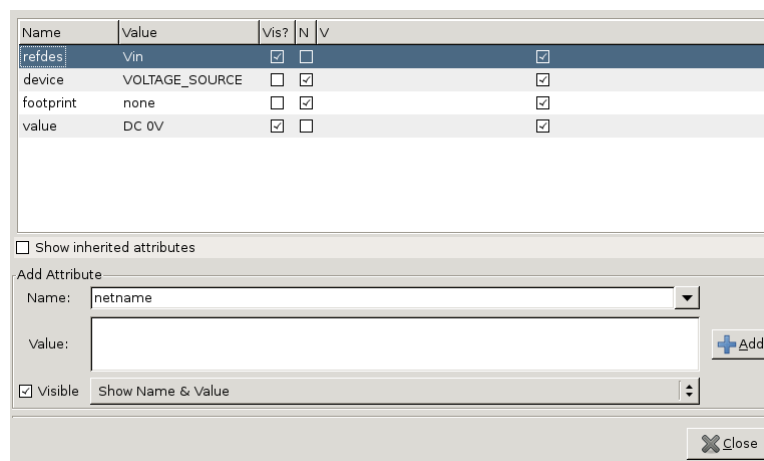


Figure 2.4. Edit Attributes dialog box.

Finally, edit the “V3” sinusoidal voltage source and change “sin 0 1 1 meg” to “sin(0V 10V 1kHz)”. We’ll do this by double-clicking the V3 symbol, which also brings up the “Edit Attributes” dialog. Change the value and close the editor.

Finally, add a connection to ground for the two voltage sources segments we’ve drawn so far. To do this, click the “add component” button or menu entry, open the “Power rails” library, and choose the “gnd-1.sym” symbol. Place an instance of this symbol at the negative terminal of both the VCC and V3 voltage sources. Close the “Select Component...” dialog.

Note that if you have correctly connected devices, the red blocks at their terminals should vanish. If you still see red blocks where devices should be connected, you have placed the symbols improperly. Move the symbol by left-clicking it and moving it until the red blocks of both devices coincide, release the mouse button, and the blocks should vanish.

When you're done with these steps, your schematic will appear as in Figure 2.5.

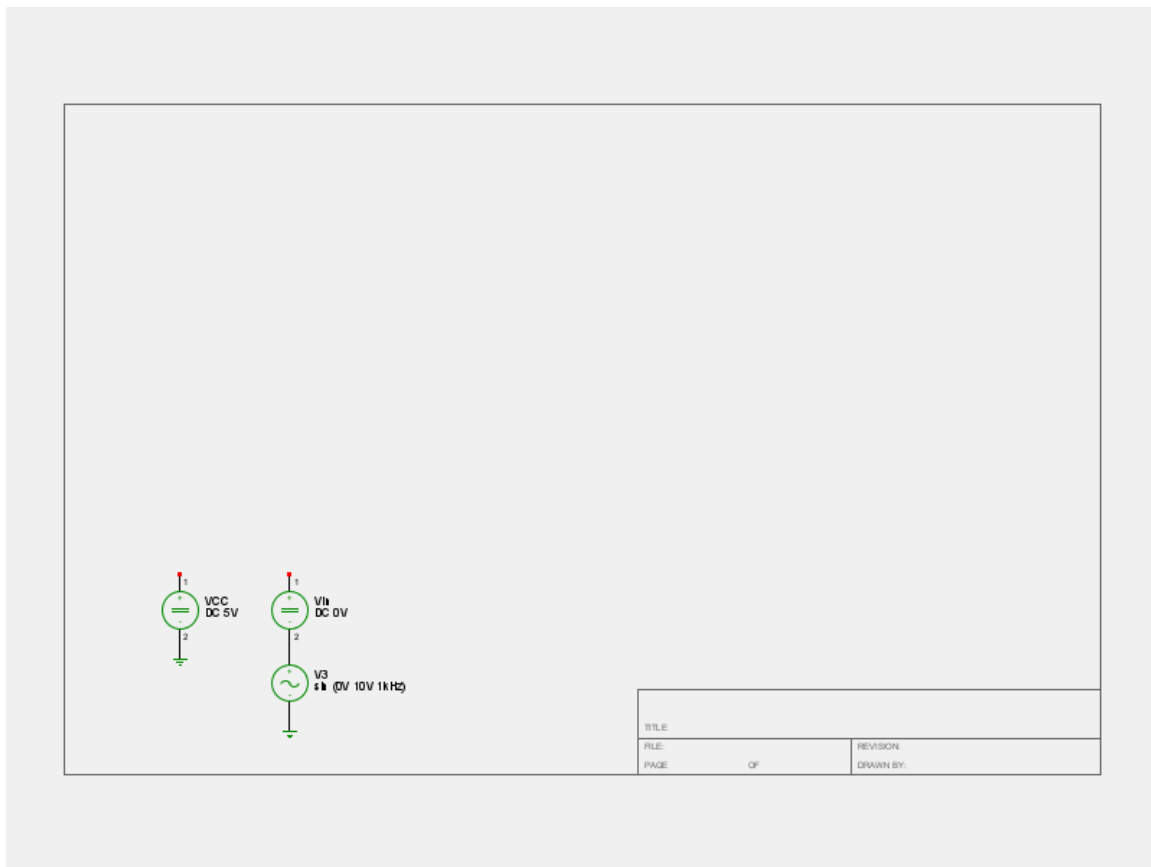


Figure 2.5. Sources for the Diode Clipper circuit.

Adding devices

Adding the remaining devices to the circuit is an exercise in selecting and placing components, rotating their symbols, and editing their attributes.

Rotating components can be done after placing them, or while placing them. To rotate a symbol after it is placed, select the symbol with the left mouse button and type “er”. The symbol will be rotated 90 degrees with each “er” command you give. Alternatively, you can select the “Rotate 90 mode” from the “Edit” menu, and then click on a point around which you want the symbol rotated. The symbol will be rotated 90 degrees around that point with each mouse click.

To rotate a symbol before placing it, choose the symbol from the Select Components dialog and then type “er” before placing it. The symbol will be placed in its rotated orientation when you click in the schematic window.

Continue creation of the Diode Clipper circuit by adding the 4 resistors that appear in Figure 2.1. Use the “Basic Devices” library and symbol “resistor-1.sym”, rotating symbols with “er” as required. Do the same for the capacitors (Basic Devices, capacitor-1.sym) and diodes (Diodes (generic) library, diode-1.sym symbol). Place these devices in the rough locations indicated in Figure 2.1, which will leave them all disconnected. The result should appear as in Figure 2.6.

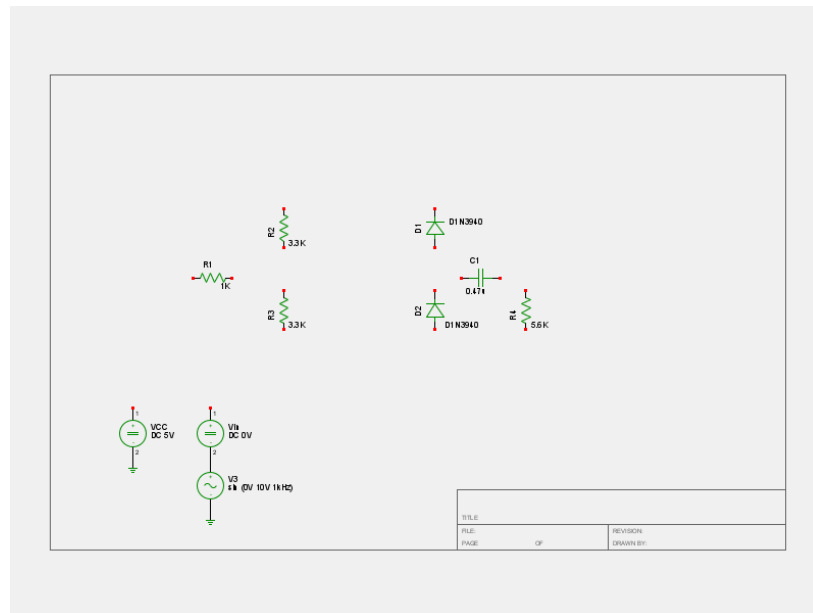


Figure 2.6. Placed components for Diode Clipper circuit.

Edit the attributes of all devices to give them the values shown in Figure 2.1. Note that all of the devices you have just added have no “value” attribute yet, so you will have to add one. Double-click each device in turn to bring up the Edit Attributes dialog. In the lower part of the dialog is an “Add Attribute” pane. For the resistors and capacitors choose “value” as the “Name” of the attribute to add, and the value of the component (e.g. “1K” for resistor R1) in the Value of that attribute. Then change the visibility to “Show Value Only” to keep the schematic legible, and click the “+Add” button.

Unfortunately, *gschem* has a tendency to put the text for the value in an ugly location, so once you close the editor you may have to move these labels. Click the background of your schematic (to deselect the component), then click the label you want to move until only it is highlighted. Drag the component value to a place that looks good in your schematic.

For the diodes, do not add “value” attributes. Diodes require a model name and an associated model “card.” For these devices, add an attribute “model-name” with value “D1N3940”

When this is complete, your schematic should look as it does in Figure 2.6.

Connecting the devices

Next, we will add “nets” to tie all the components together properly. To begin, select the “nets” tool in the `gschem` toolbar (it is to the right of the component tool), or choose the “Add” menu and the “Net” option.

Move your mouse to the lower terminal of resistor R2. When you are close, a small circle will appear around the terminal indicating that your new net will be connected there. Click your left mouse button, move to the upper terminal of resistor R3, then click the left mouse button and hit your “Esc” key. R2 and R3 will now be connected.

Do the same thing to connect diodes D1 and D2. Remember to hit “Esc” after connecting these two devices. Your partial schematic will appear as in Figure 2.7.

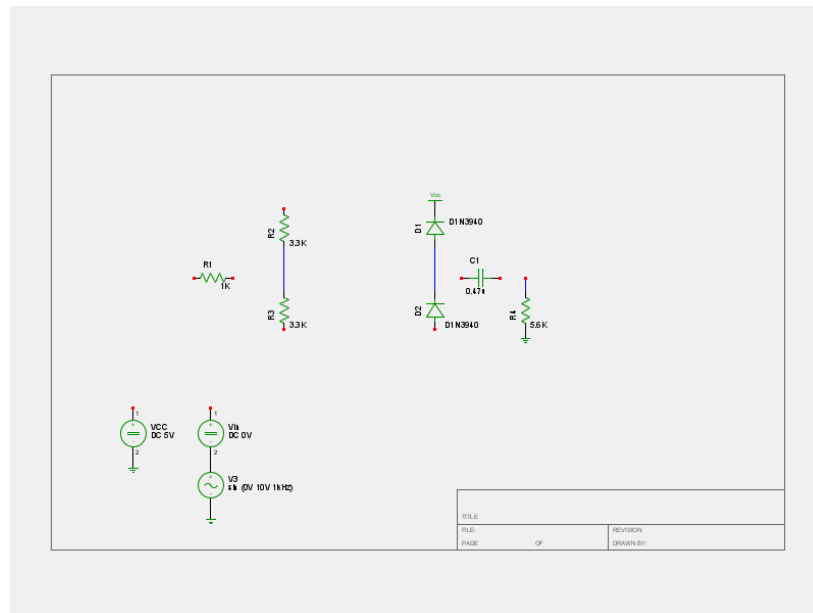


Figure 2.7. First two nets added to Diode Clipper circuit.

Our next net will connect the lower terminals of R3, D2 and R4 together. Using the net tool, click the lower terminal of R3 and drag to the lower terminal of D2. Release the mouse button at D2 but do *NOT* type “Esc.” Move your mouse to the lower terminal of R4, and click again. The three devices will now be connected. Hit “Esc” to terminate creation of this net.

Connecting R1 and C1 presents a new problem. R1 should be connected to R2, R3, D1, D2 and C1. If you simply drag a net from R1 to C1 `gschem` will *NOT* automatically

connect this net to the nets it crosses. To assure correct interconnection, click the net tool, click the right terminal of R1, then click the middle of the net between R2 and R3. Note that `gsschem` shows a white dot here, indicating that there is a connection, not just a coincidental crossing of lines. Click again at the middle of the net between D1 and D2, and then finally to the leftmost terminal of C1. Hit “Esc” to end the net. Now R1, R2, R3, D1, D2 and C1 are properly connected.

Connect the remaining unconnected terminal of R2 to the unconnected terminal of D1. Then connect the unconnected terminals of C1 and R4 together. Note that `gsschem` automatically draws a right angle in the net between these final two devices.

Finally, add a ground connection to the lower terminal of R4 (add component, “Power rails” library, symbol “gnd-1.sym”).

Your circuit, which is still not complete, will look like Figure 2.8.

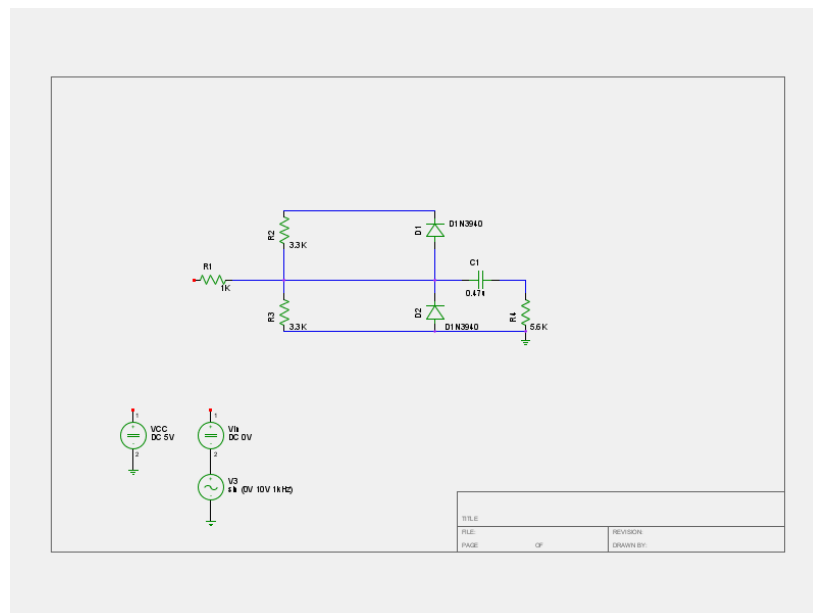


Figure 2.8. Remaining nets added to Diode Clipper circuit.

Completing the schematic

Our circuit is still not complete. The voltage sources are not yet connected to the main circuit. We will do this by creating named “nets” that connect the sources in the bottom corner to the circuit.

To begin, we’ll attach the VCC device to the upper terminal of D1. We could do this by drawing a net attaching them, but instead we’ll define a VCC “power rail.” To do this, select

the “vcc-1.sym” symbol from the “Power rails” library, and connect one to the positive terminal of the VCC voltage source, and another to the upper terminal of D1. This special Vcc symbol tells the gnetlist program to connect any terminal to which it’s attached to a common net named “vcc.”

We now want to connect the positive terminal of Vin to the remaining contact of R1. To do this we’ll place two instances of the “Input/Output (generic)” “input-1.sym” near (but not connected to) the Vin positive terminal and the R1 unconnected terminal. Use the net tool to connect each of these symbols to the one contact it is near. Using the left mouse button to select each of the two new nets, and the right mouse button to bring up a context menu, use the “Edit...” menu entry and resulting “Edit Attributes” dialog to add the “netname” attribute to the net, and set the value of that attribute to “IN”.

Finally, add an instance of the “ouput-1.sym” symbol from the “Input/Output(generic)” library and connect it to the net that connects C1 and R4. Using the Edit Attributes dialog, add a “netname” of “OUT” to the net that connects C1 and R4.

Our schematic is now complete but for a diode model. It should appear as in Figure 2.9.

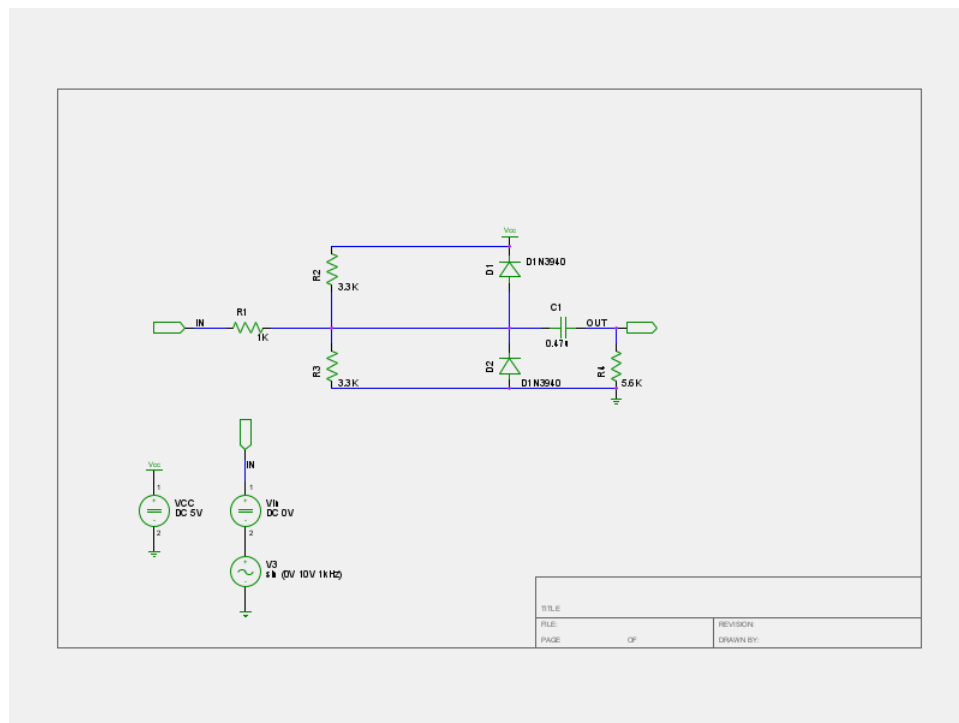


Figure 2.9. Complete Diode Clipper schematic.

Adding a Model Card

Our diode clipper circuit schematic is complete, but is still missing some information needed to generate a simulation netlist. The two diodes reference an undefined model, D1N3940. Without this model defined, **Xyce** will have no way to determine what parameters to use in the diode devices. There are multiple methods to add this model data, and we will use the one that creates the fewest unique lines of netlist code.

Select the “Add component” tool, and the “spice-model-1.sym” symbol from the “Spice simulation elements” library. Place this component in the upper right corner of your schematic, and double-click it to bring up the attribute editor. Make the following changes to the attributes of this symbol:

- Delete the “file” attribute by right-clicking its name and choosing “delete.”
- Add a “model-name” attribute with value “D1N3940”
- Add a “type” attribute with value “D”. Make the type attribute invisible.
- Add a “model” attribute (also invisible) with the value

```
IS=4e-10 RS=.105 N=1.48 TT=8e-7 CJO=1.95E-11 VJ=.4 M=.38  
EG=1.36 XTI=-8 KF=0 AF=1 FC=.9 BV=600 IBV=1e-4
```

You should enter all of these on a single line in the attribute editor, or `gnetlist` will generate a netlist with syntax errors. It is possible to force `gnetlist` to generate multiline model cards, but that will be left as an exercise for the reader.

Your attribute editor should appear as in Figure 2.10.

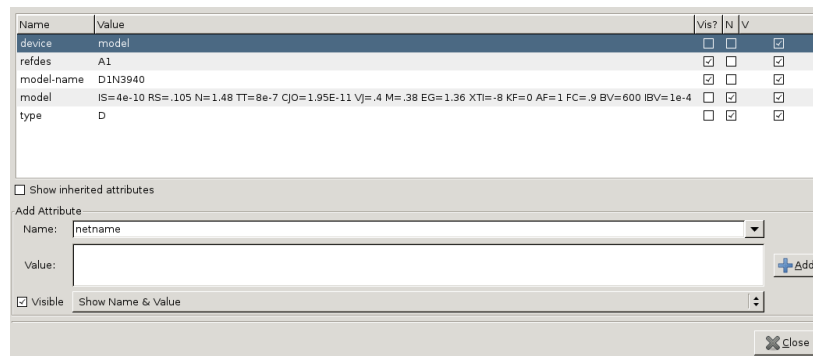


Figure 2.10. Model component attributes for diode clipper.

At this stage, your schematic contains all the elements necessary to generate a valid simulation netlist except for an analysis type and output control. Save the schematic and give it a name (such as DiodeClipper.sch).

Generating the netlist

We are now ready to create our first netlist with gEDA and friends. Using the DiodeClipper.sch schematic file you just created, run the gnetlist program with the following command-line options:

```
gnetlist -o diodeclipper.cir -g spice-sdb DiodeClipper.sch
```

This command tells gnetlist to use its “spice-sdb” back-end to generate a SPICE netlist called “diodeclipper.cir”. The “spice-sdb” back-end is the best one to use to generate **Xyce** netlists, even though it is not an exact match. At this time, there is no **Xyce**-specific gnetlist back-end.

The resulting netlist produced by gnetlist will look like this:

```
* gnetlist -o diodeclipper.cir -g spice-sdb DiodeClipper.sch
*****
* Spice file generated by gnetlist                                     *
* spice-sdb version 4.28.2007 by SDB --                               *
* provides advanced spice netlisting capability.                     *
* Documentation at http://www.brorson.com/gEDA/SPICE/ *
*****
===== Begin SPICE netlist of main design =====
.MODEL D1N3940 D (IS=4e-10 RS=.105 N=1.48 TT=8e-7 CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
D2 0 2 D1N3940
D1 2 Vcc D1N3940
C1 2 OUT 0.47u
R4 0 OUT 5.6K
R3 0 2 3.3K
R2 2 Vcc 3.3K
R1 IN 2 1K
V3 1 0 sin (0V 10V 1kHz)
Vin IN 1 DC 0V
VCC Vcc 0 DC 5V
.end
```

Here you can see how gnetlist has interpreted our schematic elements and named circuit nodes. If your generated netlist does not look like this one, or if it contains any “unconnected” nodes, re-check your schematic.

We are still not ready to run this circuit, because it is lacking analysis and output control statements.

Adding analysis and output statements

The hard work of creating a netlist from a schematic has been done. We could use the netlist produced above merely by editing it in a standard text editor (e.g. vi or emacs) to add analysis and output statements, but we can also do it within `gschem`.

To add an output statement to the schematic, re-open it in `gschem` and add two instances of the “spice-directive-1.sym” symbols from the “SPICE simulation elements” library. From each, delete the “file” attribute. In one, change the “value” attribute to be “.DC VIN -10 15 1” and in the other “.PRINT DC V(IN) V(2) V(OUT)”. This corresponds to the DC sweep example in the **Xyce** Users Guide. Save your schematic, and regenerate the netlist. The result should be as below:

```
* gnetlist -o diodeclipper.cir -g spice-sdb DiodeClipper.sch
*****
* Spice file generated by gnetlist                                     *
* spice-sdb version 4.28.2007 by SDB --                               *
* provides advanced spice netlisting capability.                     *
* Documentation at http://www.brorson.com/gEDA/SPICE/ *
*****
===== Begin SPICE netlist of main design =====
.PRINT DC V(IN) V(2) V(OUT)
.DC VIN -10 15 1
.MODEL D1N3940 D (IS=4e-10 RS=.105 N=1.48 TT=8e-7 CJO=1.95E-11 VJ=.4 M=.38 EG=1.36
D2 0 2 D1N3940
D1 2 Vcc D1N3940
C1 2 OUT 0.47u
R4 0 OUT 5.6K
R3 0 2 3.3K
R2 2 Vcc 3.3K
R1 IN 2 1K
V3 1 0 sin (0V 10V 1kHz)
Vin IN 1 DC 0V
VCC Vcc 0 DC 5V
.end
```

In this particular instance we have taken advantage of knowing how `gnetlist` has numbered the net on the anode of the diode (in this case, it is node 2), which we only knew because we had previously generated the netlist. In general this is not safe, as later development of the circuit could change the assignment of node numbers. It is generally safer to give nets that you want to print non-numeric netnames. Giving nets numeric netnames is dangerous, as `gnetlist` might assign some other unnamed net that number (a known issue in `gnetlist/spice-sdb`). We have done just that for IN and OUT.

Note also that our “spice directive” components have been written to the netlist verbatim. This component can be used to force any particular text into the netlist generated by `gnetlist` with `spice-sdb`, and can be a powerful tool for coping with oddities of **Xyce** that aren’t dealt with directly by the `spice-sdb` back-end.

Simulation with Xyce

Once we have a suitable netlist generated, complete with analysis and print statements, we can run **Xyce** over it to produce simulation results.¹

```
Xyce diodeclipper.cir
```

Xyce will complete quickly and produce a file called “diodeclipper.cir.prn” with the simulation results in it. Some representative lines of this output are shown below.

Index	V(IN)	V(2)	V(OUT)
0	-1.00000000e+01	-6.41596339e-01	9.72092671e-28
1	-9.00000000e+00	-6.36033187e-01	9.65377367e-27
2	-8.00000000e+00	-6.29569220e-01	1.29703606e-26
3	-7.00000000e+00	-6.21845977e-01	1.66372880e-28
4	-6.00000000e+00	-6.12257417e-01	3.82700013e-28
[...]			
25	1.50000000e+01	5.64159843e+00	-1.05830968e-26
End of Xyce(TM) Simulation			

This is **Xyce**’s standard columnar output. Other output formats are supported, but this will suffice for the purposes of this application note.

Plotting the results

Just as **Xyce** does not provide schematic capture capability, neither does it provide capability for graphical display of data. These capabilities require use of some other plotting software. Choices for this plotting software include the open-source tool `gnuplot` (<http://www.gnuplot.info/>), spreadsheet programs such as `openoffice-scalc` (<http://www.openoffice.org/>) or `Microsoft Excel`, or any other tool of your choice. Alternate Xyce output formats are available for use in existing open-source SPICE data viewers such as `gwave` (<http://gwave.sourceforge.net/>), `gaw` (<http://www.rvq.fr/linux/gaw.php>), or `nutmeg`, and commercial products such as `Cadence PSpice AD`² or `Tecplot`.

¹Again, Sandia users running a precompiled binary would use the `runxyce` wrapper script instead.

²Of course, PSpice AD has its own schematic capture tool and simulation engine, and if you had it you probably wouldn’t need this application note!

In this section, we will use gnuplot to display the DC swept diode clipper to produce a graph similar to the one that appears in the **Xyce** Users Guide.

By following the steps in the previous section, you have produced a file `diodeclipper.cir.prn` containing the simulation results for the DC sweep of the diode clipper circuit. The first column of the data is an index of the result and can be ignored. The second column is $V(IN)$, and we will use it as the independent variable for plotting.

Open gnuplot and issue the following commands:

```
plot 'diodeclipper.cir.prn' using 2:2 with lines title "V(IN)"
replot 'diodeclipper.cir.prn' using 2:3 with lines title "V(2)"
replot 'diodeclipper.cir.prn' using 2:4 with lines title "V(OUT)"
```

The first of these commands reads the output file and plots the the second column against itself (plotting $V(IN)$ against $V(IN)$). The next line plots the third column against the second, and the next plots the fourth column against the second, overlaying them on the first curve instead of replacing the first curve (“replot” instead of “plot”). Each command specifies “with lines” to override gnuplot’s default behavior of plotting points instead of curves. Each command also specifies a label for the line that will appear in the legend. This will reproduce the curves in the DC sweep graph that appears in the Xyce Users’ Guide. It should appear as in Figure 2.11.

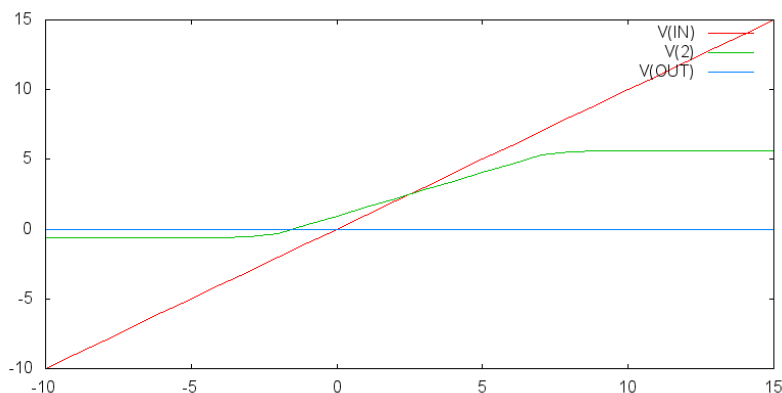


Figure 2.11. DC Sweep plot for diode clipper circuit.

Changing the analysis

In the earlier sections, we set up our diode clipper schematic to contain two “spice-directive” symbols so that the resulting netlist would run in **Xyce** with no further editing.

To change the diode clipper netlist to run a transient rather than a DC simulation, one need only modify the netlist to replace the “.DC” line with a “.tran” line, and to change the print line from “.print DC” to “.print tran”.

Though this operation could be performed by opening `gschem` and changing the “spice-directive” elements, once a netlist has been generated by `gnetlist` it is a simple text file that can be edited in any plain-text editor. If you need to run the same circuit through multiple analyses, it may be easier to copy the netlist and edit it directly than to go back, re-edit the schematic, and re-generate the netlist.

Either way, you should now be able to modify your diode clipper netlist to perform the transient calculation in the **Xyce** Users’ Guide, chapter 3, section 3 (“Transient Analysis”).

Summary

In this chapter we’ve laid out the basic steps for using gEDA to create a simulation input for **Xyce**. These steps are:

- Lay out the components and connect them with nets
- Modify the “value” attribute of simple components (resistors, capacitors, sources, etc.)
- Add “model-name” attributes for components that require models (diodes, transistors, etc.)
- Add “spice-model” symbols with appropriate attributes to define the models needed
- Add “spice-directive” symbols to emit appropriate simulator control statements
- Give any nets that you want to print non-numeric “netname” attributes rather than letting `gnetlist` assign them numeric names itself.
- Generate a netlist from the schematic using `gnetlist` and the “spice-sdb” back-end.

3. Creating hierarchical designs

While the steps of the previous chapter will provide all the tools you will need for creating single schematics for simple designs, it is sometimes necessary to create a hierarchical design, in which sub-circuits are created in separate schematics, and the sub-circuits combined in a higher-level design. Both gEDA and **Xyce** provide tools for making such designs.

`gschem` supports a notion of hierarchical schematics and has simple menu items for moving up and down a hierarchy, but as of this writing it is difficult to use this feature along with the “spice-sdb” netlist generation back-end.

Since the hierarchical schematic tools of `gschem` are difficult to use with the netlister, we will focus instead on spice-sdb’s capability for generating hierarchical *netlists* from separate schematics.

As a starting point for our discussion of hierarchical design, we’ll produce a simple common emitter amplifier circuit in a flattened design, and then break it into a two-level design with the amplifier itself in a separate subcircuit that can be used at a higher level.

The common emitter exercise in this chapter is patterned on a similar exercise based on the commercial product “LT Spice” in the “Hands-On Radio” column by Ward Silver. This column is published in the magazine *QST*, the monthly magazine of the American Radio Relay League. The exercise is described in Experiments #83 through #85[2, 3, 4], which are available to members of that organization at their web site <http://www.arrl.org/Hands-On-Radio> or in book form from the ARRL store <http://www.arrl.org/shop/ARRL-s-Hands-On-Radio-Experiments-Volume-2>[5].

This section is only a basic introduction to using the hierarchical netlisting capabilities of spice-sdb. For much more detail, there is no substitute for the documentation written by Stuart Brorson, the author of the spice-sdb `gnetlist` back-end. This documentation is available on the web at <http://www.brorson.com/gEDA/SPICE/intro.html> and is also distributed along with gEDA. Specific information about hierarchical design is in section 4.12 of that documentation.

The Common Emitter Amplifier — flat design

Figure 3.1 shows the schematic for a simple common emitter amplifier, driving circuitry, and load.

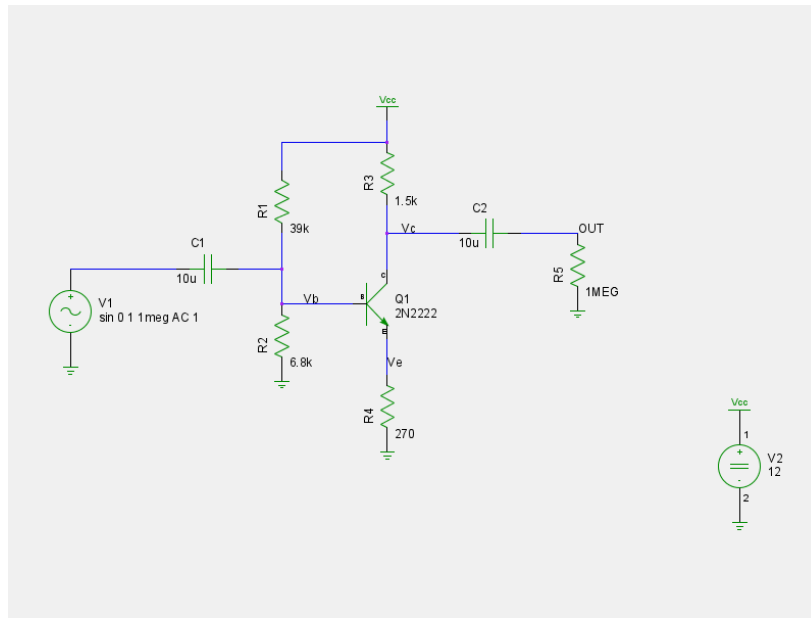


Figure 3.1. Schematic for flattened common emitter amplifier circuit

There is nothing particularly complicated about this schematic, and the basic steps of the last chapter are sufficient to create it. We will not give step-by-step instructions for creating it. There are, however, points worth noting.

Remember to add “value” attributes to all the resistors and capacitors. Set the values of these “value” attributes to the resistances and capacitances in the figure.

Use the “npn-1.sym” symbol in the “basic devices” library for Q1. Add a “model-name” attribute to Q1, and make it visible as “value only”. For this circuit, since we have only one 2N2222 transistor, we’ll add the model directly to the symbol rather than adding a separate “spice-model” symbol. To do this, add a “model” attribute to the Q1 transistor and give it the value:

```
Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307 Ise=14.34f  
Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1 Cjc=7.306p  
Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75 Tr=46.91n
```

```
Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10
```

Again, enter all of this text on a single line in the attribute editor, to assure proper netlist generation. These parameters give **Xyce** all of the information it needs to simulate Q1 with the performance of a 2N2222 transistor.

In addition to the “model-name” and “model” attributes, you also need to add a “type” attribute with value “NPN” to device Q1. This will be used by spice-sdb to produce a correctly formatted “.MODEL” line appropriate to an NPN transistor.¹

Add “netname” parameters to the three nets connected to the Q1 transistor so we can print them more easily. Use “Vc” for the net connected to the collector, “Ve” for the net connected to the emitter, and “Vb” for the net connected to the base, as shown in figure 3.1. Remember that to add a netname to a net, you must highlight the net, right-click to bring up the context menu, and “Edit...” to bring up the Edit Attributes dialog.

Finally, add three “spice-directive” symbols, removing the “file” attribute of each one. To each of these directive symbols, add one of the following lines.

```
.op
.ac oct 10 .01 1e5
.print AC Vdb(Ve) Vp(Ve) Vdb(out) Vp(out)
```

Once all the components are placed and given values, and the SPICE directives added, your schematic should look like Figure 3.2. Save the schematic with the file name “CommonEmitterI.sch”.

The netlist is generated as before:

```
gnetlist -o common_emitterI.cir -g spice-sdb CommonEmitterI.sch
```

which will produce a netlist:

```
* gnetlist -o common_emitterI.cir -g spice-sdb CommonEmitterI.sch
*****
* Spice file generated by gnetlist                                     *
* spice-sdb version 4.28.2007 by SDB --                               *
* provides advanced spice netlisting capability.                     *
* Documentation at http://www.brorson.com/gEDA/SPICE/                 *
*****
```

¹This need for a “type” attribute is peculiar to these generic transistor symbols in the “Basic Devices” library. There are other options, and we’ll see some of those options later in this tutorial.

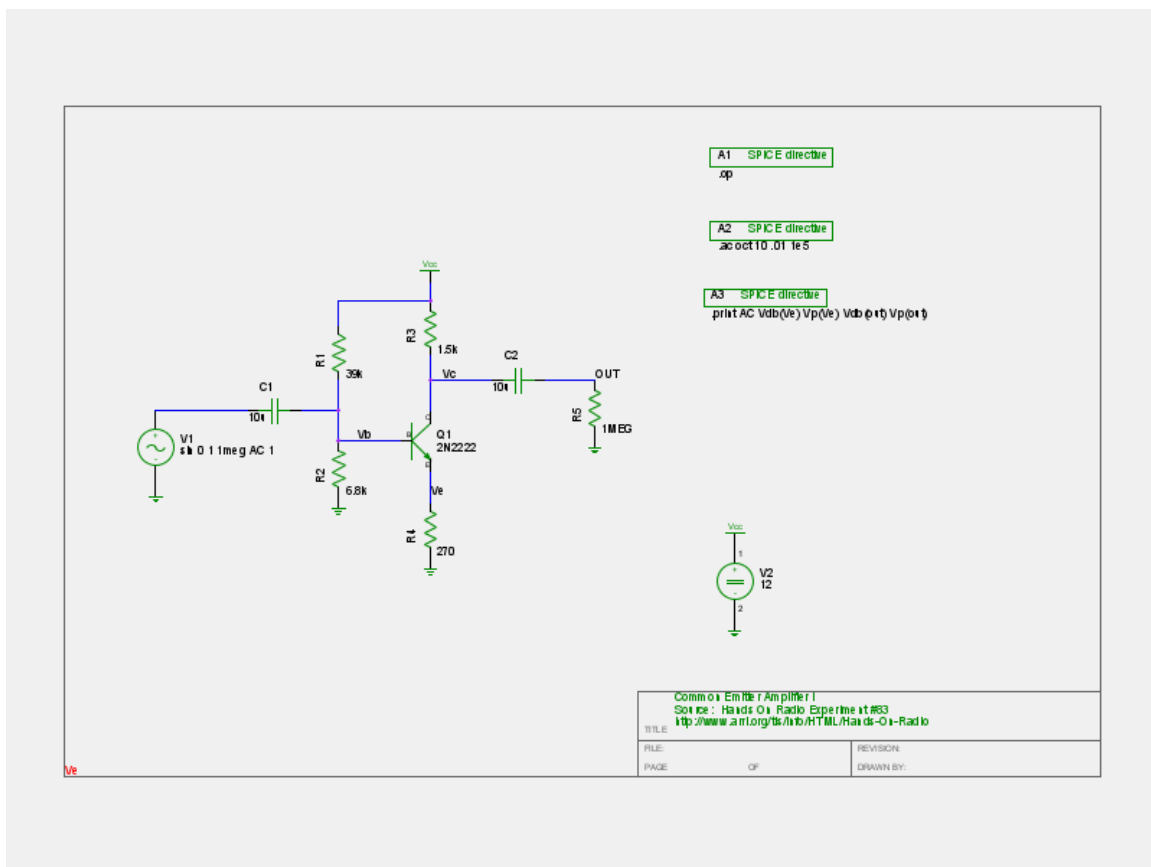


Figure 3.2. Complete schematic for flattened common emitter amplifier circuit

```

===== Begin SPICE netlist of main design =====
.print AC Vdb(Ve) Vp(Ve) Vdb(out) Vp(out)
.ac oct 10 .01 1e5
.op
R5 0 OUT 1MEG
V1 1 0 sin 0 1 1meg AC 1
V2 Vcc 0 12
C2 Vc OUT 10u
R4 0 Ve 270
R3 Vc Vcc 1.5k
Q1 Vc Vb Ve 2N2222
.MODEL 2N2222 NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307 Ise=14.34f I
R2 0 Vb 6.8k
R1 Vb Vcc 39k
C1 1 Vb 10u
.end

```

Again, if your netlist does not look like this one, check your schematic against Figure 3.2.

Run the simulation in Xyce, and it will produce tabular output in the file “common_emitter1.cir.FD.prn” that contains the response of the circuit (in dB and phase) for a range of input frequencies between .01 and 1e5Hz.

Common Emitter Amplifier — hierarchical netlist version

As noted before, `gschem` does provide a capability to create hierarchical schematics, and there is a menu called “Hierarchy” that has commands to move up and down the hierarchy. This feature is quite powerful and does make moving between levels of a hierarchical schematic convenient. Unfortunately, this feature is somewhat newer than the `spice-sdb` netlister back-end, and it does not play very well with `spice-sdb` as there are a number of issues in `spice-sdb` that interfere with its use. Further, when this feature is used, `gnetlist` builds a flattened netlist with all the hierarchy somewhat obscured.

`spice-sdb` does, however, have its own capability for dealing with hierarchical designs, and it can produce netlists that use the subcircuiting capability of SPICE (and by extension, **Xyce**). This method preserves the hierarchical structure of the schematic in the netlist format. In this section we will demonstrate how to use this version of hierarchical design.

Overview

The creation of a hierarchical netlist proceeds according to the following general scheme.

- Schematics for low-level components are created using input and output symbols to mark where they will be connected to the upper levels.
- A special “subcircuit” symbol is added to the schematic to instruct the netlister that it should generate SPICE/**Xyce** subcircuit syntax.
- The low-level schematics are netlisted with `gnetlist`.
- A symbol is created to represent the subcircuit at higher levels. Pins on the symbol are associated with the input and output symbols in the subcircuit’s schematic.
- Higher level schematics are drawn using the symbol for the subcircuit to represent the lower-level circuit. These symbols are marked with attributes to direct the netlister to the appropriate netlist file for the subcircuit.
- The high-level circuit is netlisted with `gnetlist`. It will incorporate the netlist files of the lower level, and reference them through subcircuit instantiation lines.

The Common Emitter Amplifier subcircuit

To begin our hierarchical design, we start at the lower level, the amplifier itself. We'll remove the input source, the Vcc power supply, and the output load components of our original amplifier circuit and replace them with symbols that gnetlist spice-sdb will use to tie the lower level of the schematic in to the higher level.

The complete schematic for the common emitter amplifier subcircuit appears in Figure 3.3. There are several points to note about this schematic.

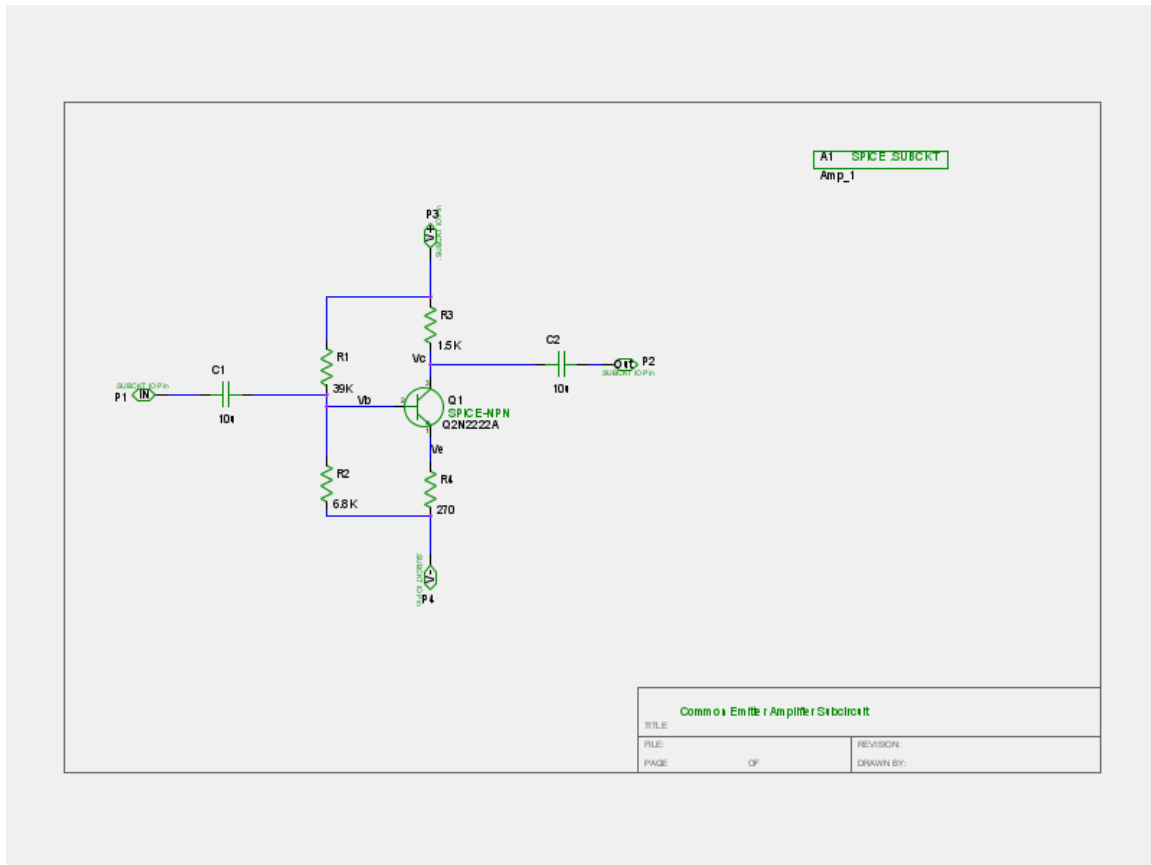


Figure 3.3. Schematic for the common emitter amplifier subcircuit

- The V1 and V2 voltage sources and the R5 resistor have been removed and replaced with “spice-subcircuit-IO-1.sym” symbols from the “SPICE Simulation Elements” library, as have been the “Vcc” voltage rail symbol and the ground connections.
- V1 has been replaced by a spice-subcircuit-IO symbol named “P1” (that is, its

“refdes” attribute is P1). It has also been given a “name” attribute “IN” that is used solely for display purposes in the schematic.

- The R5 resistor is replaced by a spice-subcircuit-IO symbol with refdes “P2” and name “Out”.
- The Vcc connection is now a pin labeled “P3” with name “V+”
- Both terminals that had previously been attached directly (and separately) to ground are now tied together and connected to a fourth spice-subcircuit-IO symbol with refdes “P4” and name “V-”.
- A single instance of the “SPICE .SUBCKT” symbol (A1, symbol “spice-subcircuit-LL-1.sym”) has been added to the schematic. This is the symbol `gnetlist` will use to create a valid SPICE (or **Xyce**) “.subckt” line to define the subcircuit.
- In this version of the circuit we’ve used a slightly different symbol for the bipolar transistor Q1. This time, we’ve used the symbol “spice-npn-1.sym” in the “SPICE Simulation Elements” library rather than the “nnp-1.sym” symbol in the “Basic Devices” library. `spice-sdb` recognizes this symbol as an NPN transistor without our needing to add a “type” attribute as we did in the previous section.

Q1 has the following attributes:

- refdes = Q1 (this will be created by `gschem` automatically)
- model-name=Q2N2222A
- ```
model=Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307
+ Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+ Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+ Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10
```

Note that we have broken the very long model specification into multiple lines this time, and to make this work correctly we need to add “+” continuation characters as the first character of each line after the first. This text will be written out to the netlist verbatim, and by having the “+” signs first on the lines **Xyce** will recognize these extra lines as being continuations of the previous line. Leaving these out will create a syntax error in the resulting netlist.

The “SPICE .SUBCKT” symbol (`spice-subcircuit-LL-1.sym`) has a “model-name” attribute of “Amp\_1”. This is the name by which the subcircuit will be identified to higher level schematics.

Create this schematic in `gschem` and save it as `Amp1.sch`. Then generate its netlist using:

```
gnetlist -o Amp1.cir -g spice-sdb Amp1.sch
```

The result will be the following netlist:

```

* Begin .SUBCKT model *
* spice-sdb ver 4.28.2007 *

.SUBCKT Amp_1 1 2 3 4
===== Begin SPICE netlist of main design =====
Q1 Vc Vb Ve Q2N2222A
.MODEL Q2N2222A NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307
+ Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+ Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+ Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)
R4 4 Ve 270
R3 Vc 3 1.5K
R2 4 Vb 6.8K
R1 Vb 3 39K
C2 Vc 2 10u
C1 1 Vb 10u
.ends Amp_1

```

There are several important things to note:

- The subcircuit has four input nodes on the .SUBCKT line. These correspond exactly to the “spice-subcircuit-IO” symbols in order of refdes number.
- The .model card for the Q1 transistor is inside the subcircuit.
- The name of the subcircuit on the .SUBCKT line is the model-name of the A1 “spice-subcircuit-LL-1.sym” symbol.
- The multiline “model” attribute of the Q1 transistor has properly been output as a multiline .MODEL card acceptable to **Xyce** and any version of SPICE.

We have now constructed the lower level subcircuit schematic and its associated netlist. It is now time to turn our attention to the higher level circuit.

## Creating a symbol for the subcircuit

We are now in a position to create a single symbol that will represent our common emitter amplifier subcircuit. Note that in this section we are providing only a brief tutorial

on how to create this symbol, and that for more detail on symbol creation in `gschem` you should consult the documentation at [http://wiki.geda-project.org/geda:gschem\\_symbol\\_creation](http://wiki.geda-project.org/geda:gschem_symbol_creation).

Since we are creating a symbol and not a schematic, we don't want our window to contain the default title block. Open `gschem` and drag your mouse so that it selects the entire title block and turns it orange. Then type "d" to delete it. Your `gschem` window will now be empty except for a faint grid.

When you open `gschem` you will be zoomed out much too far for good symbol creation. Zoom in several times using the "z" key. We're going to create a symbol that is 10 boxes wide and six boxes high, so zoom in until that number of boxes is a significant fraction of the window.

Check that snap-to-grid is on, and that text "Grid(100,100)" appears in the bottom information line of the `gschem` window. If it does not, type "os" (or choose "Options → Toggle Snap On/Off") until it does.

Make sure that grid snap size is set to 100 (this is the default).

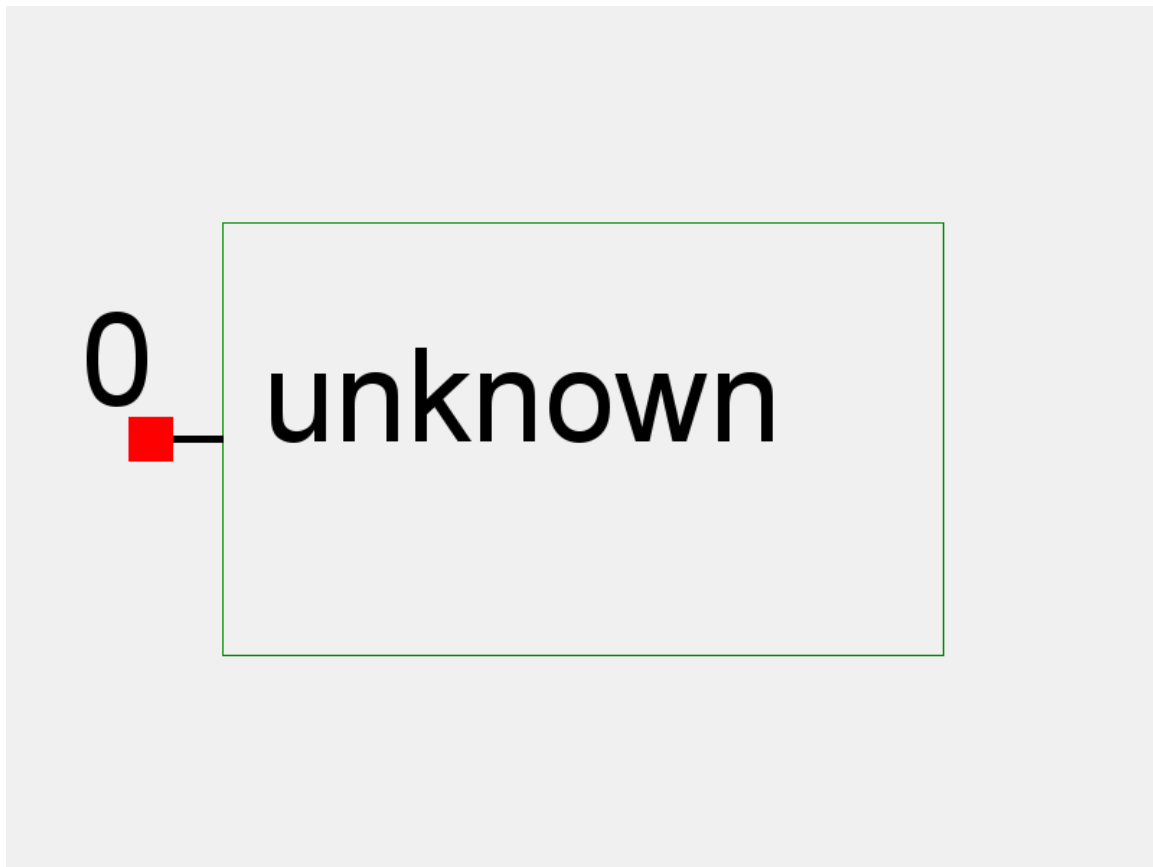
In the Add menu, chose "box". Click on one of the grid intersections, move your mouse so that a box 6 squares high and 10 squares wide appears, and click again to create it. This is the outline of the symbol we will use to represent our common emitter amplifier subcircuit.

Now it's time to add pins and labels to the symbol. Select "pin" from the "Add" menu. Move your mouse one grid square left of the left side of the green box, three grid squares from the top. Click the mouse, then mode to the edge of the box and click again. Your symbol should appear as in Figure 3.4.

We will now change the attributes of this pin so it will be useful. We must set the "pintype", "pinlabel", "pinnumber" and "pinseq" attributes, all of which already exist, but have useless values. Exit "pin mode" by hitting "Esc", and double click on the white bar of the pin. An attribute editor will appear. Use it to set the attributes:

- pintype in (uncheck the "Vis?" box)
- pinlabel In
- pinnumber 1 (uncheck the "Vis?" box)
- pinseq 1 (uncheck the "Vis?" box)

This will give your pin the name "In" and it will be recognized as an input. The pinnumber and pinseq attributes will be used by the netlister to connect this pin to appropriate nodes of the subcircuit.



**Figure 3.4.** First pin added to symbol

Return to your symbol drawing by adding pins for Out, “V+”, and “V-”.

“Out” should have

- pintype out (uncheck the “Vis?” box)
- pinlabel Out
- pinnumber 2 (uncheck the “Vis?” box)
- pinseq 2 (uncheck the “Vis?” box)

V+ should have

- pintype pwr (uncheck the “Vis?” box)
- pinlabel V+

- pinnumber 3 (uncheck the “Vis?” box)
- pinseq 3 (uncheck the “Vis?” box)

V- should have

- pintype pwr (uncheck the “Vis?” box)
- pinlabel V-
- pinnumber 4 (uncheck the “Vis?” box)
- pinseq 4 (uncheck the “Vis?” box)

We now need to add a “refdes” and “device” attribute to the entire symbol (not to any particular element of the symbol). To do this, exit “pin mode” by hitting “Esc”, and then right click on the black background of the window. Choose “Add Attribute”, and create an attribute with name “refdes” and value “X?”. Move this text to the upper right of the symbol. Then create another attribute with name “device” and value “CEAmplifier”. Move this to the bottom right of the symbol.

Finally, rotate the V+ and V- labels so they are horizontal. To do so, click on the text and type “er” to rotate them, then drag them into a place that looks good. The result should appear as in Figure 3.5.

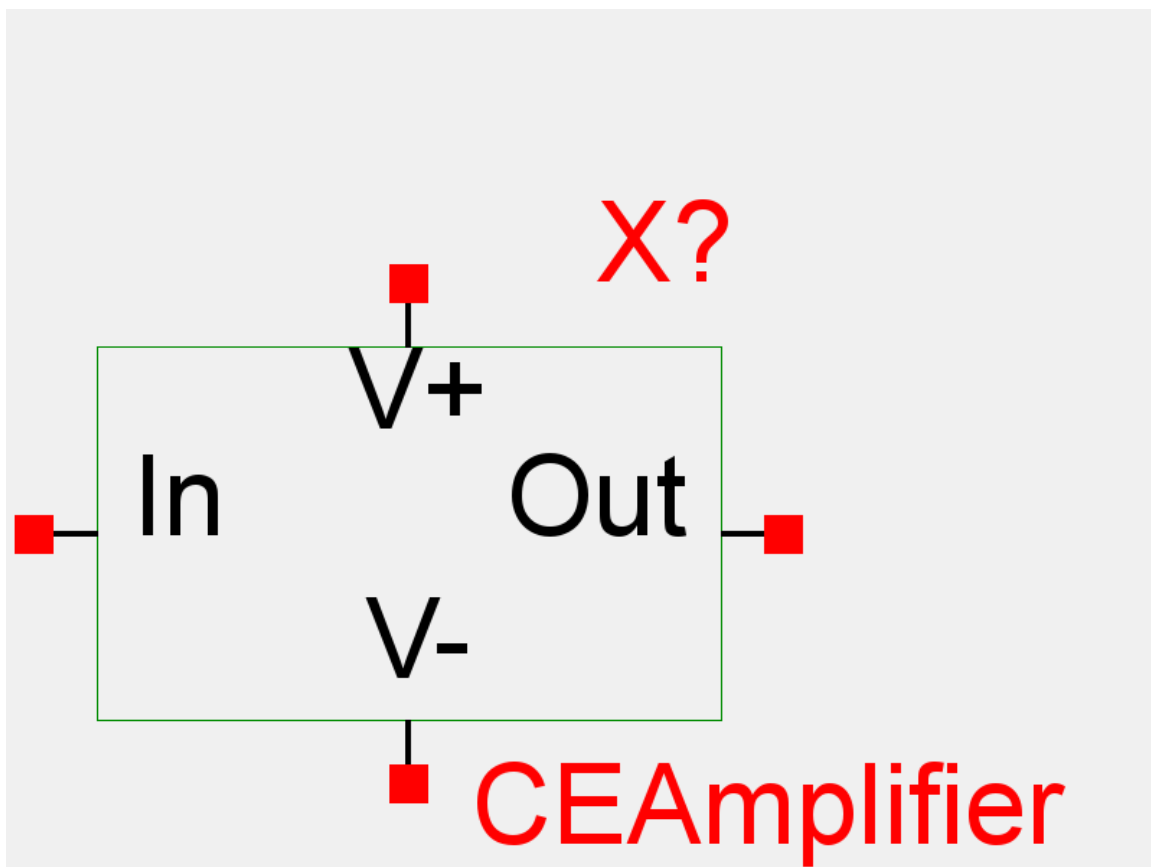
The last step is easily overlooked but is essential to making the symbol work correctly. In the “Edit” menu choose “Symbol Translate...”, enter “0” into the dialog box and click “OK”. This translates the symbol to the origin, so when it is read into a schematic later it will be able to be placed properly.

Now, create a subdirectory named “symbols” under the directory where your design is being created. Save your symbol in this directory with the name “CEAmp.sym”. Our symbol is now ready to use.

## Drawing the higher level schematic

The first step in creating our higher level schematic is to let `gschem` know where to find our new symbol. In the directory where you will do your schematic editing (and in which you have just created a “symbols” directory and saved your symbol), create a file called “gafrc” and, using a plain-text editor, add the following line to it:

```
(component-library "./symbols")
```



**Figure 3.5.** Complete Amplifier symbol

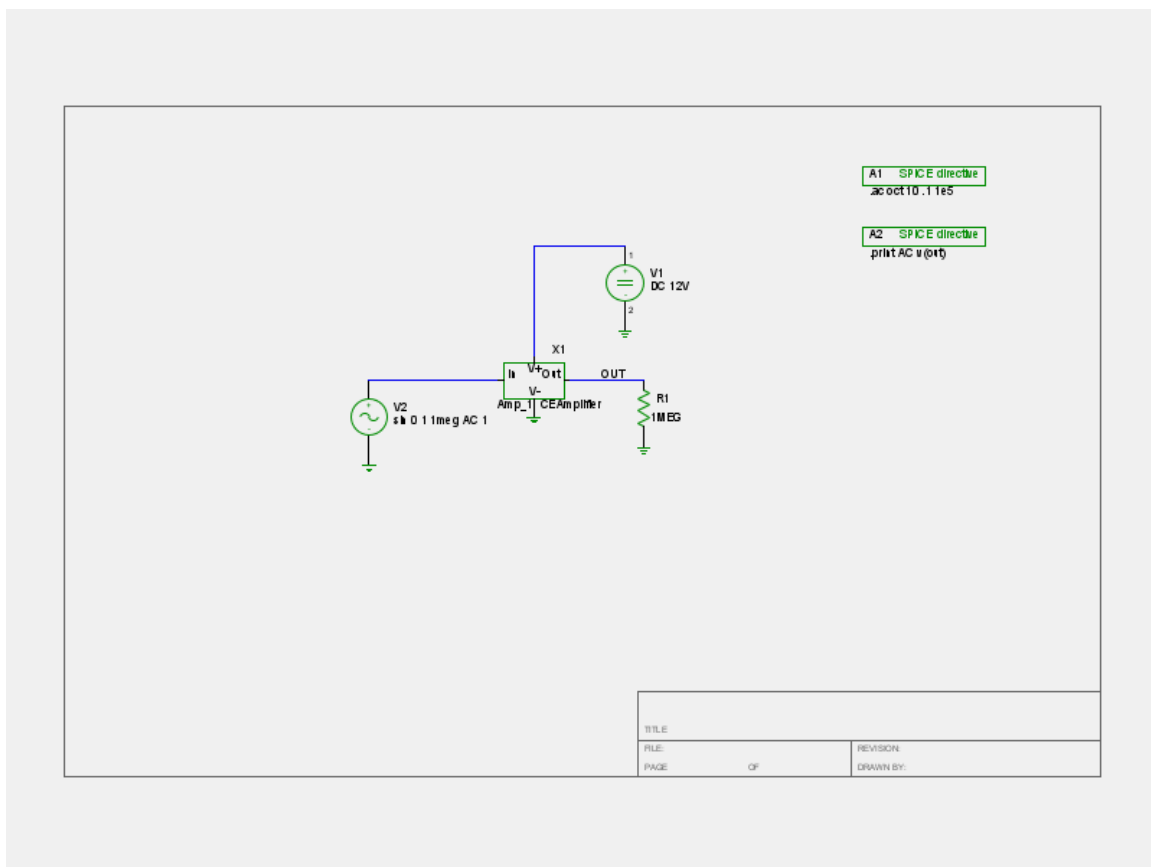
Now, when you open `gschem` your new symbol will be available as a “symbols” library in the Select Component dialog.

Open `gschem` on a new schematic. Add your new symbol to this schematic by choosing “Add Component” and selecting your symbol from the “symbols” library that now appears at the top of the Libraries tab. Place it in the center of the schematic, and add the other components that appear in figure 3.6.

Note that the input,  $V_{cc}$ , and output load resistor are all the same as they were in our original flat circuit of Figure 3.2, but the basic amplifier has been replaced with a single symbol.

Note also that we have added SPICE directive symbols with the “.ac” and “.print” lines we need to control the simulation and get output. We will only print the output signal this time.

Before we can netlist this circuit, we have to tie the X1 symbol to the subcircuit we created before. To do this, simply add a “model-name” attribute with value “Amp\_1” (the model-name we gave our subcircuit earlier) and a “file” attribute with value “Amp1.cir” (the file



**Figure 3.6.** Common Emitter Amplifier driver and load

name we generated from the subcircuit schematic) to the X1 symbol.

Save the schematic as “CE\_Amplifier\_top.sch”.

Now we can netlist with `gnetlist -o CE\_Amplifier\_top.cir -g spice-sdb CE\_Amplifier\_top.s` which will produce the following netlist:

```
* gnetlist -o CE_Amplifier_top.cir -g spice-sdb CE_Amplifier_top.sch

* Spice file generated by gnetlist *
* spice-sdb version 4.28.2007 by SDB -- *
* provides advanced spice netlisting capability. *
* Documentation at http://www.brorson.com/gEDA/SPICE/ *

*vvvvvvvvv Included SPICE model from Amp1.cir vvvvvvvvv

* Begin .SUBCKT model *
```



```

* spice-sdb ver 4.28.2007 *

.SUBCKT Amp_1 1 2 3 4
===== Begin SPICE netlist of main design =====
Q1 Vc Vb Ve Q2N2222A
.MODEL Q2N2222A NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=255.9 Ne=1.307
+ Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+ Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+ Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)
R4 4 Ve 270
R3 Vc 3 1.5K
R2 4 Vb 6.8K
R1 Vb 3 39K
C2 Vc 2 10u
C1 1 Vb 10u
.ends Amp_1

*^^^^^^^^ End of included SPICE model from Amp1.cir ^^^^^^^
*
===== Begin SPICE netlist of main design =====
.print AC v(out)
.ac oct 10 .1 1e5
R1 OUT 0 1MEG
V2 1 0 sin 0 1 1meg AC 1
V1 2 0 DC 12V
X1 1 OUT 2 0 Amp_1
.end

```

Note that the entire contents of the Amp1.cir subcircuit have been reproduced in the top-level circuit, and that the top-level nets have been correctly attached to the input nodes of the X1 subcircuit of model Amp\_1.

## Summary

To summarize this process, we performed the following steps:

- Created a subcircuit schematic, with connections to the upper level defined by “spice-subcircuit-IO-1.sym” symbols and a subcircuit declaration using the “spice-subcircuit-LL-1.sym” symbol to define the name of the model.
- Netlisted the subcircuit schematic

- Created a symbol to represent the subcircuit, with a pins “spice-subcircuit-IO-1.sym” symbol in the subcircuit. Pin sequence numbers (“pinseq” attributes) were assigned to pins to match the ordering of the subcircuit IO symbols.
- Created a “gafr” file to tell `gschem` where to find our new symbol
- Created a higher-level schematic that used the symbol, attaching attributes to match the model name and file name of the subcircuit.
- Netlisted the upper level schematic

## 4. Accessing **Xyce**-specific Netlist Features

So far, everything we have done with `gschem` and `gnetlist` has used generic SPICE features, and the netlists we have produced will work in any flavor of SPICE. But **Xyce** does have numerous netlist language extensions that are not specifically accounted for by the `spice-sdb gnetlist` backend. Some of these can be used anyway, and others require minor patches to the `spice-sdb` backend.

Fortunately, most of the features that require changes to `spice-sdb` can also be worked around in a slightly different manner and used with an unmodified version of `spice-sdb` as shipped with gEDA.

We will begin in the next sections by working through an example of a parameterized subcircuit with advanced analog behavioral modeling using an unmodified gEDA install. The first section shows the netlist we wish to create (an example from the **Xyce** Test Suite), and in the next section we work through a schematic that can produce a netlist that is functionally equivalent. In the following section we will show how a small patch to `spice-sdb` (available in the gEDA bug tracker) can simplify the process further, producing exactly the desired netlist.

### The Nonlinear Resistor Netlist

The netlist below is one of the test cases from the **Xyce** test suite, and was designed to demonstrate **Xyce**'s subcircuit parameter extension and "B" source expression extensions. This netlist contains the following **Xyce** extensions:

- Time integrator parameter specification using the `.options timeint` directive.
- Parameter definition using the `.param` statement.
- User-defined functions using the `.func` statement.
- Subcircuit definition (`.subckt` line) with parameter specification and default values (`PARAMS:`).
- Subcircuit instantiation (X device) using parameter specification (`PARAMS:`).

```

Nonlinear Resistor Circuit for Test Suite
.tran 10us 10ms
.options timeint reltol=1.0e-6 maxord=2
.print tran v(1) v(2) v(3)

C1 1 0 400uF IC=400V
L1 1 2 15mH IC=0A
vmon 2 2a 0
R1 2a 3 4
Xnlr1 3 0 nlr_PS_04 PARAMS: R0=0.15 E1=4 R1=6

.Subckt nlr_PS_04 1 2
+ Params: R0=0.15 E1=4 R1=6
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}
Vmon 1 4 0
BGabs 0 101 I = {IF(TIME < .1p, 0, 100*abs(I(Vmon)))}
Cabs 101 0 1
Rabs 101 0 1E12
.Func Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}
BEnlr 4 2 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+)
+)}
.ends
.END

```

**Figure 4.1.** Nonlinear Resistor Netlist

- B source expressions using conditionals, time-dependent expressions, and user-defined functions.

Most of these features are easily accessible from gEDA schematics, with the exception of the “PARAMS:” extension of the subcircuit definition. There is no way in the current version of spice-sdb to get additional text onto the end of a subcircuit definition.

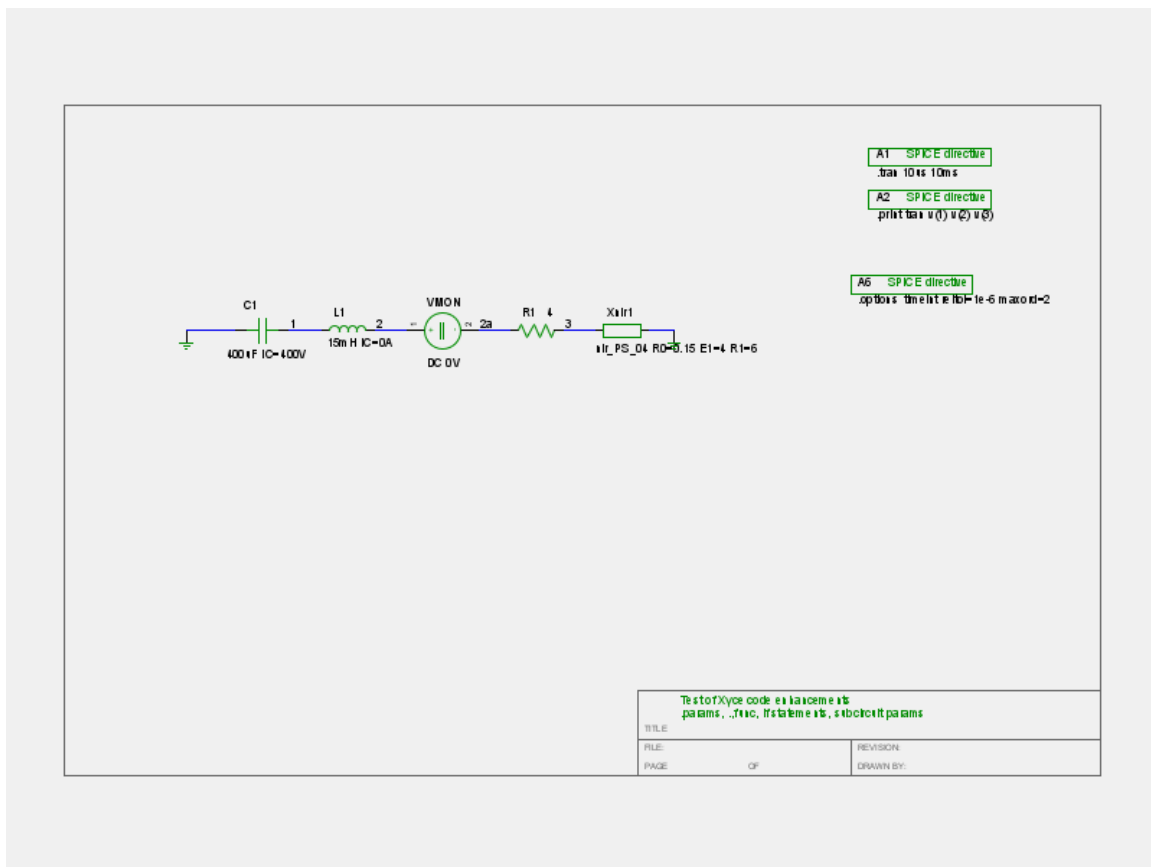
**Xyce** does, however, support a syntax for passing parameters into subcircuits even if the subcircuit definition does not include a PARAMS: block with parameter defaults. This allows us to work around this deficiency in spice-sdb without modifying that back-end. We will do just that in the following section.

## The nonlinear resistor schematic — workaround version

Figure 4.2 shows a top-level schematic for the nonlinear resistor circuit.

Creation of this schematic is straightforward. Note the following:

- The capacitor and inductor initial conditions (IC=) are simply tacked on to the component value. That is, when setting the “value” attribute, the capacitor value gets “400uF IC=400V”.
- The “Xnlr1” component uses the “resistor-2.sym” symbol from the “Basic Devices” library rather than some custom symbol. This symbol is so generic in appearance that it should be suitable for any two-terminal custom device. But we had to make a number of changes to its attributes:
  - Change its “refdes” attribute from “R2” to “Xnlr1”. The X at the beginning of the refdes is critical, as it tells the spice-sdb backend to generate a subcircuit call rather than a resistor instance.
  - Change its “device” attribute from “resistor” to “behavioral-model” (a device attribute unrecognized—and therefore ignored—by spice-sdb)
  - Add a “file” attribute with value “nlr\_PS\_04.cir”. We will shortly create a schematic for this subcircuit with that name.
  - Add a “model-name” attribute with value “nlr\_PS\_04 R0=0.15 E1=4 R1=6”. Note that this model name not only contains the name of the subcircuit, but also the parameters we are trying to pass to that subcircuit.
  - Make all attributes other than refdes and model-name invisible in the attribute editor.

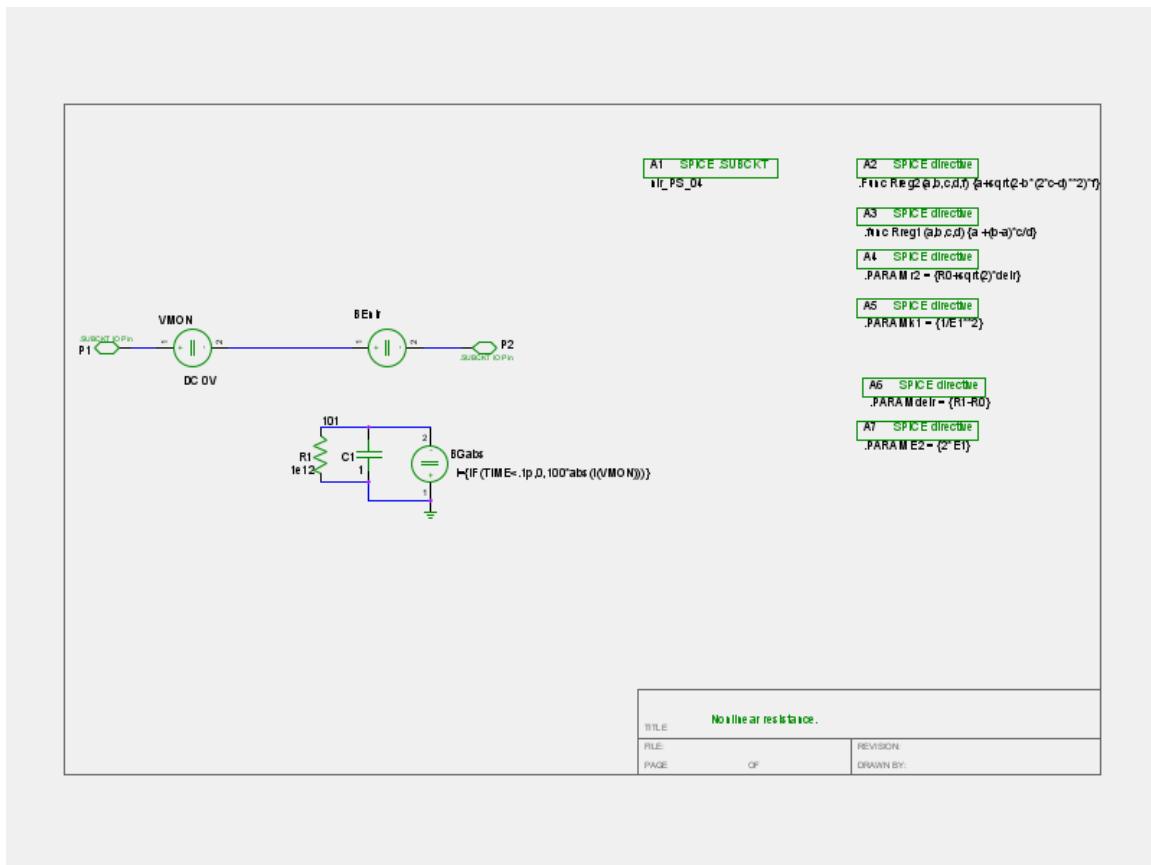


**Figure 4.2.** Nonlinear Resistor Top Level Schematic

- Add spice-directive symbols for the .tran, .print, and .options timeint lines.
- We have given all nets in this schematic “netname” attributes. Since we have named them all, it is OK for us to give some of them numeric names, as gnetlist will not be naming any nets automatically.
- We are saving this top-level schematic with the name “nlrcs10.sch”.

This top level schematic references a netlist of a subcircuit we haven’t created yet. That is our next task.

Figure 4.3 is the subcircuit schematic for the nonlinear resistance itself.



**Figure 4.3.** Nonlinear Resistor Subcircuit Schematic

The important things to note here:

- The BGabs and BEnr symbols are just instances of the DC voltage source symbol vdc-1.sym, with the first letter of their “refdes” attribute changed from V to B. Any of

the voltage source symbols would have been suitable in this regard, we just picked one. A custom symbol could also have been created. The important thing is that the refdes have “B” as its first letter, so that spice-sdb will create a correct line in the netlist.

- BGabs has its POSITIVE terminal connected to ground (it is difficult to see that in the figure). This is important for getting the sign of the current right. Positive current always flows from the positive to negative terminals of voltage sources (including B sources).
- The net attached to the negative terminal of BGabs has been given a netname of “101” so we can use it in expressions of the BEnlr source.
- There is a spice-subcircuit-LL.sym symbol with model-name “nlr\_PS\_04”.
- All of the .func and .param statements that were given in Figure 4.1 are inserted into the schematic using spice-directive symbols. Ordering is significant — **Xyce** requires that expressions in all .param statements depend only on parameters previously defined. Make sure your directive symbols have sequential refdes values as in the figure.
- The value of the BEnlr source is not shown in the schematic, because it is too long and would clutter the schematic. The “value” attribute has been set to invisible. That value is:

```
V={I(Vmon)*IF(
+ V(101)<E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101)<E2, Rreg2(R0,k1,E1,V(101),delr),R2
+)
+)}
```

Here again we have explicitly added continuation characters to the lengthy value so that it will be netlisted correctly. One could also have written the value as a single line.

Once the subcircuit schematic is created, it is saved as “nlr\_PS\_04.sch” and netlisted as usual. When netlisted, the result is saved in “nlr\_PS\_04.cir”:

```

* Begin .SUBCKT model *
* spice-sdb ver 4.28.2007 *

.SUBCKT nlr_PS_04 1 3
===== Begin SPICE netlist of main design =====
```



```

.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}
.func Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

BEnlr 2 3 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+)
+)}

R1 0 101 1e12
C1 0 101 1
BGabs 0 101 I={IF(TIME<.1p,0,100*abs(I(VMON)))}
VMON 1 2 DC 0V
.ends nlr_PS_04

```

Note that unlike the subcircuit netlist of Figure 4.1, this subcircuit does not contain a “Params:” block in the “.SUBCKT” line. Thus, the parameters R0, R1 and E1 have no default value, and those parameters MUST be specified explicitly in the subcircuit instantiation or given a value in a .PARAM statement.

Fortunately, we have done exactly that in our top-level schematic. Now that we have the low-level schematic we can run gnetlist on “nlrcs10.sch” to obtain:

```

* gnetlist -o nlrcs10.cir -g spice-sdb nlrcs10.sch

* Spice file generated by gnetlist *
* spice-sdb version 4.28.2007 by SDB -- *
* provides advanced spice netlisting capability. *
* Documentation at http://www.brorson.com/gEDA/SPICE/ *

*vvvvvvvvv Included SPICE model from nlr_PS_04.cir vvvvvvvvv

* Begin .SUBCKT model *
* spice-sdb ver 4.28.2007 *

.SUBCKT nlr_PS_04 1 3
===== Begin SPICE netlist of main design =====
.PARAM E2 = {2*E1}

```

```

.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}
.func Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

BEnlr 2 3 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+)
+)}

R1 0 101 1e12
C1 0 101 1
BGabs 0 101 I={IF(TIME<.1p,0,100*abs(I(VMON)))}
VMON 1 2 DC 0V
.ends nlr_PS_04

~~~~~ End of included SPICE model from nlr_PS_04.cir ~~~~~
*
*===== Begin SPICE netlist of main design =====
.options timeint reltol=1e-6 maxord=2
.print tran v(1) v(2) v(3)
.tran 10us 10ms
Xnlr1 3 0 nlr_PS_04 R0=0.15 E1=4 R1=6
R1 2a 3 4
VMON 2 2a DC 0V
L1 1 2 15mH IC=0A
C1 1 0 400uF IC=400V
.end

```

With the exception of the “PARAMS:” block in the subcircuit definition and the absence of a “PARAMS:” keyword in the subcircuit instantiation line, this can be seen to be the same as the circuit in Figure 4.1.

But what if we want to use this nonlinear resistor in a lot of different places of our circuit, and want there to be a default value for the three parameters? Normally, in **Xyce** we would specify these defaults in the .SUBCKT definition, and they would be applied if the subcircuit were instantiated without any parameters being set. This workaround does not allow that mechanism.

The workaround would be to create .PARAM statements at the top level of the schematic that sets R0, R1, and E1 to their default values. Any X line that instantiates the “nlr\_PS\_04” subcircuit without specifying parameters would use those values. Any X line that *does*

specify the parameters would override the defaults.

This is not a very pretty way to set parameter defaults, as it peppers the top-level design with `.param` statements setting default values for the lower level design, but it does work, and requires no modifications to `spice-sdb`.

In summary:

- It is possible to work around the inability of `spice-sdb` to create advanced `.subckt` definitions with **Xyce**'s "PARAMS:" block simply by passing parameters to the subcircuit without the "PARAMS:" keyword, and doing without the subcircuit's declaration of default parameter values.
- **Xyce**-specific statements such as `.FUNC` or `.PARAM` can be pushed into a netlist simply by adding `spice-directive` symbols.
- It is possible to push default values to the subcircuit using `.PARAM` statements in `spice-directive` symbols.
- It is not strictly necessary to generate custom symbols for every subcircuit design if an existing symbol can be pressed into service simply by changing its attributes.
- While no special symbol exists for **Xyce** (or SPICE) B source (behavioral modeling source), existing voltage source symbols can be used just by changing their `refdes` attribute, and by specifying the entire behavioral modeling expression as the voltage source's value attribute.

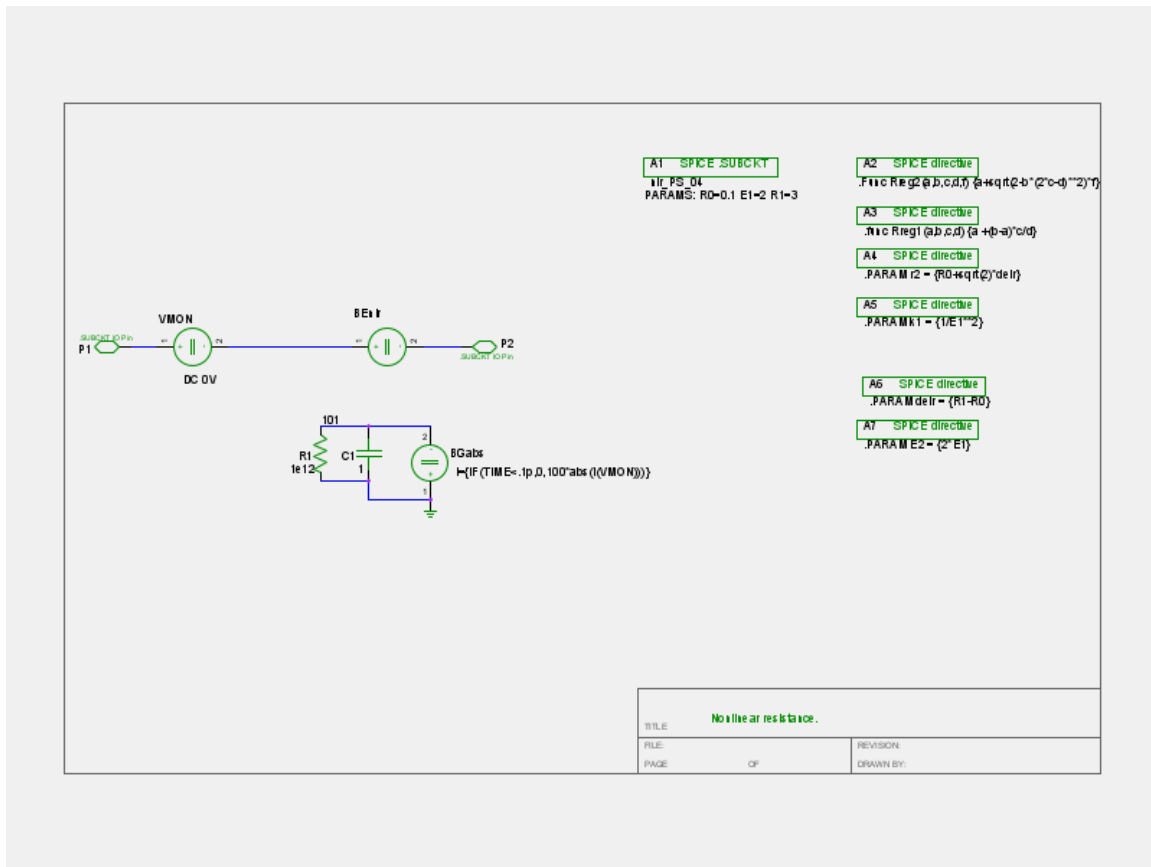
## Patching `spice-sdb`, and an improved **Xyce** nonlinear resistor schematic

Fortunately, gEDA users have recognized a need for the `spice-sdb` `gnetlist` backend to allow augmentation of the `.SUBCKT` line just as **Xyce** would need. As a result, one user has submitted a bug report to the gEDA issue tracker that contains a simple patch to `spice-sdb` that does just that.

The issue report at <https://bugs.launchpad.net/geda/+bug/698736> asks for exactly this feature, and has attached to it a patch that provides it. The patch modifies only the `spice-sdb` file, and does not require recompiling any of the gEDA suite. If you do not know how to apply patches to files, or if you do not want to modify your installation of gEDA, you can consider this section optional material.

Once applied, the patch allows any text in the "value" attribute of a "spice-subcircuit-LL" symbol to be tacked on to the end of the `.subckt` line generated by `gnetlist`. We can then

use almost the same schematic that we did before for the nlr\_PS\_04 schematic, but add a “value” attribute with the text “PARAMS: R0=0.1 E1=2 R1=3” to the spice-subcircuit-LL symbol. It will appear as in Figure 4.4.



**Figure 4.4.** Nonlinear Resistor Subcircuit Schematic

In the top-level schematic, the only thing we will do is modify the `Xnlnr1` model-name so that it has the keyword “PARAMS:” in between the actual model name and the parameter specifications. It will appear as in Figure 4.5.

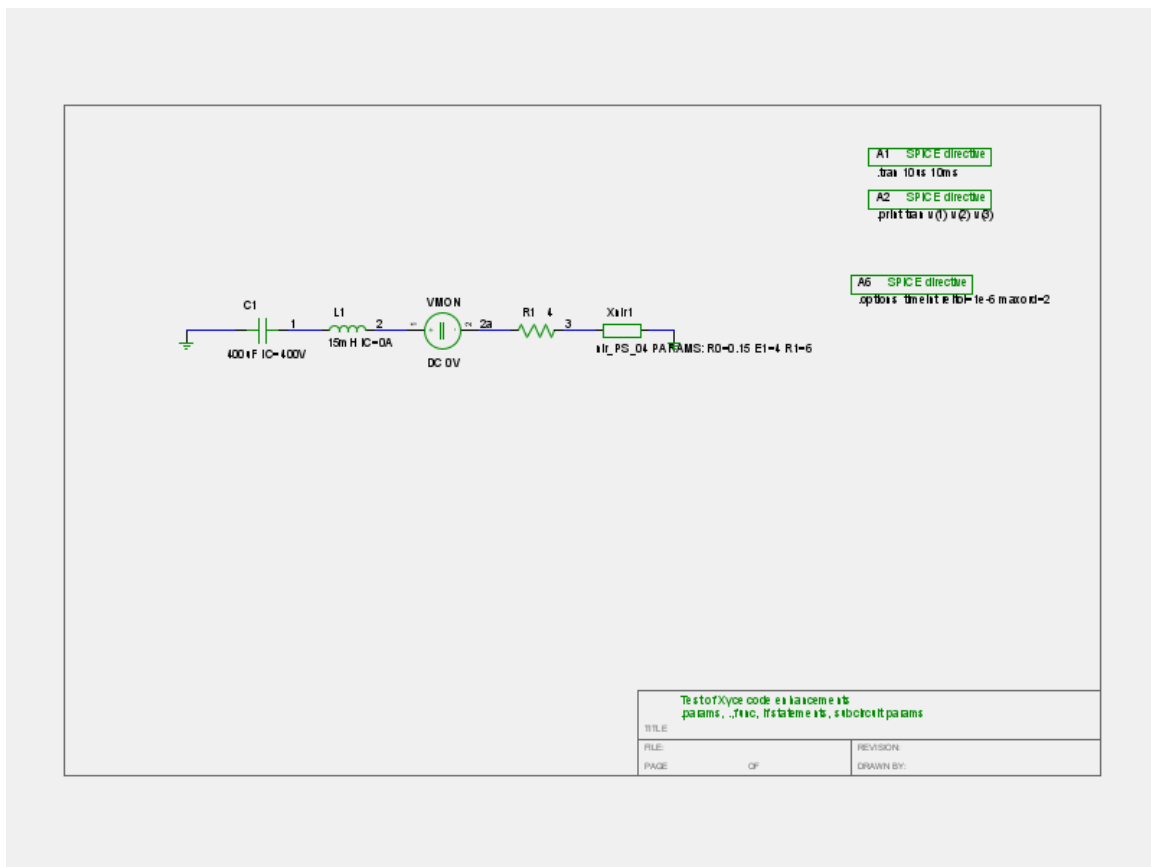
Everything else remains the same, but now the default values for the subcircuit parameters are specified in the subcircuit schematic and subcircuit definition in the netlist directly, rather than hacked in with `.PARAM` statements at top level. This subcircuit is somewhat more reusable than before.

The netlisted version of the subcircuit is exactly as expected:

```

* Begin .SUBCKT model *

```



**Figure 4.5.** Nonlinear Resistor Top-Level Schematic

```

* spice-sdb ver 4.28.2007 *

.SUBCKT nlr_PS_04 1 3 PARAMS: R0=0.1 E1=2 R1=3
===== Begin SPICE netlist of main design =====
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}
.func Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

BEnlr 2 3 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+)
+)}

R1 0 101 1e12
C1 0 101 1
BGabs 0 101 I={IF(TIME<.1p,0,100*abs(I(VMON)))}
VMON 1 2 DC 0V
.ends nlr_PS_04

```

From this, it is clear that the patch does indeed make `gnetlist` more easily able to create **Xyce** netlists with **Xyce** subcircuit extensions. It is hoped that the patch will soon become an official part of the `spice-sdb` distribution.

# A Installing gEDA on Windows

Like **Xyce**, gEDA was conceived for and originally written to support Unix-like operating systems including Linux and Mac. It is therefore not a perfect fit for Windows users, as it retains the look and feel of the original targeted operating systems, and is not as simple to install as most Windows programs.

Also like **Xyce**, the desirability of a Windows port has outweighed the difficulties, and ports have been made. DJ Delorie, a gEDA developer, has been providing pre-built packages of gEDA utilities for several years, and these are the easiest to get going. Still, there are several manual steps to take to get it running. The steps of this appendix need only be completed once. After that, the software is just as usable on Windows as it is on other operating systems, and all of the examples in this applications note will work exactly as described.

Begin by visiting <http://www.delorie.com/pcb/geda-windows/>. You will find a link to “snapshots”, <ftp://ftp.delorie.com/pub/geda-windows/snapshots/>. Click this link and locate the files named “build-xxxxyyzz.zip” where xxxxyyzz is the date the file was created. Choose the most recent date (which at the time of this writing was 20130508) and download this zip file. Be patient, it is a large download and apparently a slow network connection. This file contains everything you need to run gEDA on Windows.

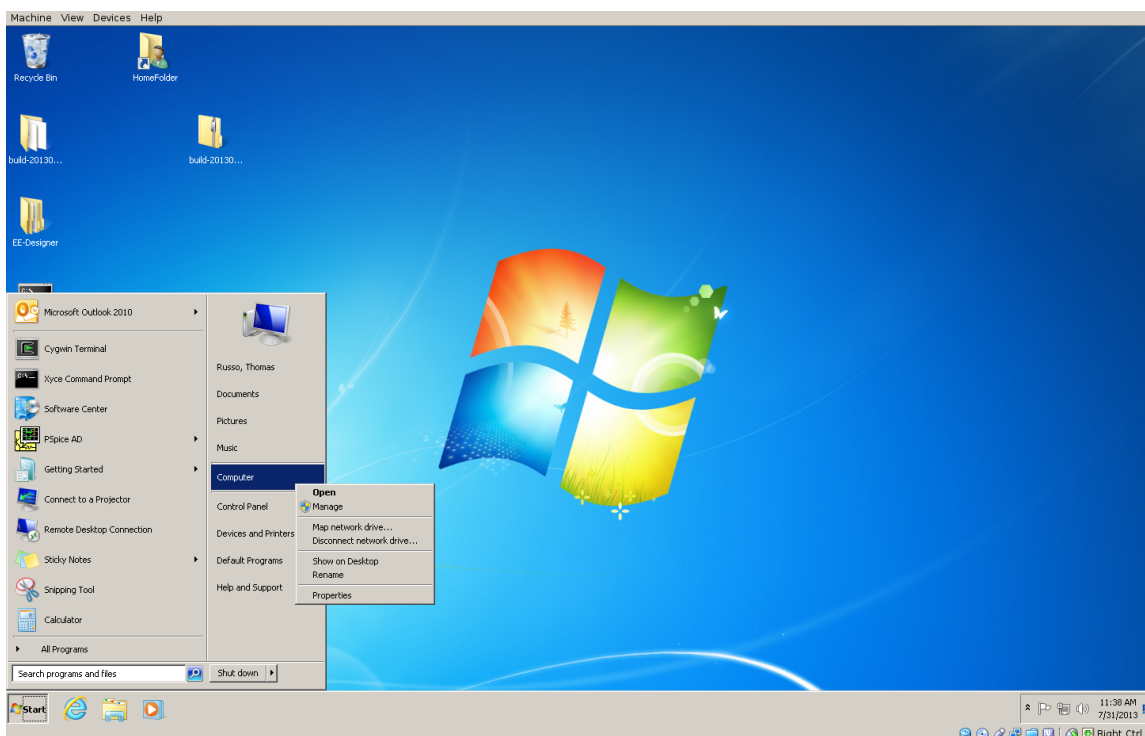
Once the download is complete, double click its icon and extract the files to a new directory (I had it extract to a “build-20130508” folder on my desktop).

When the extraction is complete, open the folder you just created and run the program “geda-runtime.exe”. This program is an installer for certain DLLs that gEDA requires.

Next you must set three environment variables. From your desktop, click the Start button, right-click the “Computer” item, and then choose “Properties,” as in Figure A.1

From the resulting window, choose “Advanced System Settings” and on the resulting “System Properties” window, click the “Environment Variables” button (Figure A.2).

The first environment variable you must set is PATH. This environment variable usually exists already, so you will find it in the upper scrolling box. Select this variable and click the “Edit...” button. Add the complete path to the “bin” folder of your extracted gEDA build. To find the complete path, open the folder in the explorer, then right click on its name in the explorer window and choose “Copy Address as Text”. **DO NOT OVERWRITE THE EXISTING VALUE OF PATH.** This variable is very important, and if you replace its existing value with your new text you could cause problems on your system. Add your new text by clicking at the beginning of the existing text, pasting the text you just copied, and then adding a semicolon to separate your new text from the previous contents. When you are sure you have done this correctly, click the “OK” button.



**Figure A.1.** Selecting Computer Properties

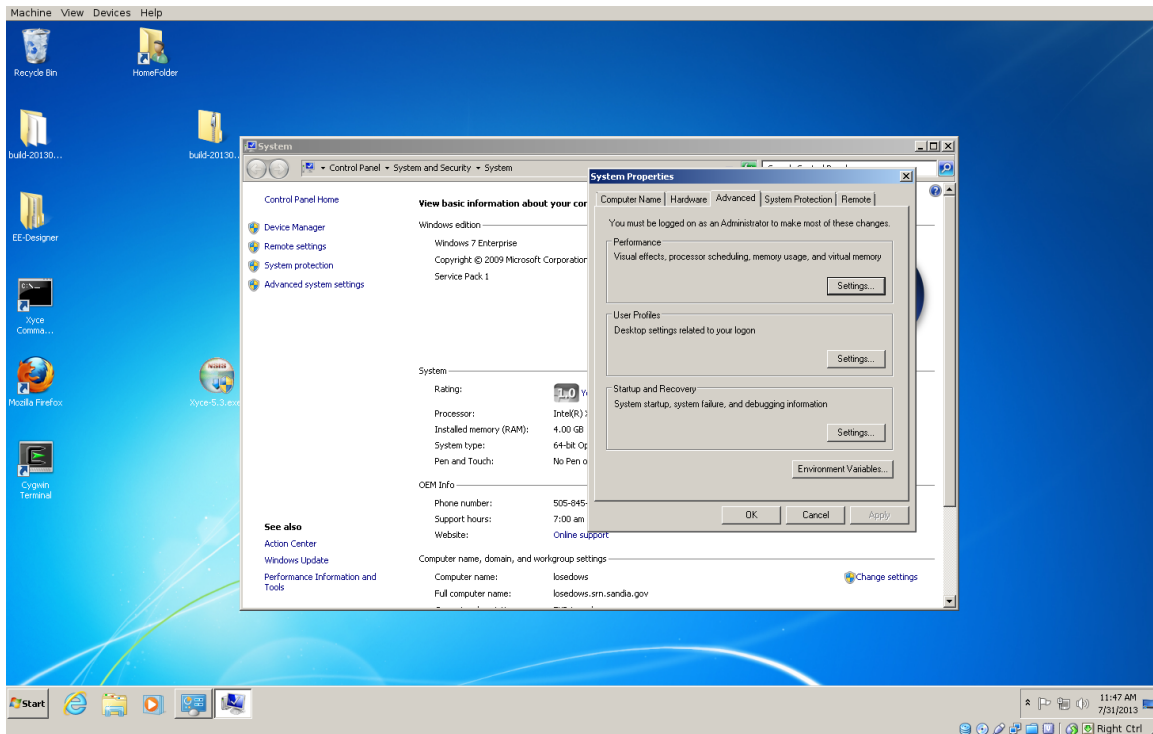
The next two environment variables you need will not exist yet, so you have to click “New...” for them.

Create one new environment variable called “GUILE\_LOAD\_PATH” and give it the full path to the “share\guile\1.8” subfolder of your build folder. Again, the easiest way to get this complete path is to open that folder in the explorer and then right-click in the location name at the top, selecting “Copy Address as Text.” (Figure A.3) Paste this address into the dialog box as the value of the variable.

Finally, add an environment variable called “GEDADATA” and give it the full path to the “share\gEDA” subfolder of your build folder.

Once you have created these environment variables, you have only one step left. The “gdk” system shipped in these builds does not come with a “loaders.cache” file that all of the programs require, so it is necessary to create one. Open a terminal window: click the start menu and type “cmd” into the “search programs and files” box, then click “cmd.exe.” (Alternatively, if you are a Sandia customer and have installed one of Sandia’s **Xyce** binaries for Windows, you can just double-click the “Xyce Command Prompt” icon.) Next, navigate to the folder “lib\gdk-pixbuf-2.0\2.10.0” of your build folder with the “cd” command (see example in Figure A.4). Enter the following command to create the loader cache:



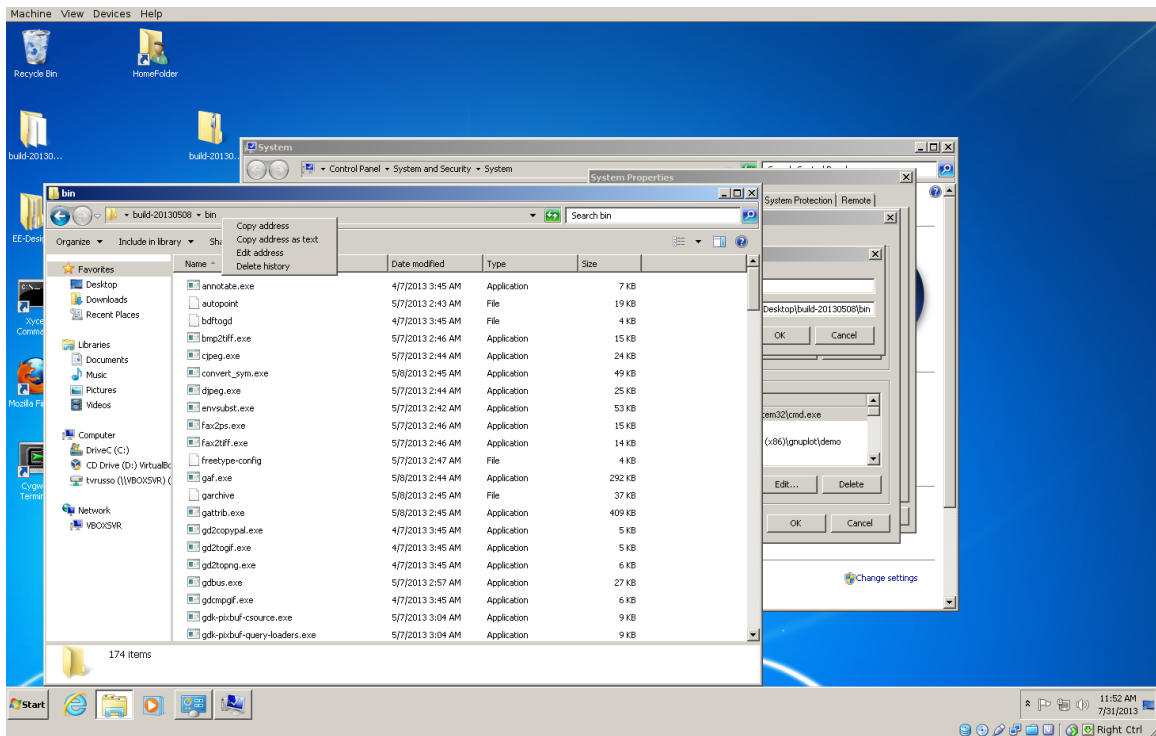


**Figure A.2.** Advanced System Settings

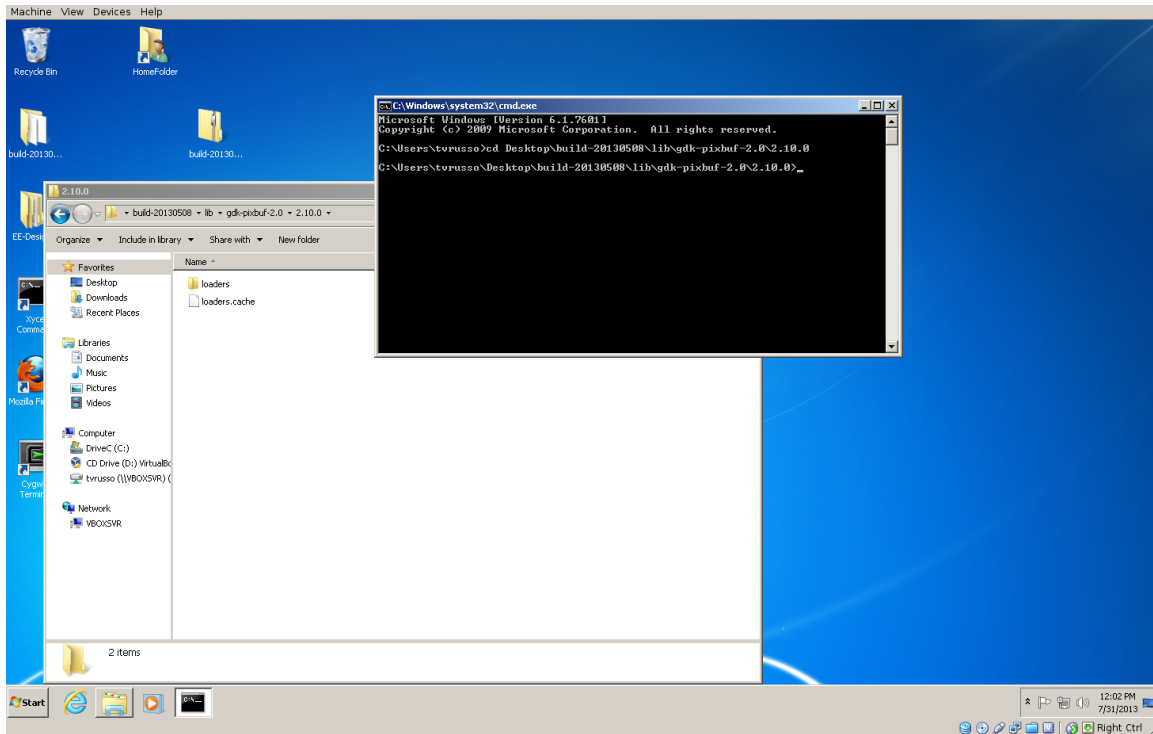
```
gdk-pixbuf-query-loaders > loaders.cache
```

Once you have completed the steps of extracting the build files, installing the geda-runtime, adding the environment variables and initializing the loaders.cache file your installation is completed. `gschem` is now accessible both as a clickable program in the `build-xxxxxxx\bin` folder or from the command line by typing “`gschem`”. If you see a number of error messages about not being able to find the “loaders.conf” file, double check that you have created it as described above, and that it is in the correct directory.

`gnetlist`, like **Xyce** itself, is only usable from the command line. Its command line syntax on Windows is identical to its command line syntax on Unix-like systems, so the instructions in the main body of this application note will all work correctly.



**Figure A.3.** Copying the full path



**Figure A.4.** Navigate to the “lib\gdk-pixbuf-2.0\2.10.0” directory

## References

- [1] Eric R. Keiter, Ting Mei, Thomas V. Russo, Eric L. Rankin, Richard L. Schiek, Heidi K. Thornquist, Jason C. Verley, Deborah A. Fixel, Roger P. Pawlowski, and Keith R. Santarelli. Xyce parallel electronic simulator: User's guide, version 6.0. Technical Report SAND2013-WWWW, Sandia National Laboratories, Albuquerque, NM, 2013.
- [2] H. Ward Silver. Hands-on radio experiment 83. *QST*, 93(12), 2009.
- [3] H. Ward Silver. Hands-on radio experiment 84. *QST*, 94(01), 2010.
- [4] H. Ward Silver. Hands-on radio experiment 85. *QST*, 94(02), 2010.
- [5] H. Ward Silver. *Hands-On Radio Experiments Volume 2*. The American Radio Relay League, Inc., 2013.

## DISTRIBUTION:

1 MS 0899      Technical Library, 9536 (electronic copy)





