HeiChips 2026 Analog Workshop

Simon Dorrer

2026-08-05

Table of contents

Workshop Slides

The slides presented during the workshop are available here: HeiChips2026_Analog-Workshop_Slides.pdf.

Important

This workshop is based on a more advanced chip design tutorial for the ihp-sg13g2 PDK and the IIC-OSIC-TOOLS container: A Tutorial on Open-Source Analog-Mixed Signal Chip Design with the ihp-sg13g2 Open-PDK. If you want to go beyond this workshop, for example digital macros, top-level assembly, padframe generation, and tapeout preparation, that tutorial is the recommended next step.

1 Workshop Overview

Welcome to the analog design workshop of HeiChips 2026, the Summer School on Open-Source Chip Design taking place from August 3 to 7, 2026 at the European Institute for Neuromorphic Computing (EINC) in Heidelberg, Germany. In one week of lectures, hands-on workshops, and a hackathon, you design your own chip and tape it out on the IHP 130nm CMOS5L Open Source PDK.

This workshop covers the analog mixed-signal track on **Wednesday, August 5, 2026**:

Table 1: Workshop schedule on Wednesday.

Time	Session
09:00 – 09:15	Open-Source Analog Mixed-Signal: Overview
09:15 – 10:30	Open-Source Analog Mixed-Signal: Schematic (Xschem, Ngspice)
11:00 – 12:30	Open-Source Analog Mixed-Signal: Layout (KLayout, DRC, LVS, PEX)
13:15 – 14:45	Open-Source Analog Mixed-Signal: Exercises

Throughout the day we implement one complete analog macro, a CMOS inverter used as an analog amplifier, in the `ihp-sg13cmos5l` Open-PDK. We go through the full flow:

1. Schematic entry and symbol creation in Xschem (see Section 4)
2. Simulation with Ngspice using dc, ac, and transient testbenches (see Section 4.4)
3. Layout in KLayout (see Section 5)
4. Verification with DRC, LVS, and PEX using KLayout and Magic + Netgen (see Section 5.4)
5. Post-layout simulation with the extracted parasitics (see Section 5.5)
6. Exercises where you apply the flow on your own (see Section 6)

All required tools are provided through a Nix shell, so no container is needed. Follow the Prerequisites in the repository README before the workshop:

```
git clone https://github.com/HeiChips/heichips26-analog-workshop.git
cd heichips26-analog-workshop
nix-shell             # enter the tool environment (do this in every new terminal)
make clone-pdk        # install the ihp-sg13cmos5l Open-PDK into this repository
make klayout-setup    # install the KLayout productivity plugins
```

> **Note**
>
> The HeiChips VM has Nix already pre-installed. If you work on your own machine, follow LibreLane's Nix-based installation guide first.

Inside the Nix shell, export the PDK environment variables once per terminal from the repository root:

```
export PDK_ROOT=$(pwd)/IHP-Open-PDK
export PDK=ihp-sg13cmos5l
```

1.1 Guidelines & Cheatsheets

The following guidelines and cheatsheets are helpful during the workshop and for your own projects afterwards. Some of them are located directly in the `doc/` folder of this repository.

- Designer's Guidelines
- Linux Cheatsheet
- Xschem Cheatsheet
- Ngspice Cheatsheet
- KLayout Cheatsheet (in `doc/klayout/`)
- KLayout Productivity Suite / Plugins

- ihp-sg13cmos5l Layout Calculation Cheatsheet (in `doc/ihp-sg13cmos5l-Open-PDK/`)
- ihp-sg13cmos5l Layout Rules and Process Specification (in `doc/ihp-sg13cmos5l-Open-PDK/`)
- ihp-sg13cmos5l LV and HV NMOS / PMOS Techsweeps (in `doc/sizing/`)

2 Repository and Makefile Overview

2.1 Directory Structure

The workshop repository is organized as follows:

```
heichips26-analog-workshop/
  config/       Shared tool configuration (.spiceinit for Ngspice, klayoutrc)
  doc/          Cheatsheets, PDK documentation, and sizing techsweeps
  inverter/     The analog macro implemented in this workshop
  workshop/     Sources of this website
 flake.nix        Nix environment with all required tools
 Makefile         PDK setup and tool launch targets
 README.md        Quick-start instructions
```

The root `Makefile` provides the setup and launch targets. Running `make` without arguments prints all available targets.

Table 2: Root Makefile targets used in this workshop.

Target	Description
`make clone-pdk`	Clone the IHP Open-PDK plus the `ihp-sg13cmos5l` PDK into `IHP-Open-PDK/` and compile the Verilog-A models with OpenVAF
`make klayout-setup`	Install the KLayout plugins into your user directory `~/.klayout/salt/`, once per machine (see Section 5.2)
`make klayout`	Open KLayout in edit mode with the `sg13cmos5l` technology preloaded

2.2 The Inverter Macro Folder

The `inverter/` folder contains the complete analog macro with the following structure:

Table 3: Folder structure of the `inverter` macro.

Folder	Content
`schematic/xschem/`	Xschem schematic and symbols (`inverter.sch`, `inverter.sym`, `inverter_pex.sym`)
`testbenches/xschem/`	The dc, ac, and transient testbenches plus Python plotting scripts
`layout/`	KLayout GDS files (`inverter.gds`)
`netlist/`	Exported netlists: `schematic/` (for LVS), `layout/` (extracted), `pex/` (parasitic)
`verification/`	DRC and LVS reports (`drc/`, `lvs/`)
`scripts/`	Sizing notebooks (`sizing/`), vendored `sak-*` verification scripts, rendering
`final/`	Build deliverables for top-level integration (GDS, LEF, LIB, Verilog stub, and the layout render in `render/`)

> **Tip**
>
> Read the inverter README first to get familiar with the folder layout and the available Makefile targets. Everything below is documented there in more detail.

2.3 Makefile Targets of the Macro

All design steps are invoked through the macro `Makefile` from within the `inverter/` folder. Running `make` without arguments prints all targets. The most important ones for this workshop are:

Table 4: Macro Makefile targets.

Target	Description
`make sim-xschem [TB=<tb>]`	Run one Xschem testbench in batch mode (netlist + `ngspice -b`)
`make sim-view-xschem [SCRIPT=<script>]`	Plot the exported simulation data with Python
`make sim-all`	Run all three testbenches in sequence
`make klayout-drc`	Run KLayout DRC on the layout
`make magic-drc`	Run Magic DRC on the layout
`make klayout-lvs`	Export the schematic netlist and run KLayout LVS
`make magic-lvs`	Export the schematic netlist and run Magic + Netgen LVS

Target	Description
`make magic-pex`	Run parasitic extraction with Magic
`make klayout-verify`	DRC and LVS with KLayout in one command
`make magic-verify`	DRC, LVS, and PEX with Magic in one command
`make build-top`	Build the deliverables (LEF, LIB, Verilog stub, GDS copy, render)
`make all`	Verify, build, and simulate everything
`make clean`	Delete all generated files and folders (deliverables, netlists, reports, simulation outputs)

Most targets accept optional variables, for example `CELL=<cellname>` to select a specific cell (defaults to `inverter`), `EXT_MODE=<1|2|3>` for the PEX mode, and `DRC_LEVEL=<precheck|macro|regular>` for the KLayout DRC rule set. `TB` and `SCRIPT` default to `<CELL>_tb_tran` and `plot_<CELL>`, so both simulation targets also run without arguments. See the inverter README for the full list.

> **Warning**
>
> KLayout-PEX (kpex) does not support the `ihp-sg13cmos5l` PDK yet, so the `klayout-pex` target currently fails. Use `make magic-pex` for parasitic extraction.

3 The Analog Macro: A CMOS Inverter

The `inverter` macro is the analog example of this workshop. It is a CMOS inverter drawn at transistor level and verified with a complete verification chain: DRC, LVS, and PEX with KLayout and Magic + Netgen.

> **Note**
>
> An inverter can act as a digital circuit, when the input is driven by a digital signal and the output switches between the supply rails. Here we characterize the inverter as an analog circuit. We are interested in its continuous voltage transfer characteristic and its small-signal voltage gain around the operating point.

Figure 1 shows the circuit. The NMOS M_1 and the PMOS M_2 form the complementary inverter. The transistors M_{dummy1} and M_{dummy2} are dummy devices. They do not contribute to the circuit function but improve matching in the layout, because the active transistors then see identical neighboring structures on both sides.

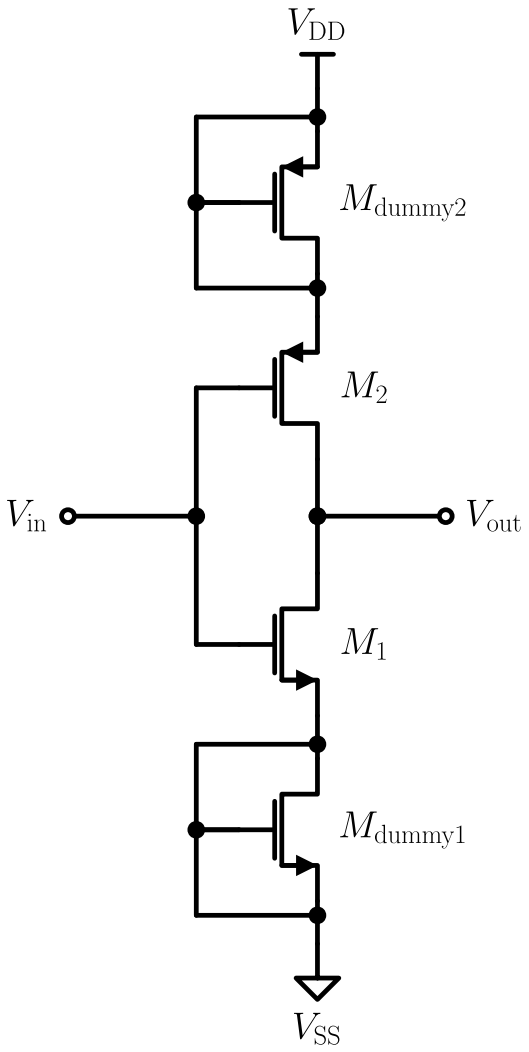

Figure 1: The CMOS inverter circuit with dummy transistors.

3.1 Transistor Sizing

The inverter is sized with the g_m/I_D method (see Transistor Sizing Using g_m/I_D Methodology for the theoretical background). Instead of relying on textbook square-law equations, the method looks up the actual device behavior from pre-characterized SPICE simulations. The full sizing flow is implemented in the Jupyter notebook `scripts/sizing/sizing_inverter.ipynb`, which uses `pygmid` to query the pre-characterized `ihp-sg13cmos5l` lookup tables in `scripts/sizing/data/`. An overview of the techsweeps is available in `doc/sizing/`.

3.1.1 Design Tradeoffs: Choosing g_m/I_D and L

Before sizing anything, two numbers have to be picked: the transconductance efficiency g_m/I_D and the channel length L. Everything else follows from them, so it is worth understanding what each one buys and what it costs.

g_m/I_D says how much transconductance you get per ampere of bias current, in S/A. It is a normalized, technology-independent measure of the **inversion level** of the device.

Figure 2 shows the tradeoffs in one picture. The arrows point from the center towards the corner where each design parameter reaches its **best** value, so the same parameter is at its worst in the opposite direction.

Read it corner by corner:

- **Strong inversion, short** L (bottom left): f_T and input impedance Z_{in} are maximal, gate capacitance C_{gg} and area are minimal. This is the high-speed corner.
- **Strong inversion, medium** L (left): minimal thermal noise PSD S_{TH} and maximal current density I_D/W.
- **Moderate inversion, short** L (bottom): the speed-efficiency product $f_T \cdot g_m/I_D$ is maximal, the best overall compromise between bandwidth and current.
- **Moderate inversion, long** L (top): the output resistance $r_{ds} = 1/g_{ds}$ is maximal. This is where current sources belong.
- **Weak inversion, long** L (top right): intrinsic gain g_m/g_{ds} is maximal, while flicker corner frequency f_{co}, noise factor γ, flicker noise S_{FL}, offset V_{OS}, and power P are minimal. This is the low-power, low-noise, high-gain corner.
- **Weak inversion, medium** L (right): overdrive V_{od} and $V_{ds,sat}$ are minimal, so this corner gives the most voltage headroom.

Two corners stay empty on purpose. Strong inversion with a long channel and weak inversion with a short channel optimize nothing, they combine the drawbacks of both settings, so in practice they make little sense.

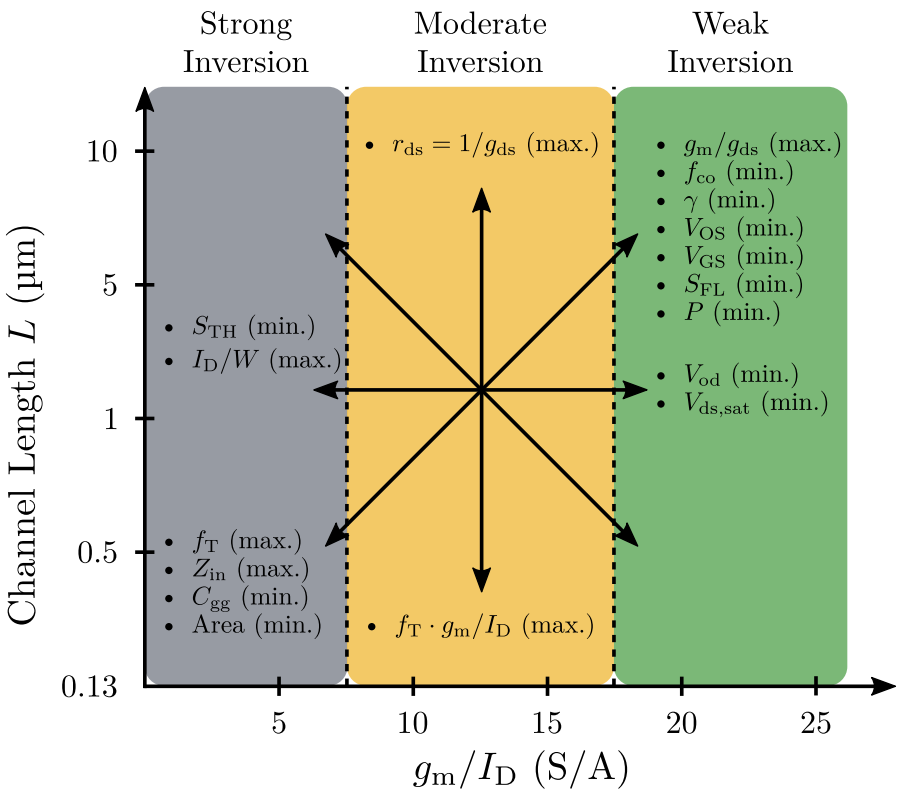

Figure 2: Overview of key design tradeoffs depending on the channel length L and the transconductance efficiency g_m/I_D. Taken from Tradeoffs in Saturation in the Analog Circuit Design course @ JKU.

Rules of thumb

Fast circuits use **strong inversion with short channels**. Low-power and low-noise circuits use **weak inversion with long channels**. Current sources use **moderate inversion with longer channels**, to maximize their output resistance. Everything else starts in moderate inversion and is moved from there.

Where does our inverter sit?

The sizing below lands at $g_m/I_D = 3.76\,\mathrm{S/A}$ with $L = 1\,\mathrm{m}$, which is deep strong inversion with a comparatively long channel, that is the corner the map leaves empty. This is a deliberate compromise for this example, not an oversight: the stage has to drive a 10 pF load, which demands a lot of g_m and therefore a lot of current, while the long channel keeps the intrinsic gain and the matching of this very simple circuit acceptable. It also explains the modest measured gain of about 31 dB. Keep the map in mind when you size your own blocks, where you are free to choose the operating point rather than inheriting it.

3.1.2 Sizing the Inverter

The notebook runs through the following steps:

1. Define the design targets: $V_{DD} = 1.5\,\mathrm{V}$, $V_{cm,in} = V_{cm,out} = V_{DD}/2 = 0.75\,\mathrm{V}$, $I_{out} = 0.75\,\mathrm{mA}$, $C_{load} = 10\,\mathrm{pF}$, and $L = 1\,\mathrm{m}$. L is chosen longer than the minimum to increase intrinsic gain and reduce mismatch, at the cost of area and bandwidth.
2. Look up g_m/I_D, g_m/g_{ds}, g_m/C_{gg}, and I_D/W for the chosen L and operating-point voltages.
3. Compute the required widths $W = I_D/(I_D/W)$ for NMOS and PMOS.
4. Round the continuous widths to a practical finger configuration with equal finger counts for NMOS and PMOS (for better layout matching), and recompute the small-signal parameters with the final widths.
5. Estimate the open-loop gain $A_{ol} = -g_{m,tot}/g_{ds,tot}$ and the output resistance $R_{out} = 1/g_{ds,tot}$.

The resulting device geometry is summarized in Table 5. These are exactly the values you will enter in the schematic in Section 4.

Table 5: Sized device geometry of the inverter.

Device	L (m)	Fingers (ng)	W / Finger (m)	Total W (m)
NMOS M_1	1.0	20	1.0	20.0
PMOS M_2	1.0	20	6.0	120.0
NMOS M_{dummy1}	1.0	2	1.0	2.0

Device	L (m)	Fingers (ng)	W / Finger (m)	Total W (m)
PMOS M_{dummy2}	1.0	2	6.0	12.0

With the final widths, the notebook computes $g_{\text{m,tot}} = g_{\text{m,NMOS}} + g_{\text{m,PMOS}} = 3.35\,\text{mS} + 2.92\,\text{mS} = 6.28\,\text{mS}$ and $g_{\text{ds,tot}} = 145\,\text{S}$. From these values the key metrics can be estimated:

$$|A_{\text{ol}}| = \frac{g_{\text{m,tot}}}{g_{\text{ds,tot}}} = \frac{6.28\,\text{mS}}{145\,\text{S}} = 43.2 = 32.72\,\text{dB},$$

$$R_{\text{out}} = \frac{1}{g_{\text{ds,tot}}} = \frac{1}{145\,\text{S}} = 6.89\,\text{k},$$

$$f_{\text{cu}} = \frac{1}{2\pi\, R_{\text{out}}\, C_{\text{load}}} = \frac{1}{2\pi \cdot 6.89\,\text{k} \cdot 10\,\text{pF}} = 2.31\,\text{MHz},$$

$$f_{\text{T}} = \frac{g_{\text{m,tot}}}{2\pi\, C_{\text{load}}} = \frac{6.28\,\text{mS}}{2\pi \cdot 10\,\text{pF}} = 100\,\text{MHz}.$$

Table 6 compares these hand calculations against the ac simulation results from the testbench in Section 4.4.1. The agreement is very good, which confirms the lookup-table approach.

Table 6: Calculated vs. simulated open-loop characteristics of the inverter.

Quantity	$g_{\text{m}}/I_{\text{D}}$ calculation	ac open-loop simulation
Open-loop dc gain A_{ol}	32.72 dB	31.33 dB
Open-loop cut-off frequency f_{cu}	2.31 MHz	2.89 MHz
Unity-gain frequency f_{T}	100 MHz	107 MHz

Tip

The notebook is fully parametric. Changing L, the target $g_{\text{m}}/I_{\text{D}}$, or I_{out} and re-running the cells immediately produces a new sizing point. Try it yourself: decrease the transistor length to $L = 0.5\,\text{m}$ and observe how the small-signal parameters and the resulting gain and cut-off frequency change. What would you expect beforehand?

3.2 Batch Simulation with the Makefile

Every testbench can be run non-interactively from the `inverter/` folder. This is useful for reproducible results and scripting.

```
cd inverter
make sim-xschem TB=inverter_tb_dc_vout
make sim-xschem TB=inverter_tb_ac_ol
make sim-xschem TB=inverter_tb_tran
make sim-all                                # all three in sequence
make sim-view-xschem SCRIPT=plot_inverter # plot the exported data with Python
```

`sim-xschem` netlists the testbench with `xschem netlist` and then calls `ngspice -b` directly, so `make` blocks until the run finishes and sees its exit status. Because the run is headless, the `plot` commands in the testbench are a no-op. Every testbench instead exports its results with `wrdata` to `testbenches/xschem/plot_simulations/data/`, from where the Python script plots them. Figure 3 and Figure 4 show the plotted results.

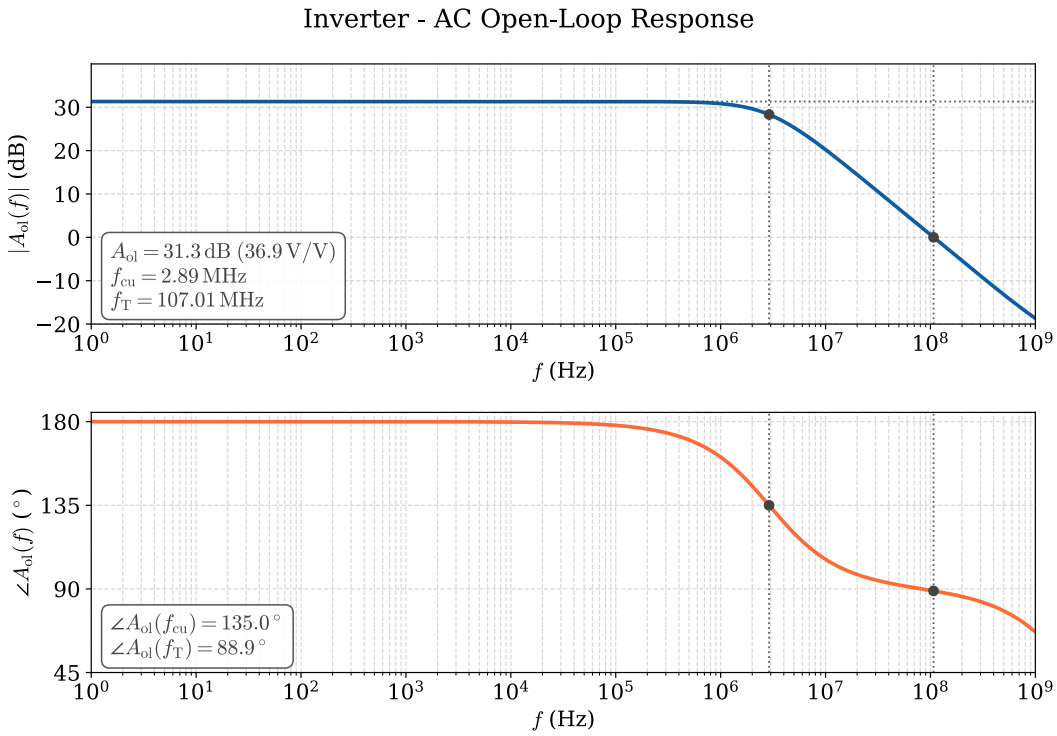

Figure 3: Simulated open-loop frequency response of the inverter (`inverter_tb_ac_ol`).

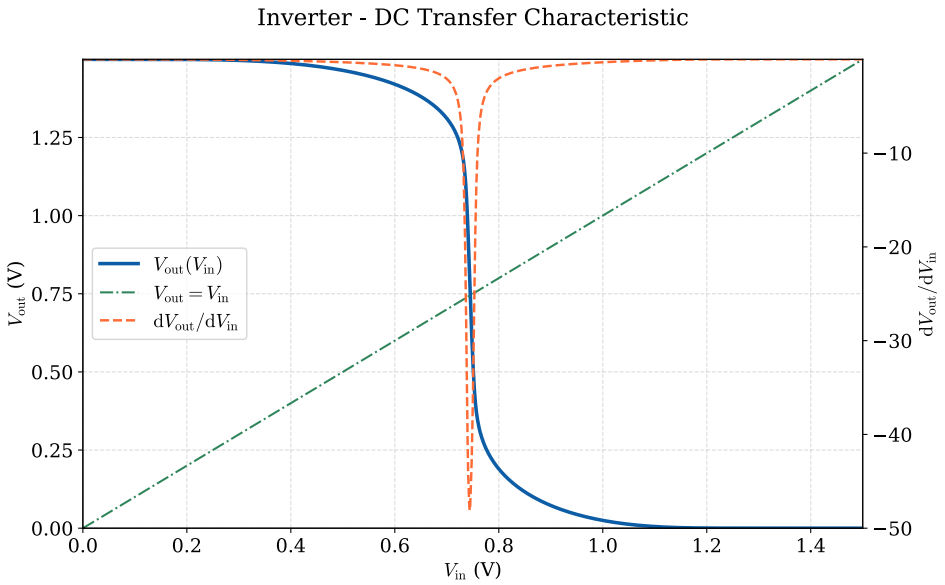

Figure 4: Simulated voltage transfer characteristic of the inverter (`inverter_tb_dc_vout`).

4 Hands-On: Schematic Entry

In this session we draw the inverter schematic, create its symbol, and complete the testbenches together in Xschem.

4.1 Getting Started with Xschem

Xschem is a schematic capture tool where circuits are built from symbols, wires, and net labels. Everything is stored in plain text files (`.sch` for schematics, `.sym` for symbols), which makes them friendly for version control and search-and-replace in a text editor.

Start Xschem from the schematic folder of the macro so that the local `xschemrc` is picked up (it configures the PDK libraries and search paths):

```
cd inverter/schematic/xschem
xschem                  # empty session
xschem inverter.sch     # or load the schematic directly
```

The most important shortcuts are listed in Table 7. A complete list is available in Xschem under `Help -> Keys` and in the Xschem Cheatsheet.

Table 7: Essential Xschem shortcuts.

Shortcut	Action
`Ins`	Insert a symbol from the component library
`w`	Draw a wire
`m`	Move the selected objects
`c`	Copy the selected objects
`Del`	Delete the selected objects
`q`	Edit the properties of the selected object
`Shift + R`	Rotate the selection
`Shift + F` / `Shift + V`	Mirror the selection horizontally / vertically
`u` / `Shift + U`	Undo / redo
`f`	Zoom full
`Ctrl + s`	Save
`e`	Descend into the schematic of the selected symbol
`i`	Descend into the symbol view of the selected symbol
`Ctrl + e`	Return to the parent schematic
`Shift + T`	Toggle the `ignore` flag on an instance (excluded from netlist)
`n`	Generate a netlist
`Ctrl + o`	Open a schematic or symbol

Tip

Since Xschem is a text-based schematic editor, you can also open `.sch` and `.sym` files in a text editor and modify them there. This is especially useful for search-and-replace operations.

4.2 Drawing the Inverter Schematic

We now draw the schematic shown in Figure 5 step by step. The reference is `inverter.sch`.

1. **Place the transistors.** Press `Ins`, and pick `sg13_lv_nmos.sym` and `sg13_lv_pmos.sym` from the PDK library. Place the NMOS M_1 at the bottom and the PMOS M_2 at the top.
2. **Set the device parameters.** Select a transistor and press `q`. Enter the values from Table 5, for example for M_1: `l=1.0u`, `w=20.0u`, `ng=20`, `m=1`. Note that `w` is the total width and `ng` is the number of fingers. Keep `spiceprefix=X`, because the PSP transistor models of the PDK are SPICE subcircuits and must be instantiated with an `X` prefix.

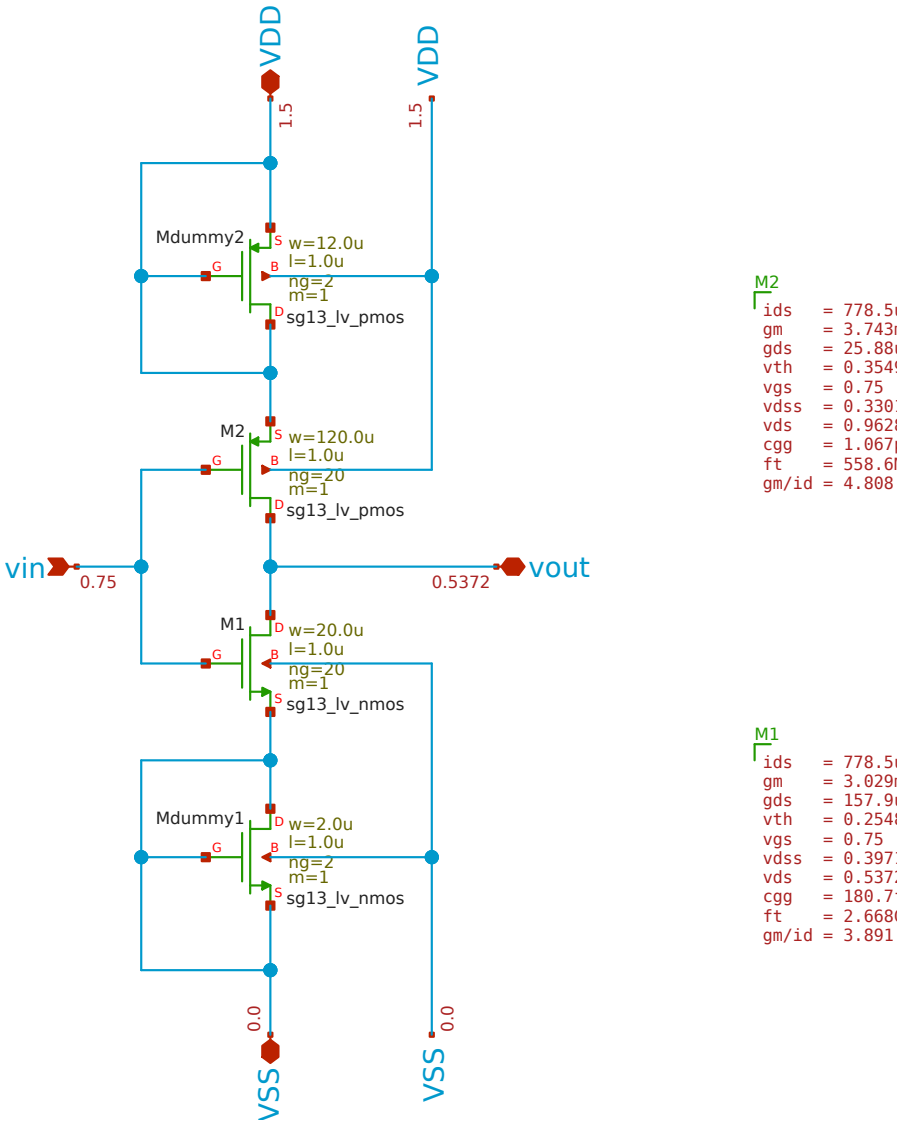

Figure 5: The finished `inverter` schematic in Xschem, with dummy transistors and operating-point annotations.

3. **Place the dummy transistors.** Copy each device with `c` and change the properties to the dummy sizing (`w=2.0u`, `ng=2` for the NMOS and `w=12.0u`, `ng=2` for the PMOS). Wire all four dummy terminals to the respective supply rail (gate, source, drain, and bulk of M_{dummy1} to `VSS`, those of M_{dummy2} to `VDD`), so the dummies are off and only their geometry matters.
4. **Wire the circuit.** Press `w` and connect the gates of M_1 and M_2 to the input net, the drains to the output net, the NMOS source and bulk to `VSS`, and the PMOS source and bulk to `VDD`.
5. **Add the pins.** Press `Ins` and place `ipin.sym` for the input `vin`, `iopin.sym` for the output `vout`, and `iopin.sym` for `VDD` and `VSS`. The pin names define the port names of the subcircuit later used for LVS, so name them exactly `vin`, `vout`, `VDD`, and `VSS`.
6. **Annotate (optional).** Place `annotate_fet_params.sym` next to M_1 and M_2 and set `ref=M1` and `ref=M2`. After a simulation, these display the operating-point parameters of the devices directly in the schematic (as visible in Figure 5).
7. **Save** with `Ctrl + s` as `inverter.sch`.

4.3 Drawing the Inverter Symbol

To use the inverter in a testbench, it needs a symbol. Figure 6 shows the finished symbol, the reference is `inverter.sym`.

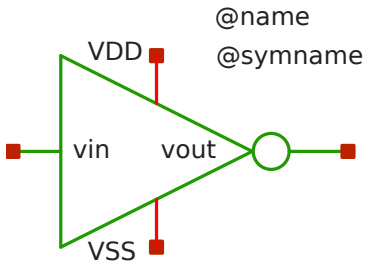

Figure 6: The `inverter` symbol in Xschem.

1. **Auto-generate the symbol.** With `inverter.sch` open, press `A` (Shift + a). Xschem creates `inverter.sym` with one pin per schematic pin, placed on a rectangular outline.
2. **Open the symbol.** Load it with `Ctrl + o` (or select a placed instance and press `i`).
3. **Redraw the shape.** Delete the auto-generated rectangle and draw the triangle with the output bubble using the line (`l`) and arc tools. Keep the pin boxes, they carry the electrical information (`name` and `dir` attributes).
4. **Arrange the pins.** Place `vin` on the left, `vout` on the right, `VDD` on top, and `VSS` at the bottom. Add the small text labels with `t`.
5. **Check the global properties.** Press `q` with nothing selected. The symbol must have `type=subcircuit` and `format="@name @pinlist @symname"`. With `type=subcircuit`,

Xschem descends into `inverter.sch` during netlisting and emits the full transistor-level subcircuit.

6. **Save** the symbol.

You can now place the symbol in any schematic with `Ins`. Select it and press `e` to descend into the underlying circuit, and `Ctrl + e` to go back up.

4.4 Completing the Testbenches

Three testbenches characterize the inverter. Skeletons are already prepared in `testbenches/xschem/`, and we finish them together:

- `inverter_tb_dc_vout.sch` sweeps the input from 0 to V_{DD} and records the voltage transfer characteristic (see Figure 7).
- `inverter_tb_ac_ol.sch` measures the open-loop frequency response (see Figure 8).
- `inverter_tb_tran.sch` applies a small sine around the operating point and simulates in the time domain (see Figure 9).

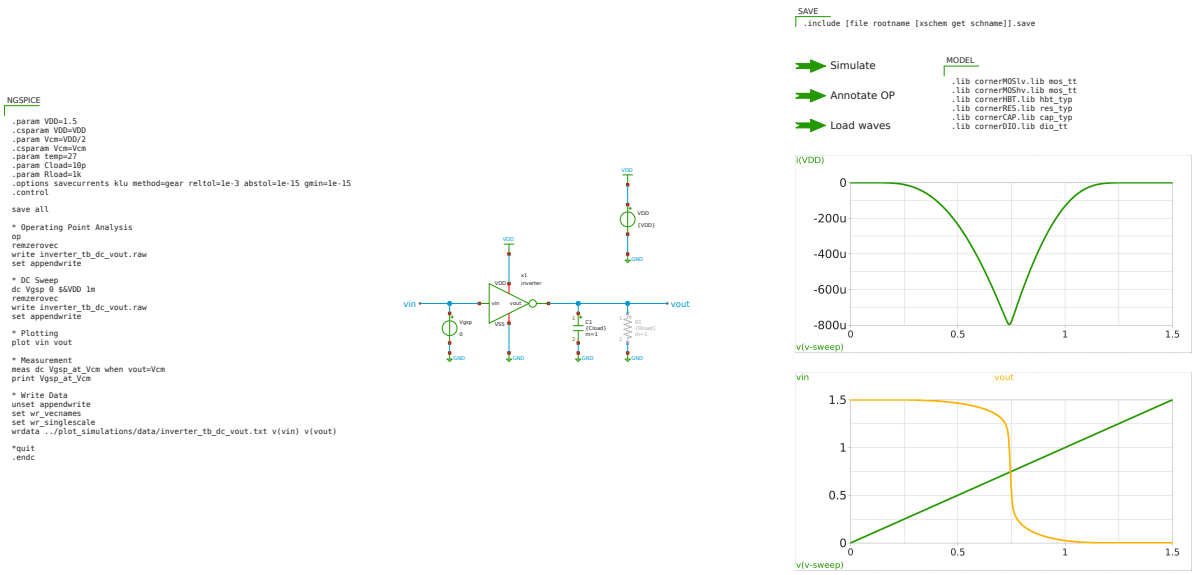

Figure 7: Testbench `inverter_tb_dc_vout` (dc input sweep).

Each testbench is operated with the three launcher buttons in the schematic. Hold `Ctrl` and click **Simulate** to netlist and start Ngspice, **Load waves** to load the results into the integrated waveform viewer (the embedded graphs), and **Annotate OP** to back-annotate the operating point into the schematic.

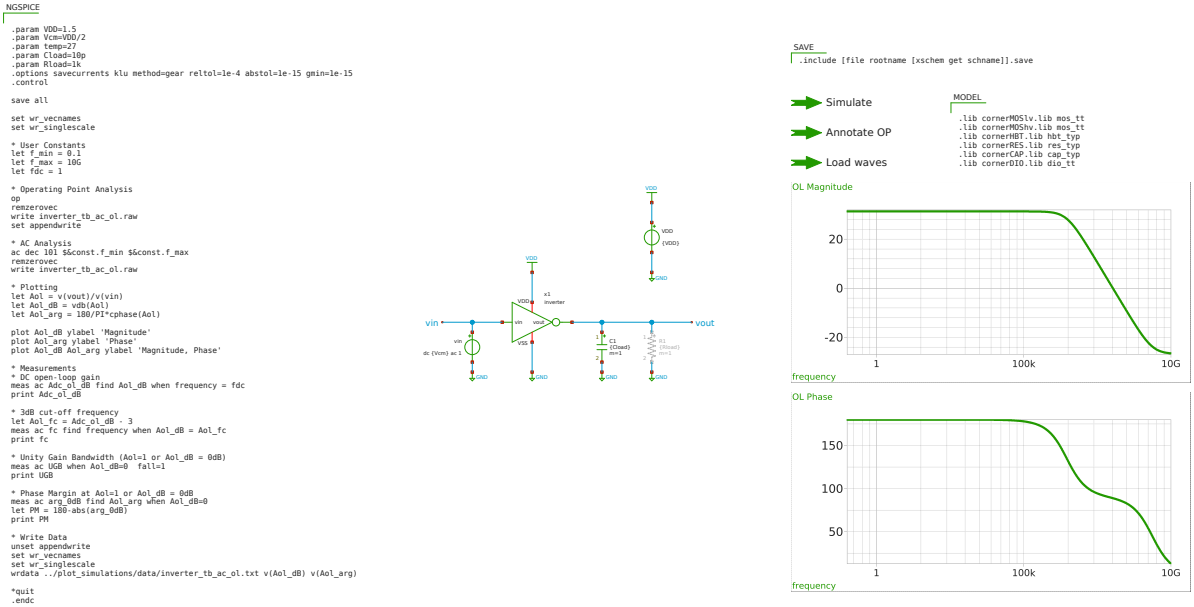

Figure 8: Testbench `inverter_tb_ac_ol` (open-loop ac characterization).

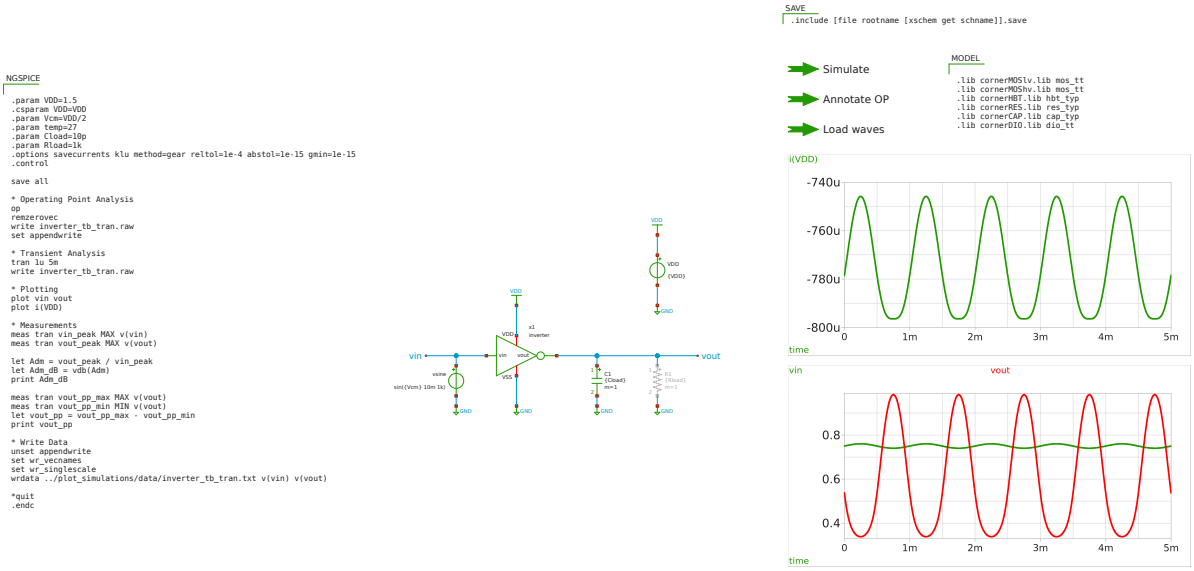

Figure 9: Testbench `inverter_tb_tran` (transient response).

4.4.1 Anatomy of the AC Testbench

We look at the ac testbench in detail, because it contains every ingredient a complete analog testbench needs. Open `inverter_tb_ac_ol.sch` and identify the following parts:

1. The device under test. An instance `x1` of `inverter.sym`. Two spare instances `x2` (`inverter.sym`) and `x3` (`inverter_pex.sym`) are also placed, with the attribute `spice_ignore=true` so they are excluded from the netlist. They are used later for the post-layout comparison in Section 5.5.

2. Supplies and bias. A voltage source `VDD` with `value={VDD}` powers the circuit between the `VDD` rail and ground. The input source `vin` has the value

```
dc {Vcm} ac 1
```

The `dc {Vcm}` part biases the input at the switching threshold $V_{\mathrm{cm}} = V_{\mathrm{DD}}/2$, where the inverter has its highest gain. The `ac 1` part sets the small-signal stimulus amplitude to 1 V. This does not overdrive the circuit, because the ac analysis linearizes the circuit around the operating point. With an input of exactly 1, the output magnitude directly equals the voltage gain, so `v(vout)/v(vin)` is a normalized transfer function.

3. The load. A capacitor `C1` with `value={Cload}` (10 pF) loads the output. A load resistor `R1` is placed as well but disabled with `spice_ignore=true`. Toggle it with `Shift + T` if you want to study resistive loading.

4. The model includes. A `code_shown.sym` block named `MODEL` pulls in the PDK device models with the typical corner:

```
.lib cornerMOSlv.lib mos_tt
.lib cornerMOShv.lib mos_tt
.lib cornerRES.lib res_typ
.lib cornerDIO.lib dio_tt
```

The bare file names work because the shared Ngspice configuration `config/.spiceinit` adds the PDK model directories to the Ngspice source path and loads the compiled PSP OSDI models. The Nix shell points Ngspice at this file automatically. To simulate a different process corner, replace `mos_tt` with, for example, `mos_ss` or `mos_ff`.

5. The control block. A second `code_shown.sym` block named `NGSPICE` contains the parameters, simulator options, and the `.control` section:

```
.include ../../../netlist/pex/inverter_magic_pex_3.spice
.param VDD=1.5
.param Vcm=VDD/2
.param temp=27
.param Cload=10p
.options savecurrents klu method=gear reltol=1e-4 abstol=1e-15 gmin=1e-15
.control
save all

* User Constants
let f_min = 0.1
let f_max = 10G
let fdc = 1

* Operating Point Analysis
op
remzerovec
write @schname.raw
set appendwrite

* AC Analysis
ac dec 101 $&const.f_min $&const.f_max
remzerovec
write @schname.raw
...
.endc
```

Read it top to bottom:

- The `.include` line loads the extracted PEX netlist, which defines the subcircuit `inverter_pex`. It is not used while `x1` is active, but it must be present for the post-layout swap in Section 5.5.
- The `.param` lines define the supply, the input bias, the temperature, and the load as parameters, so one edit changes the whole testbench.
- The `.options` line selects the KLU sparse solver and tightens the tolerances for a clean ac result.
- Inside `.control`, first an operating point analysis (`op`) runs and its result is written to the `.raw` file. This is what **Annotate OP** reads.
- Then the ac analysis sweeps 101 points per decade from 0.1 Hz to 10 GHz and appends the result to the same `.raw` file.

6. The measurements. After the analysis, the control block computes the open-loop gain and extracts the key numbers:

```
let Aol = v(vout)/v(vin)
let Aol_dB = vdb(Aol)
let Aol_arg = 180/PI*cphase(Aol)

meas ac Adc_ol_dB find Aol_dB when frequency = fdc    * dc gain at 1 Hz
let Aol_fc = Adc_ol_dB - 3
meas ac fc find frequency when Aol_dB = Aol_fc        * -3 dB cut-off frequency
meas ac UGB when Aol_dB=0 fall=1                      * unity-gain bandwidth
meas ac arg_0dB find Aol_arg when Aol_dB=0
let PM = 180-abs(arg_0dB)                             * phase margin
```

The phase starts at 180° because the inverter is an inverting amplifier. With the sizing from Section 3.1 you should measure approximately $A_{\mathrm{ol}} = 31.3\,\mathrm{dB}$, $f_{\mathrm{c}} = 2.9\,\mathrm{MHz}$, and UGB $= 107\,\mathrm{MHz}$.

7. The data export. The final `wrdata` command writes the traces to `plot_simulations/data/`, so the Python plotting script and the batch flow from Section 3.2 can use them.

8. Graphs and the save file. The embedded graph boxes display magnitude and phase after **Load waves**. A small `SAVE` code block includes `<testbench>.save`, which lists the nodes to store.

> **Tip**
>
> The dc and transient testbenches follow exactly the same pattern. Only the analysis command changes: `dc Vgsp 0 $&VDD 1m` sweeps the input source in 1 mV steps, and `tran 1u 5m` simulates 5 ms with a sine input `sin({Vcm} 10m 1k)`, a 10 mV amplitude at 1 kHz. Complete these two testbenches yourself. In the dc sweep, at which input voltage does `vout` cross $V_{\mathrm{cm}} = 0.75\,\mathrm{V}$? The simulation reports approximately 0.745 V, slightly below $V_{\mathrm{DD}}/2$.

4.5 Generating the PEX Symbol

For the post-layout simulation in Section 5.5 we prepare a second symbol now, so that the extraction flow can reorder its netlist pins to match it. The reference is `inverter_pex.sym`.

1. Copy the symbol file: `cp inverter.sym inverter_pex.sym`.
2. Open `inverter_pex.sym` in Xschem and press `q` with nothing selected to edit the global properties.
3. Change the type from `subcircuit` to `primitive`.

The two settings behave differently during netlisting:

- `type=subcircuit`: Xschem descends into the matching schematic (`inverter.sch`) and writes the full transistor-level subcircuit into the netlist. The instance line references that generated subcircuit.
- `type=primitive`: Xschem does not descend and only writes the instance line `x1 vin vout VDD VSS inverter_pex`. The subcircuit definition must come from somewhere else, in our case from the extracted PEX netlist pulled in with the `.include` line of the testbench.

Since the symbol file is named `inverter_pex.sym`, the `@symname` in the instance line resolves to `inverter_pex`, which matches the subcircuit name that the PEX flow generates. The pin order of this symbol is also the reference that `make magic-pex` uses to reorder the pins of the extracted netlist, so schematic symbol and extracted subcircuit always stay compatible.

5 Hands-On: Layout

In this session we draw the layout of the inverter in KLayout and verify it with DRC, LVS, and PEX.

5.1 Getting Started with KLayout

Open KLayout in edit mode with the `sg13cmos5l` technology preloaded from the repository root:

```
make klayout
```

Some basics before drawing:

- **Data base unit.** When creating a new layout (`File > New Layout`), set the database unit (DBU) to 0.001 m. This is important for the IHP Open-PDK.
- **Layers.** Each drawing layer has three roles: `MetalX.drawing` carries the actual geometry, `MetalX.pin` marks pin shapes, and `MetalX.text` carries the pin labels used by LVS.
- **PCells.** The PDK provides parametric cells (PCells) in the `SG13_dev` library, for example `nmos`, `pmos`, resistors (`rppd`, `rsil`, `rhigh`), capacitors (`cap_mfringe`, `moscap_n`, `moscap_p`), taps (`ntap1`, `ptap1`), the `guard_ring`, and `via_stack`.

The most important shortcuts are listed in Table 8, taken from the KLayout cheatsheet in this repository.

Table 8: Essential KLayout shortcuts with the productivity plugins installed.

Shortcut	Action
`Q`	Properties of the selected object / PCell
`M`	Move quickly
`A`	Align tool
`C`	Copy object
`P`	Path / drawing tool (double-click to finish)
`R`	Draw rectangle
`T`	Place text
`Shift + P`	Set pin (input, output, …) for LVS
`O`	Place vias during routing (path mode)
`E` / `Ctrl + E`	Descend into / ascend from an instance
`Shift + R`	Rotate instance by 90°
`Shift + H` / `Shift + V`	Flip instance horizontally / vertically
`F`	Zoom full
`U` / `Shift + U`	Undo / redo
`K` / `Shift + K`	Ruler+ / clear all rulers
`0` … `9`	Layer focus (see below)

5.2 The KLayout Productivity Suite

The KLayout Productivity Suite is a collection of plugins developed at JKU that make manual layout much faster. They were installed with `make klayout-setup` in Section 1, or are already present from an earlier installation, since they live in your user directory `~/.klayout/salt/`. The full documentation with animations is here. A short summary:

Table 9: The KLayout productivity plugins in a nutshell.

Plugin	What it does
Move Quickly Tool (`M`)	Rapid repositioning of shapes and instances with mouse or keyboard, including diagonal moves and multi-object selection
Align Tool (`A`)	Align objects to each other by clicking reference features (middle, corner, or edge markers)
Pin Tool (`Shift + P`)	Place LVS-ready pins (pin shape plus text label) with automatic layer detection from the current layer selection
Layer Shortcuts (`0…9`)	Fast layer visibility switching: `1…7` focus on one metal with its vias, `8` gate poly, `9` diffusion, `0` shows the default layers, `,` hides them, `Shift + 1…Shift + 9` extend the current focus by one more layer group

Plugin	What it does
Netlist Import (`File > Import > Netlist`)	Import a SPICE netlist and place the cell instances with technology-specific mapping. Great for starting a layout from the schematic netlist
Library Manager (`File > Cell Library Manager`)	Manage hierarchical layouts and cell libraries, and export a clean layout for tapeout
Auto Backup (`File > Automatic Backups`)	Periodic layout backups with configurable interval and rotation
Vector File Export (`File > Export Vector File`)	Export the layout view as PDF or SVG

5.3 Drawing the Inverter Layout

We now draw the layout of the `inverter` cell. The finished result is shown in Figure 10.

1. **Create the cell.** `File > New Layout`, top cell name `inverter`, DBU 0.001 m.
2. **Place the transistor PCells.** Drag `nmos` and `pmos` from the `SG13_dev` library into the layout. Place the PMOS row above the NMOS row.
3. **Match the schematic.** Select each PCell and press `Q`. Set the same `w`, `l`, and `ng` values as in the schematic (Table 5): the NMOS with $20 \times 1.0\,\mathrm{m}$ fingers and the PMOS with $20 \times 6.0\,\mathrm{m}$ fingers, both with $L = 1.0\,\mathrm{m}$. Add the two dummy devices next to the active devices in the same way.
4. **Substrate and well contacts.** Place `ptap1` / `ntap1` contacts (or a `guard_ring`) to tie the substrate to `VSS` and the n-well of the PMOS to `VDD`.
5. **Route.** Connect the gates (input), the drains (output), and the source rails with `Metal1` and `Metal2`. Use the path tool `P`, place vias with `O` or with `via_stack` PCells. Keep signal routing on the lower metals within a macro. Use the layer shortcuts `1`, `2`, `8`, `9` to focus on the layer you are working on.
6. **Add the pins.** Select the target metal layer and press `Shift + P` to place the pins `vin`, `vout`, `VDD`, and `VSS`. The pin tool places the pin shape on `MetalX.pin` and the label on `MetalX.text`, which is exactly what LVS expects.
7. **Save** as `layout/inverter.gds`.

> **Tip**
>
> The layout drawing order follows the matching rules of analog design: active devices in the center with common orientation, dummies at the edges, and taps close to the devices. Also check the layout tips in the KLayout cheatsheet.

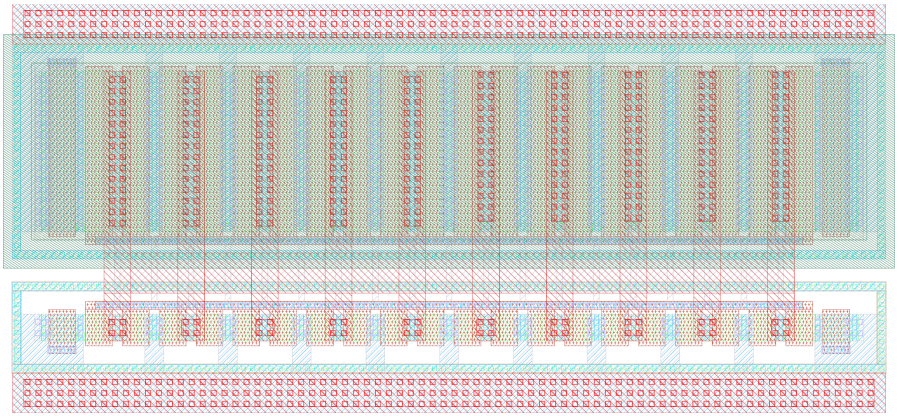

Figure 10: Render of the finished `inverter` layout.

5.4 DRC, LVS, and PEX

The layout is verified with the Makefile targets from Section 2.3, executed in the `inverter/` folder. Both KLayout and Magic + Netgen flows are available. The `sak-*` scripts driving them are vendored from IIC-OSIC-TOOLS in `scripts/`.

5.4.1 Design Rule Check

```
make klayout-drc      # KLayout DRC, default DRC_LEVEL=macro
make magic-drc        # Magic DRC (full rule set)
```

The `DRC_LEVEL` variable selects the KLayout rule set: `precheck` runs only the core manufacturing rules for fast iteration, `macro` (the default) adds off-grid, zero-area, and pin checks for block-level sign-off, and `regular` adds density and antenna checks for full-chip sign-off. Reports are written to `verification/drc/`. A successful run ends with:

```
INFO | ================================================================================
INFO | KLayout DRC Check Passed: No DRC violations detected in the layout.
INFO | ================================================================================
```

> **Tip**
>
> In the KLayout GUI, a DRC run can be launched with `F9`, and the DRC options are behind `F10`. Violations are then browsed interactively in the marker database.

5.4.2 Layout Versus Schematic

```
make klayout-lvs      # CDL netlist exported from Xschem, compared by KLayout
make magic-lvs        # SPICE netlist exported from Xschem, compared by Magic + Netgen
```

Both targets first export the schematic netlist from Xschem automatically, then extract the layout netlist and compare the two. Reports are written to `verification/lvs/`. The `ntap` and `ptap` substrate contacts are ignored during LVS in both flows, so the taps in the layout do not need matching schematic devices. A successful KLayout run prints:

```
INFO | | Status           | PASS                                    |
INFO | | Mode             | COMPARE                                 |
INFO | | Outcome          | Comparison mode: PASS (netlists match). |
INFO | | Warnings         | 0                                       |
INFO | | Errors           | 0                                       |
```

The Magic + Netgen flow reports:

```
Cell pin lists are equivalent.
Device classes inverter and inverter are equivalent.

Final result: Circuits match uniquely.
```

> **Note**
>
> The Netgen report additionally lists property warnings about a `mm_ok` property that exists only on the layout side. This warning is expected with the current PDK and can be ignored as long as the final result reads `Circuits match uniquely.`

> **Tip**
>
> In the KLayout GUI, an LVS run can be launched with `F11`, and the LVS options are behind `F12`, where the exported schematic netlist (`netlist/schematic/inverter_klayout.cdl`) has to be selected manually. Set `deep` as the run mode.

5.4.3 Parasitic Extraction

```
make magic-pex                  # full-RC extraction (EXT_MODE=3, default)
make magic-pex EXT_MODE=2       # C-coupled capacitances only
make magic-pex EXT_MODE=1       # C-decoupled capacitances only
```

PEX extracts the parasitic capacitances and resistances of the drawn wires into a SPICE netlist in `netlist/pex/`, for example `inverter_magic_pex_3.spice`. The subcircuit inside is named `inverter_pex`, and its pin order is automatically rearranged to match the PEX symbol from Section 4.5. A port check runs at the end and fails the target if any pin of the extracted subcircuit is left floating.

For the full-RC mode, three tuning parameters control how much of the resistor network is kept (`THRESHOLD`, `MINRES`, `MINDELAY`). The defaults are fine for this workshop, details are in the inverter README.

5.5 Post-Layout Simulation

Now we close the loop and simulate the extracted layout in the same testbenches:

1. Open a testbench, for example `inverter_tb_ac_ol.sch`.
2. Select the active instance `x1` (`inverter.sym`) and press `Shift + T` to set its `spice_ignore` flag. It is now excluded from the netlist.
3. Select the spare instance `x3` (`inverter_pex.sym`) and press `Shift + T` to activate it. Wire positions are identical, so nothing else changes.
4. The `.include ../../../netlist/pex/inverter_magic_pex_3.spice` line at the top of the control block now supplies the subcircuit definition. If you extracted with a different `EXT_MODE`, adapt the file name.
5. Re-run the simulation with `Ctrl` + click on **Simulate**.

Compare the post-layout results against the schematic-level results. What differences do you see, and why?

Think about it before simulating. The extracted netlist adds the wiring resistance and the capacitance of every drawn wire to the circuit. The dc transfer characteristic should barely change, because almost no current flows into the parasitics at dc. In the ac response, the added output and internal capacitances and the interconnect resistance shift the poles, so expect a slightly reduced bandwidth and small changes in the measured gain. If your post-layout results deviate strongly, that is usually a sign of a layout problem, for example a too narrow or too long wire on a critical net.

6 Exercises

Now it is your turn. Pick one of the two exercises (or do both if you are fast). They reuse the complete flow of Section 4 and Section 5 on a modified circuit. Work on your own, and ask us whenever you get stuck.

> **Note**
>
> A reference implementation of the exercises will be provided in a separate branch of the repository after the workshop.

6.1 Exercise 1: Self-Biased Single-Ended Inverter Amplifier

The bare CMOS inverter has poor PVT robustness as an amplifier, because nothing pins the input bias to the high-gain point at $V_{\mathrm{DD}}/2$. A feedback resistor R_{f} between output and input closes a dc loop that forces the operating point to the crossover ($V_{\mathrm{in}} \approx V_{\mathrm{out}} \approx V_{\mathrm{DD}}/2$) regardless of the process corner. This also linearizes the transfer characteristic and turns the inverter into a usable single-ended amplifier. The price is a reduced input resistance of the resulting amplifier.

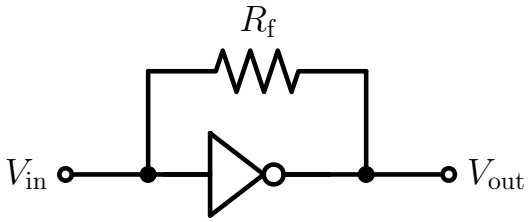

Figure 11: Self-biased inverter amplifier with feedback resistor R_{f}.

Recommended steps:

1. Copy the macro folder and delete the generated files:

```
cp -r inverter amp
cd amp
make clean
```

2. Rename `TOP` in the `Makefile` to `TOP = amp`. All targets derive their file paths from `TOP` (and `CELL`, which defaults to `TOP`), so the design files must carry the same name.
3. Rename the Xschem schematic, symbol, and testbench files:

```
for f in schematic/xschem/inverter* testbenches/xschem/inverter*; do
    mv "$f" "$(echo "$f" | sed 's/inverter/amp/')"
done
```

Since Xschem files are plain text, update the remaining `inverter` references inside the renamed files with search-and-replace, for example the `inverter.sym` instances and the `.include` of the PEX netlist in the testbenches.

4. Rename the KLayout layout file and the top cell:

   ```
   mv layout/inverter.gds layout/amp.gds
   ```

   Also rename the top cell inside the GDS from `inverter` to `amp`: open the layout in KLayout, rename the cell, and save. The verification targets require matching file and top-cell names.

5. Open `amp.sch` in Xschem and add the feedback resistor between the `vin` and `vout` nets, as shown in Figure 11. Use the `rhigh` device of the PDK (`sg13cmos5l_pr/rhigh.sym`), the high-sheet poly resistor, and set $W = 0.5\,\mathrm{m}$ and $L = 25\,\mathrm{m}$ with `q`. Keep `spiceprefix=X` and connect the `body` terminal to `VSS`. The symbol computes its value from the geometry, which gives $R_\mathrm{f} \approx 74\,\mathrm{k}$ for these dimensions.

6. Think first, then simulate. What do you expect to happen to the dc transfer characteristic, the ac response, and the measured gain? Then re-run the three testbenches and check whether the results match your expectations. Note that the ac open-loop testbench drives `vin` directly with a source, so to observe the self-biasing effect, also try disconnecting the input bias and let the feedback set the operating point.

7. Vary R_f and observe the trade-off. Too small, and the resistor loads the amplifier output and reduces the gain. Too large, and the time constant with the input capacitance limits the low end of the signal band. The resistance scales with the length, roughly 3 k per m at $W = 0.5\,\mathrm{m}$, so $L = 10\,\mathrm{m}$ gives about 30 k and $L = 100\,\mathrm{m}$ about 300 k. Keep W at 0.5 m and sweep L.

8. Propagate the resistor to the layout (`layout/amp.gds`) with the `rhigh` PCell, using the same $W = 0.5\,\mathrm{m}$ and $L = 25\,\mathrm{m}$ as in the schematic (also place dummy resistors of minimum length above and below it), and re-run the verification chain with `make klayout-verify` and `make magic-verify`.

6.2 Exercise 2: Three-Stage Ring Oscillator with Output Buffer

An odd number of inverters in a loop has no stable operating point and oscillates. Build a three-stage ring oscillator from the inverter cell and use a fourth inverter as an output buffer that decouples the ring from the load, as shown in Figure 12.

Recommended steps:

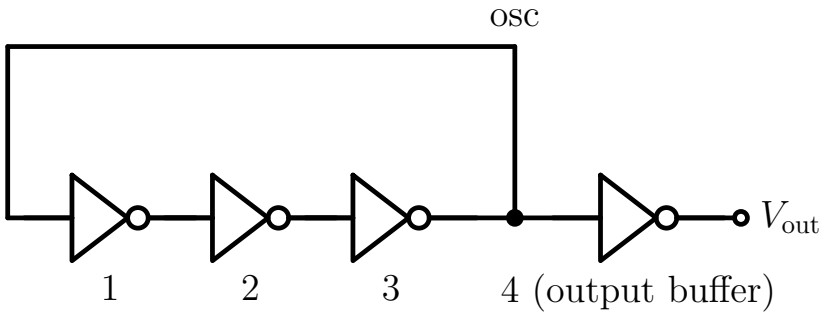

Figure 12: Three-stage ring oscillator with the fourth inverter as output buffer.

1. Follow the same setup recipe as in Section 6.1 with `ringosc` instead of `amp`: copy the folder, run `make clean`, rename `TOP` in the `Makefile` to `TOP = ringosc`, rename the Xschem schematic, symbol, and testbench files, and rename the KLayout layout file to `layout/ringosc.gds`. One difference: do not rename the top cell inside the GDS yet, the `inverter` cell stays in use as the unit cell (see the last step).

2. Restore the unit cell from the original macro, you instantiate it four times:

```
cp ../inverter/schematic/xschem/inverter.sch ../inverter/schematic/xschem/inverter.sym s
```

3. Open `ringosc.sch` (it still contains the copied transistor-level inverter), delete its content, and place four instances of `inverter.sym`. Connect three of them in a ring, and tap the ring node with the fourth inverter as the buffer. Add the pins `vout`, `VDD`, and `VSS`.

4. Update the symbol `ringosc.sym` and the PEX symbol `ringosc_pex.sym`: remove the `vin` pin, the ring oscillator has no input. Alternatively, regenerate the symbol from the schematic (press `A`), following Section 4.3 and Section 4.5.

5. Adapt the renamed transient testbench `ringosc_tb_tran.sch`: remove the input source, and measure the oscillation frequency. Since the delay per stage is small, simulate a few hundred nanoseconds with a fine step. Measure the frequency with two `meas tran` commands that time consecutive rising crossings of V_{cm}, and compute $f_{\text{osc}} = 1/T_{\text{osc}}$ from them:

```
meas tran t_rise1 when v(vout)=Vcm rise=1
meas tran t_rise2 when v(vout)=Vcm rise=2
let T_osc = t_rise2 - t_rise1
let f_osc = 1/T_osc
print f_osc
```

   Add `.csparam Vcm=Vcm` next to the `.param Vcm=VDD/2` line, otherwise `Vcm` is not visible inside the `.control` block. Make sure the stored transient window contains at least two

rising edges. The expected relation is $f_{\mathrm{osc}} = 1/(2 \cdot N \cdot t_{\mathrm{d}})$ with $N = 3$ stages and t_{d} the effective stage delay. Delete the testbenches that are no longer relevant.

6. Think about the startup. An ideal simulation can rest on the metastable point where all stages sit at V_{cm}. If the ring does not start, kick it with an initial condition, for example `.ic v(osc)=0`.

7. Resize the inverter and watch the frequency. Open `inverter.sch` and vary W, L, and the finger count `ng` of M_1 and M_2, then re-run the transient simulation for each variant. Predict the outcome before you simulate, and use Section 3.1.1 as your guide.

8. Draw the top-level layout in `layout/ringosc.gds`, following Section 5.3: create a new top cell `ringosc`, place four instances of the existing `inverter` cell, and route the ring. Re-run DRC and LVS, extract the parasitics with `make magic-pex`, and compare the post-layout oscillation frequency against the schematic level. The parasitics add delay per stage, so the frequency drops.

7 Acknowledgements

This workshop was created for HeiChips 2026, organized by the Novel Computing Technologies group at Heidelberg University. It is based on the ihp-sg13g2 AMS chip design tutorial by Simon Dorrer and Harald Pretl, Institute for Integrated Circuits and Quantum Computing, Johannes Kepler University Linz. Many thanks to the maintainers of the open-source tools and the IHP Open-PDK that make this flow possible.