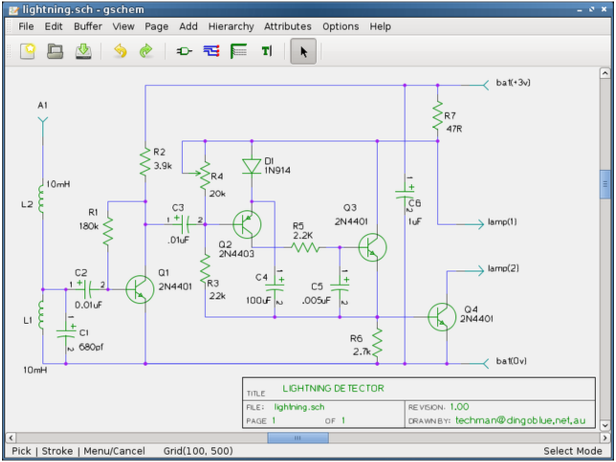
Xorn

A new approach to scripting for gEDA/gaf

Roland Lutz

February 5, 2017

The gEDA project



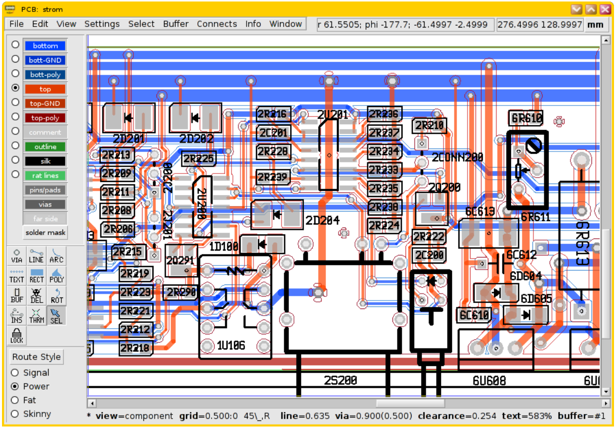
gschem

The gEDA project

```
$ gnetlist -g PCB -o - stack_1.sch
Loading schematic [stack_1.sch]
SP_BUS_L        U210-1
unnamed_net1    U211-13 U212-4
BUS_SP_L        U211-11 U212-11
CLK_SP_L        U211-14 U212-14
SPUP_L          U211-5 U212-5
Vcc             U210-20 U211-16 U212-16 U109-24
SEL_DS_L        U210-19 U211-4 U109-18
STACK_BUS_L     U109-20
BUS_STACK_L     U109-21
...
```

gnetlist

The gEDA project

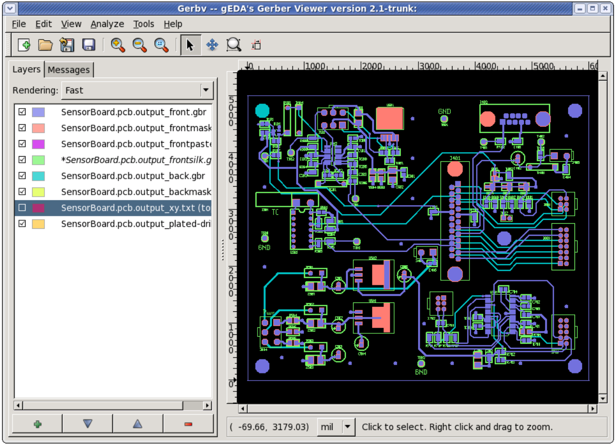


r 61.5505; phi -177.7; -61.4997 -2.4999 276.4996 128.9997 mm

* view=component grid=0.500:0 45_R line=0.635 via=0.900(0.500) clearance=0.254 text=583% buffer=#1

PCB

The gEDA project



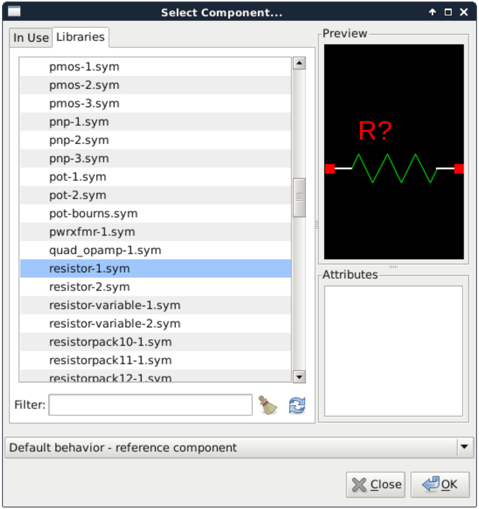
Gerbv -- gEDA's Gerber Viewer version 2.1-trunk:

Rendering: Fast

☑ SensorBoard.pcb.output_front.gbr
☑ SensorBoard.pcb.output_frontmask
☑ SensorBoard.pcb.output_frontpaste
☑ *SensorBoard.pcb.output_frontsilk.g
☑ SensorBoard.pcb.output_back.gbr
☑ SensorBoard.pcb.output_backmask
☐ SensorBoard.pcb.output_xy.txt (to
☑ SensorBoard.pcb.output_plated-dri

(-69.66, 3179.03) mil Click to select. Right click and drag to zoom.

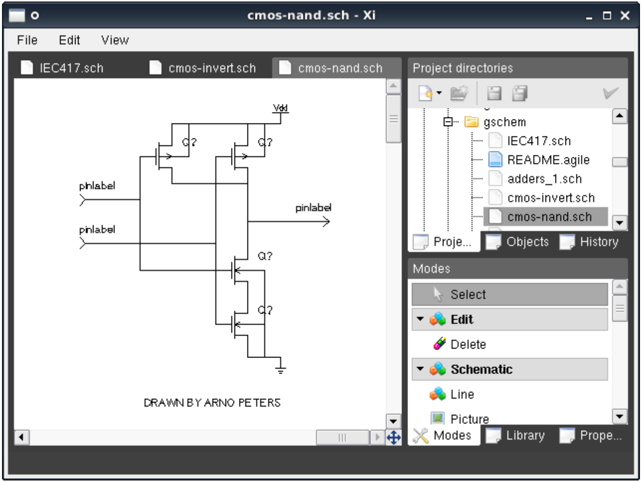
gerbv



- pmos-1.sym
- pmos-2.sym
- pmos-3.sym
- pnp-1.sym
- pnp-2.sym
- pnp-3.sym
- pot-1.sym
- pot-2.sym
- pot-bourns.sym
- pwrxfmr-1.sym
- quad_opamp-1.sym
- resistor-1.sym
- resistor-2.sym
- resistor-variable-1.sym
- resistor-variable-2.sym
- resistorpack10-1.sym
- resistorpack11-1.sym
- resistorpack12-1.sym

Filter:

Default behavior - reference component


cmos-nand.sch - Xi
File
Edit
View
IEC417.sch
cmos-invert.sch
cmos-nand.sch
Vdd
Q?
Q?
pinlabel
pinlabel
pinlabel
Q?
Q?
DRAWN BY ARNO PETERS
Project directories
gschem
IEC417.sch
README.agile
adders_1.sch
cmos-invert.sch
cmos-nand.sch
Proje...
Objects
History
Modes
Select
Edit
Delete
Schematic
Line
Picture
Modes
Library
Prope...

- What *is* scripting?
- What purpose does it fulfil?
- What are the constraints under which it is operating?

Proprietary software

Application is one opaque, immutable blob

Users want to

- chain operations into more complex functionality
- extend the user interface for specific workflows
- extend the user interface with custom more complex functionality

Vendors

- embed a scripting interpreter into the application
- export the applications's functionality as procedures to the scripting interpreter
- invoke scripts on certain actions

Free Software

Users can modify the software package, so no immediate need for scripting

But

- realizing high-level functionality in low-level languages is cumbersome and error-prone
- application's developers as well as users can express high-level "glue code" in a high-level language
- users can use scripting language interactively for working on the file currently loaded into the editor
- users can extend the application without having to re-build the package

Free Software

Q: Why not write the application in a high-level language in the first place?

- low-level functionality is often easier to express and faster in a low-level language

  ⇒ make functionality available as native modules

  ⇒ write the rest in a scripting language using these

- application start-up time

  ⇒ write a native application which embeds a scripting interpreter

Contributing code upstream

Organization

Should users be encouraged to contribute their code upstream?

Responsibility

Who is responsible for user-contributed code?

Programming

Is this code considered part of the application?

Design principles

Design goal

It should be possible to access the "interesting" functionality an application provides without actually having to load up the GUI.

$$\Downarrow$$

Design goal

All relevant gEDA functionality should be available as one or multiple libraries which can be used in other programs without having to run a gEDA application.

Design principles

Design goal

Code written in a low-level language and code written in a high-level language should be able to operate on the same file currently loaded into the editor.

- code can be written in what language makes most sense for the specific problem
- stand-alone scripts can be run from inside an application
- makes interactive console possible

Design principles

Have a clean interface between C code and high-level code

- enforce a strict object model upon which every part of the application can rely
- use a value-oriented data model
- give the storage part an understanding of what you are trying to do so queries can be optimized

libxornstorage

- `xorn_revision_t`
  a given revision of the file contents
- `xorn_object_t`
  the identity of an object across revisions
- `xorn_selection_t`
  the identity of a set of objects
- object data structures

Data structures

```
struct xorn_double2d {
    double x, y;
};

struct xornsch_net {
    struct xorn_double2d pos;
    struct xorn_double2d size;
    int color;
    bool is_bus;
    bool is_pin;
    bool is_inverted;
};
```

Strict object model

What is the bounding box of the object?

- private to the GUI
- can be derived from object data

To which nets is the object connected?

- can be recomputed at any time

Is the object selected?

- only useful in an interactive context
- conceptually isn’t part of the object state

Which attribute texts are attached to this object?

- structural data
- dedicated storage functions to access this

Making fields explicit

gEDA/gaf

- attributes are text objects
- can be freely placed and attached
- meaning of an attribute is up to the user/workflow

PCB

- attributes are a set of pre-defined key-value pairs
- private data of one part of the program
    - not part of the object state
    - keep in a private data structure
- part of the state of the object
    - dedicate an actual field in the data structure to it

What should be part of a revision?

- What data should be part of a revision?
- What data should be managed in a separate revision?
- What data shouldn't be considered in terms of a revision at all?

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <xornstorage.h>

int main()
{
    xorn_revision_t rev;
    struct xornsch_net net_data;
    xorn_object_t net_ob, *objects;
    size_t count;

    rev = xorn_new_revision(NULL);

    memset(&net_data, 0, sizeof net_data);
    net_data.pos.x = 0;
    net_data.pos.y = 200;
    net_data.size.x = 100;
    net_data.size.y = 0;
    net_data.color = 4;
    net_ob = xornsch_add_net(rev, &net_data);

    xorn_finalize_revision(rev);

    xorn_get_objects(rev, &objects, &count);
    printf("%d object(s) found\n", count);
    free(objects);

    xorn_free_revision(rev);
    return 0;
}
```

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

```
$ PYTHONPATH=./xorn/built-packages python2
Python 2.7.9 (default, Jun 29 2016, 13:08:31)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>> import xorn.storage
>>> rev = xorn.storage.Revision()
>>> net_data = xorn.storage.Net(
...     x = 0, y = 200, width = 100, height = 0,
...     color = 4)
>>> net_ob = rev.add_object(net_data)
>>> rev.finalize()
>>> rev.get_objects()
[<xorn.storage.Object object at ...>]
>>>
```

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

```
#!/usr/bin/env python2
import sys
import xorn.storage
import xorn.geda.read

rev = xorn.geda.read.read(sys.argv[1])

for ob in rev.toplevel_objects():
    data = ob.data()
    if isinstance(data, xorn.storage.Text):
        print data.text
```

Current implementation

```
v 20031231 1
L 0 100 150 100 3 0 0 0 -1 -1
L 150 100 75 0 3 0 0 0 -1 -1
L 75 0 0 100 3 0 0 0 -1 -1
T 100 300 5 10 0 0 0 0 1
graphical=1
T 500 600 5 10 0 0 0 0 1
device=none
```

Current implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<symbol xmlns="https://hedmen.org/xorn/schematic/"
        file-format-features="experimental">
  <content>
    <line x0="0" y0="1" x1="1.5" y1="1"/>
    <line x0="1.5" y0="1" x1=".75" y1="0"/>
    <line x0=".75" y0="0" x1="0" y1="1"/>
    <attribute name="graphical" x="1" y="3"
      size="10" visible="no" ...>1</attribute>
    <attribute name="device" x="5" y="6"
      size="10" visible="no" ...>none</attribute>
  </content>
</symbol>
```

Current implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<schematic xmlns="https://hedmen.org/xorn/schematic/"
           file-format-features="experimental">
  <content>
    ...
    <component x="342" y="285" angle="180" symbol="gnd-1"/>
    ...
  </content>
  ...
  <symbol id="gnd-1" name="gnd-1.sym" mode="referenced">
    <content>
      <pin x0="1" y0="3" x1="1" y1="1" inverted="yes">
        <attribute name="pinnumber" x="1.58" y="1.61"
                   size="4" visible="no" show="value">1</attribute>
        <attribute name="pinseq" x="1.58" y="1.61"
                   size="4" visible="no" show="name-value">1</attribute>
        <attribute name="pinlabel" x="1.58" y="1.61"
                   size="4" visible="no" show="value">1</attribute>
        <attribute name="pintype" x="1.58" y="1.61"
                   size="4" visible="no" show="value">pwr</attribute>
      </pin>
      <line x0="0" y0="1" x1="2" y1="1"/>
      <line x0=".55" y0=".5" x1="1.45" y1=".5"/>
      <line x0=".8" y0=".1" x1="1.2" y1=".1"/>
      <attribute name="net" x="3" y=".5" color="detached-attribute"
                 size="10" visible="no" show="name-value">GND:1</attribute>
    </content>
  </symbol>
</schematic>
```

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

```
$ cat > gnet_count.py
def run(f, netlist):
    f.write("%d packages found\n"
            % len(netlist.packages))
    f.write("%d nets found\n"
            % len(netlist.nets))
^D
$ xorn netlist \
    --symbol-library-search=/usr/share/gEDA/sym \
    -L . -g count some-schematic.sch
1 packages found
4 nets found
```

Current implementation

- `libxornstorage`
- `xorn.storage`
- `xorn.geda`
- command-line tool `xorn`
- `gnetlist`
- Guile support

Current implementation

- `xorn.guile` — Python extension which allows adding Guile as a script interpreter to an application
- `xorn.geda.netlist.guile` — Python module which re-creates the old gnetlist API procedures and exports them to Guile
- `gnet_guile.py` — netlist backend which allows using a Scheme backend with the new netlister
- `gnetlist2` — drop-in replacement for `gnetlist`
- `gnetlist` — branches conditionally to `gnetlist2`

Currently in gEDA/gaf

You can use Xorn right now! :)

For example, you could

- write a script which rearranges or renumbers the components in a set of schematics
- write a symbol generator
- write a netlist backend
- edit a gEDA file using a Python interpreter

In the future

Using the new libraries in `gschem`

- running scripts from within the application
- interactive Python console
- removes a lot of code duplication

How about PCB?

Other projects?

- makes integration between projects easier
- common user interface?