# gEDA gnetlist Users Guide

Ales Hvezda



September 21st, 2003

# Contents

# 1 Introduction

This document describes how to use gnetlist. This document and gnetlist in general are pretty ALPHA, so keep that in mind as you use it to generate netlists. As all engineers know, it is very important that you do not blindly trust tools, assuming that they will always create correct output. gnetlist is certainly no exception to this rule. It is very important that you verify *every* netlists you create. As with most programs (including all the programs in gEDA), gnetlist comes with NO WARRANTY. Blah, I hate having to say that, but I'm hoping that this warning will keep the user from assuming that gnetlist generates perfect netlists. Though if you find a bug, please let `ahvezda@geda.seul.org` know.

This document is very rough, so please e-mail all corrections to `ahvezda@geda.seul.org` or file a bug report on the gEDA homepage at `http://www.geda.seul.org`. Thanks!

# 2 Overview

`gnetlist` is the gEDA netlister. It takes as input schematic files and produces a netlist. A netlist is a textual representation of a schematic. This textual representation has all of the connections between devices completely resolved. This means that all the connections associated with a net are grouped together. The netlister also handles hierarchies of schematics.

`gnetlist` has a very flexible architecture. The main program, which is written in C, reads in a schematic (using routines from libgeda) and creates an internal representation of the schematic data. This internal representation is then manipulated by a backend which is responsible for writing the various netlist formats. The backend for each netlist format is written in scheme (specifically Guile). This architecture not only allows for an infinite number of netlist formats, but also allows the netlister to generate other reports (like bill of material lists).

As of 20001006 gnetlist has scheme backends to support the following netlist formats:

1. PCB & PCBboard - UNIX PCB netlist format.
2. Allegro netlist format
3. BAE netlist format
4. BOM & BOM2 - Bill of Material generators
5. DRC - Start of a design rule checker
6. gEDA - the native format of gEDA, mainly used for testing
7. Gossip netlist format

8. PADS netlist format
9. ProtelII netlist format
10. Spice compatible netlist format
11. Tango netlist format
12. Verilog code
13. VHDL code
14. VIPEC netlist format
15. VAMS - VHDL-AMS netlist format

This list is constantly growing. Several lacking features (as of 20001006) are: no support for buses, error detection and reporting is fairly limited, and ... (many more).

# 3 Installation

Hopefully by now you have already installed gnetlist on your machine. This document does not cover installation. You can verify the installation by running:

```
libgeda-config --version
gesym-config --version
which gnetlist
ldd `which gnetlist`
```

The first two should return the version of the installed tools (libgeda and the symbol library) and the next command should return the path to the gnetlist binary. The final command (only on Unix-like operating systems which include the `ldd` utility for listing dynamic dependencies of executable files or shared objects) will return which libraries are linked to gnetlist; all of the request libraries must be found for gnetlist to run. If these commands do not return the expected results, then most likely the gEDA tools are not installed properly. Please see the appropriate INSTALL docs (which came with the distribution) for more info on installing the gEDA tools.

# 4 Running gnetlist

It is very easy to run gnetlist. gnetlist is a pure command line interface so there is no pesky GUI to get in the way :-) For a list of command line arguments please run "`gnetlist -h`".

You need to specify the following two parameters to run gnetlists:

- `-g proc` (this specifies which backend to run against the schematics)
- `filename.sch` (this specifies the schematic files)

You can specify multiple schematics on the command line. The default filename for the generated netlist goes into "`output.net`" You can change this default location by using the `-o filename` option.

A few examples on running gnetlist:

```
gnetlist -g geda -o stack.net stack\_1.sch
```

(output netlist (in `stack.net`) for `stack_1.sch` using the gEDA native format)

There are also a few debugging flags. The first one is the `-v` flag which enables verbose mode. Verbose mode outputs a whole bunch of information on what gnetlist is doing as well a dump of the internal representation. The `-i` flag which puts gnetlist into a interactive mode is very useful in debugging scheme backends and typically is not used by the end user.

For a detailed list of command line arguments please see the gnetlist man page.

# 5 Schematic / symbol requirements

This section describes what schematics/symbols need to have to be usable with gnetlist. In order for gnetlist to work correctly, both the schematics and supporting symbols must be correct. Basically these requirements consist of attribute specification. Attributes are used through out the gEDA system to represent information. Attributes are the only way of adding information to components, nets, pins, etc... For more detailed information about the attributes mentioned in this document, please see the `attributes.txt` document (Master attribute list).

## 5.1 Symbol requirements

- All symbols must have a device= attribute.

- All pins must have the `pin#=#` attribute. This attribute will eventually change form, but for now it is required as `pin#=#`
- All pin should also have a `pinlabel=` attribute.
- For symbols which are slotted you also need the `slot=` attribute, for each slot a `slot#=#` attribute, and the `numslots=#` attribute. Slotting will also change in the near future, but for now it should be specified as above.
- For any power/gnd/arbitrary you need to put `net=` attributes inside the symbol. See the netattrib.txt document for more info.
- You can supply default values for various parameters (this is dependent on which backend you use) by taking advantage of the attribute "promotion" mechanism. See below for more info as well as the gschem documentation.
- For symbols which you want the netlister to completely ignore use the `graphical=1` attribute
- For more tips on symbols, please see the symbol creation document.

## 5.2 Schematic requirements

- Most importantly, every component you want to show up in a netlist must have a `refdes=` attribute. This is *VERY* important. gnetlist should warn you if you have a component which doesn't have a `refdes=`, but there have been bugs which do not cause this warning.
- You can label all nets using the `label=` attribute. You only need to attach this label to one net segment (of an electrically connected net) for all the net segments to inherit the label.
- You can have multiple schematics in a design (which is actually a confusing term since it means many different things to people). To use multiple schematics to create a single netlist, just specify them on the gnetlist command line.
- If you name nets the same, then these nets will be electrically connected. Same net names spawn all the specified schematics.
- There are quite a few issues that deal with hierarchy please see the hierarchy section below.

## 5.3 Random notes

- Attributes which are not attached to anything and are inside a symbol are "promoted" to the outside of the symbol when the symbol is placed inside a schematic (in gschem). These promoted attributes are always looked

at/for first before going into the symbol. So, in other words, if there is an attribute with the same name is inside a symbol and attached to the outside of the instantiated component, then the outside attribute takes precedence.

# 6 Hierarchy Support

TBA

# 7 Specific backend info

TBA

# 8 Scheme backend API

Please note that this section is still under construction. The information here should be correct, but it is not complete.

## 8.1 Overview

gnetlist operates by loading the schematic database from the .sch files, building an internal representation and then calling a function specific to the desired output netlist type which performs the actual netlisting. Each gnetlist backend is contained in a file called gnet-¡backend¿.scm. Where ¡backend¿ is the name of the particular backend. For example, gnet-switcap.scm contains the code used by "gnetlist -g switcap" and gnet-drc.scm contains the code used by "gnetlist -g drc". The backends are written in the Scheme programming language. The particular implementation of scheme is guile which stands for GNU's Ubiquitous Intelligent Language for Extensions. More information about guile may be found at http://www.gnu.org/software/guile/guile.html.

## 8.2 Entry Point

Each netlist backend is required to provide a function whose name matches the netlist type. For example, the switcap backend contained in gnet-switcap.scm must provide a function called "switcap". That is the function which gnetlist will call to initiate the netlisting. The entry point function is given a single argument which is the filename for the output netlist. Typically the first thing a netlister does is to open the output file for writing.

The following excerpt from the switcap backend shows the start of the entry point function and shows the output file being opened. At the end of the function, the output file is closed.

```
;; ---------------------------------------
;; Switcap netlist generation -- top level
;; ---------------------------------------
(define switcap
  (lambda (output-filename)
    (let ((port (open-output-file output-filename)))

;; rest of netlisting goes here

;; close the output file and return
      (close-output-port port))))
```

## 8.3 Initialization of the Backend

After opening the output netlist, any specific initializations which must be done for the particular netlist are done. In the switcap example, we must initialize a net name and reference designator (refdes) aliasing database. This is because switcap has more restrictive requirements on its net names than gschem does. In addition, the reference designators in a switcap netlist have special requirements. To deal with this situation, gnetlist provides some general purpose functions which rename nets and reference designators to comply with the target netlist requirements. More details on this later. For now, just note that the switcap backend uses the following code:

```
      ;; initialize the net-name aliasing
      (gnetlist:build-net-aliases switcap:map-net-names
          all-unique-nets)

      ;; initialize the refdes aliasing
      (gnetlist:build-refdes-aliases switcap:map-refdes
          packages)
```

The other initialization which is typically done, although not required by all netlist types, is to output some sort of header. This header may be explicitly contained in the entry point function or it may be contained in its own function for code clarity. In the switcap backend, the call is:

```
(switcap:write-top-header port)
```

Note that the convention is for any backend specific functions to have their names prefixed by the backend name. For example all switcap specific functions begin with "switcap:". Functions which are available to all backends and provided by gnetlist are prefixed by "gnetlist:".

The definition of "switcap:write-top-header" is

```
;;
;; Switcap netlist header
;;
(define switcap:write-top-header
  (lambda (port)
    (display
      "/* Switcap netlist produced by gnetlist (part of gEDA) */\n"
      port)
    (display
      "/* See http://www.geda.seul.org for more information.  */\n"
      port)
    (display
      "/* Switcap backend written by Dan McMahill             */\n"
      port)
    (display "\n\n" port)
    )
  )
```

The entry point function continues by calling functions for each section in the output netlist. The variable "packages" is predefined by gnetlist to be a list of all components in the design and "all-unique-nets" is a list of all the nets in the design. The various functions used by the backend for each section in the netlist will use these variables. For example, the main part of the switcap netlist which contains the components and their connectivity is written to the output file with

```
(switcap:write-netlist port packages)
```

## 8.4 Net Name and Reference Designator Aliasing

It is common for a target netlist type to have a more restrictive requirement for the net names than gschem does. For example, there may be restrictions on length, allowed characters, or case. To address this issue, gnetlist provides a net name aliasing feature. To use this feature, the function "gnetlist:build-net-aliases" is called as part of the initialization section of the entry point function. For example in the switcap backend,

```
;; initialize the net-name aliasing
(gnetlist:build-net-aliases switcap:map-net-names
    all-unique-nets)
```

The function "switcap:map-net-names" is a backend specific (switcap in this case) function which accepts a gschem net name as an argument and returns a modified net name which meets the requirements for the output netlist format. In the case of switcap, the requirement is ground must be called "0", nets may have no more than 7 characters, and the netlist is not case sensitive.

```
;; This procedure takes a net name as determined by
;; gnetlist and modifies it to be a valid SWITCAP net name.
;;
(define switcap:map-net-names
  (lambda (net-name)
    (let ((rx (make-regexp "^unnamed_net"))
          (net-alias net-name)
          )
      ;; XXX we should use a dynamic regexp based on the
      ;; current value for the unnamed net base string.

      (cond
       ;; Change "GND" to "0"
       ((string=? net-name "GND") (set! net-alias "0"))
       ;; remove the 'unnamed_net' part
       ((regexp-exec rx net-name)
           (set! net-alias (substring net-name 11)))
       (else net-name)
       )

      ;; Truncate to 7 characters
      (if (> (string-length net-alias) 7)
          (set! net-alias (substring net-alias 0 7))
          )
```

```
      ;; Convert to all upper case
      (string-upcase net-alias)

      )
    )
  )
```

The function "gnetlist:build-net-aliases" creates a database which later on lets you look up the output net name from the gschem net name or the gschem net name from the output net name. In addition it does the very important task of ensuring that no shorts are created by modifying the net names. As an example suppose you had a net called "MyNet" and another called "mynet" in the schematic. Those are unique but after converting both to upper case they become a single net. "gnetlist:build-net-aliases" will detect this condition and issue an error and stop netlisting.

Now that the database has been initialized, the netlister simply uses

```
(gnetlist:alias-net somenet)
```

to retrive the netlist net name from the gschem net name.

A similar set of functions are provided for reference designator aliasing.

## 8.5 Debugging Hints

A useful debugging tool is to run gnetlist in interactive mode. This is done by using the "-i" option to gnetlist. This will give you a shell where you may enter scheme commands. This provides a simple way to examine various variables and try out various functions.

An example of running gnetlist in interactive mode is shown below.

```
% gnetlist -i ../../gnetlist/examples/switcap/*.sch
gEDA/gnetlist version 20041228
gEDA/gnetlist comes with ABSOLUTELY NO WARRANTY; see COPYING for more details.
This is free software, and you are welcome to redistribute it under certain
conditions; please see the COPYING file for more details.

Loading schematic [../../gnetlist/examples/switcap/analysis.sch]
Loading schematic [../../gnetlist/examples/switcap/ckt.sch]
Loading schematic [../../gnetlist/examples/switcap/clocks.sch]
gnetlist> all-unique-nets
```

```
("unnamed_net6" "unnamed_net5" "unnamed_net4" "OUT" "unnamed_net3"
 "unnamed_net2" "unnamed_net1" "GND")
gnetlist> packages
("TIMING" "CLK1" "S7" "S8" "S6" "S5" "C3" "S4" "C2" "C1" "E1" "S3"
 "S1" "V1" "S2" "OPTIONS" "TITLE" "ANA1")
gnetlist> (quit)
%
```