# Open Source Analog IC Design

**HyungJooPark**

**Sep 15, 2022**

# CONTENTS:

CHAPTER

ONE

# ENVIRONMENT SETUP

**Note:** This tutorial requires sudo access. If you are not the owner of server or system, you should use virtual machine or docker.

Notation Guides:

- '$' means ubuntu shell commands
- '%' means tcl/tk console
- ln[1] means interactive python command
- UPPER_CASE words denote user defined path

## 1.1 Install MAGIC

### 1.1.1 1. install dependent packages

**Note:** CentOS use yum instead of apt-get package. Package names could be slightly different.

```
$ sudo apt-get install m4
$ sudo apt-get install tcsh
$ sudo apt-get install csh
$ sudo apt-get install libx11-dev
$ sudo apt-get install TCL-dev tk-dev
$ sudo apt-get install libcairo2-dev
```

### 1.1.2 2. clone Magic source

make or go to a directory for the source(DIR_FOR_SOURCE) and clone the source from git.

```
$ cd DIR_FOR_SOURCE
$ git clone git://opencircuitdesign.com/magic
```

If it dosen't work, clone the Github mirror.

- $ git clone https://github.com/RTimothyEdwards/magic

**Note:** "apt-get install magic" command installs outdated version!

### 1.1.3 3. complie and Install

This step requires a sudo acess.

```
$ cd magic
$ ./configure
$ make
$ sudo make install
```

### 1.1.4 4. test

You can run magic on random directory.

- $ magic &

Check magic's cell search path with following

- % path

If CAD_ROOT appears after "System search path is", you have to set CAD_ROOT as an environment variable pointing where magic basic libraries are installed.

- the default is '/usr/local/lib'

## 1.2 Install XSHCEM, NETGEN, NGSPICE

### 1.2.1 1. XSCHEM

Install following packages using "sudo apt-get install"

| | |
|---|---|
| libX11-6 | libx11-dev |
| libxrender1 | libxrender-dev |
| libxcb1 | libx11-xcb-dev |
| libcairo2 | libcairo2-dev |
| tcl8.6 | tcl8.6-dev |
| tk8.6 | tk8.6-dev |
| flex | bison |
| libxpm4 | libxpm-dev |
| gawk or mawk | |

Clone xshcem source and install

```
$ cd DIR_FOR_SOURCE
$ git clone https://github.com/StefanSchippers/xschem.git_
$ cd xschem
$ ./configure --prefix=ROOT_FOR_XSCHEM (default: '/usr/local')
$ make
$ make install
```

### 1.2.2 2. NETGEN

```
$ cd DIR_FOR_SOURCE
$ git clone git://opencircuitdesign.com/netgen
$ cd netgen
$ ./configure
$ make
$ sudo make install
```

### 1.2.3 3. NGSPICE

NGSPICE uses wget instead of github clone

```
$ cd DIR_FOR_SOURCE
$ wget  https://sourceforge.net/projects/ngspice/files/ng-spice-rework/37/ngspice-37.tar.
↪gz
$ tar -zxvf ngspice-37.tar.gz
$ cd ngspice-37
$ mkdir release
$ cd release
$ ../configure --with-x --enable-xspice --enable-cider --enable-openmp --with-
↪readlines=yes --disable-debug
$ make
$ sudo make install
```

## 1.3 Install SKY130-PDK

### 1.3.1 1. Google/Skywater open SKY130-PDK

Clone PDK source from github and activates submodules. Since laygo2 doesn't use digital-standard libraries, we install primitive and IO libraries only.

```
$ cd DIR_FOR_SOURCE
$ git clone https://github.com/google/skywater-pdk
$ cd skywater-pdk
$ git submodule init libraries/sky130_fd_io/latest
$ git submodule init libraries/sky130_fd_pr/latest
$ git submodule update
$ make timing
```

### 1.3.2 2. setup pdk using open-pdks

App open-pdks installs Skywater-pdk into PDK_ROOT directory and generates magic library files with them. And download some useful 3rd-party libraries such as primitives for xschem.

```
$ cd DIR_FOR_SOURCE
$ git clone git://opencircuitdesign.com/open_pdks
$ cd open-pdks
$ ./configure --enable-sky130-pdk-DIR_FOR_SOURCE/skywater-pdk --disable-alpha-sky130
```

Before the install, check these considerable options of configure script.

1. –prefix=PDK_ROOT
   - pdk files are installed in 'PDK_ROOT/share/pdk'
   - default PDK_ROOT: '/usr/local'
   - we use default path in this tutorial
2. –enable-xschem-sky130[=PATH]
   - If not disabled, the 3rd-party Sky130 setup for xschem will be installed as part of the sky130A PDK.
   - If path is omitted, or the configuration option is not specified, then the library will be pulled automatically from the repository and installed.
   - To disable the package, use –disable-xschem-sky130.
3. –enable-alpha-sky130[=PATH]
   - The 3rd-party alphanumeric layout library will be installed. Everything else are identical with 2.

If you choosed and set the configuration, you can install now. **It could take times**.

```
$ make
$ sudo make install
```

## 1.4 Set startup file for MAGIC and XSCHEM

### 1.4.1 1. MAGIC

We need a symbolic link mapping sky130 tech-files into MAGIC system search path.

- $ sudo ln -s SKY130A_INTALL_ROOT_DIR/sky130A/libs.tech/magic/* /usr/local/lib/magic/sys/

default **SKY130A_INTALL_ROOT_DIR** is '/usr/local/share/pdk'. Which is identical to the "PDK_ROOT/share/pdk".

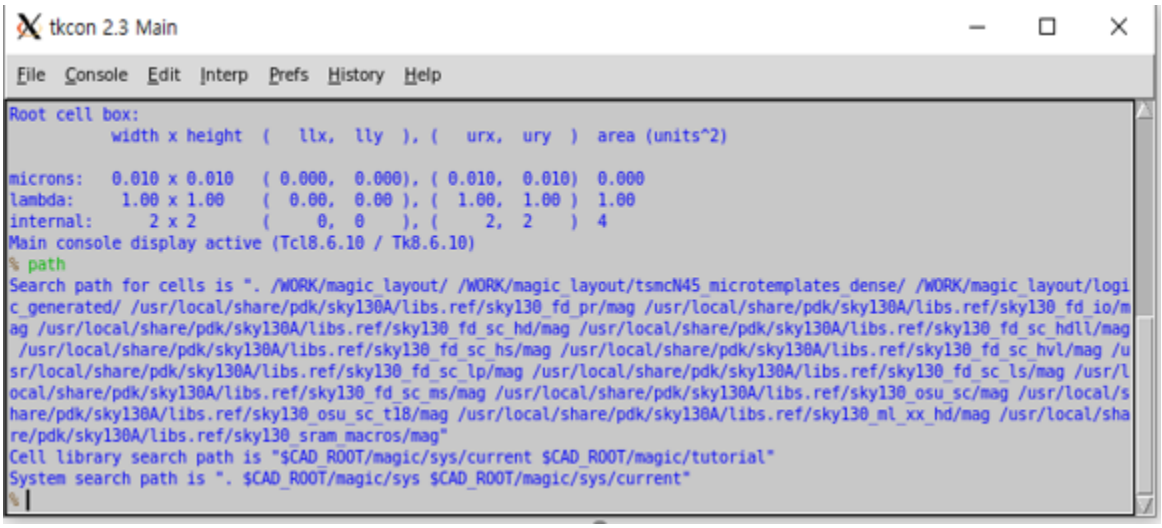
Set the .magicrc at home directory.

```
$ cd ~
$ cat > .magicrc (press enter key)
source $CAD_ROOT/magic/sys/sky130A.magicrc
```

Test whether it is connected well.

```
$ magic &
% path
```

It success if result looks like below.




### 1.4.2 2. XSHCEM

**Check xschem_sky130 library**

xschem_sky130 library should have installed in SKY130A_INSTALL_ROOT_DIR/sky130A/libs.tech

```
$ cd SKY130A_INSTALL_ROOT_DIR/sky130A/libs.tech
$ cd xschem
$ ls
```

If following directories didn't appear, you should download it manually.

```
(base) hjpark@sonic-400TCA-400SCA:/usr/local/share/pdk/sky130A/libs.tech/xschem$
 ls
decred_hash_macro  README.md     sky130_fd_pr.patch  stdcells
LICENSE            scripts       sky130_stdcells     xschemrc
mips_cpu           sky130_fd_pr  sky130_tests
(base) hjpark@sonic-400TCA-400SCA:/usr/local/share/pdk/sky130A/libs.tech/xschem$
```

**Set xschemrc**

```
$ cd ~/.xschem
$ cat > xschemrc (press enter)
  source /usr/local/share/pdk/sky130A/libs.tech/xschem/xschemrc (press ctrl+D to finish␣
↪writing)
$ xschem
```

Result should be looked like an image below.

Select the desired cell and press 'e' to check various circuits.


xschem - top.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
top.sch
Google
PERF. VPP CAP
RERAM
VARACTORS
ESD
MIM
PFET
3 PIN PFET
NFET
3 PIN NFET
RES
These symbols take W as finger length instead of total width
PFET
NFET
DIODE
PNP
XSCHEM
SNAP: 10
GRID: 20
NETLIST MODE
spice

CHAPTER

TWO

# WORKSPACE SETUP

## 2.1 Step 1 : Clone repository

Clone laygo2_workspace_sky130 from github

```
$ git clone https://github.com/niftylab/laygo2_workspace_sky130.git
$ cd laygo2_workspace_sky130
$ git submodule init
$ git submodule update
```

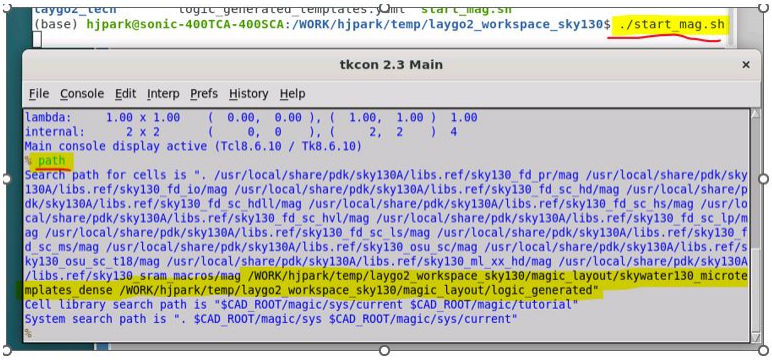
## 2.2 Step 2: Set path variables for magic

Initialize magic template cell search path for this tutorial

```
$ ./mag_set_path.sh
$ ./start_mag.sh
% path
```

These .sh scripts initialize the '.maginit_personal' file with the cell search path for 'magic'. Your magic search path should include the following directories.

- WORK_DIR/magic_layout/skywater130_microtemplates_dense
- WORK_DIR/magic_layout/logic_generated




These paths are the default template paths for the example code in laygo2.

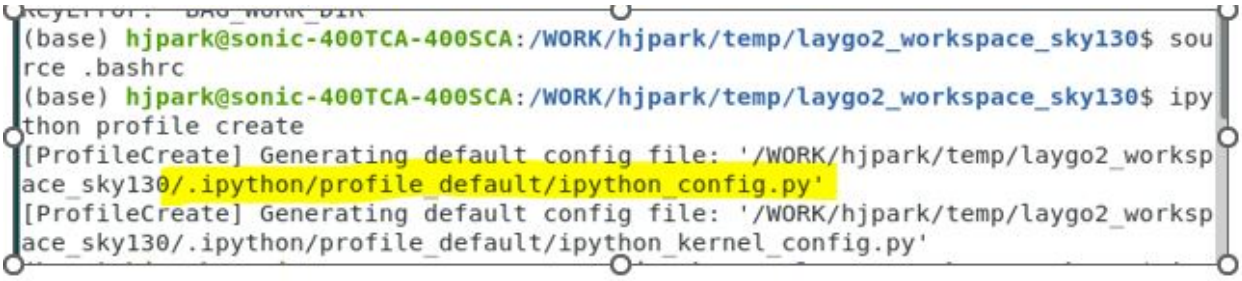
## 2.3 Step 3 : Test laygo2

**Note:** Several package prerequisites (such as Numpy) should be installed for LAYGO2. The base installation of Anaconda fulfills the package requirements.

```
$ source .bashrc
$ ipython profile create
$ cp ipython_config_init.py PROFILE_DIR/ipython_config.py
$ ipython ( or ./start_bag.sh )
```

The '.bashrc' file sets the python path of laygo2 by setting environment variables. The profile directory is printed to the shell. (Example: WORK_DIR/.ipython/profile_default/ipython_config.py) Overwrite the default 'ipython_config_init.py' to the config file in the repository.

If PROFILE_DIR is not output, or it dosen't contain WORK_DIR, check if .ipython dir exists in WORK_DIR. If so, PROFILE_DIR = WORK_DIR/.ipython/profile_default.




```
import sys
print(sys.path)
```

The printed python path should include 'WORK_DIR/laygo2'.

CHAPTER

THREE

# RUN EXAMPLE CODE

## 3.1 Goal : Generate a D Flip-Flop layout

In this tutorial, we are going to generate a D Flip-Flop cell with laygo2 and magic.

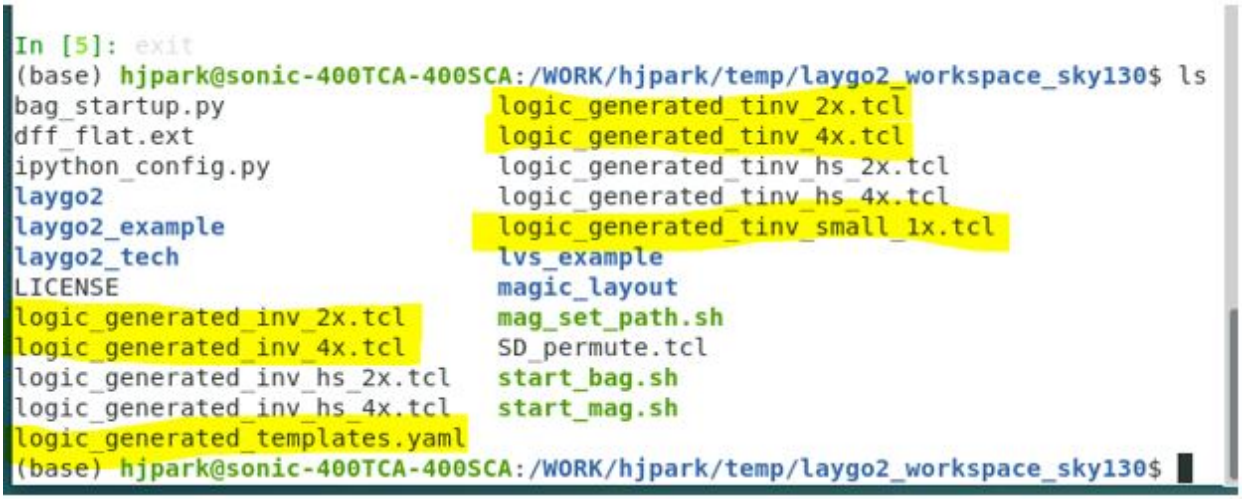
As the D Flip-Flop is composed of multiple subcells such as inverter and tri-state inverter, we should generate the subcell layouts before generating the DFF layout.

## 3.2 Step 1 : Subcell layout generation

Running the following commands in ipython generates magic TCL scripts.

```
ln [1]: run laygo2_example/inv.py
ln [2]: run laygo2_example/tinv.py
ln [3]: run laygo2_example/tinv_small_1x.py
ln [4]: exit
```

If the generators ran successfully, you should see the highlighted files in the following figure:




The actual layouts are generated by running the TCL scripts in magic by typing the following commands:

```
$ ./start_mag.sh
% source logic_generated_inv_2x.tcl
% source logic_generated_inv_4x.tcl
```

(continues on next page)

(continued from previous page)

```
% source logic_generated_tinv_2x.tcl
% source logic_generated_tinv_4x.tcl
% source logic_generated_tinv_small_1x.tcl
```

Turn off magic if you confirmed that the subcell layouts are generated (You may see warning messages about saving the designs. You can ignore the messages, as the cells are actually generated/saved).

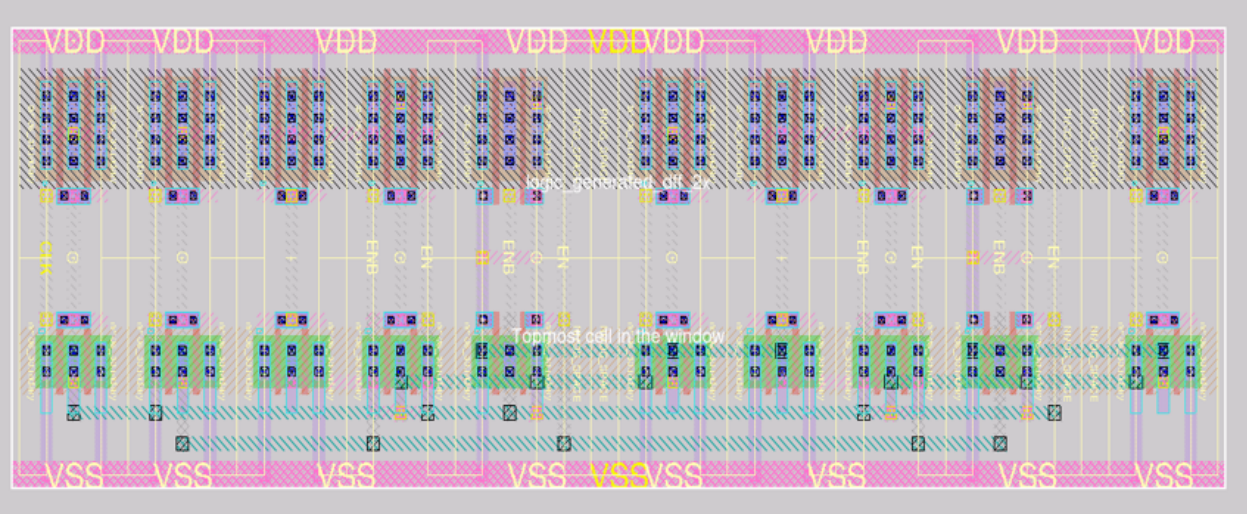
## 3.3 Step 2: DFF layout generation

Now, you can generate the DFF layout by running the following Python and TCL commands.

```
ln [1]: run laygo2_example/dff.py
ln [2]: exit
```

```
% source logic_generated_dff_2x.tcl
% source logic_generated_dff_4x.tcl
% load logic_generated_dff_2x
% select top cell
% expand
```

Now, you can see a layout of DFF as shown in the following figure.



CHAPTER

FOUR

# RUN LVS

## 4.1 Goal : Run Netgen for LVS test

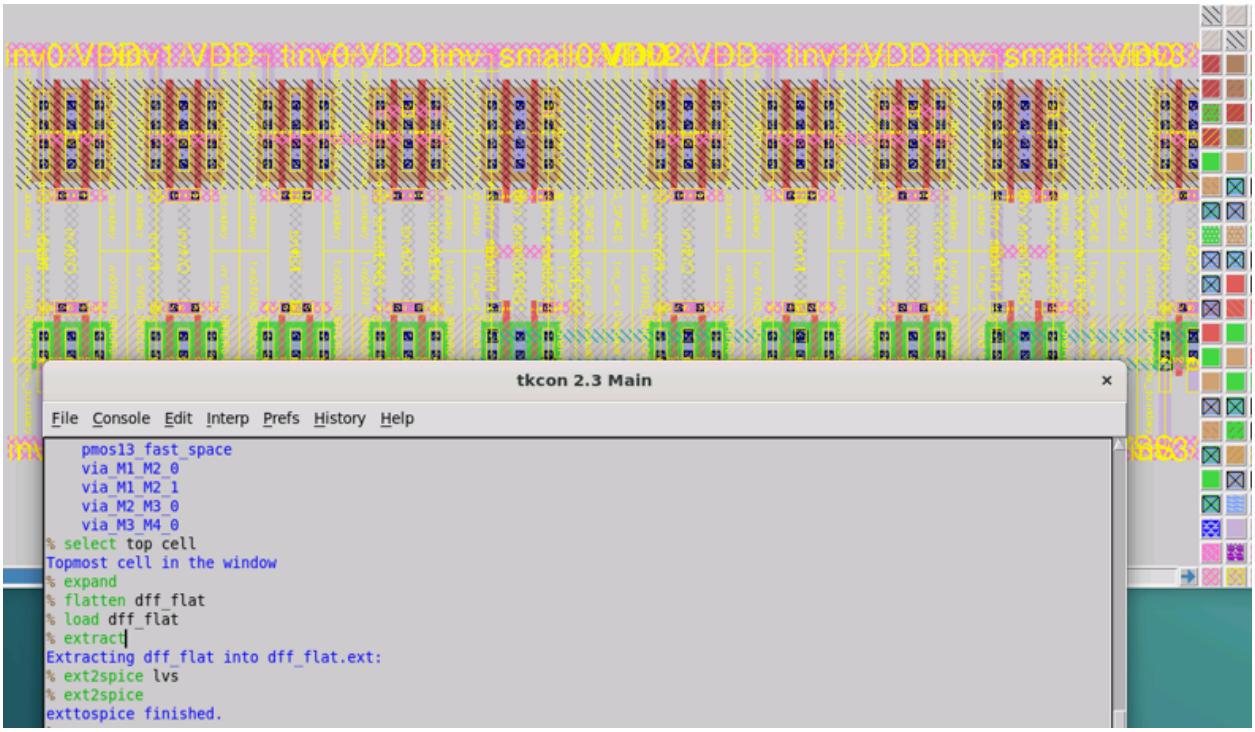
In this tutorial, we will run Netgen over the DFF cell we created in the previous section. Netgen is an open-sourced program for LVS check. As Netgen can only read SPICE netlists in a restricted format, this tutorial manipulates netlists accordingly after extraction.

## 4.2 Step 1 : Extract a SPICE netlist from the DFF cell

For unclear reasons, magic often extracts only a part of netlists from hierarchical cells. So, we have to flatten the hierarchical DFF cell before we extract the netlist.

```
% load logic_generated_dff_2x
% select top cell
% expand
% flatten dff_flat
% load dff_flat
% extract
% ext2spice lvs
% ext2spice
```

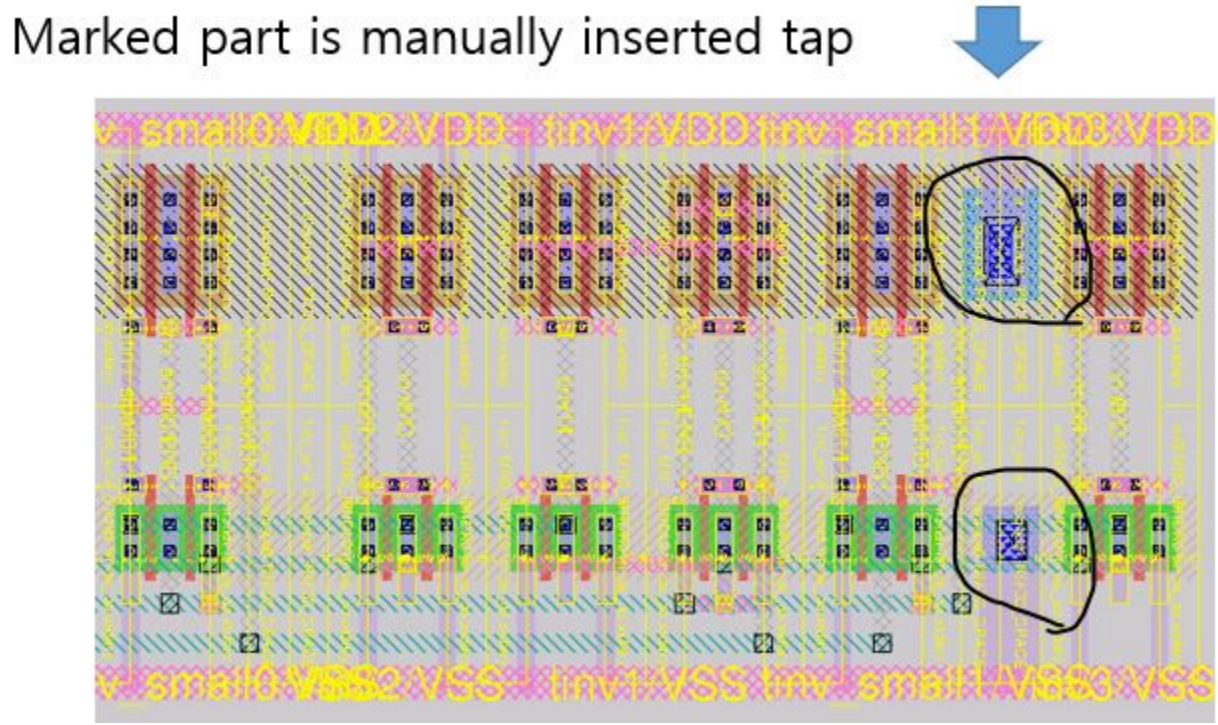
You can see the colors changed when you load the dff_flat layout.




Turn off magic. Ignore the caution again this time, as you don't need a flat layout file.

One step needs to be done before running the LVS. As the generated DFF cell does not contain taps inside (as typically the taps are placed as separate instances), we need to manually insert manual taps before running LVS.

You can either 1) manually draw taps by yourself, if you are familiar with magic. Otherwise, use the 'dff_tap.spice' file in the 'lvs_example' directory in your workspace. This tutorial uses 'dff_tap.spice'.



## 4.3 Step 2 : Edit SPICE netlists from schematic for Netgen

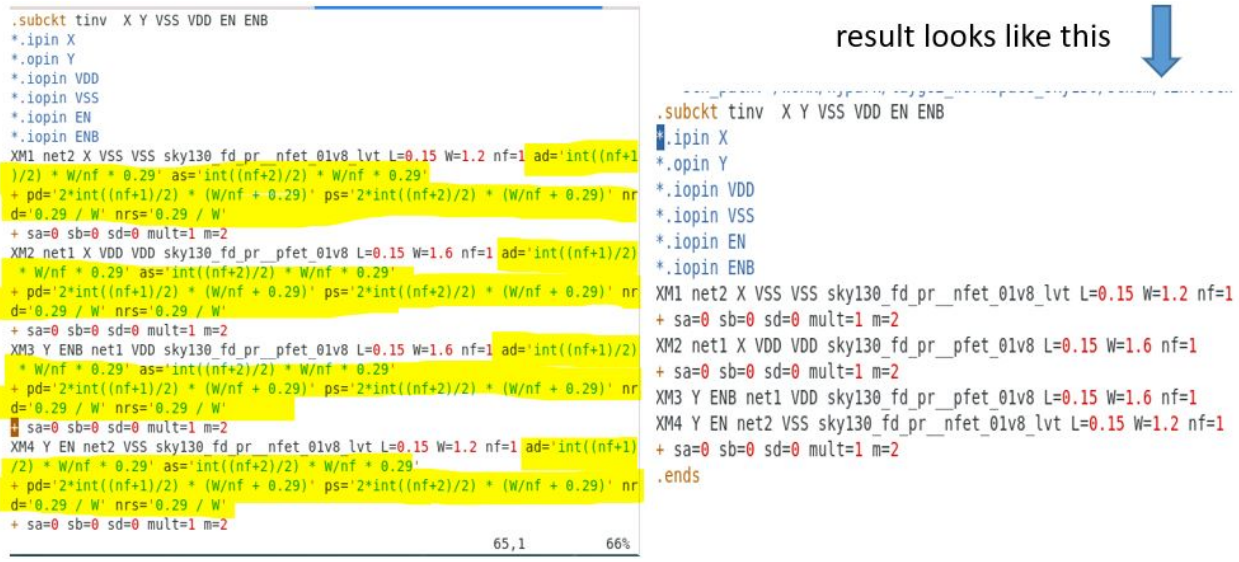
Now we have the extracted netlist from the layout. The next step is extracting the netlist from the schematic for comparison. There are many reference articles you can find out online for drawing/extracting netlists from xschem, an open-sourced schematic editor. For this tutorial, we are going to use 'DFF_tinv.spice', which is extracted from xschem and has the identical topology with 'DFF_tap' (the finger numbers are different).

You must delete the parameters defined by the equation before running Netgen. Because Netgen can't read these equations and it throws a segmentation fault! Fortunately, key parameters like width, length, and finger number are constant values. So we simply drop the parameters expressed as equations.

Copy the DFF_tinv file and edit on vim.

```
$ cd WORK_DIR/lvs_example
$ cp DFF_tinv.spice DFF_tinv_lvs.spice
$ vim DFF_tinv_lvs.spice
```

Delete mosfet parameters expressed as equations(see highlighted terms). Do this for every .subckt.

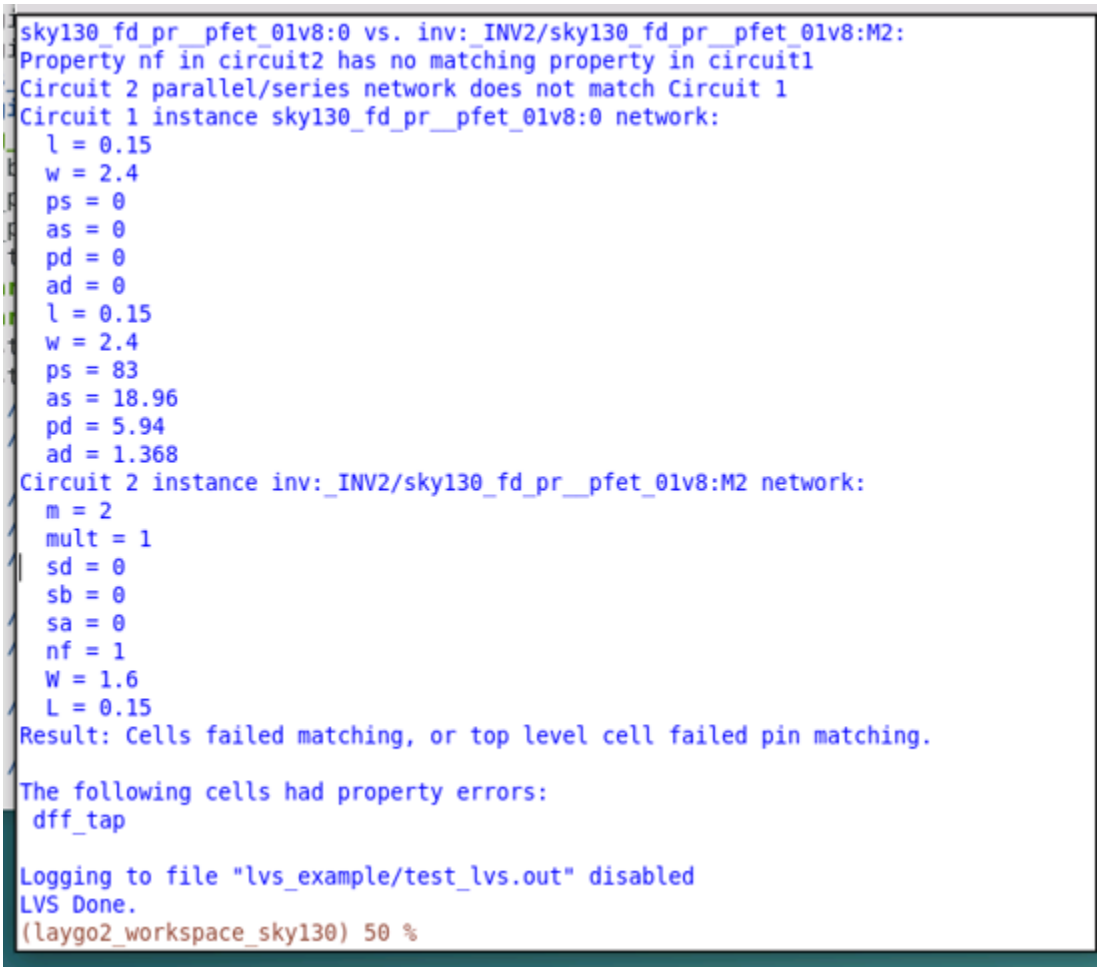



## 4.4 Step 3 : Run Netgen

Run the following commands for the LVS test with Netgen.

```
$ cd WORK_DIR
$ netgen
% lvs {lvs_example/dff_tap.spice dff_tap} {lvs_example/DFF_tinv_lvs.spice} SD_permute.
↪tcl test_lvs.out
```

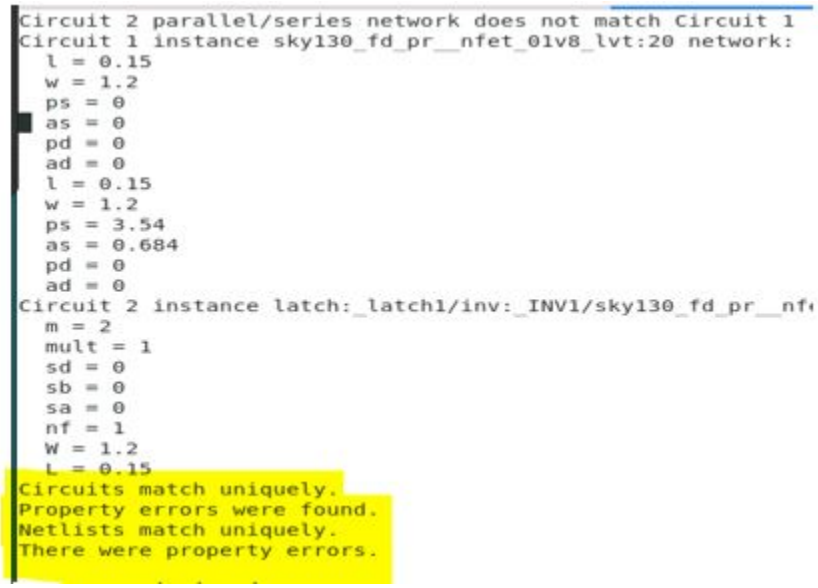
If you get a result like the following figure, you are all set!




Now let's check the output logfile. Close Netgen and open the logfile by running the following command.

```
$ vim test_out.out
```

You can see the following result.




It says netlists match uniquely. But below the note, there are several mismatch messages.

```
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property ps in circuit1 has no matching property in circuit2
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property as in circuit1 has no matching property in circuit2
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property pd in circuit1 has no matching property in circuit2
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property ad in circuit1 has no matching property in circuit2
 M circuit1: 1   circuit2: 2   (delta=1, cutoff=0)
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property mult in circuit2 has no matching property in circuit
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property sd in circuit2 has no matching property in circuit1
sky130_fd_pr__nfet_01v8_lvt:20 vs. latch:_latch1/inv:_INV1/sk
Property sb in circuit2 has no matching property in circuit1
```

The parameter mismatch messages come from 1) parameter naming differences and 2) the finger differences between xschem and magic. For example, magic uses 'l' for the MOSFET channel length whereas xschem uses 'L'. Netgen cannot catch that they are actually equivalent. For fingered transistors, xschem uses m=2 (or nf = 2), whereas magic prints parameters of each finger of MOSFETs. **As a result, these two circuits are identical even though Netgen throws the parameter errors.** Even when there are serious parameter mismatches, you can catch them by reading the log.

CHAPTER

FIVE

# LIBRARY MANAGEMENT

## 5.1 Personal Library

Create a directory with the same name as the library in the magic_layout directory

```
(base) hjpark@sonic-400TCA-400SCA:/WORK/laygo2_workspace_sky130$ cd magic_layout/
(base) hjpark@sonic-400TCA-400SCA:/WORK/laygo2_workspace_sky130/magic_layout$ ls
logic_generated  skywater130_microtemplates_dense
(base) hjpark@sonic-400TCA-400SCA:/WORK/laygo2_workspace_sky130/magic_layout$ mkdir newLib
(base) hjpark@sonic-400TCA-400SCA:/WORK/laygo2_workspace_sky130/magic_layout$ ls
logic_generated  newLib  skywater130_microtemplates_dense
(base) hjpark@sonic-400TCA-400SCA:/WORK/laygo2_workspace_sky130/magic_layout$
```

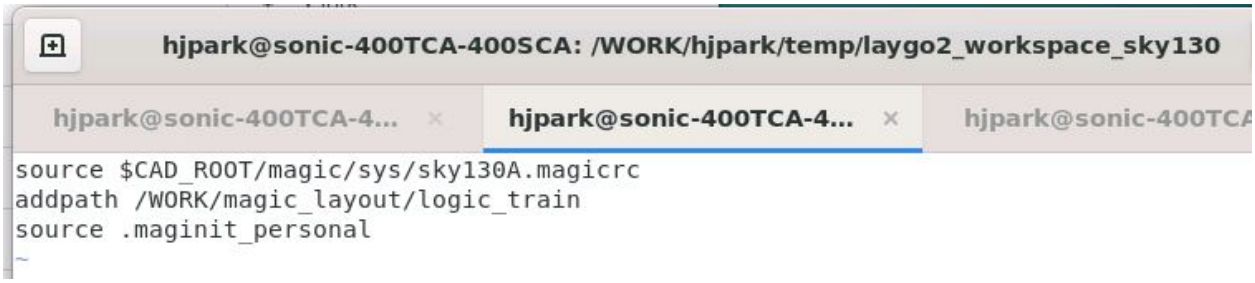
Add new library path in '.maginit_personal' file in workspace directory.


Now you can use this library. See "Testing the new libraries" section for instructions on how to determine which library your Python code will generate layout files to.

## 5.2 Public Library

You can create library directory outside of workspace directory also.


Add new public library path in .maginit file.




If you are working on a team project, You can use this kind of public library. However, this method requires editing and sharing the tracked '.maginit' on Github. In this case, consider using a **forked workspace repository**. When a project master forks the repository and the team members clone the forked repository instead of the default workspace repository, tracked files can be reliably modified and shared via Github.

## 5.3 Testing the New Libraries

To change generation path, you have to change some variables in python code. First, change the libname into new library name(logic_generated -> logic_train).

```
# Design hierarchy
libname = 'logic_train'
cellname = cell_type+'_'+str(nf)+'x'
ref_dir_template = './'#laygo2_generators_private/magic/logic/'
ref_dir_MAG_exported = './'#laygo2_generators_private/magic/logic/tcl/'
# End of parameter definitions ######
```

Then, change 'libpath' into where logic_train directory is placed. The 'libpath' is a parameter of the function "laygo2.interface.magic.export". In this tutorial, libpath = "/WORK/magic_layout"

```
# 7. Export to physical database.
print("Export design")

# Uncomment for BAG export
laygo2.interface.magic.export(lib, filename=ref_dir_MAG_exported +libname+'_'+cellname+'.tcl', c
ellname=None, libpath='/WORK/magic_layout', scale=1, reset_library=False, tech_library='sky130A'
)
```

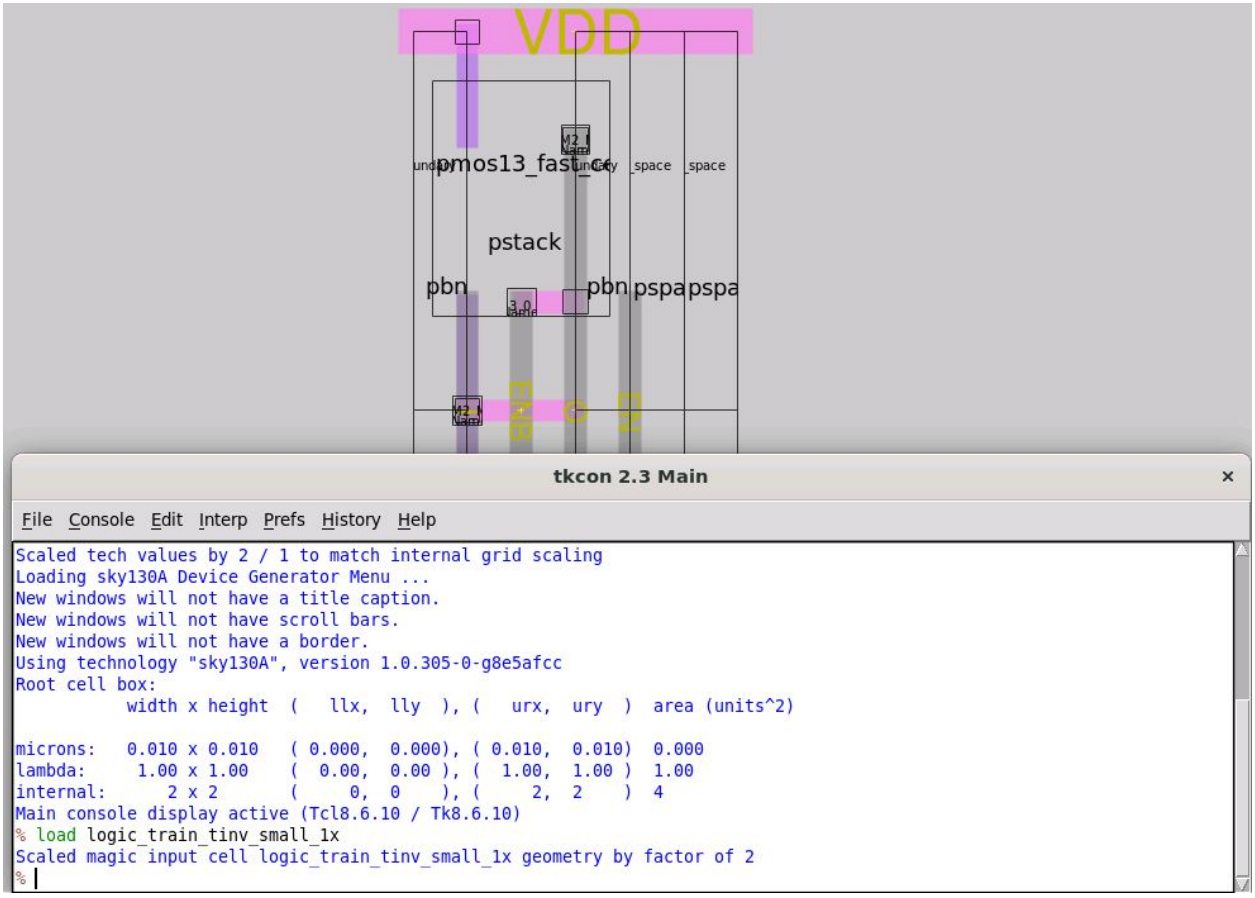
Generate TCL and mag file in the same way as in the previous section.

Close magic and check if the mag file is generated in logic_train.

Open magic again and load the layout itself in the public library.

```
% load logic_train_tinv_small_1x
```

You can see the layout is loaded. Now you can use this library through laygo2.



CHAPTER

SIX

# INDICES AND TABLES

- genindex
- modindex
- search