gEDA notes

October 14, 2011

Contents

1 gEDA configuration files

1.1 The global gafrc file

The gEDA suite consists of many different tools, which often need to know where your component libraries are. The `gafrc` file is a repository for this information. It can either be placed in `~./gEDA`, or in an individual project directory. Here's an excerpt from the `gafrc` file I keep in `~./gEDA`:

The individual tools can be further customized with individual `rc` files like `gschemrc`, which are then also placed in the `~./gEDA` directory.

1.2 The gschemrc file

The configuration options available for your local `~./gEDA/gschemrc` are shown in the "system-gschemrc" file. On my machine, this file is located in

`/usr/share/gEDA/system-gschemrc`.

1.3 PCB configuration

The configuration file for PCB will vary with the GUI toolkit PCB was compiled with. Both lesstif and gtk are supported as of `pcb-20060414`. Use

```
pcb -h
```

and look for "Available GUI hid:" to determine what's been installed and see a list of configuration options. The double-dash options correspond to X and gtk resources. Example:

Command line:
```
pcb --grid-color '#ff0000'
```

.Xdefaults (for lesstif toolkit):
```
pcb.grid-color: #ff0000
```

$HOME/.pcb/preferences (for gtk toolkit):
```
grid-color = #ff0000
```

2 Making symbols for gschem

Using `gschem` itself is one way to view and modify device attributes. This section will describe how to use `gschem` to create and modify parts to be used in schematics.

When you include a part in a schematic, `gschem` associates the part filename with the schematic. Thus, changes made to the part files outside of the schematic will automatically update to the schematic. This is bad when you "translate" the symbol when editing it, because the symbol will then move around on all the schematics it's associated with.

2.1 Making symbols for electrical and non-electrical parts

Electrical symbols are those that have pins and make connections you can describe with a netlist. Non-electrical symbols are for parts like nuts and bolts that should be included in the bill of materials but not the netlist. Non-electrical parts need the `device`, `part`, `refdes`, and `value` attributes.

2.1.1 Making a new symbol from scratch

1. Run `gschem` – bring up a blank schematic page. Get rid of the default titleblock.
2. Draw the symbol

   Lines and pins should be drawn with 0.1 inch snap to grid. Use *Options* → *Snap grid spacing* to set grid size and be sure to turn snapping on.
3. (Electrical symbols) Add pins with *Add* → *pin*

   Pins should be 0.3 inches long and have their red ends pointing away from the symbol – net connections are made at the red ends

   Pins should be at least 0.4 inches from their neighbors
4. (Electrical symbols) Add pin attributes: `pinnumber`, `pinseq`, `pinlabel`

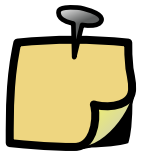
   `pinseq` just tells gschem the order to output pins when it makes a netlist.

   The `pinnumber` attribute must match up with the ones you use for your footprints. This will label pins on the netlist, so make this "value only" visible. If you want the pin number to be visible, it should be positioned 50 mils away from its pin.

   The `pinlabel` attribute is just for information. It lets you find the pin entry if you ever need to edit the raw text file for the symbol.
5. (Electrical symbols) Add the component's footprint attribute

   Make this attribute invisible, but edit the attribute (after turning the invisible text on) to select "show name and value." Then both the attribute name and value will show up when you display invisible text.

   Example: `footprint=2pin_MTA100_pol`



Look in the project's eda\footprints for available footprints. The file footnotes.org may have some footprint-specific notes. Use `$pcb footprint.fp` to see a footprint. See section 8 beginning on page 27 for more information about footprints.

6. Add the component's device attribute

   The device attribute is required for using `gnetlist` later on

   The device attribute is the basic part description, like `resistor`, `capacitor`, or `opamp`

   This should be made invisible

7. Add the component's "value" attribute

   This is the flavor of the device. This will appear on the schematic to label the symbol along with the reference designator, so make this "value only" visible. Feel free to insert spaces or symbols in this field, since it's only for the user's information.

   Resistors – the resistor value

   Opamps – the basic part number without package or other suffixes

   Zeners – the basic part number. The zener voltage can go in the optional "label" attribute

   Connectors – a description of the connector, like "polarized MTA100"

   Screws – The pitch and length, like 6-32 x 5/16 phillips.

8. Add the Faulhaber Labs part number in a "part" field. These fields can just be typed in – they don't have to be members of the pull-down list. This part field will reference another database containing purchasing information and other comments.

   Example: `part=14-6`

9. (optional) Add a `label` attribute for things like zener voltages or resistor power ratings. The `label` attribute also isn't in the default pull-down list – you have to write it in. Make this value visible. Just writing things like this in text prohibits them from rotating relative to the symbol.

10. Add a reference designator (`refdes`)

    Adding a designator like U? allows gschem to choose a value for ? when the device is in a schematic.

11. Translate the symbol to the origin

    Edit → Symbol translate

    Keep all the invisible text visible when translating. Otherwise it will be translated out of the window.

12. Save your component with the filename `partname.sym`. The save dialog box allows you to choose whether you save things as a schematic or a symbol, but it does the same thing regardless of your selection.

Figures 1, 2, and 3 show examples of what you should see when you're done with a symbol.

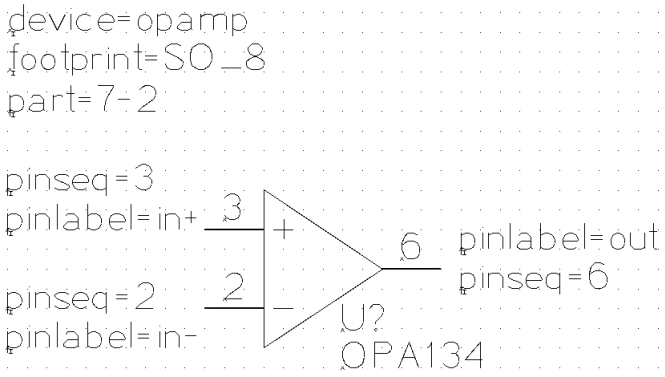


Figure 1: A nice-looking opamp symbol with all invisible attributes showing. This opamp has a separate power symbol that doesn't need any symbol attributes, only pin attributes.

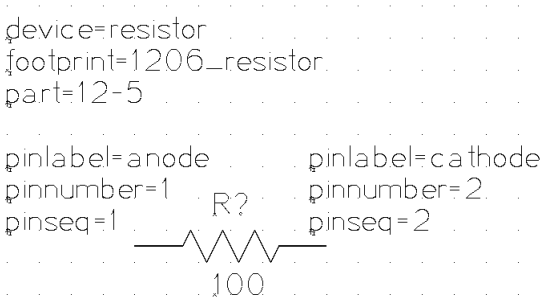


Figure 2: A nice-looking resistor with all invisible attributes showing

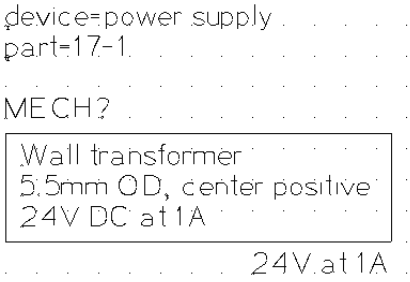


Figure 3: A non-electrical component

2.1.2 Making multi-part symbols

Multi-part symbols are those for which two or more symbols are used to represent a single part. Making them amounts to simply making multiple symbols and naming their files suggestively. For example, two symbols describing a OPA134 op amp could be named *opa134a* and *opa134b* . You'll have to remember that these two files are related.

The way **gnetlist** handles multi-part symbols seems to be a work in progress. These rules seem to hold for **geda** version 1.6.2:

1. All symbols representing the same part must share the same reference designator. For example, the *opa134a* and *opa134b* symbols could both have the `U1` reference designator, but not `U1a` and `U1b`.

2. Only the "first" part of a muti-part symbol should have anything other than the reference designator defined. Attributes like the footprint name or the part number will be ignored for parts placed after the first part.

gschem adds entries to a schematic file as they are added with the graphical editor. Symbols placed chronologically before others will appear at lower line numbers. The netlister scans the schematic file from top to bottom. If it finds two symbols with the same reference designator, it will use attributes from the first one and ignore those from the second.

2.2 Making power, ground, and other net symbols

Net symbols consist of a graphic, a pin, and a net attribute. To edit the net attribute at the schematic level, add the attribute as an overall symbol attribute as shown in Fig. 5 – not one associated with the symbol's pin. Since these net symbols only have one pin, this attribute always looks something like `name:1`. After placing the symbol on the schematic, change the name of the net attribute as needed. The name should then look like `+9V:1` or `input:1` etc.

There doesn't seem to be a way to avoid including the pin number with the name if you want to be able to change the net name on the schematic. Another way to make these symbols is to permanently associate a net name with the symbol, and then just add a text label like `+9V` to the symbol. This scheme is illustrated in Fig. 4. You would have to have a different symbol file for each net you want to have a symbol for.

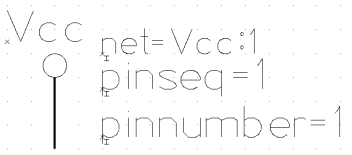


Figure 4: A power symbol with a hard-wired net attribute. The attributes to the right of the symbol are made invisible on the schematic.

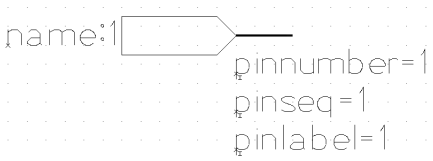


Figure 5: An off-page connector symbol with a net attribute you can change at the schematic level. The attributes to the right of the symbol are made invisible on the schematic.

2.3 Changing attributes at the schematic level

A part's default attributes can be overridden by editing the part within your schematic. Selecting the part and clicking *Edit* → *Edit* (or using the `ee` shortcut) will bring up an "Edit Attributes" dialog box. Here you can add new attributes to override the defaults. For example, adding an attribute with the name "footprint" will override the footprint attribute taken from the symbol file.

2.4 General notes

- When you start gschem on a blank page, you'll be zoomed way out. Zoom in a lot until the lines you draw snap to your visible grid.
- Multi-part devices should have a separate symbol for each part, named like part-a, part-b, etc. For example, a single op-amp should have two symbols: one for the amplifier and one for its power pins. Only the first device needs to have all the attributes – the following devices only need reference designators.
- Op-amp symbols should be 0.8 inches high by 0.6 inches long.
- If a new device has the same symbol as one you've already drawn, just edit the text file and save it as a new part.
- Canned symbols are in `/usr/share/gEDA/sym`
- No line feeds are allowed in symbol files
- Part attributes must be associated with text – you can't have a name=attribute entry in a symbol file without also having a text formatting line immediately above it. This means that the pinnumber, pinseq, and pinlabel attributes will pile up on the schematic page if you make them visible.

3 Project file etiquette

For consistency:

- A project usually consists of more than one schematic file. The collection of files should be put under version control, so that the filenames themselves never pick up a version number.
- `gsch2pcb` will tack ".pcb" on to the string you give it as "output-name," so you don't need to add it. The generated file should be moved to the layout directory. There it should be versioned to keep track of changes. The netlist should not be versioned since it is generated by the schematic.
- If a schematic has more than one page, the first page should be a table of contents.
- If the ECO changes the BOM, modify it by hand and append `_eco` to the filename. I won't bother keeping track of multiple rounds of ECOs – each new ECO should just overwrite the `_eco` BOM. If there's a website with the BOM on it, the BOM should be updated to the `_eco` version.

3.1 Directory structure

A nice directory structure looks like this:

- **project** – The top-level directory.
  - **datasheets** – Datasheets for parts used in the project.
  - **eda** – Version-controlled eda tools.
    Check out using `svn checkout svn://www.jrranalytics.com/eda`
  - **proto** – Version-controlled prototyping tools.
    Check out using `svn checkout svn://www.jrranalytics.com/proto`
  - **notes** – Design notes for the project.
    doctools – Version-controlled documentation tools.
    Check out using `svn checkout svn://www.jrranalytics.com/doctools`
  - progress.org – Progress notes to be edited with org-mode.
  - **reva** – Letter-coded revision directory.
    * **data**
    * **drawings** – Mechanical drawings for the revision.
    * **schematics** – Electrical schematic pages
      kit2 – Number-coded directory generated with the *kitgen* script.
    * **layout** – Circuit board layout
    * **pictures**

3.2 Preparing a project for version control

```
$ mkdir project
$ mkdir project/datasheets
$ mkdir project/notes
$ touch project/progress.org
$ mkdir project/reva
$ mkdir project/reva/data
$ mkdir project/reva/drawings
$ mkdir project/reva/schematics
$ mkdir project/reva/layout
$ mkdir project/reva/pictures
$ mkdir project/reva/howto
$ mkdir project/reva/code
$ svn import ./project svn://www.jrranalytics.com/projects/project -m "Initial import"
$ mv project project_nosvn
$ svn checkout svn://www.jrranalytics.com/projects/project
$ rm -rf project_nosvn
```

Now add the subversion externals:

```
$ cd project
$ svn propedit svn:externals .
```

...which will prompt you to edit a file defining the externals. Add the lines

File: svn-prop.tmp

```
...
eda svn://www.jrranalytics.com/eda
eda/notes/geda/doctools svn://www.jrranalytics.com/doctools
eda/notes/avr/doctools svn://www.jrranalytics.com/doctools
notes/doctools svn://www.jrranalytics.com/doctools
reva/howto/doctools svn://www.jrranalytics.com/doctools
```

to any other externals you'd like to define. Now a simple `svn update` will bring in the externals.

You must create a task tagged with the project name after creating a project. The top-level task name should be "`* TODO Work on` *project name*", with a link to the project location below as a comment.

3.3 Using the `makeproj.py` script

The `makeproj.py` script automates project creation by creating the directories detailed in section 3.1. To use it, first edit end script to customize your project's name, home directory, and revision letter. Then the command

```
$ python makeproj.py
```

will create the project.

The script can also make a new revision of an existing project. Just change the revcode while leaving the existing project name and directory.

4 Working with gschem

4.1 Keyboard shortcuts

Some keyboard shortcuts are shown in table 1. A comprehensive list is brought up with *Help* → *Hotkeys* or with the `hh` key sequence.

Key sequence	Function
z	Zoom in
Shift + z	Zoom out
en	Show/hide invisible text
l	Start drawing a line (like for a symbol)
s	Exit whatever mode you're in and go back to select mode
m	Begin moving a selected object
ex	Bring up the single attribute editor within a schematic

Table 1: gschem keyboard shortcuts

4.2 Adding text

The key combination `a`→`t` brings up a text entry box. Entered text can be edited using the `e`→`x` sequence. Nice-looking text sizes are shown in table 2.

Text	**Size**
Comments	12
Headings, like "notes" or "changes on this page"	18

Table 2: Text sizes for notes on schematic pages

4.3 Making pretty prints

You can print to postscript with *File* → *Print* and then choosing a filename. You can print from the command line using

```
$ gschem -s /usr/share/gEDA/scheme/print.scm -o output.ps -p input.sch
```

where `input.sch` is the schematic to be printed to `output.ps`.

4.3.1 Fixing text alignment

Pieces of text have alignment marks associated with them. When the text is printed, these marks will stay put as the text grows or shrinks to accomodate whatever fonts are used by the output device. You'll thus want to align the alignment marks if you want text objects to line up in your print. There are two tricks to this:

1. Rotate the text until the alignment mark is where you want it. Notice that `gschem` refuses to write text upside down.
2. Use the `ex` sequence to edit the alignment mark position of a text attribute associated with a symbol.

4.4 Miscellaneous notes

-

5 Working with gnetlist

5.1 Generating a netlist

The command

```
gnetlist -g PCB <input.sch> -o <output.net>
```

will create a netlist readable by both you and PCB. The file will have three columns: net name, starting pin, and ending pin.

5.2 Generating a bill of materials (BOM)

The `-g` flag for gnetlist allows you to specify a guile script to be run. Scripts for BOM creation are located in `/usr/share/gEDA/scheme`, and some are described in table 3.

Script	Description
`bom`	Reads an "attribs" file and generates a bill of materials with each component on a separate line. Columns are seperated by tab characters. Lines are not sorted.
`bom2`	Reads an "attribs" file and generates a bill of materials grouping components with the same value attribute on the same line. Columns are separated by colons. Different items in the same column are seperated by a comma character. Later versions of this backend have a quantity column appended to whatever appears in `attribs`. Find the latest version at `git.gpleda.org`.
`partslist1`	Creates a bill of materials listing each component on a separate line. Unlike the `bom` script, this always produces "refdes," "device," "value," "footprint," and "quantity" columns. No attribute file is required. Lines are sorted alphabetically by refdes. Since every line contains just one part, the quantity is always 1.
`partslist2`	Similar to `partslist1`, but lines are sorted by the value of the device attribute and not the refdes.
`partslist3`	Groups components with the same value on the same line, as in `bom2`. Lines are sorted by the value of the device attribute. The fourth column reports the number of parts in a line. Columns are seperated by the tab character, items by space.

Table 3: Scripts for BOM creation

For example, the command

```
gnetlist -g bom <input.sch> -o <output.bom>
```

will generate a BOM. You must have a file called "attribs" in the directory from which gnetlist is invoked. This file should just contain a column of attributes you want included in the BOM. An example is:

```
device
value
part
footprint
```

5.3 General notes

6 Working with gsch2pcb

The gsch2pcb back-end will generate a pcb file containing the footprints you specified in gschem. It will not handle the netlist – this must be done separately. An easy way to use it is

gsch2pcb *project*,

where this project file contains details about which schematics to include and where your footprints are.

6.1 Creating a project file

The project file allows you to link different schematic pages together into one design, and supplies gsch2pcb with some command-line arguments. The file may look like this:

```
schematics one.sch two.sch
output-name board
elements-dir /home/john/audio/geda_resources/jps_footprints
```

where I guess gsch2pcb refers to footprints as "elements."

6.2 Element locations

Canned element files are located in

/usr/share/pcb/newlib

where "newlib" refers to the "new" way of creating elements for pcb. These elements are created graphically using pcb. Another "oldlib" style involves using the GNU m4 macro processor to parametrically define footprints.

The very nice footprints stored in

~/hobby/eda/footprints/luciani

were taken from

http://www.luciani.org/geda/pcb/pcb-footprint-list.html,

a collection created by John C Luciani Jr. These can be moved to my footprints directory and renamed so that gsch2pcb can find them. I like to modify pad shapes so that only pin 1 is square, leaving the rest oblong. This can be done by changing the hex code at the end of the "pad" statements inside the element definition file. The code 0x0000 sets the shape to oblate, while 0x0100 sets it to square.

6.3 Moving on to PCB

If all goes well with gsch2pcb, you should just be able to run pcb on the file it generates. The footprints it brings in will be stacked on top of each other, but you can use *Select→Disperse all elements* to break them up.

7 Working with PCB

7.1 Using the command line

Use <:> to bring up command-line entry, which can be dismissed with <*esc*> .

7.2 Changing the board size

PCB sets the physical board to be a rectangle the size of your working area by default. This working area size can be set using **file** → **preferences** → **sizes** . However, the physical board dimensions and shape can be made different from the working area by drawing on the "outline" layer.

To make a custom board outline, use the line tool to draw a 10-mil wide line on your outline drawing layer. Make sure this layer isn't associated with any physical board layers by looking at **file** → **preferences** → **layers** → **groups** . The outline group number should be different from every other. The 10-mil thickness is a convention, and the center of the outline drawing lines will become the actual board outline.

7.3 Bringing in a netlist

The `gsch2pcb` program will generate a netlist file for you to bring into the `*.pcb` file it also generates. Bring it in using *file* → *load netlist file*. As an added feature, `gsch2pcb` generates a `*.cmd` file that will rename footprint pins to coincide with their schematic parts. This file has to be processed using pcb's command line entry, accessed with *window* → *command entry*. Once there, enter

```
ExecuteFile(*.cmd)
```

to process the command file. Finally, hit "w" to make rubber bands show up.

The rat lines are pretty thick by default. To change them, edit the "rat-thickness" setting in whatever configuration file you have. I like 1 mil.

7.4 Working with layer groupings

PCB allows you to associate different logical routing "layers" with "groups." The idea is to allow color-coding of different traces according to their purposes. All traces belonging to the same layer will have the same color. For example, power traces could be associated with the "power" layer and all be colored red. Each logical layer must be associated with one unique group, and each group ultimately becomes the artwork for one of a board's physical layers. Thus, you can't have all power traces belong to the same layer if they exist on different sides of the board. You can have them all be the same color though, simply by creating "power-solder" and "power-component" layers with the same colors.

The layers "solder side" and "component side" are always defined since all boards have two sides. You can't draw traces directly on them though – you need to associate them and yet another layer with the same group. To draw traces on the component side, you have to create a layer called something like "component" and map it to the same group as "component side" using *file* → *preferences* → *layers*.

The number of groups available will increase as you add logical layers. Specifically, the number of groups will be the same as the number of layers, excluding the predefined solder and component side layers. So, you have to have four custom layers defined to make a four-layer board.

7.5 Silkscreen preferences

The default minimum silkscreen text width of 10 mils makes text and reference designators hard to read. Setting this to 5 mils is acceptable for Sierra Proto Express and makes things more readable. Using *file* → *preferences* → *sizes* to change the "minimum silk width" parameter will change the width of all current and future silkscreen text.

7.6 Adding a ground/power plane

Planes of copper are added by drawing rectangles. These planes become associated with the first nets they're connected to. To draw a ground plane, select the layer associated with your ground plane and draw a rectangle on it. All pins and pads sharing the same physical layer will now be isolated from the rectangle by their individual "clearance" distances. To actually connect one and thus connect your plane to a net, select the thermal tool and then the pin/via to be connected. A thermal relief will show up. Optimizing the rat's nest now with *o* will make little rat circles on pads that want to connect to the plane.

Key	Function
j	Toggle a track's "clears polygon" attribute. This allows you to connect tracks to planes or copper pours.
k	Increase isolation radii around vias or through-hole pins in a copper polygon.
$< shift > k$	The opposite of *k*
\	Toggle "thin draw" mode for drawing only the ouline of polygons. You can't see though filled-in polygons to see tracks on the other side of the board.

Table 4: Key bindings important for working with ground planes and copper pours. Hotkeys modify the object the cursor hovers over.

- The thermal tool can only be used on pins or vias – not pads. Pads can be connected to planes by drawing a track between the two and using *j*.
- New lines drawn on a polygon will not be isolated from it unless *settings* → *new lines, arcs clear polygons* is checked.

7.7 Adding tracks

Tracks are added using the line tool. Turning on 135° mode as described in table 5 makes the tracks look prettier. As with everything else, you must activate the layer you wish to draw on before you draw anything.

Key	Function
.	Toggle between line and track mode
/	Toggle between 45° and 135° track modes

Table 5: Key bindings important for making tracks with the line tool.

7.8 Adding vias

Vias are added using the via tool. I think the track-to-via routing flow is in flux with PCB, but what works is dropping a via on top of a track to associate it with a net. When switching layers during routing, drop a via on the last point of a track, then continue the track from the via on another layer. Use the number keys to switch routing layers. Some keys important to vias are listed in table 6. Note that the "via size" is the overall via diameter.

Key	Function
<*shift*>v	Via size +5mil
<*shift*><*ctrl*>v	Via size -5mil
<*alt*>v	Via drill +5mil
<*shift*><*alt*>v	Via drill -5mil
k	Clearance in polygons +2mil
<*shift*>k	Clearance in polygons -2mil

Table 6: Key bindings important for making vias with the via tool.

As for via sizes, the important numbers come from the board fabrication houses. A 15-mil drill is a good minimum via drill size, and an overall via size of 30 mils will give an annulus of 7.5 mils – a typical minimum annulus is 5 mils. A polygon clearance of 10 mils is a good minimum size, and 12 mils is safely above that. For routing more power, remember that a 15-mil via has a diameter of about 50 mils. The hole plating thickness is roughly the same as copper thickness for boards plated with 1 oz/ft^2. So the minimum via size can handle quite a bit of power. Table 7 lists some good via sizes to start with.

	Drill (mils)	Clearance (mils)	Via size (mils)
Signal	15	12	30
Power	22	12	42

Table 7: Some good via sizes

7.9 Adding mounting holes

Adding mounting holes works just like manually adding any other element. Use *file* → *load element data to paste-buffer* and select your mounting hole footprint file. Then you can use the mouses to place the holes on the board.

7.10 Part placement notes

- For prototyping, a nice "pitch" for 1206 resistors and capacitors is 110 mils or greater.

7.11 Finished board checklist

1. Run the design rule checker (DRC) on your layout using **connects** → **design rule checker** . This uses the sizes specified in **file** → **preferences** → sizes. Good sizes for Sierra Proto Express are shown in table 8.

Size	Value (mils)	Comment
Minimum copper spacing	6	Sierra allows 6 mils
Minimum copper width	6	Sierra lumps this together with spacing as "minimum trace and space"
Minimum touching copper overlap	10	This is how PCB knows that two conductors are touching, and not a constraint imposed by the board house
Minimum silk width	5	Sierra allows down to 5 mils
Minimum drill diameter	15	Sierra allows 15 mils
Minimum annular ring	5	Sierra allows 5 mils

Table 8: Good DRC settings for use with Sierra Proto Express

7.12 Creating Gerber files

I usually create a directory called "submit" in the layout directory to contain Gerber (RS-274X) files. Create these files using **file** → **export layout** → **gerber** . These can be inspected using the `gerbv` program (installed separately but part of gEDA).

8 Footprint notes

- To print footprints at 1:1 scale, use

  ```
  pcb -x ps --psfile "output.ps" --media Letter --show-legend <footprint>
  ```

  to dump all the footprint layers to the "output.ps" postscript file.

- Holes for through-hole parts should be 15 mils larger than the pin diameter
- The "clearance" number is the anti-copper width/diameter to be added around pads, pins, or vias that intersect a copper plane. This isolates them from the plane. The number is added to the thickness number to get the total outer dimension of the anti-copper feature. Using 2000 here gives 10 mils of isolation, which is fine for plated holes. Unplated holes require 2400, for 12 mils of isolation.
- Solder mask reliefs should be at least 3 mils. For pads and pins, the number is the overall solder mask opening – the "thickness" plus 600.
- Use 10 mil line thicknesses for silkscreen part notes and elementlines.
- Measuring distances in PCB: use < *ctrl* >m to place a reference crosshair for relative coordinates

8.1 John Luciani's footprints

John Luciani has a nice collection of footprints for PCB at

`www.luciani.org`

I take his footprints and make some standard changes before they go in my footprint directory. The original footprint code is shown in Fig. 6.

```
Element[0x0 "SMD" "" "" 0 0 -10161 -12011 0 100 0x0]
(
   Pad[-5511 -393 -5511 393 6299 2000 8299 "input" "1" 0x0100]
   Pad[5511 -393 5511 393 6299 2000 8299 "input" "2" 0x0100]
   ElementLine[-10161 5043 -10161 -5043 1000]
   ElementLine[-10161 -5043 10161 -5043 1000]
   ElementLine[10161 -5043 10161 5043 1000]
   ElementLine[10161 5043 -10161 5043 1000]
)
```

Figure 6: Luciani's `1206` footprint code

First of all, remember that units are 1/100 of mils, since all definitions are in square brackets instead of parenthesis. Thus, 1000 → 10 mils. The standard changes I make are:

1 The first field of the element line becomes `""` instead of `0x0`. These are the same thing, but using `""` is more consistent with other documentation I have.

2 The second field of the element line becomes `""` instead of `SMD`. This field is clobbered by `gshem2pcb`, so it doesn't matter what the initial value is. Having something here is a distraction.

3 The fifth and sixth fields of the element line become 1000. These fields are the initial placement of the footprint on the pcb, and 1000 puts them on the board by 10 mils.

```
Element["" "" "" "" 1000 1000 -10161 -12011 0 100 0x0]
(
   Pad[-5511 -393 -5511 393 6299 2000 8299 "anode" "1" 0x0100]
   Pad[5511 -393 5511 393 6299 2000 8299 "cathode" "2" 0x0100]
   ElementLine[-10161 5043 -10161 -5043 1000]
   ElementLine[-10161 -5043 10161 -5043 1000]
   ElementLine[10161 -5043 10161 5043 1000]
   ElementLine[10161 5043 -10161 5043 1000]
)
```

Figure 7: My 1206 footprint code

Figure 7 shows my resulting footprint.
And here are some more miscellaneous notes:

- Luciani.org footprints are named using dimensions in 1/100 of mm. Thus, multiply inch dimensions by 2540 to determine his equivalents.

8.2 Leaded footprint example

An example of a good leaded component is shown in Fig. 8. I include this just because Luciani's footprint for this type of element used deprecated syntax.

```
Element ["" "" "" "" 1000 1000 0 10000 0 100 0x0]
(
   Pin [0 0 7000 2000 7600 4600 "anode" "1" 0x0100]
   Pin [150000 0 7000 2000 7600 4600 "cathode" "2" 0x0000]
   ElementLine[30000 23000 120000 23000 1000]
   ElementLine[30000 -23000 120000 -23000 1000]
   ElementLine[30000 23000 30000 -23000 1000]
   ElementLine[120000 23000 120000 -23000 1000]
   ElementLine[10000 0 30000 0 1000]
   ElementLine[120000 0 140000 0 1000]
)
```

Figure 8: My AX_900L_455W_1500LS_32LD footprint code. This is for an axial-leaded part with 0.9 inch body length, 0.455 inch width, 1.5 inches between leads, and 0.032 inch lead diameter. This could be that big inductor used on the LED driver.

8.3 Modifying existing footprints

1. Start up PCB with no arguments to bring up an empty layout.
2. **File → Load element data to paste-buffer** . Select the footprint (PCB calls them footprints before they're put on the board, and elements after) you want to modify, and place it somewhere.
3. Use the select tool to select the element, then **Buffer → Cut selection to buffer** . All the menus will be grayed out as PCB waits for you to choose where you'd like to "grip" the selected items. I like to choose the diamond-shaped insertion point marker at this point. After choosing, use <*esc*> to release the objects.
4. Select **Buffer → Break buffer elements to pieces** .
5. Select **Buffer → Paste buffer to layout** . The footprint will look different, because it won't have pads anymore – just traces where the pads used to be (only fully-formed footprints have pads, and we broke up our footprint). Place the collection of pieces somewhere in the layout.
6. Make the modifications you want to make. Hover over the traces that will become pads and use <*n*> to assign pin numbers. Of course, right now they'll be called "linenames."
7. Use the selection tool to select everything you want to be in the new footprint.
8. Select **Buffer → Cut selection to buffer** . Select a gripping point and use <*esc*> to release the objects. This gripping point will become the insertion point of the new footprint – marked with a diamond-shape.
9. Select **Buffer → Convert buffer to element** .
10. Select **Buffer → Paste buffer to layout** . Now you'll have pads instead of traces. Hovering over a pad and pressing <*q*> will toggle pads between round and square shapes. Hovering over a pad and pressing <*d*> will toggle pin number visibility. You can't change the pin numbers at this point, but it's nice to make sure they're correct. Hovering over the insertion point and pressing <*n*> will prompt you for an element name. Whatever name you enter will be clobbered by the reference designator set by gsch2pcb, but it's nice to choose a default position for the label and the text size at this point. The keys <*s*> and <*shift*> <*s*> will adjust text size.
11. When you're satisfied with your footprint, select **Buffer → Cut selection to buffer** . Select your insertion point again and use <*esc*> to release the object.
12. Save the footprint with **Buffer → Save buffer elements to file** . I like to end my footprint filenames with an fp extension. You're done!

9 Engineering change orders (ECOs)

9.1 Changing part footprints

The PCB file format contains raw element (footprint) code and not file paths. Thus, changes to footprint files will not change layout files. If you need to change a footprint, delete the bad footprint and save your layout file. Run `gsch2pcb`, specifying your current layout file as the output file. The program will generate a new file containing the appropriately named missing footprint read from its source. This new layout file information can be brought into your current layout, thereby updating the footprint. You'll be told exactly what to do when `gsch2pcb` is run.

9.2 ECOs that affect the netlist or BOM

ECOs that affect the netlist or BOM must be recorded on both the schematic pages where they apply and in a new BOM. The procedure is as follows:

1. Add a note on the schematic under a "Changes on this page" heading, and place a label referring to the note next to the affected parts or nets.

2. Edit the BOM within gnumeric to reflect the change. Append `_eco` to the BOM filename. Multiple levels of changes won't be tracked – all ECOs just overwrite the most recent BOM.

10 Kit and order generation

The scripts `kitgen.py` and `buygen.py` can help with building kits and generating orders for vendors. The design flow is as follows:

10.1 Configure `kitgen.py`

- Make sure the relative path from the **scripts** directory to the schematics directory is correct.
- Set the *kitnum* variable to correspond to a new kit.
- Set the *kitqty* variable used to set the number of units to be built.
- Make sure the name of your description file is correct.

File: buttprog/eda/scripts/kitgen.py

```
...
#----------------------------Begin configuration-------------------------

# The relative path from the project/eda/scripts directory to the
# schematics directory.
# Should be something like ../../reva/schematics with just the rev letter
# changed.
schpath = '../../reva/schematics'

# The kit number (should just be an integer)
kitnum = 6

# The kit quantity (how many boards do you want to build)
kitqty = 1

# List of part descriptions
descfile = '../purchasing/descriptions.dat'

#. ----------------------------End of configuration-----------------------
```

10.2 Run the kitgen script

You can create a symbolic link to the script in the schematics directory using

```
$ ln -s ../../eda/scripts/kitgen.py kitgen.py
```

...and any file paths you've entered can stay the same. The script will always refer to files relative to the **scripts** directory.

```
$ python kitgen.py


* /home/john/projects/jrr/buttprog/reva/schematics/kit6 already exists.  I'll overwrite it.
-------------------Output from gnetlist-------------------
Loading schematic [/home/john/projects/jrr/buttprog/reva/schematics/buttprog.sch]
----------------------------------------------------------
* Found 21 parts.
* Fill kit by editing kit6_fill.dat with emacs.  Then run buygen.py
  to see what needs to be ordered.
* Assembly BOM written to kit6_build.bom and kit6_build.tex. Execute 'latex
  kit6_build.tex' to process .tex file.
```

10.3 Edit the kitx/kitx_fill.dat file with emacs to fill the kit by hand

This fill file will be clobbered every time kitgen is run. This is on purpose, as kitgen has no way of determining what has changed in the schematic file. When new parts come in to fill shortages, the whole kit needs to be filled all over again. This is the only way the kit will reflect the latest schematic.

Resist the urge to print out the fill file and write in quantities by hand. This basically ensures that the kit will be out-of-date with respect to the schematic.

What if I have parts with long lead times? Is there a way to track the status of a kit?

10.4 Process the kitx/kitx_build.tex file with LaTeX

The kitgen.py script will generate a kitx/kitx_build.tex file for use in populating circuit boards. This file can be processed into a pdf with the sequence:

```
$ latex kitx/kitx_build.tex
$ dvips -t letter kitx/kitx_build.dvi -o
$ ps2pdf kitx/kitx_build.ps
```

10.5 Configure buygen.py

- Make sure the relative path from the **scripts** directory to the schematics directory is correct.
- Make sure all the vendor files you'd like to use are entered.

File: buttprog/eda/scripts/buygen.py

```
...

...
```

10.6 Run the buygen script

This will create the summary file `kitx/kitx_summary.dat` if it hasn't been created already. This file may have repeated parts in it, since buygen just blindly throws every part number match between the shortage list and the vendor files into the summary file. You then have to choose which vendor you'd like to source the repeated part, and remove the others. This is done by editing the summary file.

```
$ python buygen.py
```

10.7 Edit the summary file with emacs to remove repeated vendors

The `kitx/kitx_summary.dat` file will have a "remove" column. Placing an "x" in any row's remove column will cause that row to be deleted the next time buygen is run. Buygen will issue `--fix-->` lines until all the repeats have been removed.