Efabless Recommended Open Source Analog Design Flow

Tim Edwards August 8, 2023
with contributions from Harald Pretl and Mitch Bailey

Introduction

Analog design flows can be complex, consisting of much manual design work and iterations with tools, and a much closer connection to the physical design and device models than is typical for a digital design flow. Much analog design is done using commercial tools, but except for the most cutting-edge designs, all of the basic tasks of an analog design flow can be handled by open source tools. Below is a list of open source tools and methods recommended by Efabless for taking an analog design from concept to layout.

The recommended analog design flow consists of these components:

1. Design concept
2. Schematic entry
3. Schematic capture (netlisting)
4. Simulation
5. Layout
6. Device extraction
7. Physical verification (DRC, ERC, and LVS)
8. Parasitic extraction

We have a recommendation of open source tools to use for each of these steps:

1. Design concept: Octave or Python. There are many Python packages available for different kinds of analog design, and the “numpy” and “scipy” packages are generally available and provide much of the same capability as Octave scripting.
2. Schematic entry: Xschem and Xcircuit. Xschem is the most popular and has an interface look and feel similar to commercial tools.
3. Schematic capture: Xschem and Xcircuit. Both tools can generate hierarchical netlists for both LVS and simulation.
4. Simulation: ngspice and Xyce. Ngspice has a well-developed interactive control command interface, and Xyce has a well-developed multiprocessing capability. Both tools can handle Verilog-A models.
5. Layout: magic and klayout. Klayout is closer to commercial tools in look and feel and the representation of layout as objects. Magic has well-developed capabilities for extraction, connectivity checks, wiring, and an interactive DRC checker. Klayout can be scripted in Ruby or Python, while magic can be scripted in Tcl/Tk. Magic can import SPICE netlists and generate all devices and subcells. Klayout and magic both have scripted parameterized device generators.

6. Device and parasitic extraction: Magic
7. DRC: Magic (interactive) and Klayout (noninteractive)
8. LVS: netgen
9. ERC: CVC(RV) Circuit Validity Checker (Reliability Verification)

In addition to these standard design steps, there are some special-purpose tools that may be needed for certain types of design:

1. Field equation solver: Fastercap and FastHenry
2. Mixed-mode (analog and digital) simulation: ngspice (xspice) and Xyce
3. Cosimulation: ngspice (from a specific repository branch) and Xyce

The most complete process design kits (PDKs) for use with the open source tools are the open PDKs:

1. sky130: SkyWater 130nm CMOS with 6 layers of metal
2. GF180MCU: Global Foundries 180nm CMOS with 5 layers of metal

The open PDKs support all of the open source tools enumerated above through the open PDKs installer:

1. open_pdks

There are complete solutions available through docker that contain complete setups of both the open PDKs and the open source tools. The tools and PDKs can also be built and installed locally on any system manually, which is described in the next section. Complete solutions with pre-installed components include the following:

1. volare: This is, in effect, a packaged version of the output of open_pdks. The difference between this and open_pdks is effectively the same as the difference between obtaining pre-compiled software from a package manager, and compiling a piece of software from source.
2. Open Galaxy: This is a platform belonging to Efabless that provides a virtual server with all pre-installed PDKs and tools.
3. IIC OSIC tools: This is a container (docker) solution that provides a virtual server with all pre-installed PDKs and tools.

Compiling and Installing the open-source tools

This section explains how to obtain, compile, and install all of the open source tools mentioned above. In the case of using one of the complete solutions mentioned above, this step is not necessary, as all tools and PDKs will be provided. For complete control over tool and PDK versions and installation locations, compiling the latest version of all tools locally is recommended.

Magic:

The first tool to compile and install is Magic, one of the tools that can be used for layout work, which also can perform device extraction for LVS and simulation, and parasitic extraction for sign-off simulation. It also can read and write LEF and DEF files, read SPICE netlists into (unwired) layout, read and write layout GDS, make plots, and more. It is used by the open_pdks installer to build out parts of the PDK that are not available in the sources, and so needs to be installed and running before the PDKs can be built and installed.

Documentation:

Documentation on Magic can be found at http://opencircuitdesign.com/magic/. This is a front page with a number of button selections in the sidebar which lead to various useful sub-pages. Go to “Download” for instructions on how to download either the git repository sources or the source tarball. “Install” has instructions for compiling and installing Magic. “Using Magic” has a complete command-line command reference, and “Technology Files” has a complete reference to the technology file format. “Documentation” has a number of older published papers about Magic, online translations of the manual pages for Magic, and online translations of the magic tutorials.

There is an additional reference written by Harald Pretl called the “Magic cheat sheet” which has a good summary of command-line commands and key bindings for the beginner user. That document can be found here: https://github.com/iic-jku/osic-multitool/blob/main/magic-cheatsheet/magic_cheatsheet.pdf (part of OSIC-Multitools).

Download:

The best way to obtain the source code for Magic is to download it from github. The main source repository is kept on the server at opencircuitdesign.com; any updates are mirrored to github overnight, so the github version is never more than a day out of date (and is usually up-to-date). Issues should be posted on the github site, and pull requests can be made from github as well. The source repository is: https://github.com/RTimothyEdwards/magic/.

Installation:

Magic should compile on all major OS versions except Windows. On Windows, the use of WSL-2 is recommended, where magic can be compiled and installed according to the same instructions that would be used for an Ubuntu Linux system. There is a file in the source code called “INSTALL_MacOS.md” that should be referred to when compiling on Mac OS.

Magic should be compiled with the Tcl/Tk interpreter enabled. Magic can be compiled without interpreter support for backwards compatibility with very old versions; this is not recommended. For a complete list of prerequisites, see the documentation page listed above and click on “Install” in the sidebar, and read the section “System Requirements”. In summary, on a Linux system, the prerequisites can be installed with

```
apt install m4 tcsh csh libx11-dev tcl-dev tk-dev libcairo2-dev
```

and optionally, for Open-GL support, use

```
apt install mesa-common-dev libglu1-mesa-dev
```

For users of Fedora, CentOS, and similar Linux systems, the equivalent package installer commands are:

```
yum install m4 tcsh csh libX11-devel tcl-devel tk-devel cairo-devel
```

and

```
yum install mesa-libGL mesa-libGLU
```

After installing prerequisites, you need to run from the command line:

```
./configure --enable-cairo-offscreen
make
sudo make install
```

Most users who want to run the OpenGL graphics interface will probably need to enable Cairo graphics for off-screen rendering, which requires the configuration option above, although in some cases it may not be needed. If when running Magic with OpenGL graphics the layer icons on the right-hand side of the window appear black or filled with random bit patterns, then Cairo off-screen rendering is needed.

The default installation puts the executable in the path `/usr/local/bin/magic` and various files needed at run-time go to `/usr/local/lib/magic/`. Note that “magic” itself is a shell script launcher, not an executable.

Usage:

By far, the most common way to run magic is with the command line invocation

```
magic -d <type> -rcfile <path> [<cellname>]
```

Where `<type>` is either `OGL` or `XR` for OpenGL or Cairo graphics, respectively. The startup script file `<path>` is determined by the PDK (see the section on PDK installation, below). Generally, the name of a cell (which has a database file in the `.mag` format) is given on the command line without the file extension. However, other file types such as GDS files, DEF files, or Tcl scripts can also be passed on the command line, with the file extension included.

The other common way to run magic is in batch mode, usually as part of a script, either taking commands from standard input or from a Tcl script file:

```
magic -dnull -noconsole -rcfile <path> << EOF
(commands…)
EOF
```

or

```
magic -dnull -noconsole -rcfile <path> <script>
```

Specific usage of Magic for layout, extraction, DRC, and other purposes will be explained in a later section.

In this document, it will be assumed that the PDK is installed using the default options including the default installation location. That will put the PDK files in the path `/usr/local/share/pdk/` followed by the specific PDK name. Given these defaults, the command line to start magic is

```
magic -d XR -rcfile \
/usr/local/share/pdk/sky130A/libs.tech/magic/sky130A.magicrc
```

for the Sky130 technology, and

```
magic -d XR -rcfile \
/usr/local/share/pdk/gf180mcuD/libs.tech/magic/gf180mcuD.magicrc
```

for the GF180MCU technology.

The “rcfile” startup script is responsible for loading the correct technology file into magic, loading the parameterized device generator scripts, and ensuring that any setup commands required for the technology have been applied.

Instead of writing the long command line, the startup script file can be copied to any working directory where Magic is being used and renamed “`.magicrc`”, or a symbolic link called “`.magicrc`” can be made that points back to the startup script in the PDK. If that is done, then magic can be started with just

```
magic -d XR
```

and the correct startup script will be applied.

XSchem:

Xschem is a schematic editor and schematic capture tool with the general look and feel of commercial tools. It supports generation of SPICE netlists, and has integrated support of simulation with node tracing and graphing of results.

Documentation:

For documentation on xschem, refer to the SourceForge page at this URL:
https://xschem.sourceforge.io/stefan/index.html

Download:

Xschem can be downloaded by cloning the repository from this URL:
https://github.com/stefanschippers/xschem.git

Installation:

Prerequisites include most of the same prerequisites which should have already been installed for use with Magic (Tcl/Tk and X11 and Cairo graphics) In addition, libXpm is required:

```
sudo apt install libxpm-dev
```

The xschem installation is a straightforward standard build with:

```
./configure [--prefix=<root_install_path>]
make
sudo make install
```

Usage:

Use of xschem with the open PDKs generally requires copying the startup Tcl script file from the PDK:

```
/usr/local/share/pdk/sky130A/libs.tech/xschem/xschemrc
```

to the working directory where xschem is being run (or making a symbolic link).

Netgen:

Netgen is a full-featured LVS tool which can compare two netlists, either SPICE or verilog, including combinations of the two. It is built on a Tcl interpreter and has a command-line interface. It handles hierarchy, parallel and series device networks, property comparison, and pin comparison.

Documentation:

Documentation on Netgen can be found at http://opencircuitdesign.com/netgen/. This is a front page with a number of button selections in the sidebar which lead to various useful sub-pages. Go to “Download” for instructions on how to download either the git repository sources or the source tarball. “Install” has instructions for compiling and installing Netgen. “Reference” has information on using Netgen and a complete command-line command reference.

Download:

The best way to obtain the source code for Netgen is to download it from github. The main source repository is kept on the server at opencircuitdesign.com; any updates are mirrored to github overnight, so the github version is never more than a day out of date (and is usually up-to-date). Issues should be posted on the github site, and pull requests can be made from github as well. The source repository is: https://github.com/RTimothyEdwards/netgen/.

Installation:

Netgen should compile on all major OS versions except Windows. On Windows, the use of WSL-2 is recommended, where Netgen can be compiled and installed according to the same instructions that would be used for an Ubuntu Linux system.

Netgen should be compiled with the Tcl/Tk interpreter enabled. Netgen can be compiled without interpreter support for backwards compatibility with very old versions; this is not recommended. For a complete list of prerequisites, see the documentation page listed above and click on “Install” in the sidebar, and read the section “System Requirements”. In summary, on a Linux system, the prerequisites can be installed with

```
apt install tcl-dev tk-dev
```

or

```
yum install tcl-devel tk-devel
```

Since these are also requirements for Magic, they should already be installed. There are generally no non-default options needed for configuration, so installing netgen is simply

```
./configure
make
sudo make install
```

Usage:

There is a GUI version of Netgen that can be invoked with "`netgen -gui`". However, it is more common to run Netgen in batch mode. If the netlists are both simple single files, then the easiest way to run netgen is with the "lvs" command on the command line:

```
netgen -batch lvs <lvs_options>
```

If the netlists require special handling for read-in, such as reading multiple files from different sources, or special setup, then it works best to put all the commands into a script file and source the script file from the command line:

```
netgen -batch source <script_file>
```

The script file will then invoke the "lvs" command at the end of the script.

The arguments passed to the "lvs" command for a simple comparison of two netlist files are:

```
lvs "<netlist1> <cell1>" "<netlist2> <cell2>" <setup_file> <output_file>
```

Where the `<setup_file>` comes from the PDK and the `<output_file>` is an arbitrary name for the output log (default "comp.out"). For the Sky130 PDK, the setup file is:

```
/usr/local/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl
```

And for the GF180MCU PDK, the setup file is:

```
/usr/local/share/pdk/gf180mcuD/libs.tech/netgen/gf180mcuD_setup.tcl
```

The setup file will be found automatically if it is copied to the local working directory where netgen is being run and renamed to "`setup.tcl`" (or a symbolic link can be made to the file in the PDK).

Ngspice:

Ngspice is an analog simulation tool based on Berkeley SPICE 3 but with much additional development to keep its capabilities in line with commercial SPICE simulators. Ngspice implements all of the main analysis types such as operating point, DC, AC, and noise analysis, and has its own command interpreter to create control blocks in a testbench with additional capabilities for loops, conditionals, measurement, and more. It has a built-in event-driven digital simulator called xspice which is capable of co-simulation of digital components.

Documentation:

The ngspice documentation can be found at the website on SourceForge at the URL:

https://ngspice.sourceforge.io/

Download:

See:

https://ngspice.sourceforge.io/download.html

for download instructions. SourceForge has its own git server, for which the download command is

```
git clone git://git.code.sf.net/p/ngspice/ngspice
```

Installation:

Follow the instructions in the “INSTALL” file in the repository top level directory for a “fully-featured” ngspice:

```
./autogen.sh –adms
mkdir release
cd release
../configure --with-x --enable-xspice --disable-debug --enable-cider \
--with-readline=yes --enable-openmp --enable-adms --enable-osdi
make 2>&1 | tee make.log
sudo make install
```

Usage:

Specifically for the Sky130 PDK, ngspice will take a long time to start up and will not interpret fingered devices correctly unless there is a specific startup file in the working directory where ngspice is invoked. To ensure that ngspice works correctly, copy the file

```
/usr/local/share/pdk/sky130A/libs.tech/ngspice/spinit
```

to the current working directory and rename the file to “`.spiceinit`”. The GF180MCU PDK does not need any special setup for ngspice.

Klayout:

Klayout is a useful layout viewer and editor that has a look and feel that is more like commercial tools than is Magic; klayout uses an object-based database and reads and writes directly from and to GDS format. It supports reading and writing of OASIS format. It can run DRC and LVS batch jobs.

Documentation:

https://www.klayout.de/doc.html

Download:

See information on this web page:

https://www.klayout.de/build.html

The page links to a number of packaged versions of klayout for all major OS versions including Windows and Mac OS. If you want to compile the most recent version from source, then download the git repository version from this URL:

https://github.com/KLayout/klayout

and install using the instructions from the web page above. The main package prerequisites required for compiling klayout from source are the ruby development package and the Qt development package.

Usage:

Klayout maintains a list of accessed locations in the user's home directory under `.klayout/`, so it is only necessary to set up the technology for each PDK once.

To set up each PDK for klayout, start klayout, which starts with a mainly blank window. From the menu, select "Tools → Manage Technologies". The pop-up window has a list of technologies on the left that contains only "(Default)", and a blank window on the right with a message "Choose a category". With the mouse pointer in the left window, right-click to get a pop-up menu, and click on "Import Technology". Then navigate to the following:

```
/usr/local/share/pdk/sky130A/libs.tech/klayout/tech/
```

and import the file "`sky130A.lyt`", for the Sky130 technology; and

```
/usr/local/share/pdk/gf180mcuD/libs.tech/klayout/tech/
```

and import the file "`gf180mcu.lyt`" for the GF180MCU technology.

Subsequently, when running klayout, any of the imported technologies can be selected by going to the technology icon in the icon bar at the top (The “T” inside a circle), clicking on the down arrow to get the drop-down menu, and choosing the technology to use.

Xyce:

Xyce is a SPICE simulator developed by Sandia National Labs, and was developed primarily to enable fast simulation through multiprocessing. It supports all of the main analysis types, supports some accelerated digital simulation with digital components (which is not compatible with xspice), and supports co-simulation with iverilog.

Documentation:

Documentation for Xyce can be found at Sandia National Labs at the following public URL:

```
https://xyce.sandia.gov/documentation-tutorials/
```

Download:

Xyce has a development repository on github and can be downloaded with:

```
git clone https://github.com/Xyce/Xyce
```

Installation:

Xyce compile and install is a complicated process requiring a number of packages to be installed, and a custom library called Trilinos needs to be compiled first.

```
git clone https://github.com/trilinos/Trilinos
```

For CentOS/Fedora systems, the prerequisites are:

```
sudo yum install blas-devel lapack-devel suitesparse-devel openmpi-devel
```

I had difficulty with the “mpic” compiler from openmpi not being found in standard locations and had to do the following:

```
cd /usr/bin
sudo ln -s /usr/lib64/openmpi/bin/mpicc
sudo ln -s /usr/lib64/openmpi/bin/mpic++
sudo ln -s /usr/lib64/openmpi/bin/mpif77
sudo ln -s /usr/lib64/openmpi/bin/mpicxx
```

For the Trilinos package, do:

```
cd Trilinos
mkdir build
cd build
```

Go to the web page https://xyce.sandia.gov/documentation-tutorials/building-guide/ and find the section "**Example "reconfigure" script for Parallel Trilinos:**" and copy the text into a file called "`reconfigure`". Edit the file and modify "`SRCDIR`" to point to the top-level directory of the Trilinos source. Modify "`ARCHDIR`" to point to `XyceLibs/Parallel` in the same location as Trilinos, Xyce, and other packages being built for the analog flow. Then do

```
chmod a+x reconfigure
./reconfigure
make
make install
```

This should be sufficient for compiling Trilinos. For Xyce,

```
cd Xyce
./bootstrap
mkdir build
cd build
../configure CXXFLAGS="-g" ARCHDIR="../../XyceLibs/Parallel" \
CPPFLAGS="-I/usr/include/suitesparse" --enable-mpi CXX=mpicxx \
CC=mpicc F77=mpif77 --enable-stokhos --enable-amesos2
make -j 16
sudo make install
```

The specific configuration line differs between OS variants; see the same web page mentioned above and go to the section "**Parallel Builds**" for information about OS-specific configuration lines.

Usage:

Xyce normally runs in batch mode like other variants of SPICE. There are hooks from the xschem schematic entry tool that can set up and run Xyce automatically; see the xschem documentation for details.

CVC:

Documentation:

CVC(RV) [Circuit Validity Check (Reliability Verification)] runs static, voltage aware ERC checks. (Be warned - there are at least 2 other, unrelated EDA programs also called CVC.)

Download:

```
git clone https://github.com/d-m-bailey/cvc.git
```

Installation:

```
autoreconf -vif
./configure --disable-nls
make
sudo make install
```

Usage:

CVC takes a single spice file (no includes) as input and does not handle many of directives available in spice (global signals, subcircuit parameterizations, connect statements, etc). For this reason, the recommended input for open source designs is the extracted layout. CVC also requires a device model and a power definition file, both in formats unique to CVC.
The base files for the sky130A/B and gf180mcuC process user_project_wrapper and user_analog_project_wrapper are available at https://github.com/d-m-bailey/extra_be_checks.git.

```
git clone https://github.com/d-m-bailey/extra_be_checks.git
```

The `extra_be_checks` directory has a `run_cvc` command that will extract the layout and use the results for CVC. Copy the appropriate `lvs_config.*.json` file from `extra_be_checks/tech/<technology>` directory and modify it for your project.

```
LVS_ROOT=<path>/extra_be_checks
$LVS_ROOT/run_cvc lvs_config.json
```

A summary of the results are displayed and the details can be found in the cvc.log and cvc.error.gz files located in `WORK_ROOT` (default location is <design> directory).

Building and installing the open PDKs:

The next important step is to obtain, build, and install the open PDKs. As of this writing, there are two open PDKs that are supported by the open_pdks installer. These are “sky130” and “gf180mcu”. Of all the open source tools, only Magic is required to be installed prior to running the PDK installer, which makes use of Magic for a number of tasks.

Each open PDK is a collection of libraries in open repositories containing files in standard open formats like GDS, SPICE, LEF/DEF, liberty, and verilog. These file formats are sufficient to completely describe most process libraries like standard cells, I/O cells, and primitive devices. There are additional components like DRC and extraction rule decks, tables of parasitic capacitance, and parameterized cell generators that do not have standard open formats. Some of this information is in the open PDK primary sources, and some in the open PDK documentation. The open_pdks installer is responsible for creating a common structure around all libraries and files; ensuring that the resulting files are compatible with selected open source tools, and providing additional files for open source tools when needed (such as DRC rules converted from documentation to a format understood by magic or klayout).

Processes are often defined by a foundry as having multiple *variants* which all belong to the same basic process but may involve different sets of process options; each variant corresponds to a unique set of masks used during fabrication. Typically, the variants will be different numbers of metal layers available for routing, or a MiM capacitor option, or similar option. Foundries have their own unique naming schemes for identifying a process variant. To maintain some consistency across foundry processes, open_pdks enumerates its own process variants by adding a suffix such as “A”, “B”, “C”, etc., to the process name. The suffix string is arbitrary, but each unique string corresponds to an entire PDK to be installed. Because most process options do not affect all files (*i.e.,* an extra top-level metal layer will have little or no impact on a standard cell library), open_pdks optimizes the installed PDK by making symbolic links from files in one process variant to another if the contents are the same. That way, many process variants can be created without greatly increasing the disk space needed to hold the files, but with each process variant acting like its own self-contained PDK.
For example, the Sky130 PDK has two variants “A” and “B” defined by open_pdks. Variant “Sky130A” is the standard process, and variant “Sky130B” is a variant that supports the manufacture of ReRAM devices. The GF180MCU process has four variants “A”, “B”, “C”, and “D”. Variant “GF180MCUA” has three metal layers; variant “GF180MCUB” has four metal layers; variant “GF180MCUC” has five metal layers, and variant “GF180MCUD” has five metal layers with a thicker top metal. Where examples are given in this document, the variants “Sky130A” and “GF180MCUD” will be used unless otherwise noted.

The tool open_pdks is largely a set of scripts and Makefiles that builds out and populates a common structure for a PDK that can be recognized by the open source tools. The two existing open PDKs, “sky130” and “gf180mcu”, are known to open_pdks. Additional PDKs (open or otherwise) can be added to open_pdks as additional subdirectories, using the same Makefile format.

Documentation:

Documentation for open_pdks can be found at http://opencircuitdesign.com/open_pdks/. The primary documentation can be found by clicking on the “Install” button in the sidebar. The “Reference” button takes you to a page with additional information on the file structure generated by the PDK installation, and the usage of the Python scripts that make up the main part of the build and install process.

Download:

The best way to obtain the source code for open_pdks is to download it from github. The main source repository is kept on the server at opencircuitdesign.com; any updates are mirrored to github overnight, so the github version is never more than a day out of date (and is usually up-to-date). Issues should be posted on the github site, and pull requests can be made from github as well. The source repository is: https://github.com/RTimothyEdwards/open_pdks/.

Installation:

Instructions on the open_pdks website are the best detailed resource for installation. This document explains the most common install setups commonly needed.

The open_pdks build process is a multi-step process:

1. The user runs the “configure” script to declare which PDK(s) to install, and can select specific libraries to ignore or extra libraries to install. The “configure” process builds the Makefile that is used in the next step.
2. Next, the user runs “make”. The first step of the “make” process is to clone the necessary source repositories into the open_pdks “sources/” subdirectory.
3. The next step of “make” is to completely build out the PDK for each defined process variant, which is done in a staging area in open_pdks in a subdirectory named for the process variant (*e.g.*, “sky130/sky130A/”).
4. Finally, the user runs “make install”. The files are copied from the staging directory into the final install target directory, creating symbolic links where possible to reduce disk space.
5. After the build and install, the user can run “make clean” to remove the contents of the staging directory, “make veryclean” to additionally remove the log files generated during the build and install processes, and “make distclean” to additionally remove the cloned repository sources.

It is recommended to always install and clean up the staging and source directories to save disk space, which can be considerable for a PDK. It is also recommended to keep the install target in a system-wide available directory which does not have write access for typical users. The PDK contents are not intended to be altered, and keeping them in a read-only filesystem area prevents accidental modifications to the PDK files. It is possible to use a PDK from the staging area prior to running “make install”. This is recommended only to test the PDK; the PDK should not normally be used from the staging area.

A typical build of the Sky130 PDK looks like this:

```
./configure --enable-sky130-pdk --enable-sram-sky130
make
sudo make install
make distclean
```

A typical build of the GF180MCU PDK looks like this:

```
./configure --enable-gf180mcu-pdk --enable-osu-sc-gf180mcu
make
sudo make install
make distclean
```

While it is possible to build PDKs for multiple foundry processes (*e.g.*, Sky130 and GF180MCU) in a single step, this is not recommended due to the large amount of disk space used by each PDK and the amount of redundant data generated during the cloning and staging steps. Each process can be built and installed separately.

Please note: The build process is multithreaded, and any error that occurs is usually lost in the middle of the terminal output. The terminal output is saved as a log file in the PDK directory (*e.g.*, `sky130/sky130A_make.log`). If an error occurs, then the Makefile will terminate with error status 2. Review the log file; since most of the Make process is done with python scripts, errors will usually have a python “Traceback” message where the error occurred.

Please also note: The build process generates a large amount of output, much of which contains warnings and occasional errors. These messages are produced by the tools being invoked by open_pdks and are not under the control of open_pdks. If the Make process does not terminate with an error status, then any error messages in the terminal output are non-fatal errors and can be ignored.

Designing in a simple Analog Flow

The “simple analog flow” assumes that the entire design is analog. The analog circuit is designed as a schematic, a layout is created to match the schematic, and the final layout is exported to GDS.

Filesystem Setup

There is no single mandatory way to set up a filesystem for analog design. Generally, though, a guiding principle should be that any component of a design that could be valuable for re-use should exist in its own self-contained area. Otherwise, the recommendation is that the filesystem should follow the following structure:

```
<top-level project directory>
     xschem/  magic/  lef/  gds/  netlist/  macros/
```

This locates most related files in a common area: Schematic and symbol files in `xschem`, Magic database files in `magic`, abstract layout views in `lef`, final layout in `gds`, and SPICE netlists in `netlist`. Where subcomponents of the design could be independent IP, the `macros` directory can be populated with symbolic links to other projects. The use of symbolic links makes the subcomponents more portable.

Ideally, the project itself should be a git repository, to provide a robust method of version tracking. If made into a git repository, it is recommended that the following file types be excluded from versioning by listing them in a `.gitignore` file, because they are either intermediate results or reproducible on demand: `*.ext` (magic intermediate extraction files), `*.raw` (SPICE raw files). `macros/` should be added to `.gitignore` since they should be their own repositories.

Schematic Entry

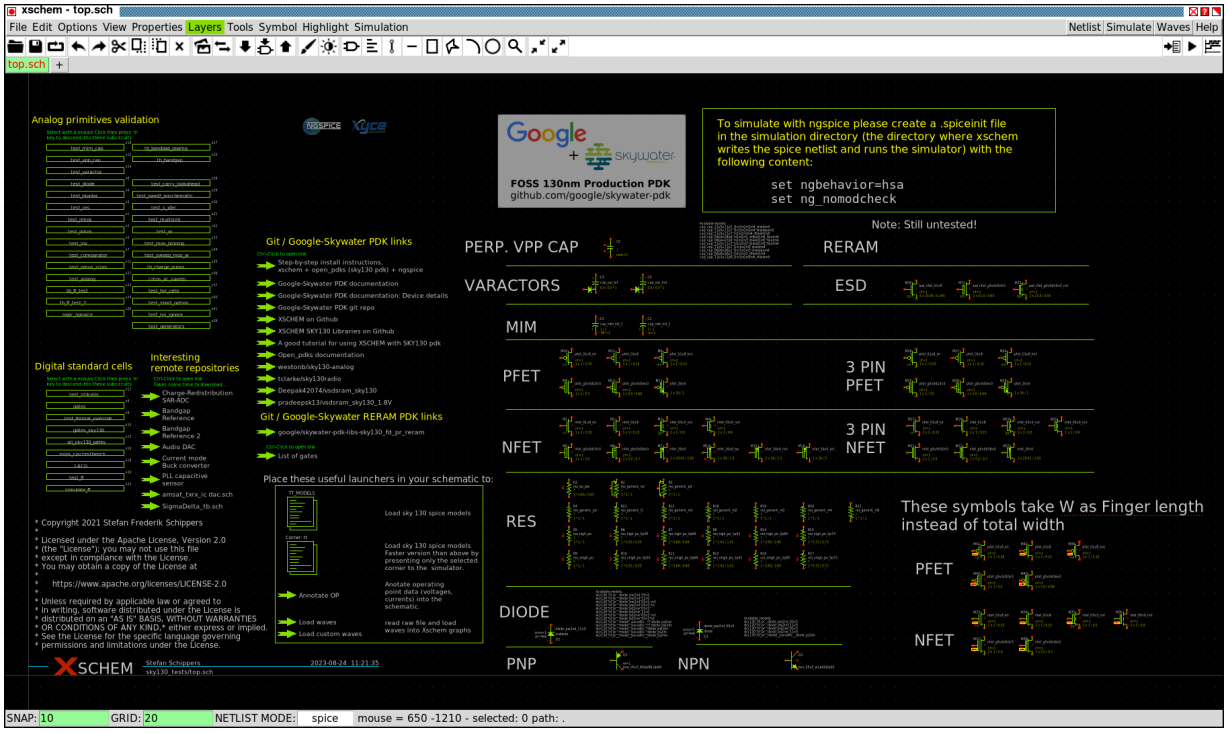
Schematics are drawn using xschem. To set up the xschem environment for use, starting from the project top-level directory, do (using the Sky130 process as an example):

```
cd xschem
ln -s /usr/local/share/pdk/sky130A/libs.tech/xschem/xschemrc
```

Then you can run, without arguments,

```
xschem
```

and you should get the default window with useful information about xschem and the PDK, with examples, that looks like the following:



These videos are good introductions to using xschem for schematic entry:

> https://www.youtube.com/watch?v=q3ZcpSkVVuc from Efabless (courtesy of Matt Venn and featuring xschem developer Stefan Schippers)
> https://www.youtube.com/watch?v=BpPP2hE_eK8 from Brad Minch

The default page of xschem is also a good place to find tutorials. On the left side is a set of buttons that link to different tutorial schematics; select any of them and type the key `e` to access the tutorial page.

It is highly recommended to work hierarchically when possible, and to contain any part of the circuit that is meant to become layout in its own self-contained schematic, to create a separate testbench schematic in which the schematic intended for layout is instantiated, and to keep all non-layout components in the testbench only (zero-volt sources or zero-ohm ideal resistors used for probing voltage or current are acceptable, as they are recognized and handled appropriately by Magic and Netgen).

Schematic Capture

Generate netlists from a testbench for simulation. Generate netlists from the schematic intended for layout for importing into layout. When generating a netlist for layout import, be sure to select menu item "Simulation" → "LVS netlist: Top level is a .subckt". This puts the ".subckt" … ".end" block around the subcircuit and adds pins, which the import routine will handle.

Netlists are generated from the "Netlist" button at the top of the xschem window. By default, any netlists will be placed in the user home directory under:

```
~/.xschem/simulations/
```

This behavior can be modified by editing the `xschemrc` file and changing the value of "netlist_dir". Modifying the `xschemrc` file requires that the file be a copy of the PDK file, not a symbolic link. For this document, the location will be assumed to be modified to the `netlist/` subdirectory of the project directory, as described in "Filesystem Setup" above.

It is also possible to run xschem in batch mode to generate a netlist using the command line as:

```
xschem -n -x -s -q -o ../netlist <top_level_schematic>
```

The `-n` option generates the netlist, while `-s` specifies SPICE format and `-o` is followed by the path to the location of the output netlist file. The other options are appropriate for batch mode operation. See the output of "`xschem --help`" for command-line option details.

Simulation from schematic netlist

It is necessary to generate a netlist of the full testbench prior to running simulation. Once the netlist has been generated, the simulation can be run using the "Simulation" button at the top of the xschem window. Of course the simulation can be run from the terminal with a command line as well.

For use with the open PDKs, each schematic should have "code" or "code_shown" blocks on the testbench. The traditional use case is to use a "code" block to declare the library; this code block can be named for the library corner, such as "TT_MODELS", and the value content should contain the line

```
.lib $::SKYWATER_MODELS/sky130.lib.spice tt
```

The "`SKYWATER_MODELS`" variable name is defined in the `xschemrc` file and is understood by Xschem, which makes the appropriate variable substitution when writing the output netlist file. The "code_shown" block should have the testbench analysis; for ngspice, that would be the `.control … .endc` block.

Because the "code" blocks write text verbatim into the output, the netlist can be made compatible with any SPICE simulator, so it is easy to generate output for Xyce as well as for ngspice. To choose which tool to use to run the simulation, select the menu item "Simulation→ Configure simulators and tools".

Schematic import into layout

To set up the layout environment, do (using the Sky130 process as an example):

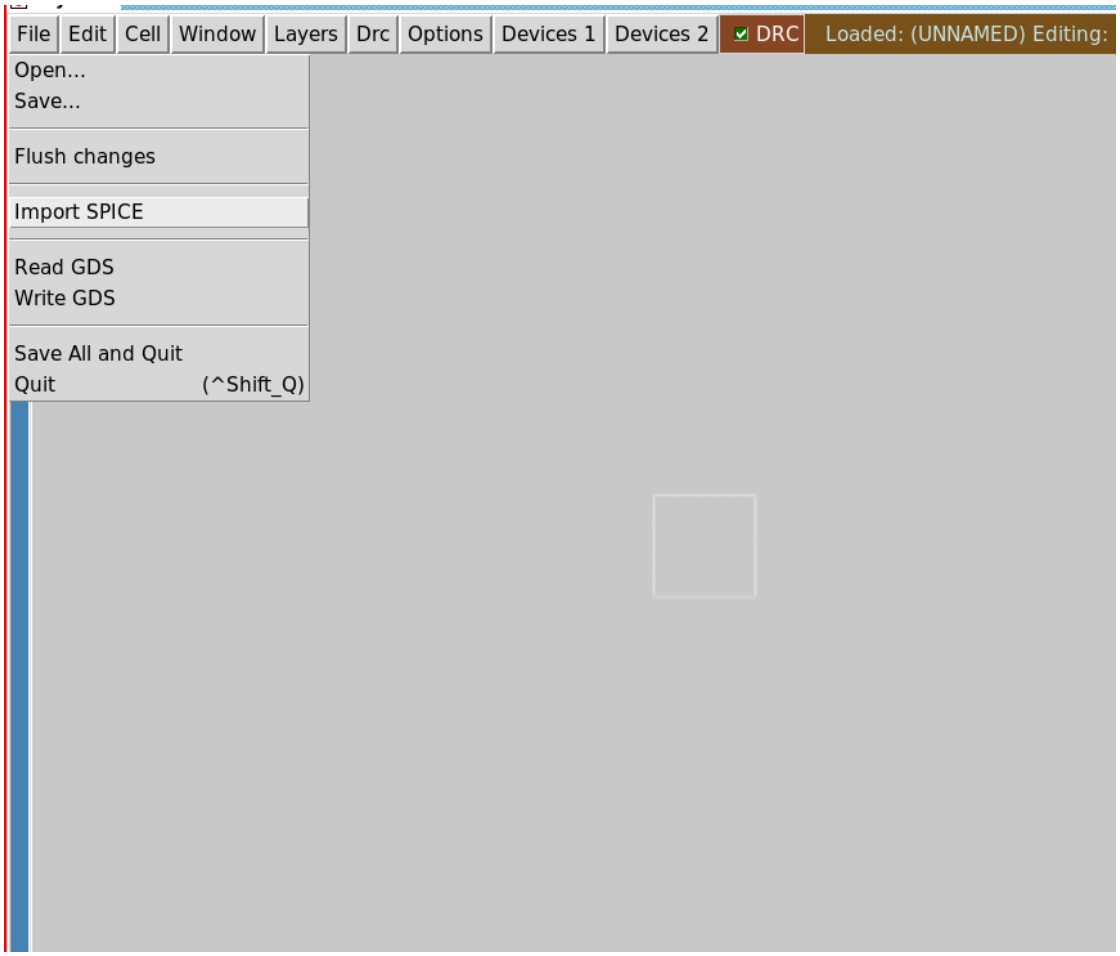
```
cd magic
```

```
ln -s /usr/local/share/pdk/sky130A/libs.tech/magic/sky130A.magicrc .magicrc
```

Run magic from this directory:

```
magic -dXR
```

You should get a layout window and a text window. From the layout window, select the menu item "File" followed by "Import SPICE". This action generates a pop-up window where you can navigate to the directory with the schematic netlist and import the SPICE netlist (the netlist of the subcircuit to become layout, not the testbench).



Layout

DRC

ERC

Device extraction and LVS

To set up the LVS environment, do (using the Sky130 process as an example):

```
cd netlist
ln -s /usr/local/share/pdk/sky130A/libs.tech/netgen/sky130A_setup.tcl setup.tcl
```

Parasitic extraction

Simulation from parasitic extracted layout netlist

Generation of GDS files

Integration of an analog design into the Efabless Caravan chip

Appendix 1: Quick Default Installation (Ubuntu and Debian Linux)

This is a summary of the installation of the set of tools outlined above that should work in most cases. This procedure will put all of the tool repository sources into a directory under the user's home directory called "gits". Everything in the instructions is executed by commands on the Linux command line in a terminal. Comment lines start with "echo" so that they can be cut and pasted into the terminal without causing errors.

```
cd ~
echo Create common directory for tool sources
mkdir -p gits
cd ~/gits
echo Install prerequisite packages (assumes Ubuntu-like OS)
sudo apt install python3 build-essential gcc
sudo apt install m4 tcsh csh libx11-dev tcl-dev tk-dev libcairo2-dev
sudo apt install mesa-common-dev libglu1-mesa-dev
echo Get the Magic VLSI layout editor
git clone https://github.com/RTimothyEdwards/magic
cd magic
./configure --enable-cairo-offscreen
make
sudo make install
make clean
cd ~/gits
echo Get the open_pdks installer and build Sky130 and GF180MCU
git clone https://github.com/RTimothyEdwards/open_pdks
cd open_pdks
./configure --enable-sky130-pdk --enable-sram-sky130
make
sudo make install
make veryclean
./configure --enable-gf180mcu-pdk --enable-osu-sc-gf180mcu
make
sudo make install
make veryclean
make distclean
cd ~/gits
echo Get the netgen LVS tool
git clone https://github.com/RTimothyEdwards/netgen
cd netgen
./configure
make
sudo make install
make clean
cd ~/gits
echo Get the xschem schematic editor
```

```
git clone https://github.com/stefanschippers/xschem.git
cd xschem
./configure
make
sudo make install
make clean
cd ~/gits
echo Get the ngspice circuit simulator
git clone git://git.code.sf.net/p/ngspice/ngspice
cd ngspice
./autogen.sh –adms
mkdir release
cd release
../configure --with-x --enable-xspice --disable-debug --enable-cider \
--with-readline=yes --enable-openmp --enable-adms --enable-osdi
make 2>&1 | tee make.log
sudo make install
cd ~gits
echo Get the KLayout layout editor
sudo apt install qtcreator qtbase5-dev qt5-qmake
sudo apt install qtmultimedia5-dev  libqt5xmlpatterns5-dev
sudo apt install libqt5svg5-dev qttools5-dev-tools qttools5-dev
sudo apt install python3-dev
git clone https://github.com/KLayout/klayout
cd klayout
./build.sh
echo Put Klayout in a normal system executable path
sudo mv bin-release /usr/local/share/klayout
sudo ln -s /usr/local/share/klayout/klayout /usr/local/bin/klayout
sudo cat > /etc/ld.so.conf.d/klayout-x86_64.conf << EOF
/usr/local/share/klayout
EOF
sudo ldconfig
```

Appendix 2: Quick Default Installation (CentOS and Fedora Linux)

This is a summary of the installation of the set of tools outlined above that should work in most cases. This procedure will put all of the tool repository sources into a directory under the user's home directory called "gits". Everything in the instructions is executed by commands on the Linux command line in a terminal. Comment lines start with "echo" so that they can be cut and pasted into the terminal without causing errors.

```
cd ~
echo Create common directory for tool sources
mkdir -p gits
cd ~/gits
echo Install prerequisite packages (assumes CentOS-like OS)
sudo yum install python3 build-essential gcc
sudo yum install m4 tcsh csh libX11-devel tcl-devel tk-devel cairo-devel
sudo yum install mesa-libGL-devel mesa-libGLU-devel
echo Get the Magic VLSI layout editor
git clone https://github.com/RTimothyEdwards/magic
cd magic
./configure --enable-cairo-offscreen
make
sudo make install
make clean
cd ~/gits
echo Get the open_pdks installer and build Sky130 and GF180MCU
git clone https://github.com/RTimothyEdwards/open_pdks
cd open_pdks
./configure --enable-sky130-pdk --enable-sram-sky130
make
sudo make install
make veryclean
./configure --enable-gf180mcu-pdk --enable-osu-sc-gf180mcu
make
sudo make install
make veryclean
make distclean
cd ~/gits
echo Get the netgen LVS tool
git clone https://github.com/RTimothyEdwards/netgen
cd netgen
./configure
make
sudo make install
make clean
cd ~/gits
echo Get the xschem schematic editor
```

```
git clone https://github.com/stefanschippers/xschem.git
cd xschem
./configure
make
sudo make install
make clean
cd ~/gits
echo Get the ngspice circuit simulator
git clone git://git.code.sf.net/p/ngspice/ngspice
cd ngspice
./autogen.sh –adms
mkdir release
cd release
../configure --with-x --enable-xspice --disable-debug --enable-cider \
--with-readline=yes --enable-openmp --enable-adms --enable-osdi
make 2>&1 | tee make.log
sudo make install
cd ~/gits
echo Get the KLayout layout editor
sudo yum install qt-creator qt-devel
git clone https://github.com/KLayout/klayout
./build.sh
echo Put Klayout in a normal system executable path
sudo mv bin-release /usr/local/share/klayout
sudo ln -s /usr/local/share/klayout/klayout /usr/local/bin/klayout
sudo cat > /etc/ld.so.conf.d/klayout-x86_64.conf << EOF
/usr/local/share/klayout
EOF
sudo ldconfig
```

Appendix 3: Quick Default Installation (Mac OS)

This is a summary of the installation of the set of tools outlined above. This procedure will put all of the tool repository sources into a directory under the user's home directory called "gits". Everything in the instructions is executed by commands on the Mac OS command line in a terminal. Comment lines start with "echo" so that they can be cut and pasted into the terminal without causing errors. These instructions have been vetted on Mac OS Big Sur.

```
cd ~
echo Create common directory for tool sources
mkdir -p gits
cd gits
echo
echo Install homebrew
echo
echo Note:  The command below takes a few minutes to install stuff
/bin/bash -c "$(curl -fsSL \
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
echo
echo Compile and install X11-based Tcl/Tk for magic and xschem
echo
brew install wget coreutils
wget https://prdownloads.sourceforge.net/tcl/tcl8.6.13-src.tar.gz
tar xfz tcl8.6.13-src.tar.gz
rm tcl8.6.13-src.tar.gz
cd tcl8.6.13/unix
echo Using opt2 in case homebrew tcl/tk was installed
./configure --prefix=/usr/local/opt2/tcl-tk
make -j$(nproc)
sudo make install
make clean
cd ~/gits
brew install libx11
wget https://prdownloads.sourceforge.net/tcl/tk8.6.13-src.tar.gz
tar xfz tk8.6.13-src.tar.gz
rm tk8.6.13-src.tar.gz
cd tk8.6.13/unix
./configure --prefix=/usr/local/opt2/tcl-tk \
--with-tcl=/usr/local/opt2/tcl-tk/lib --with-x \
--x-includes=/usr/local/include/X11 --x-libraries=/usr/local/lib
sed -i.bak 's/(libdir):.*$/(libdir)/g' Makefile
make -j$(nproc)
sudo make install
make clean
cd ~/gits
```

```
echo
echo Install prerequisite packages for Mac OS using homebrew
echo
brew install cairo
brew install libglu freeglut
brew install --cask xquartz
echo Run the X11 server
open -a XQuartz
echo Allow X11 permission (may or may not be necessary)
xhost + localhost
echo Set the DISPLAY environment variable
export DISPLAY=:0
echo
echo Get the Magic VLSI layout editor
echo
git clone https://github.com/RTimothyEdwards/magic
cd magic
./configure --with-tcl=/usr/local/opt2/tcl-tk/lib \
--with-tk=/usr/local/opt2/tcl-tk/lib \
--with-cairo=$(brew --prefix cairo)/include \
--x-includes=/usr/local/include/X11 --x-libraries=/usr/local/lib \
--enable-cairo-offscreen
make -j$(nproc)
sudo make install
make clean
cd ~/gits
echo
echo Get the open_pdks installer and build Sky130 and GF180MCU
echo
git clone https://github.com/RTimothyEdwards/open_pdks
cd open_pdks
./configure --enable-sky130-pdk --enable-sram-sky130
make
sudo make install
make veryclean
./configure --enable-gf180mcu-pdk --enable-osu-sc-gf180mcu
make
sudo make install
make veryclean
make distclean
cd ..
echo
echo Get the netgen LVS tool
echo
git clone https://github.com/RTimothyEdwards/netgen
```

```
cd netgen
./configure --with-tcl=/usr/local/opt2/tcl-tk/lib \
--with-tk=/usr/local/opt2/tcl-tk/lib
make
sudo make install
make clean
cd ~/gits
echo
echo Get prerequisites needed for compiling xschem
echo
brew install libxaw libxpm jpeg libxcb
echo
echo Get the xschem schematic editor
echo
git clone https://github.com/stefanschippers/xschem.git
cd xschem
./configure
echo Modify Makefile.conf for the X11-based Tcl/Tk
sed -i.bak1 "/CFLAGS/s#-O2#-O2 -I/usr/local/opt2/tcl-tk/include#" Makefile.conf
sed -i.bak2 "/LDFLAGS/s#-lm#-lm -L/usr/local/opt2/tcl-tk/lib#" Makefile.conf
sed -i.bak3 "/LDFLAGS/s#8.5#8.6#g" Makefile.conf
make
sudo make install
make clean
cd ~/gits
echo
echo Get ngspice from Homebrew
echo
echo brew install ngspice
echo
echo Compile the ngspice circuit simulator from source
echo
brew install automake autoconf libtool
brew install bison
git clone git://git.code.sf.net/p/ngspice/ngspice
cd ngspice
./autogen.sh
mkdir release
cd release
../configure --with-x --enable-xspice --disable-debug --enable-cider \
--enable-pss --enable-osdi \
BISON=/usr/local/opt/bison/bin/bison \
YACC=/usr/local/opt/bison/bin/yacc
make -j$(nproc)
sudo make install
```

```
make clean
cd ~/gits
```