102726
Design of Analog and Mixed Integrated Circuits and Systems
Lab Manual

Francesc Serra Graells
http://www.cnm.es/~pserra/uab/damics
paco.serra@imb-cnm.csic.es

Contents

1 Objectives **2**

2 Installation **2**

3 CMOS Technology **5**
3.1 Design Rules 6
3.2 Device Models 7

4 My Operational Amplifier **10**
4.1 Schematic Entry 10
4.2 Electrical Simulation 14
4.3 Circuit Optimization 19
4.4 PCell-Based Netlist-Driven Layout Design 21
4.5 Design Rule Checker 34
4.6 Circuit Extraction and Electrical Rule Checker 39
4.7 Layout Versus Schematic 45
4.8 2D and 3D Parasitics Extraction 51
4.9 Post-Layout Simulation 59
4.10 Tape-Out 59

Glossary **60**

References **61**

1 Objectives

The aim of these lab exercises is to introduce the general electronic design automation (EDA) methodology of Fig. 1 for full-custom analog integrated circuit (IC) design. For this purpose, a practical design case based on a simple operational amplifier (OpAmp) circuit is developed step-by-step, from the metal-oxide-semiconductor (MOS) transistor schematic to the physical mask layout.

☞ Students will deliver the following **computer files** for evaluation:

- The OpAmp **design report** in portable document format (PDF) containing the answers to questions **Q1-Q10** together with all the required supporting materials (e.g. hand calculations, simulation plots. . .).
- The OpAmp **layout** in graphic database system II (GDSII) format.

Before starting with the lab exercises of Section 4, the installation of the required EDA tools and the details of the target complementary metal-oxide-semiconductor (CMOS) technology are discussed in next two sections, respectively.

2 Installation

The EDA tools chosen for the design flow of Fig. 1 are freely available for both MS Windows and Linux operative systems. In particular:

Glade (GDS, LEF and DEF editor) by Peardrop Design Systems is an IC schematic and mask layout editor, programmable netlister and physical verification tool featuring Python language scripting. More information can be found at `http://www.peardrop.co.uk/glade`.

SpiceOpus (SPICE with integrated optimization utilities) by the CACD Group at University of Ljubljana is a port of the Berkeley SPICE3 electrical simulator featuring NUTMEG language scripting, together with a custom optimization tool and the Georgia Tech Research Institute XSpice high-level multi-domain event-driven engine. The resulting simulation suite can perform native mixed-signal circuit and system simulation and optimization. More information can be found in [1] and at `http://fides.fe.uni-lj.si/spice`.

The **physical design kit (PDK)** contains all the technological information needed by Glade and SpiceOpus according to Fig. 1, as well as the lab exercises themselves.

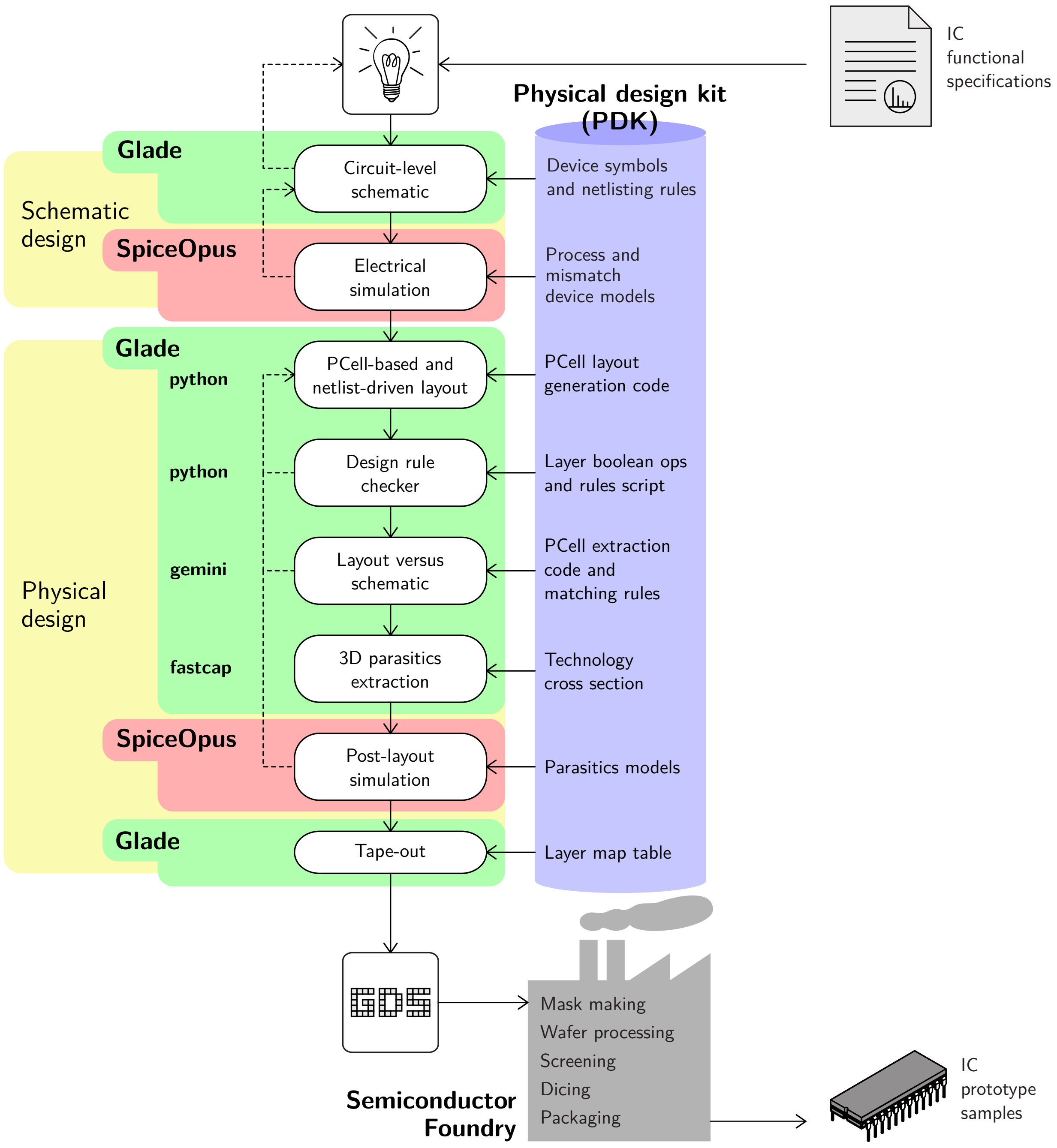


Figure 1 | General full-custom IC design methodology and proposed EDA tools.

The installation instructions to be followed before starting with the lab exercises are described below:

☞ Browse to `http://www.cnm.es/~pserra/uab/damics/lab.html`.

☞ Download and install Glade and SpiceOpus for your operative system (OS).

☞ Download and extract the lab PDK for your OS:

`damics-kit/doc`	This manual!
`/glade`	Schematic and layout library, technology files and verification rules
`/spiceopus`	Simulation models, test netlists and scripts

☞ For **MS Windows**, adjust **red paths** in:

- `glade\glade.bat`:

```
set GLADE_HOME=path_to_glade
set PATH=%GLADE_HOME%;%PATH%
set PYTHONPATH=.;.\pcells;.\verification
set GLADE_LOGFILE_DIR=.
set GLADE_DRC_WORK_DIR=.
set GLADE_DRC_FILE=.\verification\cnm25drc.py
set GLADE_EXT_FILE=.\verification\cnm25xtr_lvs.py
set GLADE_FASTCAP_WORK_DIR=.
del .\*.log
start /b glade.exe -script .\glade_init.py
```

If MS Visual C++ libs (i.e. `MSVCP*.dll`) are not already in your OS, then execute the `vcredist*.exe` installer supplied with Glade.

- `spiceopus\spiceopus.bat`:

```
set OPUSHOME=path_to_spiceopus
set PATH=.;%OPUSHOME%\bin;%PATH%
start /b spiceopus.exe -pw .
```

☞ For **Linux**, adjust **red paths** in:

- `glade/glade.sh`:

```
#! /bin/bash
export GLADE_HOME=path_to_glade
export PATH=${GLADE_HOME}/bin:${PATH}
export LD_LIBRARY_PATH=${GLADE_HOME}/lib:
       ${LD_LIBRARY_PATH}
export PYTHONPATH=.:./pcells:./verification:
       ${GLADE_HOME}/bin:${PYTHONPATH}
export GLADE_LOGFILE_DIR=.
export GLADE_DRC_WORK_DIR=.
export GLADE_DRC_FILE=./verification/cnm25drc.py
export GLADE_EXT_FILE=./verification/cnm25xtr_lvs.py
export GLADE_FASTCAP_WORK_DIR=.
rm ./*.log
glade -script ./glade_init.py &
```

- `spiceopus/spiceopus.sh`:

```
#! /bin/bash
export OPUSHOME=path_to_spiceopus
export PATH=.:${OPUSHOME}/bin:${PATH}
export LD_LIBRARY_PATH=.:${LD_LIBRARY_PATH}
spiceopus -pw . &
```

☞ Glade and SpiceOpus tools must be always launched using the above **scripts**!

3 CMOS Technology

Your OpAmp will be designed for the 2.5μm 2-polySi 2-metal CMOS technology from IMB-CNM(CSIC) (CNM25). The main native devices available from this CMOS process are the bulk P-type and N-type MOS field-effect transistor (MOSFET) and the polySi-insulator-polySi (PiP) capacitor, as shown in Fig. 2. The corresponding physical layers for the full-custom layout design are listed in Table 1.

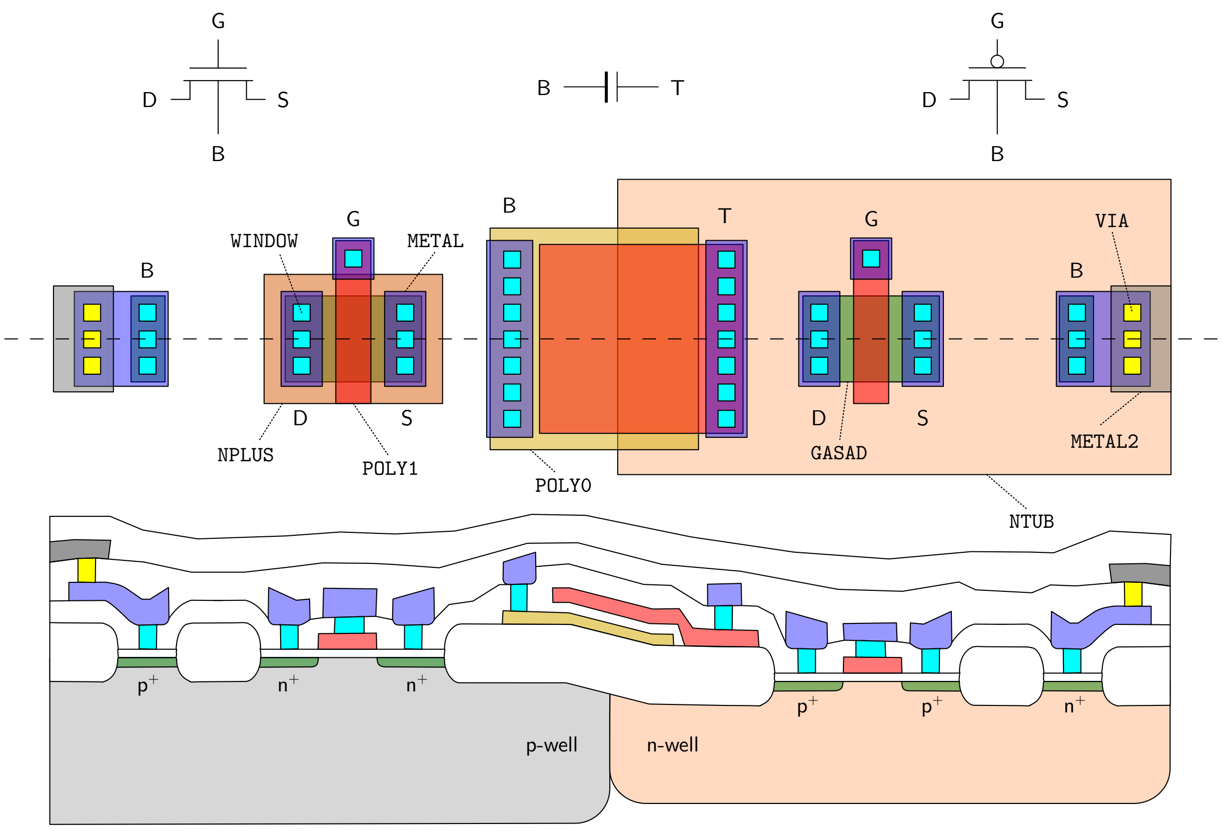


Figure 2 | CNM25 devices (top) and process cross section (bottom). Not to scale.

GDS num.	Name	Explanation
1	NTUB	N-well
2	GASAD	Active area
3	POLY0	PolySi for PiP capacitors only
4	POLY1	PolySi for MOS gate and routing
5	NPLUS	N^+ implant
6	WINDOW	Contact window
7	METAL	Metal 1
9	METAL2	Metal 2
10	VIA	Metal 1-2 via
8	CAPS	Passivation window

Table 1 | CNM25 design layer table.

3.1 Design Rules

In order to ensure the manufacturability of your CMOS circuit, its full-custom layout must be compliant with the CNM25 design rules listed in Table 2.

Ref.	Description
0.0	Design grid is 0.25um x 0.25um
1.1	N-well width >= 8um
1.2	N-well spacing and notch >= 8um
2.1	GASAD width >= 2um
2.2	GASAD spacing and notch >= 4um
2.3	N-well enclosure of P-plus active >= 5um
2.4	N-well spacing to N-plus active >= 5um
3.1	Poly0 width >= 2.5um
3.2	Poly0 spacing and notch >= 6um
3.3	Poly0 spacing to GASAD >= 6um
4.1.a	Poly1 width inside GASAD >= 3um
4.1.b	Poly1 width outside GASAD >= 2.5um
4.2	Poly1 spacing and notch >= 3um
4.3	GASAD extension of Poly1 >= 3um
4.4	Poly1 extension of GASAD >= 2.5um
4.5	Poly1 spacing to GASAD >= 1.25um
4.6	Poly0 enclosure of Poly1 >= 3um
5.1	N-plus enclosure of GASAD >= 2.5um
5.2	N-plus spacing to P-plus active >= 2.5um
5.3	N-plus spacing to Poly1 inside P-plus active >= 2um
5.4	N-plus extension of Poly1 inside N-plus active >= 1.5um
5.5	N-plus width >= 2.5um
5.6	N-plus spacing and notch >= 2.5um
6.1	Exact contact size = 2.5um x 2.5um
6.2	Contact spacing >= 3um
6.3	GASAD enclosure of Contact >= 1um
6.4	Poly1 enclosure of Contact >= 1.25um
6.5	Poly1 Contact spacing to GASAD >= 2.5um
6.6	Contact spacing to Poly1 inside GASAD >= 2um
6.9	Poly0 enclosure of Contact >= 4um
6.10	Contact spacing to Poly1 & Poly0 >= 4um
7.1	Metal1 width >= 2.5um
7.2	Metal1 spacing and notch >= 3um
7.3	Metal1 enclosure of Contact >= 1.25um
8.1	Exact via size = 3um x 3um
8.2	Via spacing >= 3.5um
8.3	Metal1 enclosure of Via >= 1.25um
8.4	Via spacing to Contact >= 2.5um
8.5	Via spacing to Poly1 >= 2.5um
9.1	Metal2 width >= 3.5um
9.2	Metal2 spacing and notch >= 3.5um
9.3	Metal2 enclosure of Via >= 1.25um
10.1	Exact passivation window size = 100um x 100um

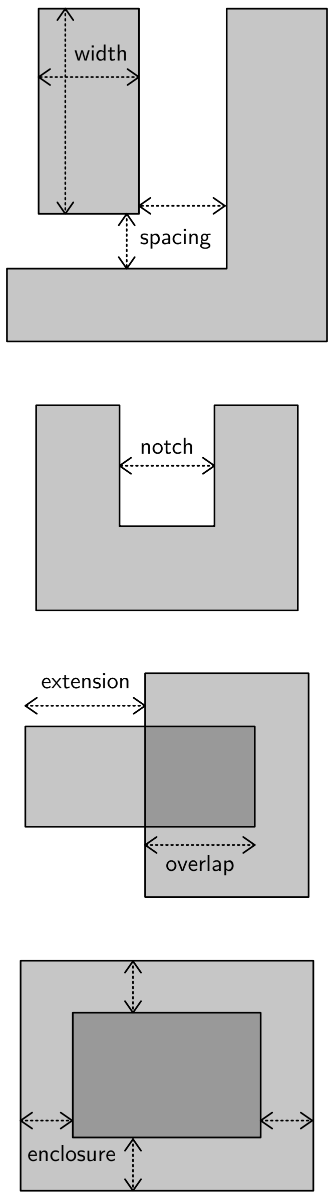


Table 2 | CNM25 design rules (left) and definitions (right).

This PDK comes with tools for assisting the designer during the edition of the full-custom layout, as explained in Section 4.4. However, due to number of rules present in Table 2 and their layer interdependency, the specific physical verification step of Section 4.5 is also included in the methodology of Fig. 1.

3.2 Device Models

The methodology of Fig. 1 predicts the final performance of your IC design through the electrical simulation of its equivalent circuit extracted from the mask layout. For this purpose, the PDK includes `damics-kit/spiceopus/cnm25mod.lib`, the full set of model parameters for the CNM25 devices of Fig. 2 to be used in Simulation Program with Integrated Circuit Emphasis (SPICE). In the case of MOS transistors, the Berkeley Short-channel IGFET Model version 3.3 (BSIM3v3) is employed [2]. In general, two types of parameters are given by the foundry to cover its technology deviations:

Process parameters deal with global deviations at wafer level (e.g. drift in gate oxide thickness), with correlation distances much larger than device dimensions. Hence, process variations affect in the same way all devices in the circuit belonging to a given type (e.g. N-type MOSFETs). Usually, process parameters are defined as worse case conditions, which can be also extended to full process, supply voltage and temperature (PVT) corners. CNM25 SPICE process corners for N-type MOS (NMOS) and P-type MOS (PMOS) transistors and PiP capacitors are combined as typical (`t`), slow (`s`) and fast (`f`) cases of threshold voltage (V_{TO}), carrier mobility (μ_0) and capacitance density (C_{j}).

Matching parameters are intended for local variations at circuit level (e.g. dopant fluctuations), with correlation distances comparable to device dimensions. In this case, variations affect in a different way each component of the circuit, even belonging to the same device type. In general, the technological mismatching of P parameter (ΔP) between a pair of equally designed devices follows the simplified Pelgrom's law [3]:

$$\sigma(\Delta P) = \frac{A_P}{\sqrt{WL}} \tag{1}$$

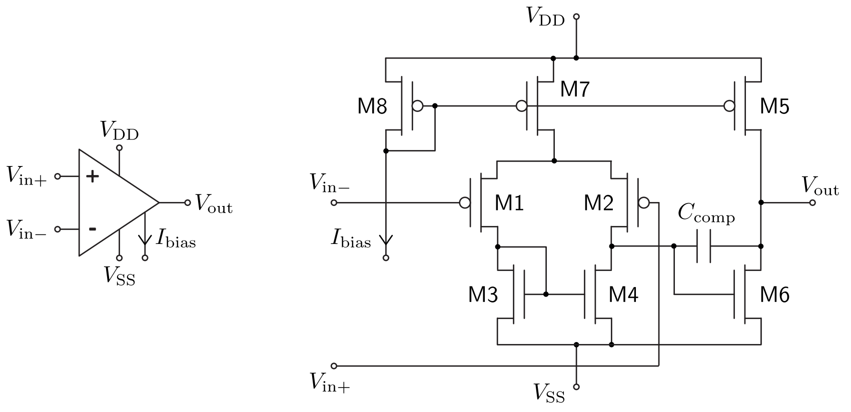
where σ is the Gaussian standard deviation, WL stands for the device area and A_P is the mismatching coefficient of the given technology. CNM25 SPICE mismatching parameters for NMOS and PMOS transistors and PiP capacitors are also supplied as local variations of V_{TO}, μ_0 and C_{j} parameters for Montecarlo simulation.

damics-kit/spiceopus/cnm25mod.lib
```
.lib common
.subckt cnm25modn d g s b param: w=3u l=3u ad=0 as=0 pd=0 ps=0 m=1
.param vth0eff = vth0n+rndgauss(avth0/sqrt(m*w*l))
.param u0eff = u0n*(1+rndgauss(rau0/sqrt(m*w*l)))
.model nbsim nmos
+ LEVEL   = 53
+ VERSION = 3.2.4        TNOM    = 27           TOX     = 3.75E-8
+ XJ      = 1.5E-7       NCH     = 1.7E17       VTH0    = {vth0eff}
+ K1      = 1.17296      K2      = -0.05        K3      = 11.2079
+ K3B     = -1.59332     W0      = 1.00727E-6   NLX     = -1E-9
+ DVT0W   = 0            DVT1W   = 0            DVT2W   = -0.032
+ DVT0    = 4.11104      DVT1    = 0.366189     DVT2    = -0.182099
+ U0      = {u0eff}      UA      = 1.72783E-10  UB      = 5E-18
+ UC      = 4.01727E-11  VSAT    = 1.848E5      A0      = 1.05122
+ AGS     = 0.111468     B0      = 1.6771E-7    B1      = -5.04982E-9
+ KETA    = -0.047       A1      = 0            A2      = 1
+ RDSW    = 3.65E3       PRWG    = 0.0338512    PRWB    = -1E-3
+ WR      = 1            WINT    = 4.55906E-7   LINT    = 9E-7
+ XL      = 0            XW      = 0            DWG     = -2.5492E-8
+ DWB     = 3.22958E-8   VOFF    = -0.124454    NFACTOR = 1.04789
+ CIT     = 0            CDSC    = 2.4E-4       CDSCD   = 0
+ CDSCB   = 0            ETA0    = 0.0354838    ETAB    = -0.07
+ DSUB    = 0.56         PCLM    = 1.96809      PDIBLC1 = 0.482853
+ PDIBLC2 = 0.01           PDIBLCB = 0            DROUT   = 0.415163
+ PSCBE1  = 5.99202E8      PSCBE2  = 5E-5         PVAG    = 0.0141775
+ DELTA   = 3.6636E-3      MOBMOD  = 1            PRT     = 0
+ UTE     = -1.5           KT1     = 0            KT1L    = 0
+ KT2     = 0              UA1     = 4.31E-9      UB1     = -7.61E-18
+ UC1     = -5.6E-11       AT      = 3.3E4        NQSMOD  = 0
+ WL      = 0              WLN     = 1            WW      = 0
+ WWN     = 1              WWL     = 0            LL      = 0
+ LLN     = 1              LW      = 0            LWN     = 1
+ LWL     = 0              CAPMOD  = 2            CJ      = 2.940466E-4
+ PB      = 0.6681951      MJ      = 0.438766     CJSW    = 5.450602E-10
+ PBSW    = 0.4            MJSW    = 0.2725869    TCJ     = 0
+ TPB     = 0              TCJSW   = 0            TPBSW   = 0
+ NOFF    = 1              ACDE    = 1            MOIN    = 15
+ TPBSWG  = 0              TCJSWG  = 0            PRDSW   = -3.85642E3
+ PVSAT   = -1.8E5         CGDO    = 9.89535E-10  CGSO    = 9.89535E-10
+ NOIMOD  = 1              AF      = 1.33         KF      = 1e-29
mn d g s b nbsim w={w} l={l} ad={ad} as={as} pd={pd} ps={ps} m={m}
.ends

.subckt cnm25modp d g s b param: w=3u l=3u ad=0 as=0 pd=0 ps=0 m=1
.param vth0eff = vth0p+rndgauss(avth0/sqrt(m*w*l))
.param u0eff = u0p*(1+rndgauss(rau0/sqrt(m*w*l)))
.model pbsim pmos
+ LEVEL   = 53
+ VERSION = 3.2.4          TNOM    = 27           TOX     = 3.75E-8
+ XJ      = 1.5E-7         NCH     = 1.7E17       VTH0    = {vth0eff}
+ K1      = 0.74278        K2      = -4.93305E-5  K3      = -77.5174
+ K3B     = -3.17908       W0      = 6.70948E-6   NLX     = 1.44524E-7
+ DVT0W   = 0              DVT1W   = 0            DVT2W   = -0.032
+ DVT0    = 1.61621        DVT1    = 0.15752      DVT2    = -0.05
+ U0      = {u0eff}        UA      = 2.65041E-9   UB      = 4.97595E-18
+ UC      = -9.99573E-11   VSAT    = 5E5          A0      = 0.804733
+ AGS     = 0.0783374      B0      = 3.55811E-7   B1      = 2.01182E-10
+ KETA    = -0.047         A1      = 0            A2      = 1
+ RDSW    = 5.41703E3      PRWG    = 0.013649     PRWB    = -1E-3
+ WR      = 1              WINT    = 5E-7         LINT    = 8E-7
+ XL      = 0              XW      = 0            DWG     = -1.44072E-8
+ DWB     = 5.72498E-8     VOFF    = -0.196491    NFACTOR = 0.924527
+ CIT     = 0              CDSC    = 2.4E-4       CDSCD   = 0
+ CDSCB   = 0              ETA0    = 0.3989455    ETAB    = -0.07
+ DSUB    = 0.56           PCLM    = 4.3768578    PDIBLC1 = 0.7281865
+ PDIBLC2 = 0.0140758      PDIBLCB = 0            DROUT   = 0.2398601
+ PSCBE1  = 8E8            PSCBE2  = 5E-5         PVAG    = 0.0099941
+ DELTA   = 0.0634845      MOBMOD  = 1            PRT     = 0
+ UTE     = -1.5           KT1     = 0            KT1L    = 0
+ KT2     = 0              UA1     = 4.31E-9      UB1     = -7.61E-18
+ UC1     = -5.6E-11       AT      = 3.3E4        NQSMOD  = 0
+ WL      = 0              WLN     = 1            WW      = 0
+ WWN     = 1              WWL     = 0            LL      = 0
+ LLN     = 1              LW      = 0            LWN     = 1
+ LWL     = 0              CAPMOD  = 2            CJ      = 3.728047E-4
+ PB      = 0.7982792      MJ      = 0.4562281    CJSW    = 3.946756E-10
+ PBSW    = 0.587129       MJSW    = 0.2658605    TCJ     = 0
+ TPB     = 0              TCJSW   = 0            TPBSW   = 0
+ NOFF    = 1              ACDE    = 1            MOIN    = 15
+ TPB     = 0              TPBSW   = 0            TPBSWG  = 0
+ TCJ     = 0              TCJSW   = 0            TCJSWG  = 0
+ CGDO    = 1.2894E-9      CGSO    = 1.2894E-9
+ NOIMOD  = 1              AF      = 1.33         KF      = 1e-29
```

```
mp d g s b pbsim w={w} l={l} ad={ad} as={as} pd={pd} ps={ps} m={m}
.ends

.subckt cnm25cpoly t b param: w=30u l=30u m=1
.param cjeff = cj*(1+rndgauss(racj/sqrt(m*w*l)))
.model cap c CJ = {cjeff}  CJSW = 0.0
ci t b cap w={w} l={l} m={m}
.ends
.endl

*** Process corners
.lib ttt
.param vth0n = 0.860363
.param vth0p = -1.52069
.param u0n = 0.0573986
.param u0p = 0.0228166
.param cj = 4.227E-4
.param avth0 = 0
.param rau0 = 0
.param racj = 0
.lib 'cnm25mod.lib' common
.endl

.lib sss
.param vth0n = 1.00467
.param vth0p = -1.73564
.param u0n = 0.0367598
.param u0p = 0.0140327
.param cj = 4.650E-4
.param avth0 = 0
.param rau0 = 0
.param racj = 0
.lib 'cnm25mod.lib' common
.endl

.lib fff
.param vth0n = 0.63934
.param vth0p = -1.20160
.param u0n = 0.0780374
.param u0p = 0.0316005
.param cj = 3.804E-4
.param avth0 = 0
.param rau0 = 0
.param racj = 0
.lib 'cnm25mod.lib' common
.endl

*** Montecarlo mismatching: AVto{n,p} = 30mVum, AUo{n,p}/Uo = 5%um and ACj/Cj = 0.5%um
.lib ttt_mc
.param vth0n = 0.860363
.param vth0p = -1.52069
.param u0n = 0.0573986
.param u0p = 0.0228166
.param cj = 4.227E-4
.param avth0 = 30e-9
.param rau0 = 5e-8
.param racj = 5e-9
.lib 'cnm25mod.lib' common
.endl
```
damics-kit/spiceopus/cnm25mod.lib

4 My Operational Amplifier

The CMOS circuit implementation chosen for your OpAmp is the classic two-stage Miller-compensated single-ended output topology of Fig. 3, consisting on two cascaded voltage amplification stages [4]. The first stage includes the PMOS differential pair M1-M2 with current steering M3-M4, all biased by the current mirror M8-M7. The second stage is built around the inverting amplifier M6 biased by the current mirror M8-M5. Finally, the nested capacitor C_{comp} is introduced for the frequency compensation of the multi-pole OpAmp. The initial circuit design parameters are shown in the same figure.



	M	W [μm]	L [μm]
M1,2	1×	12	6
M3,4	1×	12	12
M5	8×	12	6
M6	1×	48	6
M7	2×	12	6
M8	1×	12	6
C_{comp}	1×	100	100

Figure 3 | OpAmp symbol, CMOS schematic and initial device sizing.

4.1 Schematic Entry

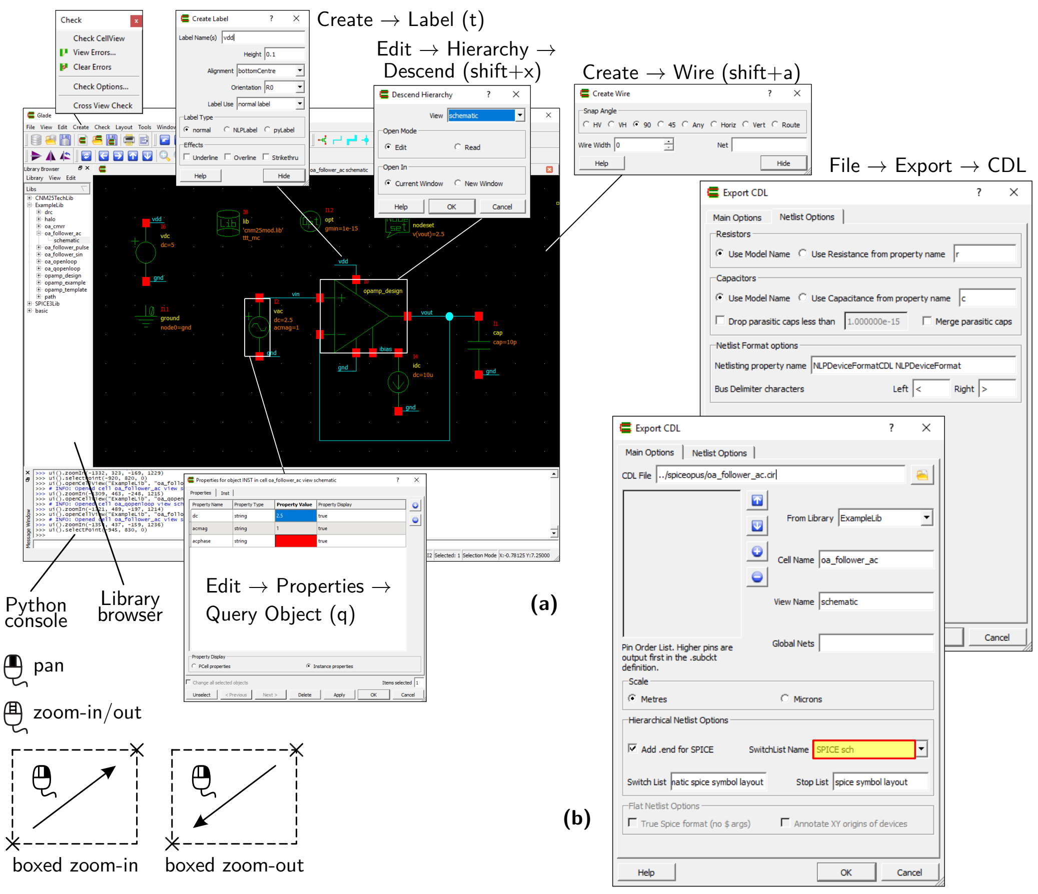
Starting with the design methodology of Fig. 1, all schematic entries at system and circuit levels are performed through the Glade editor. When in schematic mode, this tool supports symbol edition, net and pin labeling, design hierarchy and instance annotation, among other features. As an example, Fig. 4 depicts the equivalent schematic of the OpAmp presented in Fig. 3 being edited in Glade using the PDK custom library `damics-kit/glade/ExampleLib`. Taking advantage of Glade netlisting configurability, specific backend rules have been defined in all symbols to generate circuit description language (CDL) netlists compatible with SPICE3 simulation.

☞ Launch `damics-kit/glade/glade.bat` (or `.sh`).

☞ Open the OpAmp cell view `ExampleLib`→`opamp_design`→`schematic` with the library browser and **fill** the device parameter values of Fig. 3.

☞ Check for schematic connectivity errors with Check→Check Cellview.

☞ Save all cell view changes with File→Save Cell.



Export CDL — Netlist Options

Resistors: (● Use Model Name) ○ Use Resistance from property name: r
Capacitors: (● Use Model Name) ○ Use Capacitance from property name: c
☐ Drop parasitic caps less than 1.000000e-15 ☐ Merge parasitic caps
Netlist Format options: Netlisting property name: NLPDeviceFormatCDL NLPDeviceFormat
Bus Delimiter characters: Left < Right >

Export CDL — Main Options

CDL File: ../spiceopus/oa_follower_ac.cir
From Library: ExampleLib
Cell Name: oa_follower_ac
View Name: schematic
Global Nets:
Pin Order List. Higher pins are output first in the .subckt definition.
Scale: (● Metres) ○ Microns
Hierarchical Netlist Options: ☑ Add .end for SPICE; SwitchList Name: SPICE sch
Switch List: natic spice symbol layout; Stop List: spice symbol layout
Flat Netlist Options: ☐ True Spice format (no $ args) ☐ Annotate XY origins of devices

Create Wire — Snap Angle: ○ HV ○ VH (● 90) ○ 45 ○ Any ○ Horiz ○ Vert ○ Route; Wire Width: 0

Figure 4 Main window of the Glade schematic editor (a) and CDL export dialogue configuration (b) for SPICE schematics (pre-layout).

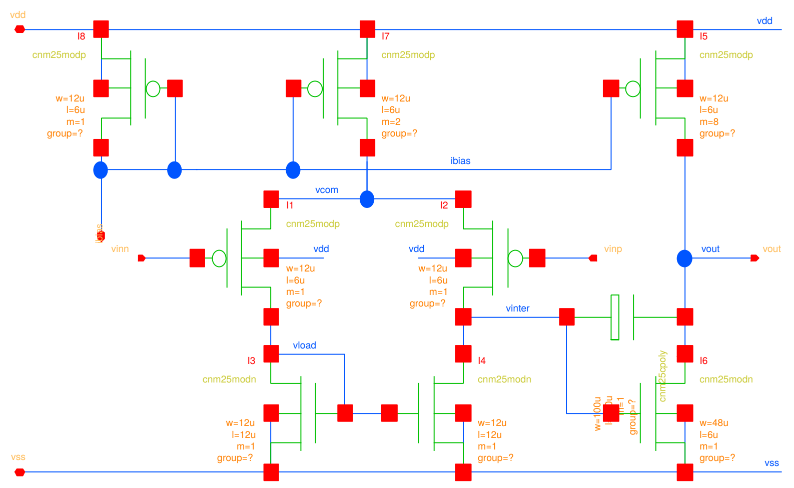


Figure 5 Glade cell view `ExampleLib→opamp_design→schematic` for the OpAmp circuit of Fig. 3 after setting all device parameter values.

The first step towards the design of an IC block is to build a suitable test bench for the analysis of its performance parameters. In this sense, the PDK features a set of test-bench examples oriented to the characterization of OpAmps, as summarized in Fig. 6. This schematics collection includes: open-loop configuration (`oa_openloop`); quasi open-loop topology (`oa_qopenloop`); follower with small-signal (`oa_follower_ac`), pulsed (`oa_follower_pulse`) or sinusoidal (`oa_follower_sin`) input, and an specific setup for the extraction of the common-mode rejection ratio (CMRR) (`oa_cmrr`).

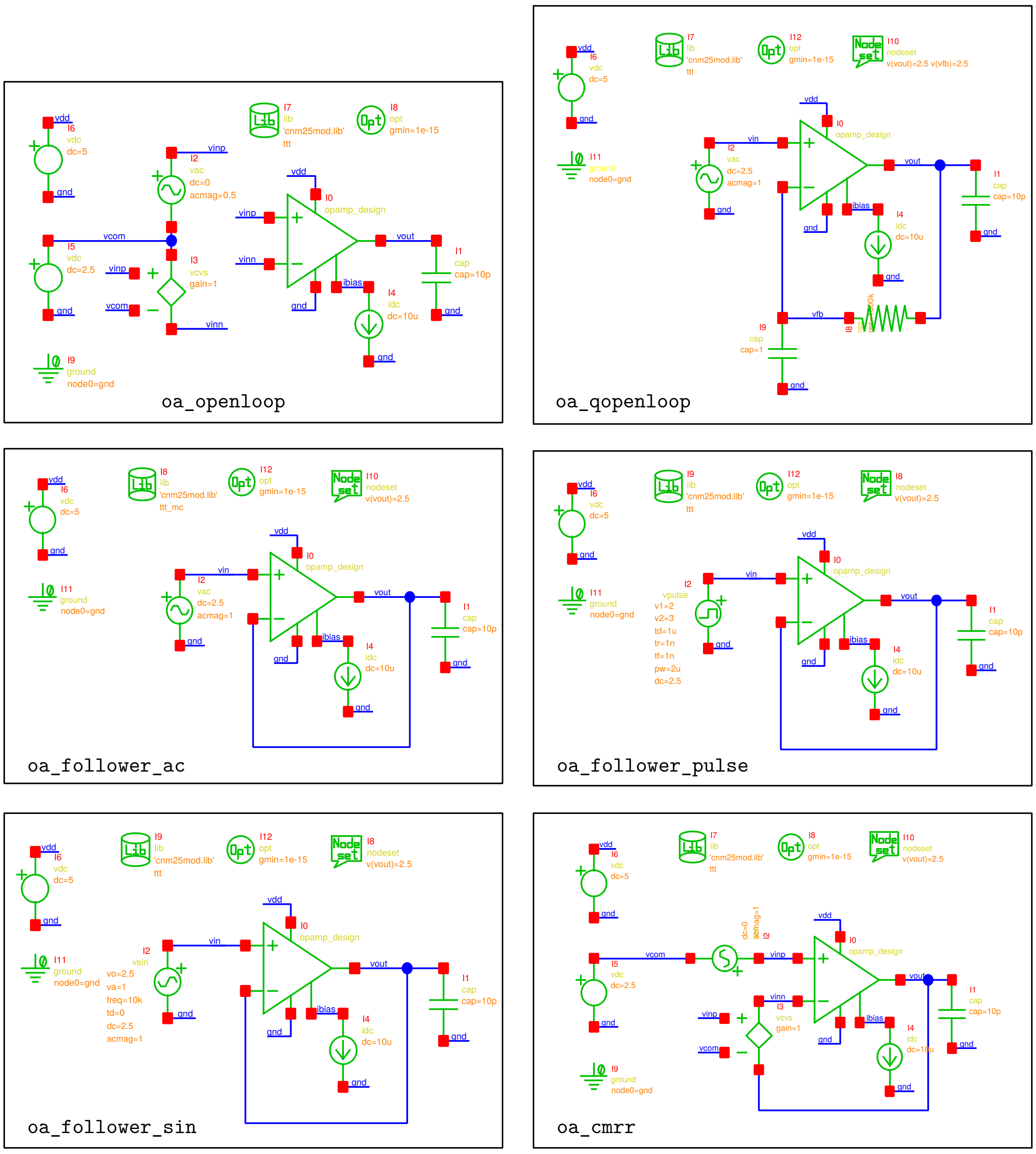


Figure 6 | Test-bench schematics available in `ExampleLib` for the characterization of the `opamp_design` circuit of Fig. 5.

When exported to CDL from Glade editor, each of these test schematics incorporates the netlist at transistor level of the CMOS OpAmp as a subcircuit thanks to the own design hierarchy, like in the example below.

oa_qopenloop.cir

```
********************************************************************************
* Library : ExampleLib
* Top Cell Name: oa_qopenloop
* View Name: schematic
********************************************************************************

********************************************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_design
* View Name:    schematic
********************************************************************************
.SUBCKT opamp_design vinn vinp vout vdd vss ibias

   xI7 vdd ibias vcom vdd cnm25modp w=24u l=6u m=4
   xI3 vload vload vss vss cnm25modn w=24u l=6u m=1
   xI1 vcom vinn vload vdd cnm25modp w=32u l=6u m=6
   xI9 vout vinter cnm25cpoly w=100u l=100u m=2
   xI4 vinter vload vss vss cnm25modn w=24u l=6u m=1
   xI2 vcom vinp vinter vdd cnm25modp w=32u l=6u m=6
   xI8 vdd ibias ibias vdd cnm25modp w=24u l=6u m=1
   xI5 vdd ibias vout vdd cnm25modp w=24u l=6u m=16
   xI6 vout vinter vss vss cnm25modn w=48u l=6u m=10

.ENDS

vI6 vdd gnd dc=5
xI0 vfb vin vout vdd gnd ibias opamp_design
cI1 vout gnd c=10p
vI11 gnd 0 0
iI4 ibias gnd dc=10u
rI8 vfb vout r=1000k
vI2 vin gnd dc=2.5 acmag=1
cI9 vfb gnd c=1

.lib 'cnm25mod.lib' ttt
.options gmin=1e-15
.nodeset v(vout)=2.5 v(vfb)=2.5

.END
```

It is important to note here that each test bench is self contained in the sense that already includes all the necessary information for its simulation. Hence, apart from the circuit under test (i.e. the OpAmp itself) and its boundary conditions (e.g. load impedance, bias current and supply voltage), these test benches also incorporate: process and matching selection for CNM25 models (`.lib`), convergence aids (`.nodeset`), initial conditions (`.ic`) or simulation options (`.options`).

Regarding the CNM25 primitive devices (i.e. N/P-type MOS devices and PiP capacitors), they are netlisted as SPICE subcircuits (x-suffix) to allow the combined modeling of process and matching technology deviations following the device model structure presented in Section 3.2.

4.2 Electrical Simulation

In general, the analog behavior of your OpAmp can be accurately described by the parameters of Table 3, which includes from large to small signal performance and also from static to dynamic response.

	Static	**Dynamic**

Large Signal	Range:
	  Differential input (IR)
	  Common mode input (CMR$\pm$)
	  Output (OR)
	Equivalent input offset (V_{off})
	Quiescent power (P_{D})
		Slew rate (SR$\pm$)
		Settling time (t_{s})
		Maximum frequency (f_{max})
		Harmonic distortion (THD)

Small Signal	Open loop differential gain (G_{DC}) Common mode rejection (CMRR_{DC}) Power supply rejection ($\text{PSRR}_{\text{DC}}\pm$)	Bandwidth (BW) $G_{\text{DC}} \times \text{BW}$ product (GBW) Phase margin (ϕ_{m}) Impedance ($Z_{\text{in,out}}$) Equivalent input noise (V_{neq})

Table 3 | OpAmp main performance parameters.

For our purposes, these figures of merit will be specified by the datasheet of Table 4. In the case of Silicon area, its final value is somehow difficult to predict before the full-custom design of its physical layout. Hence, only the sum of device areas will be taken into account based on (2), where ΔW_{cont} and ΔL_{cont} stand for the minimum depth to open a contact window according to the CNM25 design rules of Table 2.

Following the methodology of Fig. 1, the EDA tool selected for all the electrical simulations is SpiceOpus, an interactive electrical simulator based on SPICE3 [5]. Concerning netlist hierarchy, the strategy of Fig. 8 is adopted in all SpiceOpus simulations of this lab PDK. Basically, self-contained test circuits (`*.cir`) are generated from Glade schematics with the required CNM25 device models (`cnm25*.mod`). Test scripts written in NUTMEG (`*.sp3`) are launched simply by invoking their name from the SpiceOpus shell following Fig. 8. In practice, a single test script may manage more than one test circuit. Moreover, multiple simulation analysis can be executed on multiple circuit topologies and results can be combined all together in order to perform automated measurements and produce graphics for documentation.

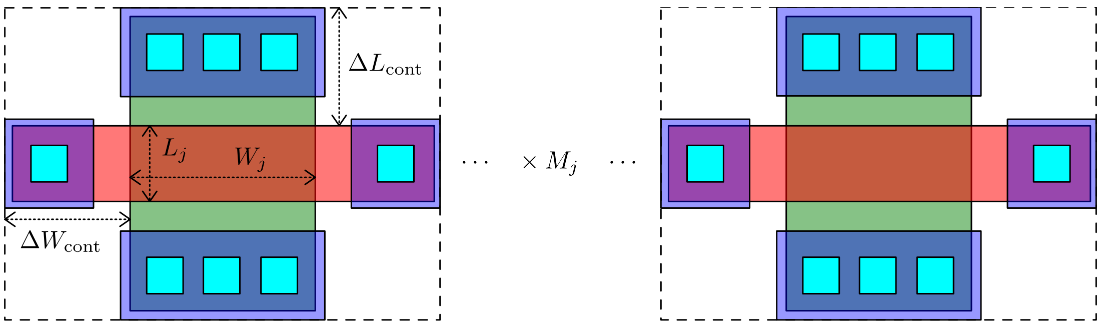
Based on the test circuits of Fig. 6, a complete set of SPICE3 test scripts are also supplied for the numerical extraction of all the OpAmp parameters of Table 4:

- `test_oa_vtc.sp3`: differential-mode input range.
- `test_oa_cmr.sp3`: common-mode input range.
- `test_oa_offset.sp3`: equivalent input offset.
- `test_oa_pd.sp3`: quiescent power consumption.
- `test_oa_bode.sp3`: Bode transfer function.
- `test_oa_cmrr.sp3`: CMRR transfer function.
- `test_oa_psrr.sp3`: PSRR transfer function.
- `test_oa_sr.sp3`: slew rate.
- `test_oa_noise.sp3`: equivalent input noise.
- `test_oa_thd.sp3`: harmonic distortion.

Parameter	Comments	Min.	Typ.	Max.	Units
IR					mV$_{pp}$
CMR	Upper Lower				V
OR					V$_{pp}$
$V_{\rm off}$	$\pm\sigma$				mV$_{rms}$
$P_{\rm D}$	$V_{\rm in} = 2.5$ V				mW
$G_{\rm DC}$					dB
$\rm CMRR_{DC}$					dB
$\rm PSRR_{DC}$	$+$ $-$				dB
SR	$+$ $-$				V/μs
$t_{\rm s}(1\%)$	$V_{\rm out} = 2$ V $\rightarrow$ 3 V $V_{\rm out} = 3$ V $\rightarrow$ 2 V				ns
$f_{\rm max}$	$V_{\rm out} = \rm OR$				kHz
THD	$V_{\rm out} = \rm OR/2$ @10kHz				%
BW	-3 dB				Hz
GBW					MHz
$\phi_{\rm m}$					deg
$V_{\rm nieq}$	1 Hz to 10 MHz				μV$_{rms}$
Area	According to (2)				mm^2

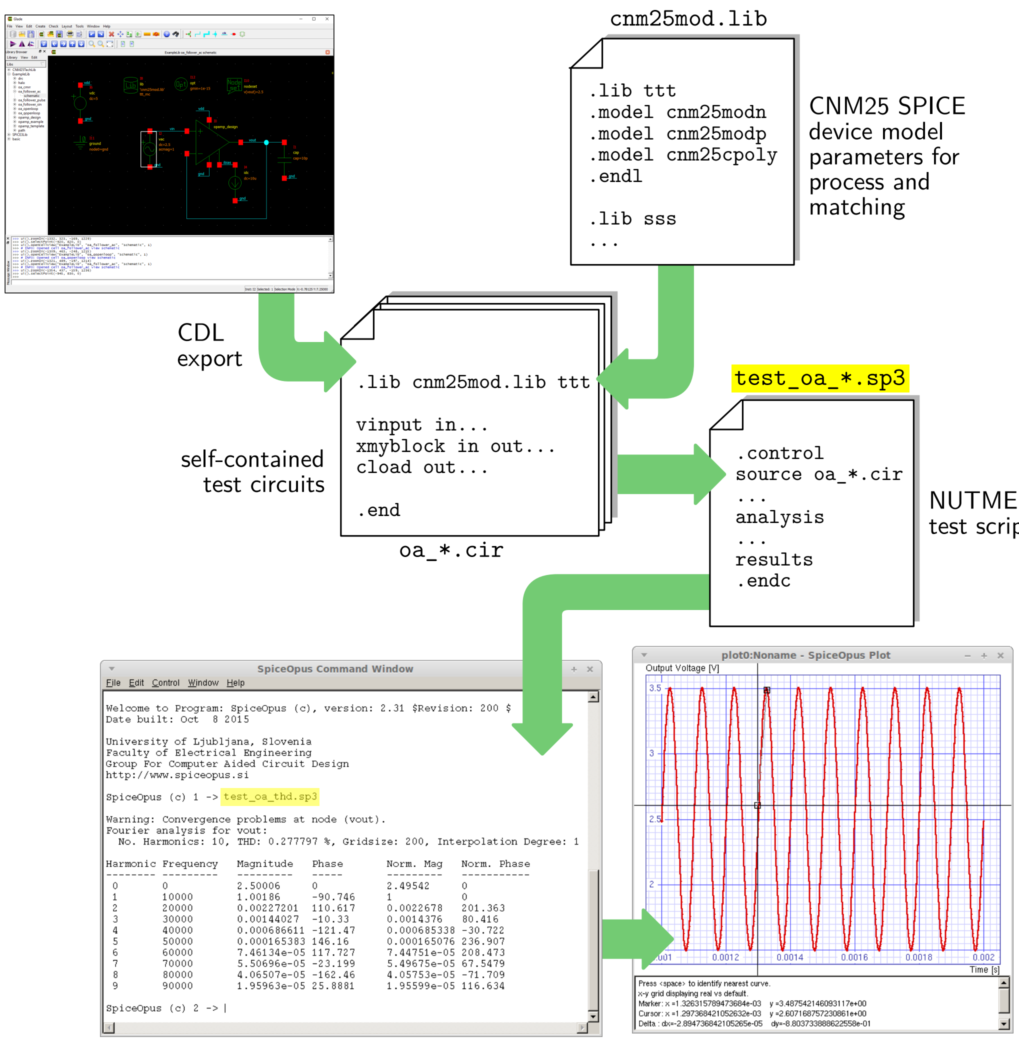
$V_{\rm DD} = +5$ V, $V_{\rm SS} = 0$ V, $I_{\rm bias} = 10$ μA and $C_{\rm load} = 10$ pF.

Table 4 | OpAmp datasheet and test conditions.



$$\text{Area} \doteq \sum_{j}^{\text{devices}} (W_j + 2\Delta W_{\rm cont})(L_j + 2\Delta L_{\rm cont})M_j \tag{2}$$

Figure 7 | Rule of thumb to estimate device area for the datasheet of Table 4.



```
.lib ttt
.model cnm25modn
.model cnm25modp
.model cnm25cpoly
.endl

.lib sss
...
```

```
.lib cnm25mod.lib ttt

vinput in...
xmyblock in out...
cload out...

.end
```

```
.control
source oa_*.cir
...
analysis
...
results
.endc
```

```
Welcome to Program: SpiceOpus (c), version: 2.31 $Revision: 200 $
Date built: Oct  8 2015

University of Ljubljana, Slovenia
Faculty of Electrical Engineering
Group For Computer Aided Circuit Design
http://www.spiceopus.si

SpiceOpus (c) 1 -> test_oa_thd.sp3

Warning: Convergence problems at node (vout).
Fourier analysis for vout:
  No. Harmonics: 10, THD: 0.277797 %, Gridsize: 200, Interpolation Degree: 1

Harmonic Frequency   Magnitude   Phase       Norm. Mag   Norm. Phase
-------- ---------   ---------   -----       ---------   -----------
 0       0           2.50006     0           2.49542     0
 1       10000       1.00186     -90.746     1           0
 2       20000       0.00227201  110.617     0.0022678   201.363
 3       30000       0.00144027  -10.33      0.0014376   80.416
 4       40000       0.000686611 -121.47     0.000685338 -30.722
 5       50000       0.000165383 146.16      0.000165076 236.907
 6       60000       7.46134e-05 117.727     7.44751e-05 208.473
 7       70000       5.50696e-05 -23.199     5.49675e-05 67.5479
 8       80000       4.06507e-05 -162.46     4.05753e-05 -71.709
 9       90000       1.95963e-05 25.8881     1.95599e-05 116.634

SpiceOpus (c) 2 -> |
```

```
Press <space> to identify nearest curve.
x-y grid displaying real vs default.
Marker: x =1.326315789473684e-03    y =3.487542146093117e+00
Cursor: x =1.297368421052632e-03    y =2.607168757230861e+00
Delta : dx=-2.894736842105265e-05   dy=-8.803733888622558e-01
```

Figure 8 | SpiceOpus netlist hierarchy used in this PDK.

damics-kit/spiceopus/test_oa_vtc.sp3

```
test_oa_vtc.sp3
* Differential mode input range
.control
delcirc all
destroy all
delete all
save all
source oa_openloop.cir
dc vi2 -2.5 2.5 5m
let vdin=v(vinp)-v(vinn)
let vout=v(vout)
plot vout vs vdin xlabel 'Diff. Input Voltage...
dc vi2 -2.5m 2.5m 5u
let vdin=v(vinp)-v(vinn)
let vout=v(vout)
plot vout vs vdin xlabel 'Diff. Input Voltage...
.endc
.end
```

damics-kit/spiceopus/test_oa_cmr.sp3

```
test_oa_cmr.sp3
* Common mode input range
.control
delcirc all
destroy all
delete all
save all
source oa_follower_ac.cir
dc vi2 0 5 5m
let vin=v(vin)
let vout=v(vout)
plot vin vout xlabel 'Input Voltage [V]'...
.endc
.end
```

damics-kit/spiceopus/test_oa_offset.sp3

```
test_oa_offset.sp3
* Offset voltage
.control
delcirc all
destroy all
delete all
setplot new
nameplot mc
let voff=0
repeat 1000
 source oa_follower_ac.cir
 save v(vin) v(vout)
 op
 let mc.voff=(mc.voff;v(vin)-v(vout))
 echo run #{length(mc.voff)}
          {round(1e5*(v(vin)-v(vout)))/100} mV
 destroy op1
 delcirc all
 end
* Gaussian fitting
let voff=voff[1,length(voff)-1]
let mn=mean(voff)*1e3
let std=sqrt(mean((voff-mean(voff))^2))*1e3
echo
echo Voffset = {round(100*mn)/100} mV
                           +/- {round(100*std)/100} mV
echo
* Histogram (25-bin)
let dbin=(max(voff)-min(voff))/25
let ybin=vector(25)
let xbin=vector(25)
let counter=0
while counter le 24
 let voffleft=min(voff)+{counter}*dbin
 let voffright=voffleft+dbin
 let xbin[{counter}]=voffleft
 let ybin[{counter}]=sum((voff ge voffleft)
                      and (voff lt voffright))
 let counter=counter+1
 endwhile
set plottype=comb
set linewidth=10
plot ybin vs xbin*1e3 xlabel 'Input Offset [mV]'
                              ylabel 'Samples []'
                  title 'Montecarlo mismatching'
set plottype=line
set linewidth=1
.endc
.end
```

damics-kit/spiceopus/test_oa_pd.sp3

```
test_oa_pd.sp3
* Queiscent power
.control
delcirc all
destroy all
delete all
save all
source oa_follower_ac.cir
op
let iddq=-i(vi6)
let pd=iddq*v(vdd)*1e6
echo Quiescent power consumption = {round(pd)}uW
.endc
.end
```

damics-kit/spiceopus/test_oa_bode.sp3

```
test_oa_bode.sp3
* Bode plot
.control
delcirc all
destroy all
delete all
save all
set units=degrees
source oa_qopenloop.cir
ac dec 50 1 1e7
let Gmag=20*log10(mag(v(vout)))
let Gph=phase(v(vout))
plot Gmag Gph xlog xlabel 'Frequency [Hz]'...
.endc
.end
```

damics-kit/spiceopus/test_oa_cmrr.sp3

```
test_oa_cmrr.sp3
* Common mode rejection ratio
.control
delcirc all
destroy all
delete all
save all
source oa_cmrr.cir
ac dec 50 1 1e7
let CMRR=-20*log10(mag(v(vout)))
plot CMRR xlog xlabel 'Frequency [Hz]'
               ylabel 'CMRR [dB]'
.endc
.end
```

damics-kit/spiceopus/test_oa_psrr.sp3

```
test_oa_psrr.sp3
* Power supply rejection ratio
.control
delcirc all
destroy all
delete all
save all
source oa_follower_sin.cir
let @vi2[acmag]=0
let @vi6[acmag]=1
ac dec 50 1 1e7
let PSRRP=-20*log10(mag(v(vout)))
plot PSRRP xlog xlabel 'Frequency [Hz]'
                ylabel 'PSRR+ [dB]'
.endc
.end
```

damics-kit/spiceopus/test_oa_sr.sp3

```
test_oa_sr.sp3
* Slew-rate +/-
.control
delcirc all
destroy all
delete all
save all
source oa_follower_pulse.cir
tran 1n 5u
let vout=v(vout)
plot vout xlabel 'Time [s]'
          ylabel 'Output Voltage [V]'
.endc
.end
```

damics-kit/spiceopus/test_oa_noise.sp3

```
test_oa_noise.sp3
* Spectral noise analysis
.control
delcirc all
destroy all
delete all
save all
source oa_follower_sin.cir
op
noise v(vout) vi2 dec 100 1 10meg
plot sqrt(noise1.onoise_spectrum)
                       vs noise1.frequency ylog
                         xlabel 'Frequency [Hz]'
 ylabel 'Equivalent Input Noise [Vrms/sqrt(Hz)]'
let vnin=sqrt(noise2.onoise_total)*1e6
echo
echo Equivalent input noise (1Hz to 10MHz) =
                             {round(vnin)}uVrms
echo
.endc
.end
```

damics-kit/spiceopus/test_oa_thd.sp3

```
test_oa_thd.sp3
* Harmonic distortion analysis
.control
delcirc all
destroy all
delete all
save all
source oa_follower_sin.cir
tran 1u 2m 1m 1u
let vout=v(vout)
plot vout xlabel 'Time [s]'
          ylabel 'Output Voltage [V]'
fourier 10k vout
.endc
.end
```

Q1. For the initial OpAmp dimensions of Fig. 3, **fill the datasheet** of Table 4:

- Launch damics-kit/spiceopus/spiceopus.bat (or .sh).
- Call all test scripts by invoking their names, e.g. test_oa_vtc.sp3.
- Measure **typ.** results from graphs and shell messages.
- Repeat for process corners (sss,fff) to measure **min.** and **max.** results.

Q2. Answer the following **questions**:

- test_oa_offset.sp3: What are the meaning of the offset voltage results? Compare with test_oa_vtc.sp3.
- test_oa_bode.sp3: Why openloop.cir and qopenloop.cir show different results? Which one is correct?
- test_oa_cmrr.sp3: How is the CMRR calculated?

4.3 Circuit Optimization

The main objective of this part is to apply the OpAmp design equations seen in class sessions [4] for the optimization of a particular parameter of the datasheet, while keeping a minimum performance for the rest of them. This design process consists on changing the OpAmp initial device sizing of Fig. 3 without altering its internal circuit topology.

In particular, the optimization of your OpAmp is arranged into two steps:

Initial solution. The starting point is obtained from the results of Section 4.2. Hence, simulation results from Table 4 need to be copied into the **Initial** column of Table 5.

Optimization. A single parameter must be chosen for your OpAmp circuit optimization, while maintaining the rest of parameters within the specification range defined by the **Spec.** column. During optimization, each MOSFET and capacitor of the OpAmp is resized to enhance the selected parameter but checking for the side effects on the rest of them. After finishing this optimization process, performance results will be displayed in the **Optim.** column of Table 5.

Parameter	Comments	Spec.	Initial	Optim.	Units
IR					mV$_{pp}$
CMR	Upper	>4			V
	Lower	<1			
OR		>3			V$_{pp}$
V_{off}	$\pm\sigma$	<10			mV$_{rms}$
P_D	$V_{in} = 2.5$ V	<1.5			mW
G_{DC}		>60			dB
CMRR$_{DC}$		>50			dB
PSRR$_{DC}$	$+$	>50			dB
	$-$	>50			
SR	$+$	>1.5			V/μs
	$-$	>1.5			
$t_s(1\%)$	$V_{out} = 2$ V $\rightarrow$ 3 V	<1500			ns
	$V_{out} = 3$ V $\rightarrow$ 2 V	<1500			
f_{max}	$V_{out} =$ OR				kHz
THD	$V_{out} =$ OR/2 @10kHz	<1			%
BW	-3dB				Hz
GBW		>1			MHz
ϕ_m		>50			deg
V_{nieq}	100 Hz to 10 MHz	<1			mV$_{rms}$
Area	According to (2)	<0.025			mm^2

$V_{DD} = +5$ V, $V_{SS} = 0$ V, $I_{bias} = 10\ \mu$A and $C_{load} = 10$ pF.

Table 5 | OpAmp optimization datasheet and test conditions.

Q3. Perform your OpAmp optimization as follows:

- **Ask the professor** before choosing the optimization parameter! Possible choices are: G_{DC}, GBW or $\mathrm{SR}\pm$.
- When optimizing your circuit, the size of **all devices** of Fig. 3 may be changed, including the PiP compensation capacitor.
- For each iteration of the optimization process, it is advised to carefully check the following **key parameters**: G_{DC}, GBW, $\mathrm{SR}\pm$, ϕ_{m}, Area and P_{D}.
- Students are encouraged to develop their **own test script** (e.g. `test_oa_datasheet.sp3`) in order to extract these 6 parameters automatically. For such a purpose, some useful SPICE3 NUTMEG code examples are given below. Further information can be found in [5].
- Even more important than the final value of the optimized parameter, the comprehensive explanation of the OpAmp **optimization strategy** is of major importance, as well as the analysis of the second order effects limiting the optimization process!
- **Report** must include the detailed optimization planning, device dimensions of the new OpAmp circuit, filled datasheet of Table 5 and all related plots.

```
Miscellaneous SPICE3 recipes

* Working with multiple circuits
source file1.cir
source file2.cir
setcirc ckt1

* Interacting with netlists
let w1=@mi1:xi0[w]
let @mi2:xi0[w]=w1

* Working with multiple analysis
ac dec 50 1 10meg
let GdB=20*log10(mag(v(vout)))
setcirc ckt2
ac dec 50 1 10meg
let GdB=20*log10(mag(v(vout)))
plot ac1.GdB ac2.GdB xlog xlabel 'Frequency...
setplot ac1

* Automatic measurements
let GdBDC=G[0]
cursor c right GdB 0
let GBW=abs(frequency[%c])/1e6

* Reporting results
echo G(DC)={round(GdBDC)}dB
echo GBW={round(GBW*10)/10}MHz

* Cleaning memory
destroy ac2
delcirc ckt2

* Parametric analysis
foreach wpar 10u 20u 30u
   let @mi6:xi0[w]={wpar}
   ac dec 50 1 10meg
   let GdBDC=20*log10(mag(v(vout)[0]))
   let ac1.GdBDC=(ac1.GdBDC;GdBDC)
   end
let mydata=ac1.GdBDC

* Exporting results
set nobreak
set noprintheader
set noprint index
set width=132
print mydata > fileout.txt

Miscellaneous SPICE3 recipes
```

4.4 PCell-Based Netlist-Driven Layout Design

Once the optimal device sizing of your OpAmp schematic has been obtained, the physical CMOS design of the IC can start according to the methodology of Fig. 1. Unlike in full-custom digital circuits, where compactness and speed are the major design targets, the main goals in analog IC design are both signal integrity and device matching. For the former, signal decoupling is improved by introducing ground guards, avoiding cross-coupling and using differential signaling when routing. As for device matching, the symmetry rules of Table 6 are strongly recommended.

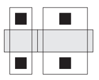
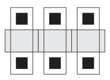
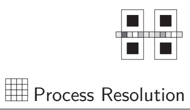
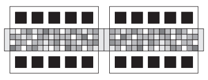
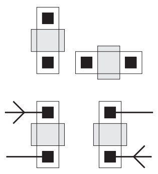
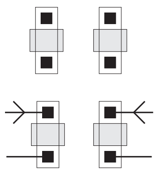
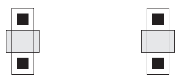
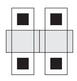
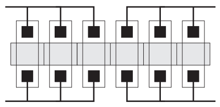
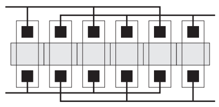
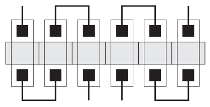
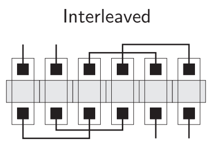
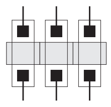
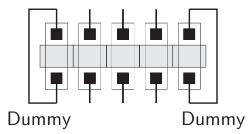
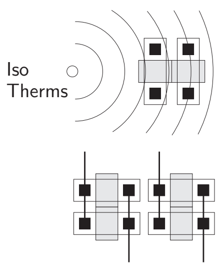
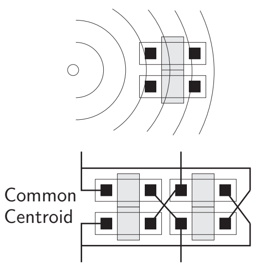
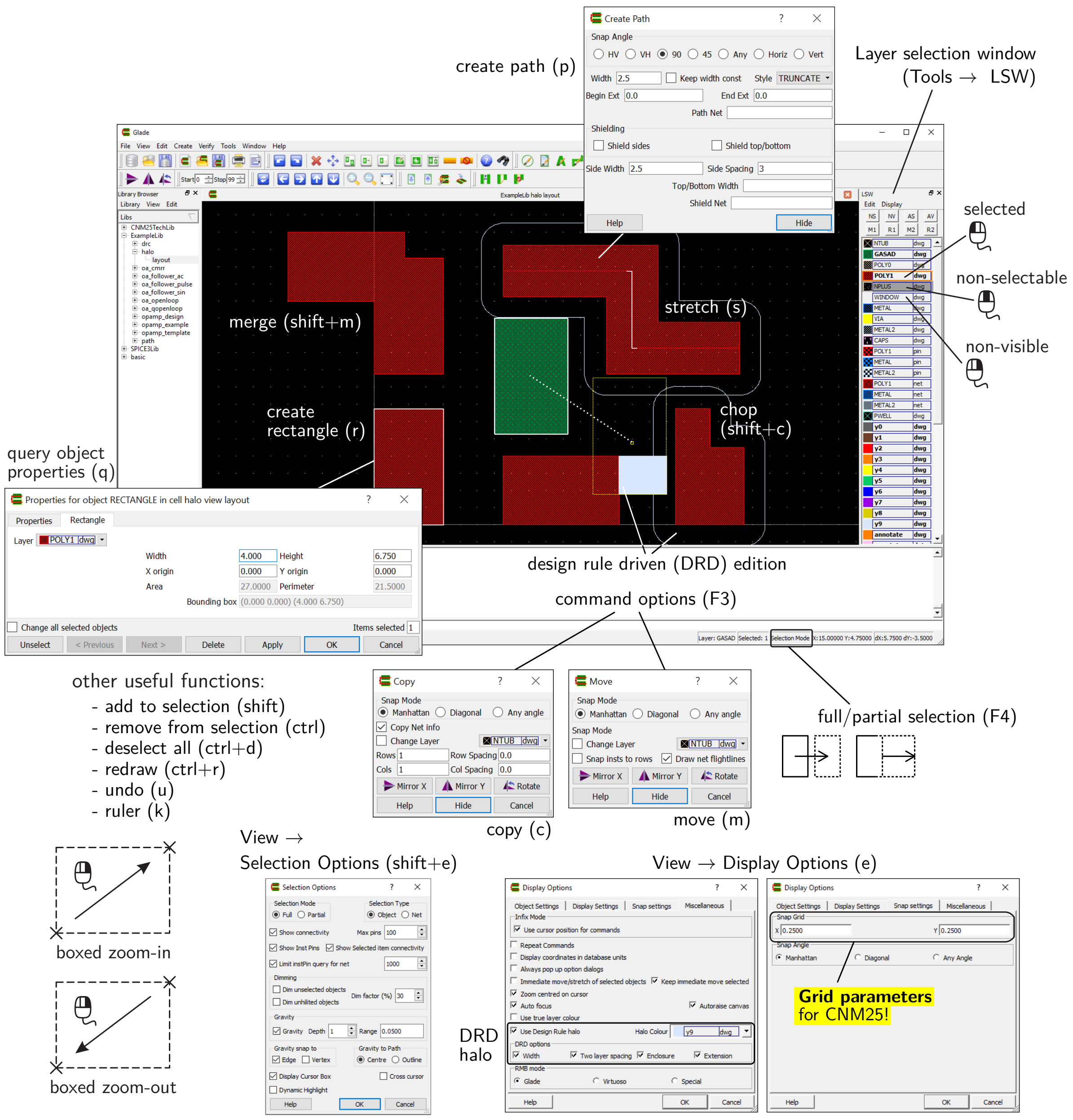
Layout Rule	Bad	Good
Unitary Elements		
Large Area		
Same Orientation		
Minimum Distance		
$(W/L)>>1$		
$(W/L)<<1$		
Same Sorround		
Same Symmetry		

Table 6 │ Analog layout style guide for best device matching [6].

Glade is the EDA tool selected in the design methodology of Fig. 1 for the full-custom edition and physical verification of IC mask layouts. This layout editor includes advanced geometrical operations (e.g. stretching, chopping, merging), full and partial object selection, multi hierarchical design browsing, lots of display options, as well as the effective management of technological layers following Fig. 9. Due to the intensive mouse usage when in interactive mode, most commands can be invoked through the corresponding bindkeys predefined in Edit→Edit Bindkeys.

One particularly useful tool of Glade is the design rule driven (DRD) edition. As illustrated in the canvas of Fig. 9, this operation mode displays the design rules on the fly, during object edition. To prevent from a negative impact on the editor speed performance, the DRD mode can be configured to check rules between the object being edited and the rest of elements belonging to the same layer only.



other useful functions:
- add to selection (shift)
- remove from selection (ctrl)
- deselect all (ctrl+d)
- redraw (ctrl+r)
- undo (u)
- ruler (k)

Figure 9 | Glade layout editor main window and associated tools.

Glade takes most of the CMOS process information from the technology file, like: layer table, layer connectivity, basic design rules, contact rules, periodic structures, among others. For this purpose, the CNM25 PDK comes with the technology file `damics-kit/glade/cnm25.tch`, which is already compiled inside the lab design library `damics-kit/glade/ExampleLib`.

damics-kit/glade/cnm25.tch

```
//
//      Name    Purpose gds_num gds_dtyp RGBA              sel? vis? fillstyle  linestyle
LAYER   NTUB    drawing  1       0        (255,230,191,255) t    t    cross      plain ;
LAYER   GASAD   drawing  2       0        (0,204,102,255)   t    t    dots1      plain ;
LAYER   POLY0   drawing  3       0        (255,230,191,255) t    t    zagr       plain ;
LAYER   POLY1   drawing  4       0        (255,0,0,255)     t    t    right_bars plain ;
LAYER   NPLUS   drawing  5       0        (255,230,191,255) t    t    points_1   plain ;
LAYER   WINDOW  drawing  6       0        (0,255,255,255)   t    t    full       plain ;
LAYER   METAL   drawing  7       0        (0,128,255,255)   t    t    zagr       plain ;
LAYER   VIA     drawing 10       0        (255,255,0,255)   t    t    solid      plain ;
LAYER   METAL2  drawing  9       0        (204,230,255,255) t    t    zagl       plain ;
LAYER   CAPS    drawing  8       0        (255,170,255,255) t    t    crosses    plain ;
LAYER   POLY1   pin     20       0        (255,0,0,255)     t    t    squares_1  plain ;
LAYER   METAL   pin     21       0        (0,128,255,255)   t    t    squares_2  plain ;
LAYER   METAL2  pin     22       0        (204,230,255,255) t    t    squares_2  plain ;
LAYER   POLY1   net     23       0        (255,0,0,255)     t    t    dots1      plain ;
LAYER   METAL   net     24       0        (0,128,255,255)   t    t    dots1      plain ;
LAYER   METAL2  net     25       0        (204,230,255,255) t    t    dots1      plain ;
LAYER   PWELL   drawing 26       0        (73,214,186,255)  t    t    cross      plain ;
//
// Layer function
//
FUNCTION POLY0 drawing ROUTING ;
FUNCTION POLY1 drawing ROUTING ;
FUNCTION WINDOW drawing CUT ;
FUNCTION METAL drawing ROUTING ;
FUNCTION VIA drawing CUT ;
FUNCTION METAL2 drawing ROUTING ;
FUNCTION POLY1 pin PIN ;
FUNCTION METAL pin PIN ;
FUNCTION METAL2 pin PIN ;
FUNCTION POLY1 net PIN ;
FUNCTION METAL net PIN ;
FUNCTION METAL2 net PIN ;
//
// Layer Connections
//
CONNECT POLY0 drawing BY WINDOW drawing TO METAL drawing ;
CONNECT POLY1 drawing BY WINDOW drawing TO METAL drawing ;
CONNECT METAL drawing BY VIA drawing TO METAL2 drawing ;
//
// Layout rules
//
MINWIDTH NTUB drawing 8.000 ;
MINSPACE NTUB drawing 8.000 ;
MINWIDTH GASAD drawing 2.000 ;
MINSPACE GASAD drawing 4.000 ;
MINENC NTUB drawing GASAD drawing 5.000 ;
MINSPACE NTUB drawing GASAD drawing 5.000 ;
MINWIDTH POLY0 drawing 2.500 ;
MINSPACE POLY0 drawing 6.000 ;
MINSPACE POLY0 drawing GASAD drawing 6.000 ;
MINWIDTH POLY1 drawing 2.500 ;
```

```
MINSPACE POLY1 drawing 3.000 ;
MINEXT POLY1 drawing GASAD drawing 2.500 ;
MINEXT GASAD drawing POLY1 drawing 3.000 ;
MINSPACE POLY1 drawing GASAD drawing 1.250 ;
MINENC POLY0 drawing POLY1 drawing 3.000 ;
MINENC NPLUS drawing GASAD drawing 2.500 ;
MINWIDTH NPLUS drawing 2.500 ;
MINSPACE NPLUS drawing 2.500 ;
MINWIDTH WINDOW drawing 2.500 ;
MINSPACE WINDOW drawing 3.000 ;
MINENC GASAD drawing WINDOW drawing 1.000 ;
MINENC POLY1 drawing WINDOW drawing 1.250 ;
MINENC POLY0 drawing WINDOW drawing 4.000 ;
MINSPACE WINDOW drawing POLY1 drawing 4.000 ;
MINSPACE WINDOW drawing POLY0 drawing 4.000 ;
MINWIDTH METAL drawing 2.500 ;
MINSPACE METAL drawing 3.000 ;
MINENC METAL drawing WINDOW drawing 1.250 ;
MINWIDTH VIA drawing 3.000 ;
MINSPACE VIA drawing 3.500 ;
MINENC METAL drawing VIA drawing 1.250 ;
MINSPACE VIA drawing WINDOW drawing 2.500 ;
MINSPACE VIA drawing POLY1 drawing 2.500 ;
MINWIDTH METAL2 drawing 3.500 ;
MINSPACE METAL2 drawing 3.500 ;
MINENC METAL2 drawing VIA drawing 1.250 ;
//
// Via rules
//
VIA dff_m1
        GASAD drawing -2.250 -2.250 2.250 2.250
        WINDOW drawing -1.250 -1.250 1.250 1.250
        METAL drawing -2.500 -2.500 2.500 2.500
;
VIA p0_m1
        POLY0 drawing -5.250 -5.250 5.250 5.250
        WINDOW drawing -1.250 -1.250 1.250 1.250
        METAL drawing -2.500 -2.500 2.500 2.500
;
VIA p1_m1
        POLY1 drawing -2.500 -2.500 2.500 2.500
        WINDOW drawing -1.250 -1.250 1.250 1.250
        METAL drawing -2.500 -2.500 2.500 2.500
;
VIA m1_m2
        METAL drawing -2.750 -2.750 2.750 2.750
        VIA drawing -1.500 -1.500 1.500 1.500
        METAL2 drawing -2.750 -2.750 2.750 2.750
;
//
// MultiPartPath rules
//
MPP nguard LAYER NTUB  drawing WIDTH 14.5 BEGEXT 7.25 ENDEXT 7.25 ;
MPP nguard LAYER GASAD drawing WIDTH 4.5 BEGEXT 2.25 ENDEXT 2.25 ;
MPP nguard LAYER NPLUS drawing WIDTH 9.5 BEGEXT 4.75 ENDEXT 4.75 ;
MPP nguard LAYER WINDOW drawing WIDTH 2.5 BEGEXT -1.25 ENDEXT 1.25 SPACE 3 LENGTH 2.5 ;
MPP nguard LAYER METAL drawing WIDTH 5.0 BEGEXT 2.5 ENDEXT 2.5 ;

MPP pguard LAYER GASAD drawing WIDTH 4.5 BEGEXT 2.25 ENDEXT 2.25 ;
MPP pguard LAYER WINDOW drawing WIDTH 2.5 BEGEXT -1.25 ENDEXT 1.25 SPACE 3 LENGTH 2.5 ;
MPP pguard LAYER METAL drawing WIDTH 5.0 BEGEXT 2.5 ENDEXT 2.5 ;

MPP p0m1 LAYER POLY0 drawing WIDTH 10.5 BEGEXT 5.25 ENDEXT 5.25 ;
MPP p0m1 LAYER WINDOW drawing WIDTH 2.5 BEGEXT -1.25 ENDEXT 1.25 SPACE 3 LENGTH 2.5 ;
MPP p0m1 LAYER METAL drawing WIDTH 5.0 BEGEXT 2.5 ENDEXT 2.5 ;

MPP p1m1 LAYER POLY1 drawing WIDTH 5 BEGEXT 2.50 ENDEXT 2.50 ;
MPP p1m1 LAYER WINDOW drawing WIDTH 2.5 BEGEXT -1.25 ENDEXT 1.25 SPACE 3 LENGTH 2.5 ;
MPP p1m1 LAYER METAL drawing WIDTH 5.0 BEGEXT 2.5 ENDEXT 2.5 ;

MPP m1m2 LAYER METAL drawing WIDTH 5.5 BEGEXT 2.75 ENDEXT 2.75 ;
MPP m1m2 LAYER VIA drawing WIDTH 3 BEGEXT -1.5 ENDEXT 1.5 SPACE 3.5 LENGTH 3 ;
MPP m1m2 LAYER METAL2 drawing WIDTH 5.5 BEGEXT 2.75 ENDEXT 2.75 ;
//
// Fastcap conductor data in um
//
METLYR METAL  drawing HEIGHT 1.000 THICKNESS 1.000 ;
VIALYR VIA    drawing HEIGHT 2.000 THICKNESS 0.800 ;
METLYR METAL2 drawing HEIGHT 2.800 THICKNESS 1.100 ;
//
// Layout generation tool
//
MAP cnm25modn TO cnm25modn_m layout ;
MAP cnm25modp TO cnm25modp_m layout ;
MAP cnm25cpoly TO cnm25cpoly_m layout ;
//
// Line Styles...
// Stipple Patterns
STIPPLE right_bars      STIPPLE
   1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
   1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
   0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
   0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
   1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
   1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
   0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
   0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
   1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
   1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
   0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
   0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
   1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
   1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
   0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
   0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
   ;
STIPPLE squares_2       STIPPLE
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
   1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
...
```

damics-kit/glade/cnm25.tch

In order to speed up the full-custom edition of the IC layout, this PDK supports the use of a parameterized cell (PCell) for each native CNM25 device. A PCell is a geometrical element, in between plain full-custom primitives (e.g. rectangle, irregular polygon, path) and semi-custom cells (e.g. logic gates), which is automatically generated according to variable sizing parameters. In the case of CNM25, layout PCells are available in `CNM25TechLib` for NMOS transistors (`cnm25modn_m`), PMOS transistors (`cnm25modp_m`) and PiP capacitors (`cnm25cpoly_m`), as shown in Fig. 10. In Glade, PCells can be programmed for each CMOS technology using Python language, like the code example shown below.

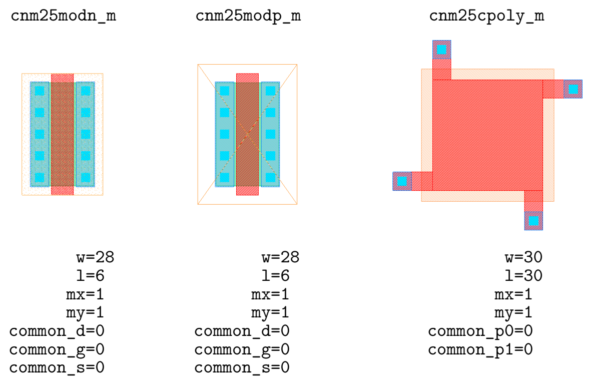


Figure 10 Glade layout PCells available in the CNM25 PDK and sizing parameters. PCell devices can be called through Create→Instance (i).

damics-kit/glade/pcells/cnm25modn_m.py

```python
from ui import *
def cnm25modn_m(cv, w=4.5e-6, l=3.0e-6, mx=1, my=1, common_d=0, common_g=0, common_s=0) :
    lib = cv.lib()
    tech = lib.tech()
    dbu = lib.dbuPerUU()
    width = abs(int(w * 1.0e6 * dbu))
    length = abs(int(l * 1.0e6 * dbu))
    xelem = abs(int(mx))
    yelem = abs(int(my))
    commd = bool(common_d)
    commg = bool(common_g)
    comms = bool(common_s)
    # Layer rules
    xygrid = int(0.25 * dbu)
    gasad_width = int(2.00 * dbu)
    gasad_space = int(4.00 * dbu)
    ntub_ov_gasad = int(5.00 * dbu)
    nplus_ov_gasad = int(2.50 * dbu)
    poly_width = int(3.0 * dbu)
    poly_space = int(3.0 * dbu)
    poly_space_gasad = int(1.25 * dbu)
    poly_ext_gasad = int(2.5 * dbu)
    cont_size = int(2.50 * dbu)
    cont_space = int(3.00 * dbu)
    cont_space_poly = int(2.00 * dbu)
    gasad_ov_cont = int(1.00 * dbu)
    poly_ov_cont = int(1.25 * dbu)
    metal_width = int(2.50 * dbu)
    metal_space = int(3.00 * dbu)
    metal_ov_cont = int(1.25 * dbu)
    # Device rules
    min_length = poly_width
    min_width = max(gasad_width, cont_size + 2*gasad_ov_cont)
    min_xelem = 1
    min_yelem = 1
    # Checking parameters
    if length%xygrid!=0 :
        length = int(xygrid * int(length / xygrid))
        cv.dbReplaceProp("l", 1e-6 * (length / dbu))
        print "** cnm25modn WARNING: l is off-grid. Adjusting element length. **"
        cv.update()
    if width%xygrid!=0 :
        width = int(xygrid * int(width / xygrid))
        cv.dbReplaceProp("w", 1e-6 * (width / dbu))
        print "** cnm25modn WARNING: w is off-grid. Adjusting element width. **"
        cv.update()
    if length < min_length :
        length = min_length
        cv.dbReplaceProp("l", 1e-6 * (length / dbu))
        print "** cnm25modn WARNING: l < minimum length. Resetting element length. **"
        cv.update()
    if width < min_width :
        width = min_width
        cv.dbReplaceProp("w", 1e-6 * (width / dbu))
        print "** cnm25modn WARNING: w < minimum width. Resetting element width. **"
        cv.update()
    if xelem < min_xelem :
        xelem = min_xelem
        cv.dbReplaceProp("mx", xelem)
        print "** cnm25modn WARNING: mx == 0. Resetting number of horizontal elements to ",
                                                                                  xelem, ". **"
        cv.update()
    if yelem < min_yelem :
        yelem = min_yelem
        cv.dbReplaceProp("my", yelem)
        print "** cnm25modn WARNING: my == 0. Resetting number of vertical elements to ",
        cv.update()                                                              yelem, ". **"
    # Calculate XY incremental offsets
    dxoffset_min = length + 2*cont_space_poly + cont_size
    dxoffset_extra = cont_size + 2*gasad_ov_cont + max(gasad_space,
                     (metal_ov_cont-gasad_ov_cont) + metal_space)
    dxoffset_alt = [ dxoffset_min + (not commd) * dxoffset_extra, dxoffset_min +
                   (not comms) * dxoffset_extra]
    dyoffset = width + max(gasad_space, (not commg) * (2*poly_ext_gasad + poly_space),
               ((not commd) or (not comms)) * (2*(metal_ov_cont-gasad_ov_cont) + metal_space))
    # 2D element array iteration
    xoffset = 0
    for x in range(xelem) :
        yoffset = 0
        for y in range(yelem) :
            # Create active
            layer = tech.getLayerNum("GASAD", "drawing")
            r = Rect(-(cont_space_poly + cont_size + gasad_ov_cont), 0, length +
                cont_space_poly + cont_size + gasad_ov_cont, width)
            r.offset(xoffset, yoffset)
            active = cv.dbCreateRect(r, layer)
            # Create gate
            layer = tech.getLayerNum("POLY1", "drawing")
            poly_ext_one_side = max(poly_ext_gasad, commg * max(gasad_space/2, ((not commd) or
                                (not comms)) * ((metal_ov_cont-gasad_ov_cont) + metal_space/2)))
            r = Rect(0, -poly_ext_one_side, length, width + poly_ext_one_side)
            r.offset(xoffset, yoffset)
            poly = cv.dbCreateRect(r, layer)
            gate_net = cv.dbCreateNet("G")
            pin = cv.dbCreatePin("G", gate_net, DB_PIN_INPUT)
            cv.dbCreatePort(pin, poly)
            # Create drain and source contacts
            layer = tech.getLayerNum("WINDOW", "drawing")
            n_cont = int((width - 2*gasad_ov_cont + cont_space) / (cont_size + cont_space))
            s_cont = 0
            if (n_cont > 1) :
                 s_cont = cont_space
            for n in range(n_cont) :
                r = Rect(-cont_space_poly - cont_size, gasad_ov_cont + n *
                    (cont_size + s_cont), -cont_space_poly, gasad_ov_cont +
                    cont_size + n * (cont_size + s_cont))
                r.offset(xoffset, yoffset)
                contact = cv.dbCreateRect(r, layer)
                r = Rect(length + cont_space_poly, gasad_ov_cont + n * (cont_size +
                    s_cont), length + cont_space_poly + cont_size, gasad_ov_cont +
                    cont_size + n * (cont_size + s_cont))
                r.offset(xoffset, yoffset)
                contact = cv.dbCreateRect(r, layer)
            # Create drain and source metal
            layer = tech.getLayerNum("METAL", "drawing")
            metal_ext_one_side = max(metal_ov_cont - gasad_ov_cont, (((x%2!=0) and commd)
                                 or ((x%2==0) and (comms))) * (dyoffset - width)/2)
            r = Rect(-(cont_space_poly + cont_size + metal_ov_cont), -metal_ext_one_side,
                -cont_space_poly + metal_ov_cont, width + metal_ext_one_side)
            r.offset(xoffset, yoffset)
            metal = cv.dbCreateRect(r, layer)
            source_net = cv.dbCreateNet("S")
            pin = cv.dbCreatePin("S", source_net, DB_PIN_INOUT)
            cv.dbCreatePort(pin, metal)
            metal_ext_one_side = max(metal_ov_cont - gasad_ov_cont, (((x%2==0) and
                                 (commd)) or ((x%2!=0) and (comms))) * (dyoffset - width)/2)
            r = Rect(length + cont_space_poly - metal_ov_cont,-metal_ext_one_side,length
                + cont_space_poly + cont_size + metal_ov_cont, width + metal_ext_one_side)
            r.offset(xoffset, yoffset)
            metal = cv.dbCreateRect(r, layer)
            drain_net = cv.dbCreateNet("D")
            pin = cv.dbCreatePin("D", drain_net, DB_PIN_INOUT)
            cv.dbCreatePort(pin, metal)
            yoffset = yoffset + dyoffset
        xoffset = xoffset + dxoffset_alt[x%2!=0]
    # Create n-plus
    layer = tech.getLayerNum("NPLUS", "drawing")
    nplus_x_ext = cont_space_poly + cont_size + gasad_ov_cont + nplus_ov_gasad
    r = Rect(-nplus_x_ext, -nplus_ov_gasad, length + nplus_x_ext + xoffset -dxoffset_alt[x%2!=0],
        width + nplus_ov_gasad + yoffset - dyoffset)
    nplus = cv.dbCreateRect(r, layer)
    # Create p-well
    layer = tech.getLayerNum("backgnd", "drawing")
    ntub_x_ext = cont_space_poly + cont_size + gasad_ov_cont + ntub_ov_gasad
    r = Rect(-ntub_x_ext, -ntub_ov_gasad, length + ntub_x_ext + xoffset -dxoffset_alt[x%2!=0],
        width + ntub_ov_gasad + yoffset - dyoffset)
    ptub = cv.dbCreateRect(r, layer)
    bulk_net = cv.dbCreateNet("B")
    pin = cv.dbCreatePin("B", bulk_net, DB_PIN_INOUT)
    cv.dbCreatePort(pin, ptub)
    # Save results
    cv.update()
```
damics-kit/glade/pcells/cnm25modn_m.py

Examples of usage for CNM25 PCells can be found in Figs. 11 and 12. As a positive side effect, PCell-based layouts are less prone to introduce violations since they have been already programmed taking into account the process design rules of Table 2.

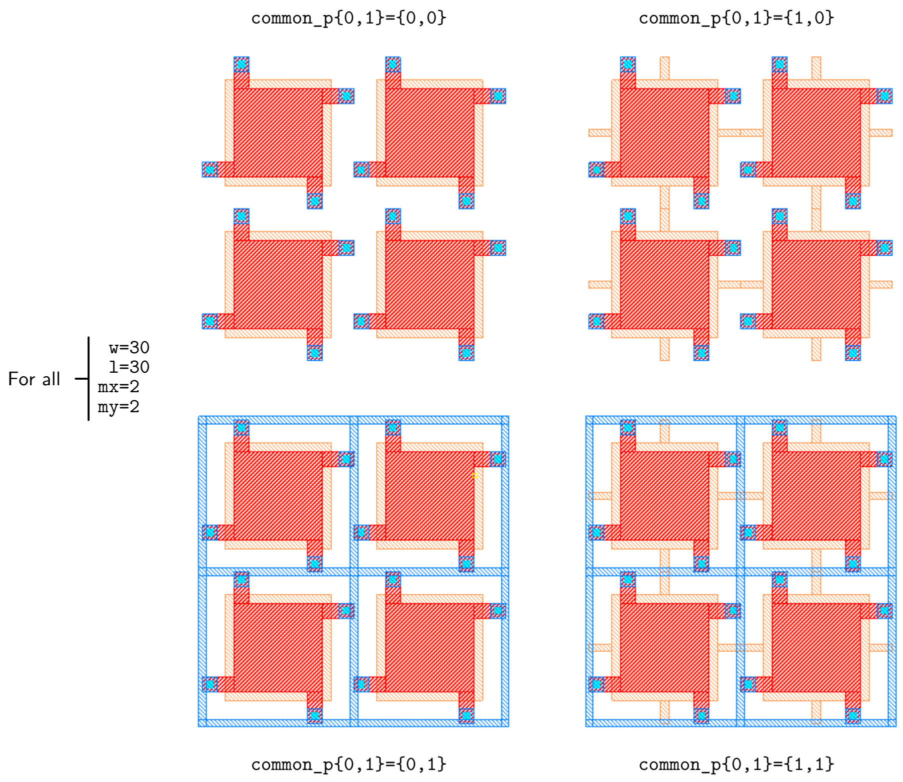


Figure 11 Example of automatic PiP capacitor generation using CNM25 PCell `cnm25cpoly_m`. Parameters for this PCell are: element width (`w`) and length (`l`), number of vertical (`mx`) and horizontal (`my`) elements, and conditional options for common bottom (`common_p0`) and/or top (`common_p1`) plates.

Concerning the connection of the above PCell devices inside your layout, Manhattan-style paths are of major help. Glade uses the physical layer connectivity defined in the technology file to switch from the current to the upper or downer routing layer through automatic contact or via generation. The CNM25 available layers for routing are `POLY1`, `METAL` and `METAL2`, as illustrated in Fig. 13.

Finally, the Glade MultiPartPath (MPP) command allows the automated generation of periodic linear structures, which are extremely useful to adapt guard rings and contact linear arrays around devices. Examples of the CNM25 MPPs are shown in Fig. 14, where `nguard` and `pguard` stand for N-type and P-type guard rings, while `p0m1`, `p1m1` and `m1m2` are intended for contact linear arrays between `METAL` and `POLY0`, `POLY1` and `METAL2`, respectively. All these layout elements are fully tunable after creation, as their periodic structure is automatically regenerated when stretched.

For further assistance with the design of the full-custom layout of your IC, Glade also features the schematic-driven layout generation tool of Fig. 15. First, this tool allows to manage matching groups at schematic level through device property `group`. Such management includes the definition of device array dimensions and distribution of unitary elements for better matching (e.g. common centroid). Second, the same tool automatically places all the required PCells in the layout and highlights their terminal connectivity according to the schematic information. However, it is designer responsibility to arrange such arrays and to route their interconnections for best device matching and signal decoupling, respectively.

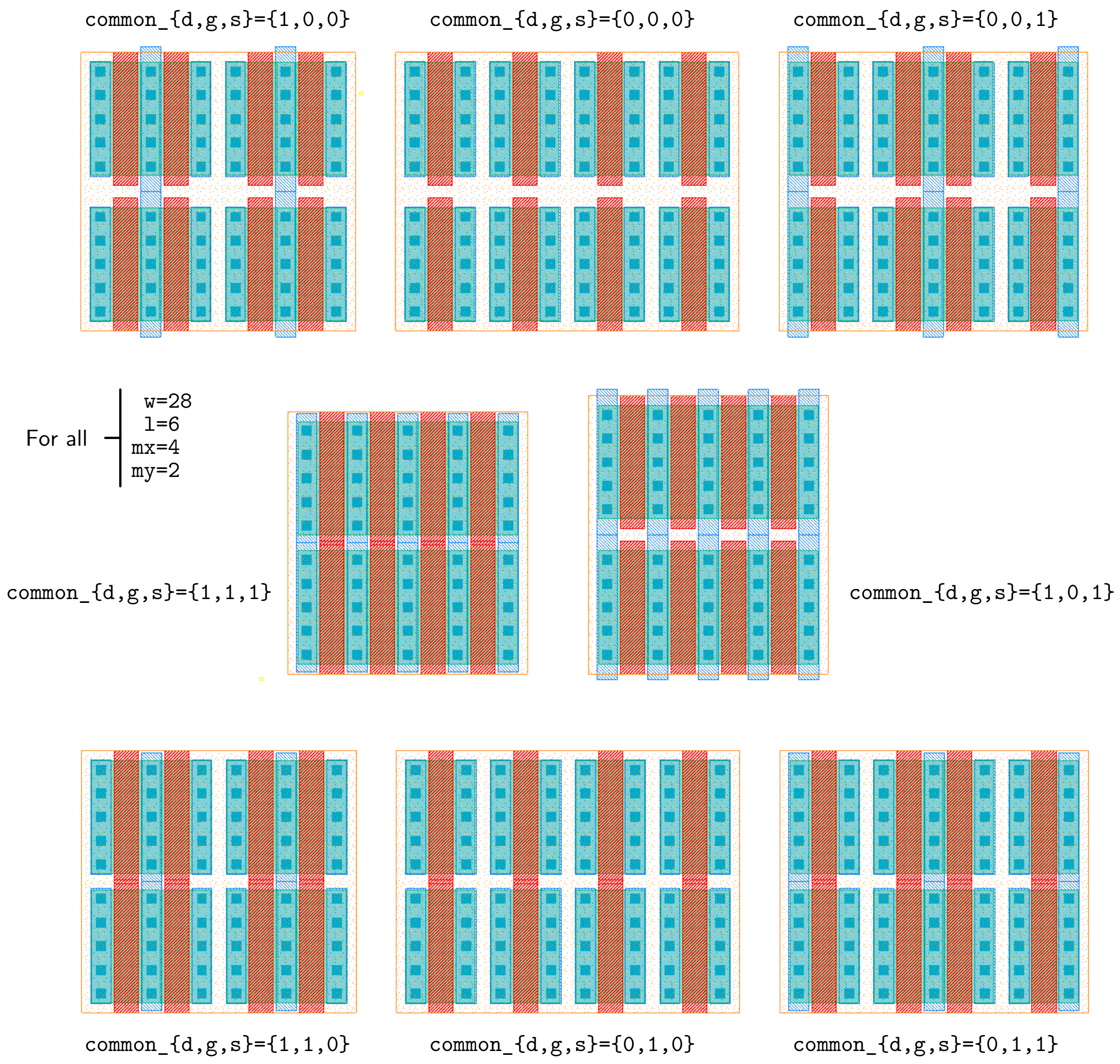


Figure 12 Example of automatic NMOS device generation using CNM25 PCell `cnm25modn_m`. Parameters for this PCell are: element width (`w`) and length (`l`), vertical (`mx`) and horizontal (`my`) multiplicity, and conditional options for common drain (`common_d`), gate (`common_g`), and/or source (`common_s`).

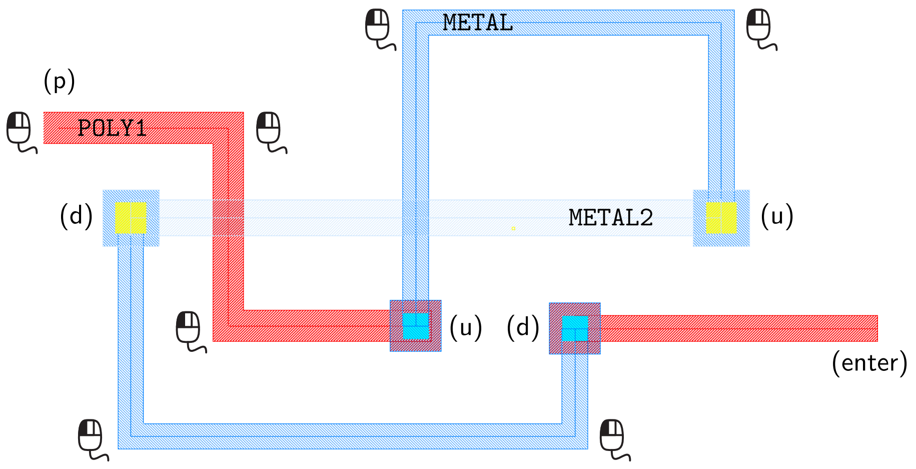
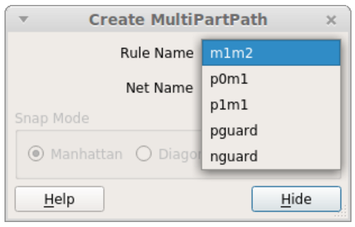


Figure 13 | Example of Glade Create→Path (p) command usage for CNM25 routing.



```
p0m1  = POLY0-WINDOW-METAL
m1m2  = METAL-VIA-METAL2
pguard = GASAD-WINDOW-METAL (P-bulk guard ring)
p1m1  = POLY1-WINDOW-METAL
nguard = NTUB-GASAD-NPLUS-WINDOW-METAL (N-bulk guard ring)
```

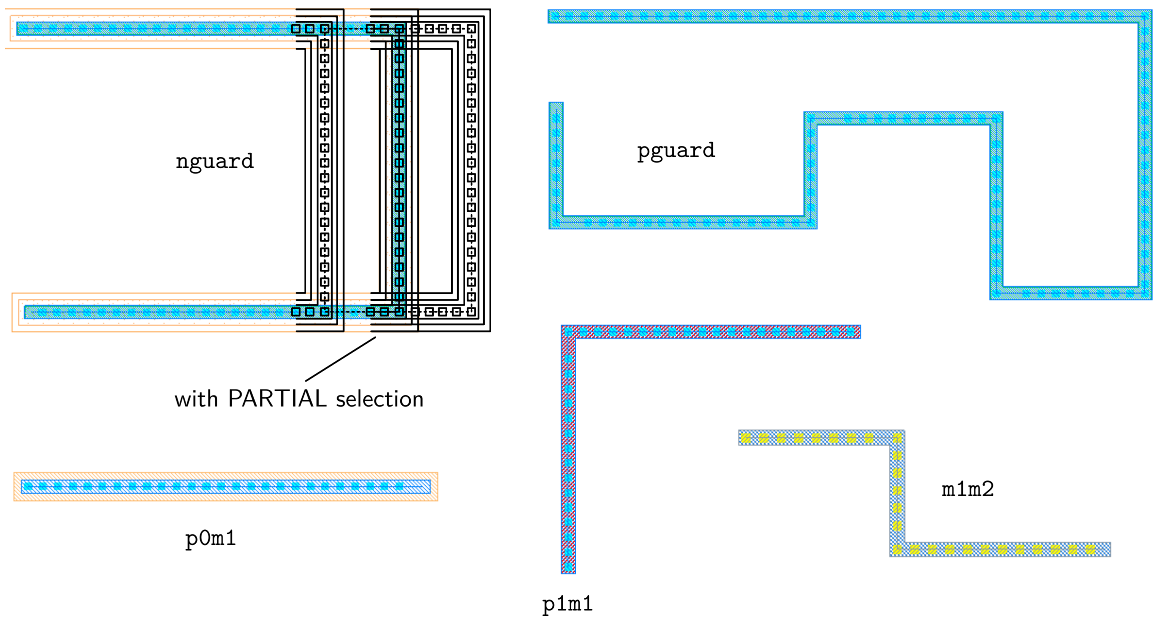
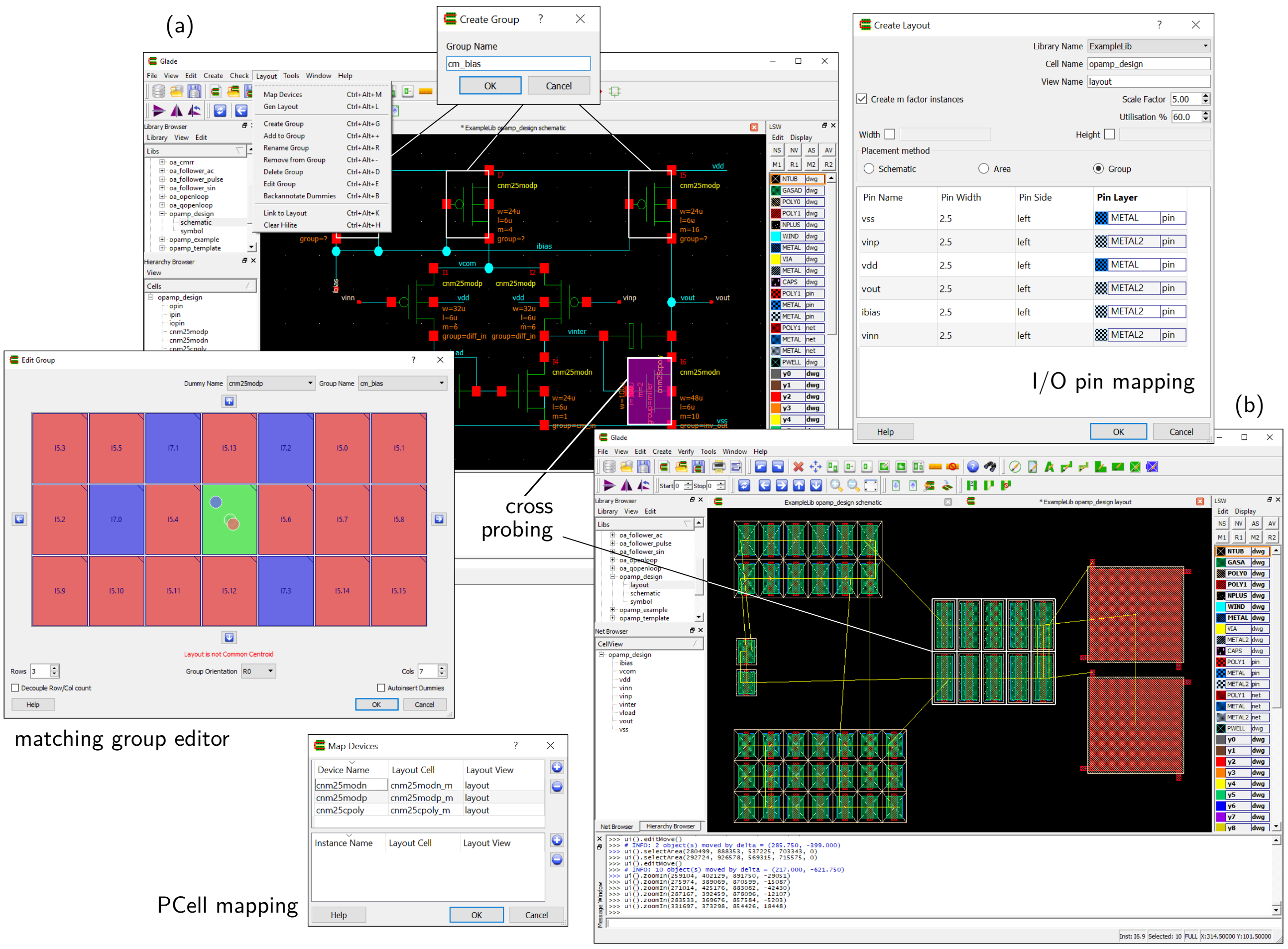


Figure 14 | CNM25 automatic and stretchable MPP structures available through the Glade Create→MultiPartPath command.



Create Group — Group Name: cm_bias

Edit Group — Dummy Name: cnm25modp; Group Name: cm_bias; Layout is not Common Centroid; Rows: 3; Cols: 7; Group Orientation: R0; Decouple Row/Col count; Autoinsert Dummies

Map Devices

Device Name	Layout Cell	Layout View
cnm25modn	cnm25modn_m	layout
cnm25modp	cnm25modp_m	layout
cnm25cpoly	cnm25cpoly_m	layout

Create Layout — Library Name: ExampleLib; Cell Name: opamp_design; View Name: layout; [✓ Create m factor instances]; Scale Factor: 5.00; Utilisation %: 60.0; Placement method: Schematic, Area, [✓ Group]

Pin Name	Pin Width	Pin Side	Pin Layer	
vss	2.5	left	METAL	pin
vinp	2.5	left	METAL2	pin
vdd	2.5	left	METAL	pin
vout	2.5	left	METAL2	pin
ibias	2.5	left	METAL2	pin
vinn	2.5	left	METAL2	pin

```
>>> ui().editMove()
>>> # INFO: 2 object(s) moved by delta = (285.750, -399.000)
>>> ui().selectArea(280499, 888353, 537225, 703343, 0)
>>> ui().selectArea(292724, 926578, 569315, 715575, 0)
>>> ui().editMove()
>>> # INFO: 10 object(s) moved by delta = (217.000, -621.750)
>>> ui().zoomIn(259104, 402129, 891750, -29051)
>>> ui().zoomIn(275974, 389069, 870599, -15087)
>>> ui().zoomIn(271014, 425176, 883082, -42430)
>>> ui().zoomIn(287167, 392459, 878096, -12107)
>>> ui().zoomIn(283533, 369676, 857584, -5203)
>>> ui().zoomIn(331697, 373298, 854426, 18448)
>>>
```

Figure 15 Usage of Glade layout generation tool for managing matching groups (a) and for automated PCell placement (b).

☞ Update device dimensions of Glade `ExampleLib`→`opamp_design`→`schematic` according to the **optimized** device sizes obtained in Section 4.3!

☞ Use the Glade layout generation tool of Fig. 15 to generate all PCell devices of your OpAmp in `ExampleLib`→`opamp_design`→`layout`.

☞ Check grid and snap parameters are set to **X=0.25** and **Y=0.25**, as in Fig. 9.

Q4. Complete the **full-custom layout** of your optimized OpAmp:

- Exploit the advantage of **paths** and **MPPs** as in Figs. 13 and 14.
- Follow the matching design **guidelines** of Table 6.
- Ensure layout compatibility with the cell **template** of Fig. 16.

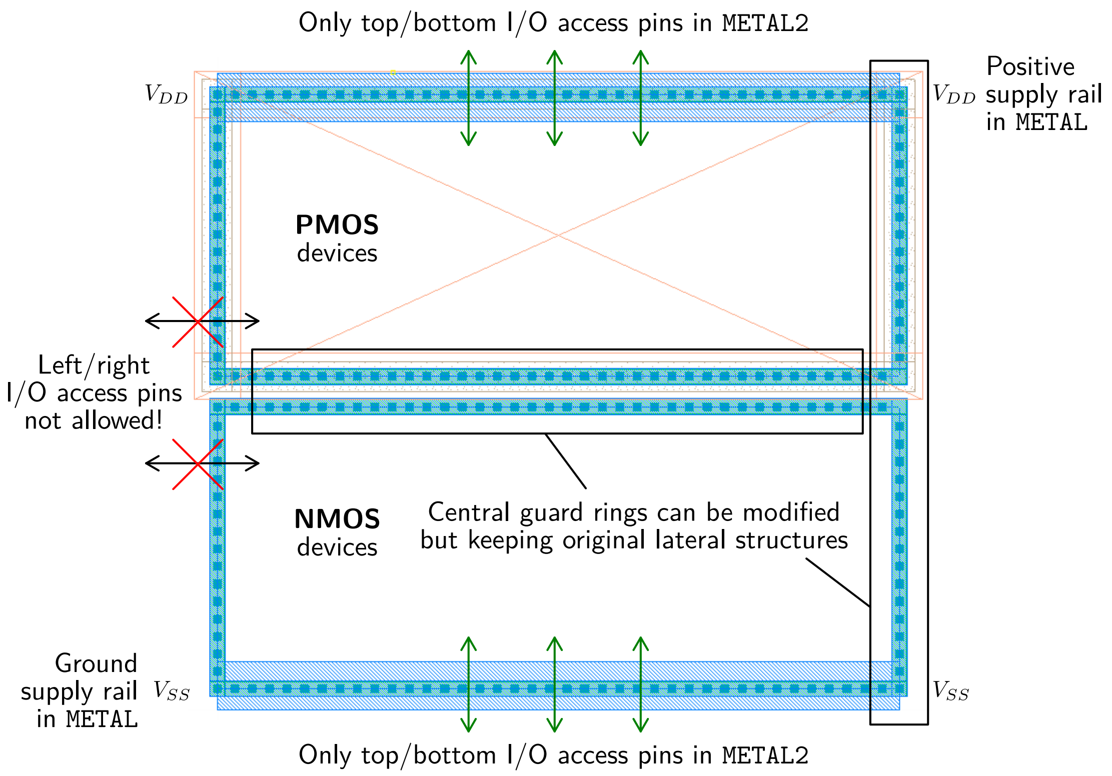

Only top/bottom I/O access pins in METAL2
V_{DD}
V_{DD}
Positive supply rail in METAL
PMOS devices
Left/right I/O access pins not allowed!
NMOS devices
Central guard rings can be modified but keeping original lateral structures
Ground supply rail in METAL
V_{SS}
V_{SS}
Only top/bottom I/O access pins in METAL2

Figure 16 Glade cell view ExampleLib→opamp_template→layout to be used for the design boundaries of your OpAmp layout. Standard-cell height is 200μm.

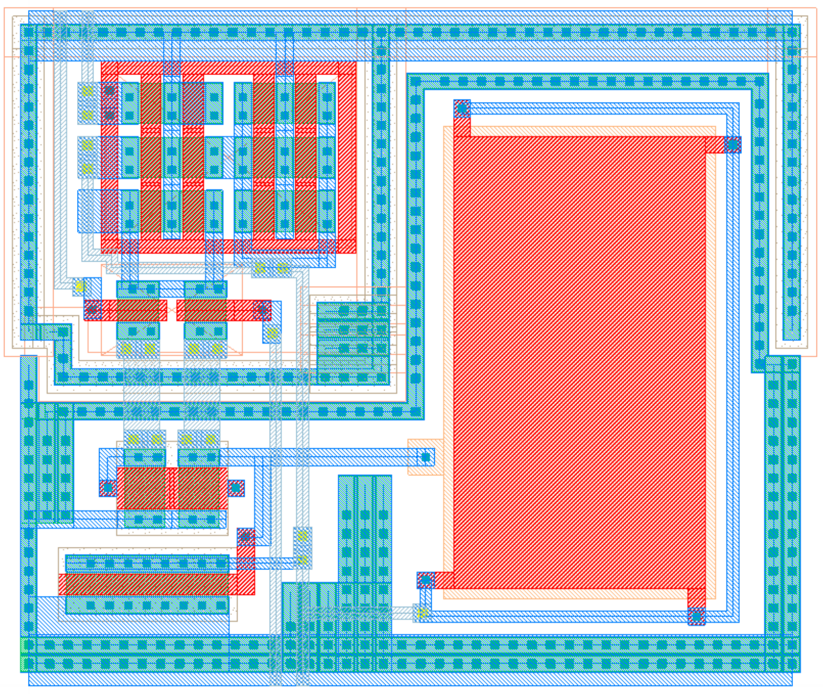


Figure 17 Glade cell view ExampleLib→opamp_example→layout to illustrate the use of the cell template of Fig. 16 and the matching guidelines of Table 6. Overall cell width is 226μm.

4.5 Design Rule Checker

Once the geometrical edition of your full-custom IC layout is completed, the first physical verification step in the methodology of Fig. 1 is the design rule checker (DRC). The main purpose of this stage is to verify the fabrication feasibility of your CMOS circuit in CNM25 from the geometrical viewpoint only (e.g. adequate spacing, width, overlap, extension, enclosure and area of the mask patterning).

In this sense, this PDK is already shipped with `damics-kit/glade/verification/cnm25drc.py`, a Python script which includes the main design rules of Table 2. Also, a sample layout with typical CNM25 structures is supplied in `ExampleLib`→`drc`→`layout` for DRC training, as shown in Fig. 18.

damics-kit/glade/verification/cnm25drc.py

```
# CNM25 2M DRC deck
def printErrors(msg) :
        n = geomGetCount()
        if n > 0 :
                print n, msg

# Initialise DRC package
from ui import *
cv = ui().getEditCellView()
drcInit(cv)

# Get raw layers
nwell     = geomGetShapes("NTUB", "drawing")
active    = geomGetShapes("GASAD", "drawing")
polygate  = geomGetShapes("POLY1", "drawing")
polycap   = geomGetShapes("POLY0", "drawing")
nimp      = geomGetShapes("NPLUS", "drawing")
cont      = geomGetShapes("WINDOW", "drawing")
metal1    = geomGetShapes("METAL", "drawing")
via12     = geomGetShapes("VIA", "drawing")
metal2    = geomGetShapes("METAL2", "drawing")
pad       = geomGetShapes("CAPS", "drawing")

# Form derived layers
gate        = geomAnd(polygate, active)
ngate       = geomAnd(gate, nimp)
pgate       = geomAndNot(gate, ngate)
cpoly       = geomAnd(polygate, polycap)
polygatecont= geomAnd(polygate, cont)
polycapcont = geomAnd(polycap, cont)
activecont  = geomAnd(active, cont)
allcon      = geomOr(geomOr(polygatecont, polycapcont),activecont)
badcon      = geomAndNot(allcon, metal1)
metal1via   = geomAnd(metal1, via12)
badvia      = geomAndNot(metal1via, metal2)
diff        = geomAndNot(active, gate)
ndiff       = geomAnd(diff, nimp)
pdiff       = geomAndNot(diff, nimp)
ntap        = geomAnd(ndiff, nwell)
ptap        = geomAndNot(pdiff, nwell)

# Form connectivity
geomConnect( [
              [ntap, nwell, ndiff],
              [cont, ndiff, metal1],
              [cont, pdiff, metal1],
              [cont, polygate, metal1],
              [cont, polycap, metal1],
              [via12, metal1, metal2]
             ] )

# Start design rule checking

print "0.0. Checking off-grid..."
geomOffGrid(nwell, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(active, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(polygate, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(polycap, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(nimp, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(cont, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(metal1, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(via12, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(metal2, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")
geomOffGrid(pad, 0.25, 1, "0.0. Design grid is 0.25um x 0.25um")

print "1.X. Checking N-well..."
geomWidth(nwell, 8, "1.1. N-well width >= 8um")
geomSpace(nwell, 8, samenet, "1.2. N-well spacing (same net) >= 8um")
geomSpace(nwell, 8, diffnet, "1.2. N-well spacing (different net) >= 8um")
geomNotch(nwell, 8, "1.2. N-well notch >= 8um")

print "2.X. Checking GASAD..."
geomWidth(active, 2, "2.1. GASAD width >= 2um")
geomSpace(active, 4, samenet, "2.2. GASAD spacing (same net) >= 4um")
geomSpace(active, 4, diffnet, "2.2. GASAD spacing (different net) >= 4um")
geomNotch(active, 4, "2.2. GASAD notch >= 4um")
geomEnclose(nwell, pdiff, 5, "2.3. N-well enclosure of P-plus active >= 5um")
geomSpace(nwell, ndiff, 5, samenet, "2.4. N-well spacing to N-plus active (same net) >= 5um")
geomSpace(nwell, ndiff, 5, diffnet, "2.4. N-well spacing to N-plus active (different net) >= 5um")

print "3.X. Checking Poly0..."
geomWidth(polycap, 2.5, "3.1. Poly0 width >= 2.5um")
geomSpace(polycap, 6, samenet, "3.2. Poly0 spacing (same net) >= 6um")
geomSpace(polycap, 6, diffnet, "3.2. Poly0 spacing (different net) >= 6um")
geomNotch(polycap, 6, "3.2. Poly0 notch >= 6um")
geomSpace(polycap, active, 6, samenet, "3.3. Poly0 spacing to GASAD (same net) >= 6um")
geomSpace(polycap, active, 6, diffnet, "3.3. Poly0 spacing to GASAD (different net) >= 6um")

print "4.X. Checking Poly1..."
geomWidth(gate, 3, "4.1.a. Poly1 width inside GASAD >= 3um")
geomWidth(geomAndNot(polygate, gate), 2.5, "4.1.b. Poly1 width outside GASAD >= 2.5um")
geomSpace(polygate, 3, samenet, "4.2. Poly1 spacing (same net) >= 3um")
geomSpace(polygate, 3, diffnet, "4.2. Poly1 spacing (different net) >= 3um")
geomNotch(polygate, 3, "4.2. Poly1 notch >= 3um")
geomExtension(active, polygate, 3 , "4.3. GASAD extension of Poly1 >= 3um")
geomExtension(polygate, active, 2.5, "4.4. Poly1 extension of GASAD >= 2.5um")
geomSpace(polygate, active, 1.25, samenet, "4.5. Poly1 spacing to GASAD (same net) >= 1.25um")
geomSpace(polygate, active, 1.25, diffnet, "4.5. Poly1 spacing to GASAD (different net) >= 1.25um")
geomEnclose(polycap, cpoly, 3, "4.6. Poly0 enclosure of Poly1 >= 3um")

print "5.X. Checking N-plus..."
geomEnclose(nimp, active, 2.5, "5.1. N-plus enclosure of GASAD >= 2.5um")
geomSpace(nimp, pdiff, 2.5, samenet, "5.2. N-plus spacing to P-plus active (same net) >= 2.5um")
geomSpace(nimp, pdiff, 2.5, diffnet, "5.2. N-plus spacing to P-plus active (different net) >= ...
geomSpace(nimp, pgate, 2, samenet, "5.3. N-plus spacing to Poly1 inside P-plus active (same ...
geomSpace(nimp, pgate, 2, diffnet, "5.3. N-plus spacing to Poly1 inside P-plus active (different ...
geomExtension(nimp, ngate, 1.5, "5.4. N-plus extension of Poly1 inside N-plus active >= 1.5um")
geomWidth(nimp, 2.5, "5.5. N-plus width >= 2.5um")
geomSpace(nimp, 2.5, samenet, "5.6. N-plus spacing (same net) >= 2.5um")
geomSpace(nimp, 2.5, diffnet, "5.6. N-plus spacing (different net) >= 2.5um")
geomNotch(nimp, 2.5, "5.6. N-plus notch >= 2.5um")

print "6.X. Checking contact..."
saveDerived(badcon, "6.0. Contact requires Metal1")
geomArea(cont, 6.25, 6.25, "6.1. Exact contact size = 2.5um x 2.5um")
geomWidth(cont, 2.5, "6.1. Exact contact size = 2.5um x 2.5um")
geomSpace(cont, 3, samenet, "6.2. Contact spacing (same net) >= 3um")
geomSpace(cont, 3, diffnet, "6.2. Contact spacing (different net) >= 3um")
geomNotch(cont, 3, "6.2. Contact notch >= 3um")
geomEnclose(active, cont, 1, "6.3. GASAD enclosure of Contact >= 1um")
geomEnclose(polygate, cont, 1.25, "6.4. Poly1 enclosure of Contact >= 1.25um")
geomSpace(polygatecont, 2.5, samenet, "6.5. Poly1 Contact spacing to GASAD (same net) >= 2.5um")
geomSpace(polygatecont, 2.5, diffnet, "6.5. Poly1 Contact spacing to GASAD (different net) >= ...
geomSpace(cont, gate, 2, samenet, "6.6. Contact spacing to Poly1 inside GASAD (same net) >= 2um")
geomSpace(cont, gate, 2, diffnet, "6.6. Contact spacing to Poly1 inside GASAD (different net) >= ...
# 6.7 and 6.8 not implemented!
geomEnclose(polycap, cont, 4, "6.9. Poly0 enclosure of Contact >= 4um")
geomSpace(cont, cpoly, 4, diffnet, "6.10. Contact spacing to Poly1 & Poly0 (different net) >= 4um")

print "7.X. Checking Metal1..."
geomWidth(metal1, 2.5, "7.1. Metal1 width >= 2.5um")
geomSpace(metal1, 3, samenet, "7.2. Metal1 spacing (same net) >= 3um")
geomSpace(metal1, 3, diffnet, "7.2. Metal1 spacing (different net) >= 3um")
geomNotch(metal1, 3, "7.2. Metal1 notch >= 3um")
geomEnclose(metal1, cont, 1.25, "7.3. Metal1 enclosure of Contact >= 1.25um")

print "8.X. Checking Via..."
saveDerived(badvia, "8.0. Via requires Metal2")
geomArea(via12, 9, 9, "8.1. Exact via size = 3um x 3um")
geomWidth(via12, 3, "8.1. Exact via size = 3um x 3um")
geomSpace(via12, 3.5, samenet, "8.2. Via spacing (same net) >= 3.5um")
geomSpace(via12, 3.5, diffnet, "8.2. Via spacing (different net) >= 3.5um")
geomNotch(via12, 3.5, "8.2. Via notch >= 3.5um")
geomEnclose(metal1, via12, 1.25, "8.3. Metal1 enclosure of Via >= 1.25um")
geomSpace(via12, cont, 2.5, samenet, "8.4. Via spacing to contact (same net) >= 2.5um")
geomSpace(via12, cont, 2.5, diffnet, "8.4. Via spacing to contact (different net) >= 2.5um")
geomSpace(via12, polygate, 2.5, samenet, "8.5. Via spacing to Poly1 (same net) >= 2.5um")
geomSpace(via12, polygate, 2.5, diffnet, "8.5. Via spacing to Poly1 (different net) >= 2.5um")

print "9.X. Checking Metal2..."
geomWidth(metal2, 3.5, "9.1. Metal2 width >= 3.5um")
geomSpace(metal2, 3.5, samenet, "9.2. Metal2 spacing (same net) >= 3.5um")
geomSpace(metal2, 3.5, diffnet, "9.2. Metal2 spacing (different net) >= 3.5um")
geomNotch(metal2, 3.5, "9.2. Metal2 notch >= 3.5um")
geomEnclose(metal2, via12, 1.25, "9.3. Metal2 enclosure of Via >= 1.25um")

print "10.X. Checking Pad..."
geomArea(pad, 10000, 10000, "10.1. Exact passivation size = 100um x 100um")
geomWidth(pad, 100, "10.1. Exact passivation size = 100um x 100um")

num_err = geomGetTotalCount()
print "** Total error count = ", num_err

# Exit DRC package, freeing memory
drcUnInit()
```

damics-kit/glade/verification/cnm25drc.py

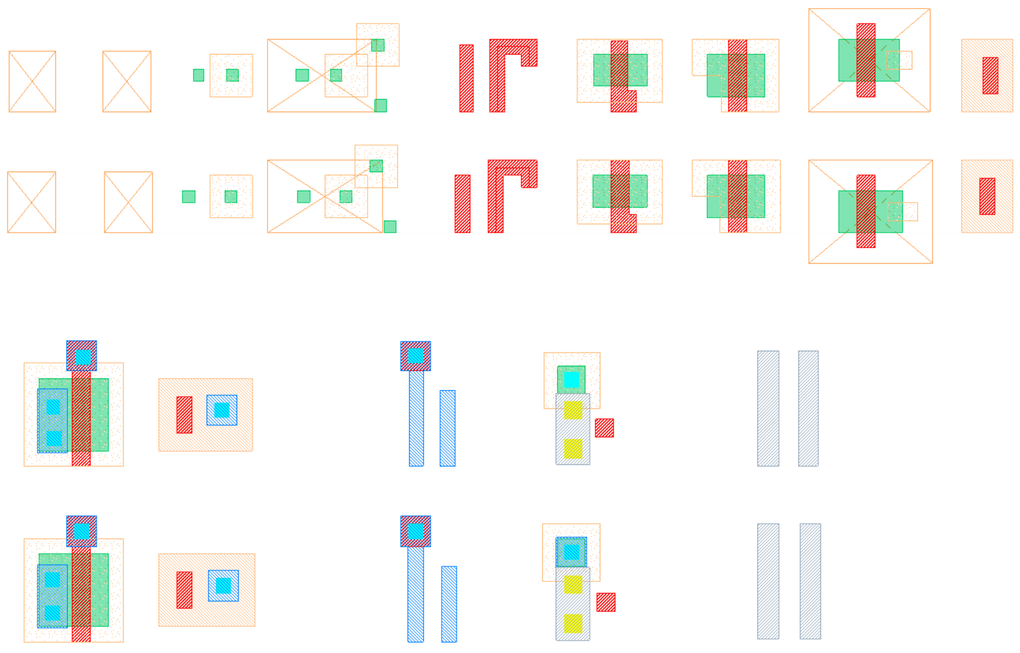
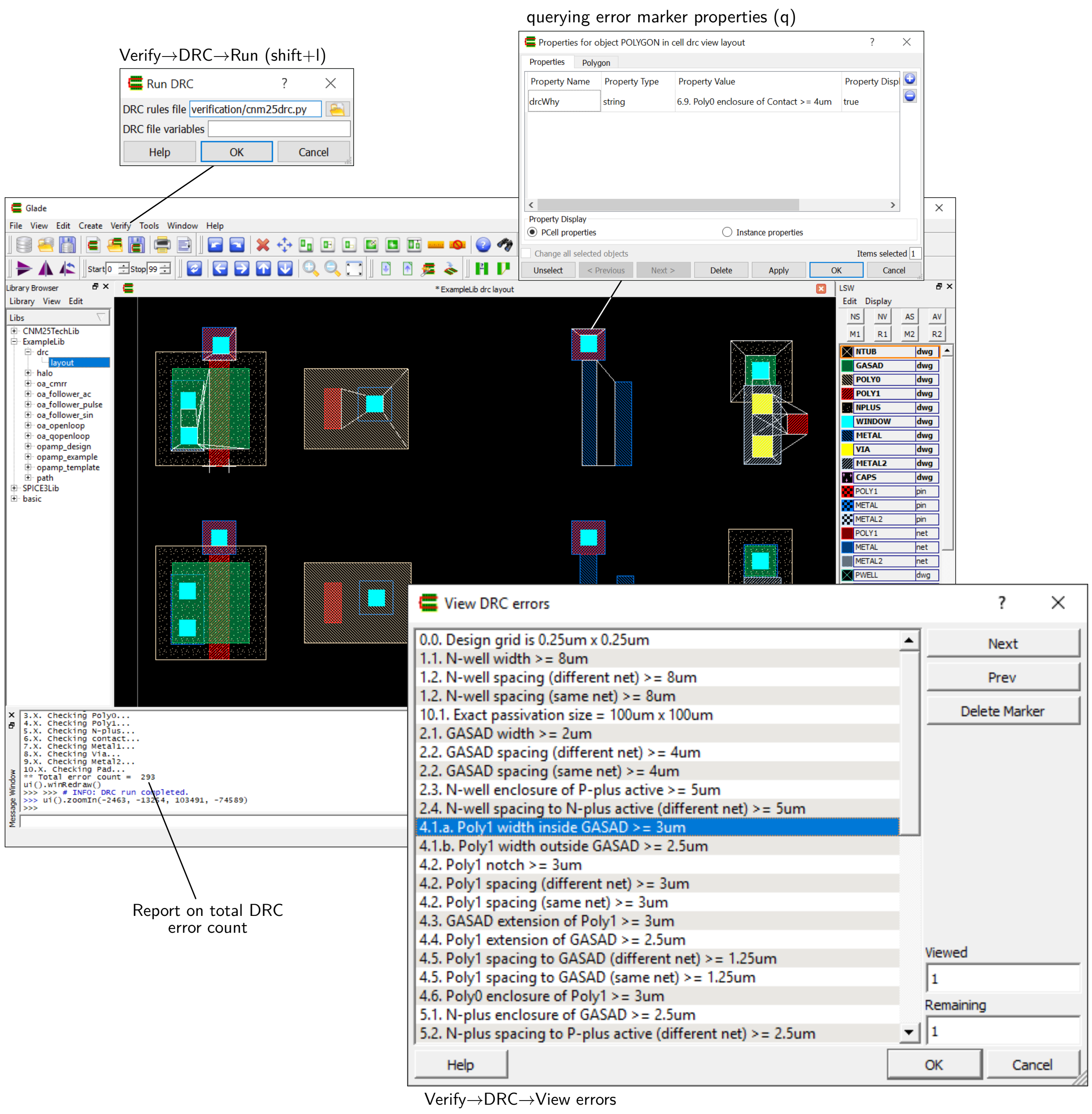


Figure 18 | Glade cell view `ExampleLib`→`drc`→`layout` includes examples of CNM25 layout structures with (top) and without (bottom) DRC errors.

Performing the full DRC verification of your OpAmp layout in Glade is as simple as following the procedure described below:

1. Open your `ExampleLib`→`opamp_design`→`layout`.
2. Execute Verify→DRC→Run (shift+I).
3. Select the rules file `damics-kit/glade/verification/cnm25drc.py` and launch the DRC process.
4. Open the DRC results browser through Verify→DRC→View Errors, or select a particular error marker and query for its properties (q), as shown in Fig. 19.
5. Correct all the reported errors according to the design rules of Table 2 and iterate above until the console window shows the following message: `** Total error count = 0`.

Q5. Execute the above verification procedure in your optimized OpAmp, and correct any rule violation in order to obtain a **DRC error-free layout**. Which are the most common DRC errors of your design?


Verify→DRC→Run (shift+l)
querying error marker properties (q)
Report on total DRC error count
Verify→DRC→View errors

Run DRC

DRC rules file: verification/cnm25drc.py
DRC file variables:

Properties for object POLYGON in cell drc view layout

Property Name	Property Type	Property Value	Property Displ
drcWhy	string	6.9. Poly0 enclosure of Contact >= 4um	true

Property Display: [✓ PCell properties] Instance properties

Items selected: 1

```
3.X. Checking Poly0...
4.X. Checking Poly1...
5.X. Checking N-plus...
6.X. Checking contact...
7.X. Checking Metal1...
8.X. Checking Via...
9.X. Checking Metal2...
10.X. Checking Pad...
** Total error count =  293
ui().winRedraw()
>>> >>> # INFO: DRC run completed.
>>> ui().zoomIn(-2463, -13254, 103491, -74589)
>>>
```

View DRC errors

```
0.0. Design grid is 0.25um x 0.25um
1.1. N-well width >= 8um
1.2. N-well spacing (different net) >= 8um
1.2. N-well spacing (same net) >= 8um
10.1. Exact passivation size = 100um x 100um
2.1. GASAD width >= 2um
2.2. GASAD spacing (different net) >= 4um
2.2. GASAD spacing (same net) >= 4um
2.3. N-well enclosure of P-plus active >= 5um
2.4. N-well spacing to N-plus active (different net) >= 5um
4.1.a. Poly1 width inside GASAD >= 3um
4.1.b. Poly1 width outside GASAD >= 2.5um
4.2. Poly1 notch >= 3um
4.2. Poly1 spacing (different net) >= 3um
4.2. Poly1 spacing (same net) >= 3um
4.3. GASAD extension of Poly1 >= 3um
4.4. Poly1 extension of GASAD >= 2.5um
4.5. Poly1 spacing to GASAD (different net) >= 1.25um
4.5. Poly1 spacing to GASAD (same net) >= 1.25um
4.6. Poly0 enclosure of Poly1 >= 3um
5.1. N-plus enclosure of GASAD >= 2.5um
5.2. N-plus spacing to P-plus active (different net) >= 2.5um
```

Viewed: 1
Remaining: 1

Figure 19 Example of Glade DRC error browsing in `ExampleLib`→`drc`→`layout` cell view.

4.6 Circuit Extraction and Electrical Rule Checker

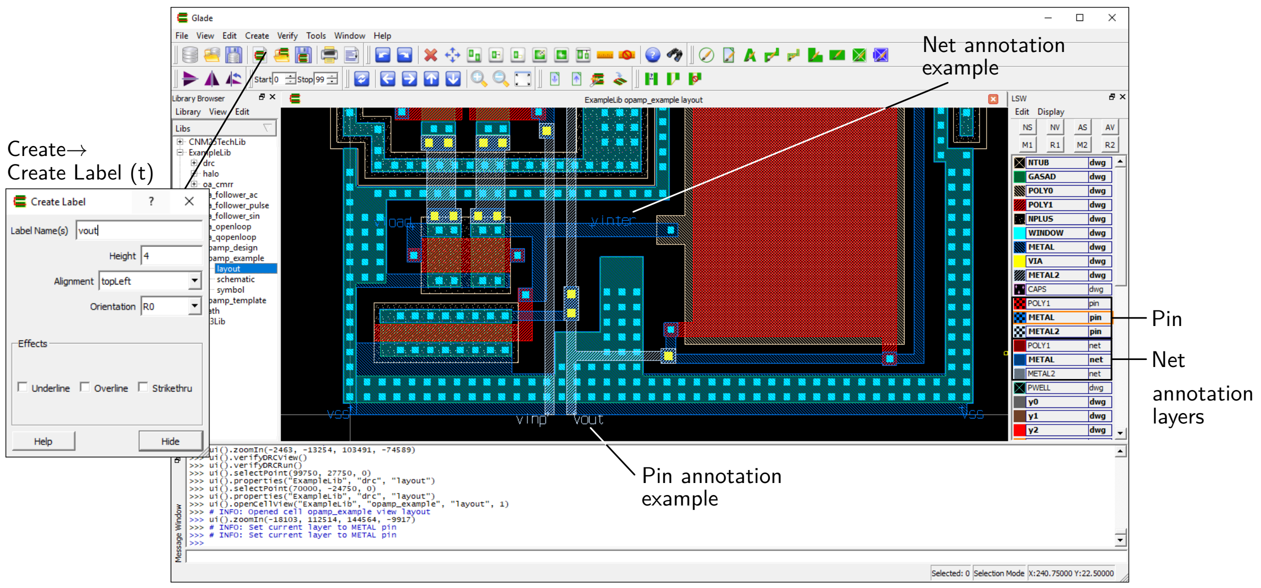
According to the full-custom design methodology of Fig. 1, the next physical verification step is the extraction of the equivalent electrical circuit from your DRC-compliant IC layout geometry. Previously, the following information about the circuit connectivity needs to be declared:

Pin annotation. In this step, the input/output (I/O) pins of the circuit cell are located in the layout. After executing the procedure listed below, the placed pins should look like in the OpAmp example of Fig. 20:

1. Select `METAL2-pin` layer from the layer selection window (LSW).
2. Create a text label with origin within the I/O pin location through Create→Create Label (t).
3. Enter the same pin name as in the OpAmp schematic of Fig. 5.
4. Iterate above for each I/O pin of the OpAmp (i.e. `vinn`, `vinp`, `vout` and `ibias`).
5. Repeat steps 1 to 4 for the supply pins (i.e. `vdd` and `vss`) but using `METAL-pin` layer.

Net annotation. This step is optional, since the internal nodes of the circuit are automatically named during the extraction process if no labels are present. Anyway, it is a good practice to specify the net names for all the internal circuit nodes in order to simplify the debugging of any connectivity error in Section 4.7. Again, Fig. 20 illustrates the following process:

1. Select the appropriate `{POLY1,METAL,METAL2}-net` layer from the LSW.
2. Create a text label with origin in the internal net area through Create→Create Label (t).
3. Enter the same net name as in the OpAmp schematic of Fig. 5.
4. Iterate above for each internal net of the OpAmp (i.e. `vcomm`, `vinter` and `vload`).



Create Label
Label Name(s) vout
Height 4
Alignment topLeft
Orientation R0
Effects
☐ Underline ☐ Overline ☐ Strikethru
Help Hide

```
>>> ui().zoomIn(-2463, -13254, 103491, -74589)
>>> ui().verifyDRCView()
>>> ui().verifyDRCRun()
>>> ui().selectPoint(99750, 27750, 0)
>>> ui().properties("ExampleLib", "drc", "layout")
>>> ui().selectPoint(70000, -24750, 0)
>>> ui().properties("ExampleLib", "drc", "layout")
>>> ui().openCellView("ExampleLib", "opamp_example", "layout", 1)
>>> # INFO: Opened cell opamp_example view layout
>>> ui().zoomIn(-18103, 112514, 144564, -9917)
>>> # INFO: Set current layer to METAL pin
>>> # INFO: Set current layer to METAL pin
>>>
```

Selected: 0 Selection Mode X:240.75000 Y:22.50000

Figure 20 | Glade cell view `ExampleLib`→`opamp_example`→`layout` with pin and net annotation examples.

Once the electrical I/O connectivity information is annotated into the layout by the designer, the automated circuit extraction itself can start. In general, this process involves the following tasks:

- Identify the location of all instances of the technology native devices present in the layout.
- Extract the connectivity between these devices (and also between them and the I/O pins).
- For each identified device, call the corresponding extraction PCells to compute its sizing parameters.

As expected, the PDK comes with an specific Python code to cover such functionalities, which is called `damics-kit/glade/verification/cnm25xtr_lvs.py` and is shown below.

damics-kit/glade/verification/cnm25xtr_lvs.py

```

# CNM25 2M extraction deck
# Initialise boolean package
from ui import *
ui = cvar.uiptr
cv = ui.getEditCellView()
geomBegin(cv)
libxtrlvs = cv.lib()

# Get raw layers
nwell     = geomGetShapes("NTUB", "drawing")
active    = geomGetShapes("GASAD", "drawing")
polygate  = geomGetShapes("POLY1", "drawing")
polycap   = geomGetShapes("POLY0", "drawing")
nimp      = geomGetShapes("NPLUS", "drawing")
cont      = geomGetShapes("WINDOW", "drawing")
metal1    = geomGetShapes("METAL", "drawing")
via12     = geomGetShapes("VIA", "drawing")
metal2    = geomGetShapes("METAL2", "drawing")

# Form derived layers
bkgnd       = geomBkgnd()
pwell       = geomAndNot(bkgnd, nwell)
gate        = geomAnd(polygate, active)
ngate       = geomAnd(gate, nimp)
pgate       = geomAndNot(gate, ngate)
cpoly       = geomAnd(polygate, polycap)
diff        = geomAndNot(active, gate)
ndiff       = geomAnd(diff, nimp)
pdiff       = geomAndNot(diff, nimp)
ntap        = geomAnd(ndiff, nwell)
ptap        = geomAnd(pdiff, pwell)

# Extract pin and net names before geomConnect
geomLabel(polygate, "POLY1", "pin", 1)
geomLabel(metal1, "METAL", "pin", 1)
geomLabel(metal2, "METAL2", "pin", 1)
geomLabel(polygate, "POLY1", "net", 0)
geomLabel(metal1, "METAL", "net", 0)
geomLabel(metal2, "METAL2", "net", 0)

# Form connectivity
geomConnect( [
        [pwell, bkgnd, pwell],
        [ptap, pwell, pdiff],
        [ntap, nwell, ndiff],
        [cont, ndiff, pdiff, polygate, polycap, metal1],
        [via12, metal1, metal2]
       ] )

# Save interconnect
saveInterconnect( [
                [pwell, "PWELL"],
                nwell,
                [ptap, "GASAD"],
                [ntap, "GASAD"],
                [ndiff, "GASAD"],
                [pdiff, "GASAD"],
                polycap,
                polygate,
                cont,
                metal1,
                via12,
                metal2,
                ] )

# Extracting devices
if geomNumShapes(ngate) > 0 :
        print "# Extract NMOS transistors"
        extractMOS("cnm25modn", ngate, polygate, ndiff, pwell)

if geomNumShapes(pgate) > 0 :
        print "# Extract PMOS transistors"
        extractMOS("cnm25modp", pgate, polygate, pdiff, nwell)

if geomNumShapes(cpoly) > 0 :
        print "# Extract PiP capacitors"
        extractDevice("cnm25cpoly", cpoly, [[polygate, "T"], [polycap, "B"]])

print "# Ending circuit extraction"
geomEnd()

# Opening extracted view and reporting results
ui.openCellView(libxtrlvs.libName(), cv.cellName(), "extracted")
cv_ex = libxtrlvs.dbFindCellViewByName(cv.cellName(), "extracted")
box = cv_ex.bBox()
objs = cv_ex.dbGetOverlaps(box,0,1)
obj = objs.first()
num_dev=0
while obj :
        if obj.isInst() :
                num_dev=num_dev+1
        obj = objs.next()
print "** Total device count = ", num_dev

```

damics-kit/glade/verification/cnm25xtr_lvs.py

During the execution of the above script (see code lines 68-78), the corresponding CNM25 extraction PCells are called for each identified device of Fig. 2. These Python PCells are in charge of extracting device sizing properties and annotating device terminal locations, like in the case of CNM25 PiP capacitors:

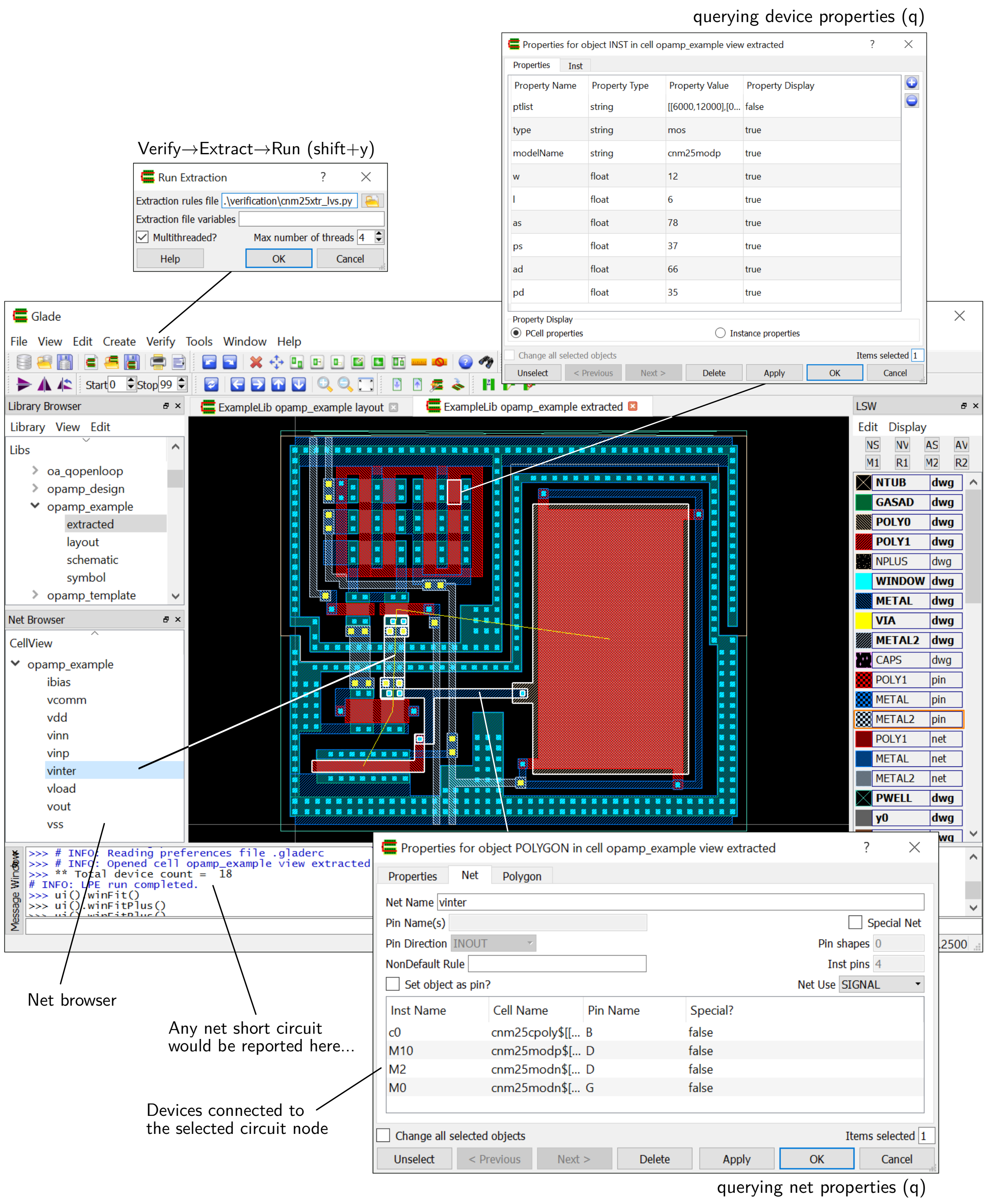
damics-kit/glade/pcells/cnm25cpoly.py

```
from ui import *
def cnm25cpoly(cv, ptlist=[[0,0],[30000,0],[30000,30000],[0,30000]]) :
    lib = cv.lib()
    dbu = float(lib.dbuPerUU())
    npts = len(ptlist)

    # Calculate total area an perimeter for arbitrary Manhattan shapes
    asum = 0.0
    perimeter = 0.0
    i = npts-1
    j = 0
    while (j < npts) :
        dx  = float(ptlist[i][0]) / dbu
        dy  = float(ptlist[i][1]) / dbu
        dx1 = float(ptlist[j][0]) / dbu
        dy1 = float(ptlist[j][1]) / dbu
        # compute perimeter
        perimeter = perimeter + ((dx1 - dx) * (dx1 - dx) + (dy1 - dy) * (dy1 - dy))**0.5
        # compute area
        asum = asum + (dx + dx1) * (dy1 - dy)

        # increment vertex
        i = j
        j = j + 1
    area = asum / 2.0

    # Derive rectangular w and l properties:
    # area = w*l
    # perimeter = 2*(w+l)
    a = 1.0
    b = -perimeter / 2.0
    c = area
    l = float((-b+(b**2-4*a*c)**0.5)/(2*a))
    w = float(area/l)

    # Update the master pcell property
    cv.dbAddProp("w", w*1e-6)
    cv.dbAddProp("l", l*1e-6)

    # Create the recognition region shape
    xpts = intarray(npts)
    ypts = intarray(npts)
    for i in range (npts) :
        xpts[i] = ptlist[i][0]
        ypts[i] = ptlist[i][1]
    cv.dbCreatePolygon(xpts, ypts, npts, TECH_Y0_LAYER);

    # Create pins
    top_net = cv.dbCreateNet("T")
    cv.dbCreatePin("T", top_net, DB_PIN_INPUT)
    bot_net = cv.dbCreateNet("B")
    cv.dbCreatePin("B", bot_net, DB_PIN_INPUT)

    # Setting device type to capacitor
    cv.dbAddProp("type", "cap")

    # Set the device modelName property for netlisting
    cv.dbAddProp("modelName", "cnm25cpoly")

    # Set the NLP property for netlisting
    cv.dbAddProp("NLPDeviceFormat", "c[@instName] [|T:%] [|B:%] [@modelName] [@w:w=%:w=30u]
                                                       [@l:l=%:l=30u] [@m:m=%:m=1]")
    cv.dbAddProp("NLPDeviceFormatCDL", "x[@instName] [|T:%] [|B:%] [@modelName] [@w:w=%:w=30u]
                                                       [@l:l=%:l=30u] [@m:m=%:m=1]")

    # Update the bounding box
    cv.update()
```
damics-kit/glade/pcells/cnm25cpoly.py

As a result of the extraction process, Glade creates the `extracted` view of your cell layout and reports any short-circuit between labeled nets in the message window. However, the above extraction script does not identify other types of electrical errors (e.g. open circuits). For such a purpose, the net browser of Fig. 21 can be used as an interactive electrical rule checker (ERC), since it highlights the physical location of a given net, including multi-layer nets. Also, the list of device terminals attached to a particular circuit net can be easily explored through the query function. In consequence, the ERC steps are as follows:

1. Open your `ExampleLib`→`opamp_design`→`layout`.
2. Execute Verify→Extract→Run (shift+y).
3. Select the script `damics-kit/glade/verification/cnm25xtr_lvs.py` and launch the extraction process.
4. Inspect for any noticeable ERC error, as depicted in Fig. 21.
5. If present, correct ERC errors in the layout and iterate Section 4.5 and 4.6.

Q6. Execute the above verification procedure in your optimized OpAmp layout to obtain an **ERC error-free extracted view**. Did you find any connectivity error?


querying device properties (q)
Verify→Extract→Run (shift+y)
Net browser
Any net short circuit would be reported here...
Devices connected to the selected circuit node
querying net properties (q)
Glade
File View Edit Create Verify Tools Window Help
Library Browser
Net Browser
LSW

Run Extraction

Extraction rules file .\verification\cnm25xtr_lvs.py
Extraction file variables
☑ Multithreaded? Max number of threads 4

Properties for object INST in cell opamp_example view extracted

Property Name	Property Type	Property Value	Property Display
ptlist	string	[[6000,12000],[0...	false
type	string	mos	true
modelName	string	cnm25modp	true
w	float	12	true
l	float	6	true
as	float	78	true
ps	float	37	true
ad	float	66	true
pd	float	35	true

Property Display: [✓ PCell properties] Instance properties
Items selected 1

Libs: oa_qopenloop, opamp_design, opamp_example (extracted, layout, schematic, symbol), opamp_template

CellView: opamp_example: ibias, vcomm, vdd, vinn, vinp, vinter, vload, vout, vss

```
>>> # INFO: Reading preferences file .gladerc
>>> # INFO: Opened cell opamp_example view extracted
>>> ** Total device count =  18
# INFO: LPE run completed.
>>> ui().winFit()
>>> ui().winFitPlus()
```

Properties for object POLYGON in cell opamp_example view extracted

Net Name: vinter
Pin Direction: INOUT
Pin shapes: 0
Inst pins: 4
Net Use: SIGNAL
☐ Special Net
☐ Set object as pin?

Inst Name	Cell Name	Pin Name	Special?
c0	cnm25cpoly$[[...	B	false
M10	cnm25modp$[...	D	false
M2	cnm25modn$[...	D	false
M0	cnm25modn$[...	G	false

Items selected 1

Figure 21 | Glade ERC inspection for `ExampleLib`→`opamp_example`→`extracted`.

4.7 Layout Versus Schematic

Passing the DRC and ERC validation steps ensures your circuit layout is both compliant with the geometrical rules of the CMOS technology and it is free of basic interconnection errors. However, these properties are useless if the resulting layout is not exactly equivalent to your optimized circuit schematic. For this reason, the design methodology of Fig. 1 also incorporates the layout versus schematic (LVS) verification step.

The basic idea behind the LVS checking is to take two circuit netlists, one obtained from the layout circuit extraction of Section 4.6 and the other one from the simulated schematic, and compare their topologies to identify element-by-element correspondences at pin, net and device levels. As a result of this comparison, open and short circuits, missing devices, device properties mismatching and pin mismatching errors can be identified in the layout. In general, LVS involves the solution of graph isomorphism problems, but taking benefit of the reduction and permutation properties of the specific devices, like in the example of Fig. 22.

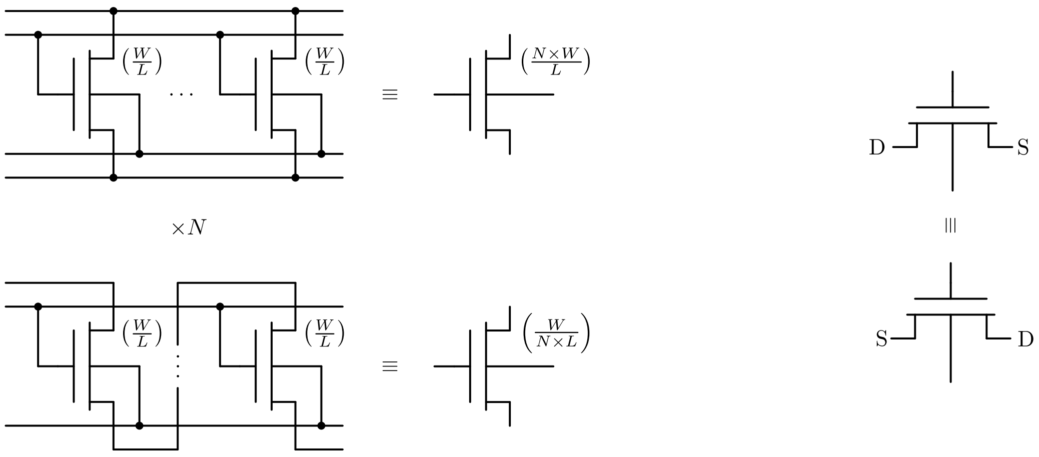

$\left(\frac{W}{L}\right)$
$\left(\frac{W}{L}\right)$
$\equiv$
$\left(\frac{N\times W}{L}\right)$
$\times N$
$\left(\frac{W}{L}\right)$
$\left(\frac{W}{L}\right)$
$\equiv$
$\left(\frac{W}{N\times L}\right)$
D
S
$\equiv$
S
D

Figure 22 | Typical reduction (left) and permutation (right) LVS rules for the case of MOS transistors.

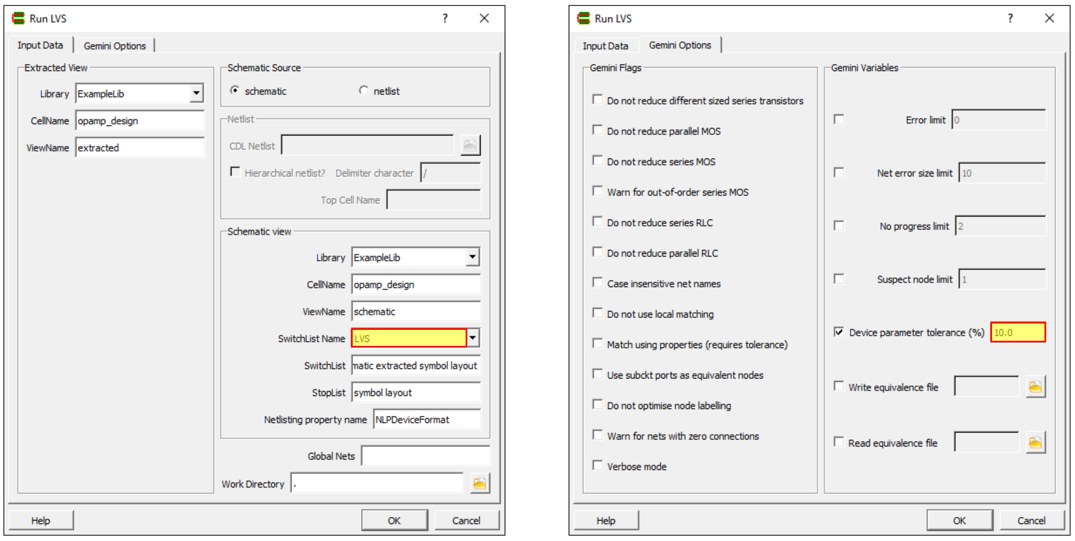
In our context, Glade relies on the venerable tool Gemini[1] [7] for the LVS verification. First, Glade generates the SPICE netlist of both the target schematic and the extracted layout, and then it calls Gemini to perform the comparison process between both netlists. The LVS verification results returned by Gemini are finally reported in the Glade message window, while both net and device errors are directly highlighted through geometrical markers in the extracted view.

In practice, the implementation of the analog layout guidelines proposed in Table 6 involve the introduction of some dummy elements inside the device matching arrays of your circuit layout to improve geometrical symmetry. These extra devices will be extracted by Glade, thus they need to be added in the schematic view as well.

[1]More information can be found at `https://www.cs.washington.edu/node/2177`.

Based on the above explanation, the complete steps to perform the LVS verification process are as follows:

1. Add the equivalent **dummy devices** of your layout to `ExampleLib`→`opamp_design`→`schematic`!
2. Open your `ExampleLib`→`opamp_design`→`extracted` view.
3. Execute Verify→LVS→Run (shift+l).
4. Configure Gemini according to Fig. 23 and run the LVS process.
5. Once completed, review the results in the message window and in the extracted view.



Run LVS

Input Data | Gemini Options

Extracted View
Library: ExampleLib
CellName: opamp_design
ViewName: extracted

Schematic Source
[✓ schematic] netlist

Netlist
CDL Netlist:
Hierarchical netlist? [ ] Delimiter character: /
Top Cell Name:

Schematic view
Library: ExampleLib
CellName: opamp_design
ViewName: schematic
SwitchList Name: LVS
SwitchList: matic extracted symbol layout
StopList: symbol layout
Netlisting property name: NLPDeviceFormat

Global Nets:
Work Directory: .

Help | OK | Cancel

Run LVS

Input Data | Gemini Options

Gemini Flags
[ ] Do not reduce different sized series transistors
[ ] Do not reduce parallel MOS
[ ] Do not reduce series MOS
[ ] Warn for out-of-order series MOS
[ ] Do not reduce series RLC
[ ] Do not reduce parallel RLC
[ ] Case insensitive net names
[ ] Do not use local matching
[ ] Match using properties (requires tolerance)
[ ] Use subckt ports as equivalent nodes
[ ] Do not optimise node labelling
[ ] Warn for nets with zero connections
[ ] Verbose mode

Gemini Variables
[ ] Error limit: 0
[ ] Net error size limit: 10
[ ] No progress limit: 2
[ ] Suspect node limit: 1
[✓] Device parameter tolerance (%): 10.0
[ ] Write equivalence file
[ ] Read equivalence file

Help | OK | Cancel

Figure 23 | Gemini configuration for the LVS verification of your `ExampleLib`→`opamp_design`→`extracted` view.

For illustration purposes, the PDK comes with the schematic of Fig. 24, which is the equivalent circuit of the OpAmp layout example used in the previous section and presented in Fig. 17. In this case, one PMOS dummy device needs to be added to match the physical transistor array of current mirrors.

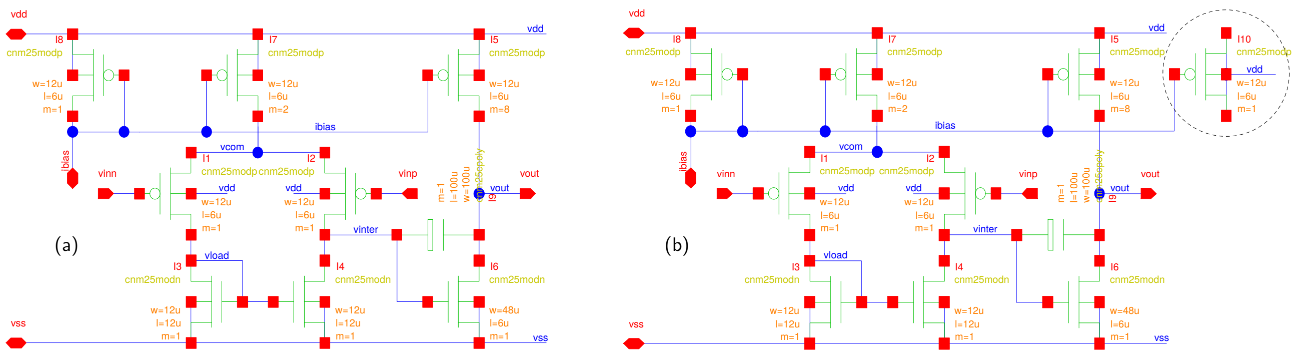


Figure 24 | Glade `ExampleLib`→`opamp_example`→`schematic` view of the OpAmp layout example used in Fig. 21 before (a) and after (b) including the corresponding layout dummy devices.

The resulting SPICE schematic netlists **before** (left) and **after** (right) dummy-element correction are:

damics-kit/glade/opamp_example.cdl

```
************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_example
* View Name:    schematic
************************************************

.SUBCKT opamp_example vinn vinp vout vdd vss ibias
*.PININFO vss:B vinp:I vdd:B vout:O vinn:I ibias:B

MI7 vdd ibias vcom vdd cnm25modp w=12u l=6u m=2
MI3 vload vload vss vss cnm25modn w=12u l=12u m=1
MI1 vcom vinn vload vdd cnm25modp w=12u l=6u m=1
CI9 vout vinter cnm25cpoly w=100u l=100u m=1
MI4 vinter vload vss vss cnm25modn w=12u l=12u m=1
MI2 vcom vinp vinter vdd cnm25modp w=12u l=6u m=1
MI5 vdd ibias vout vdd cnm25modp w=12u l=6u m=8
MI8 vdd ibias ibias vdd cnm25modp w=12u l=6u m=1
MI6 vout vinter vss vss cnm25modn w=48u l=6u m=1
.ENDS
```

damics-kit/glade/opamp_example.cdl

```
************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_example
* View Name:    schematic
************************************************

.SUBCKT opamp_example vinn vinp vout vdd vss ibias
*.PININFO vss:B vinp:I vdd:B vout:O vinn:I ibias:B

MI7 vdd ibias vcom vdd cnm25modp w=12u l=6u m=2
MI10 vdd ibias vdd vdd cnm25modp w=12u l=6u m=1
MI3 vload vload vss vss cnm25modn w=12u l=12u m=1
MI1 vcom vinn vload vdd cnm25modp w=12u l=6u m=1
CI9 vout vinter cnm25cpoly w=100u l=100u m=1
MI4 vinter vload vss vss cnm25modn w=12u l=12u m=1
MI2 vcom vinp vinter vdd cnm25modp w=12u l=6u m=1
MI5 vdd ibias vout vdd cnm25modp w=12u l=6u m=8
MI8 vdd ibias ibias vdd cnm25modp w=12u l=6u m=1
MI6 vout vinter vss vss cnm25modn w=48u l=6u m=1
.ENDS
```

During the LVS verification, Gemini compares the above circuits to the following extracted netlist:

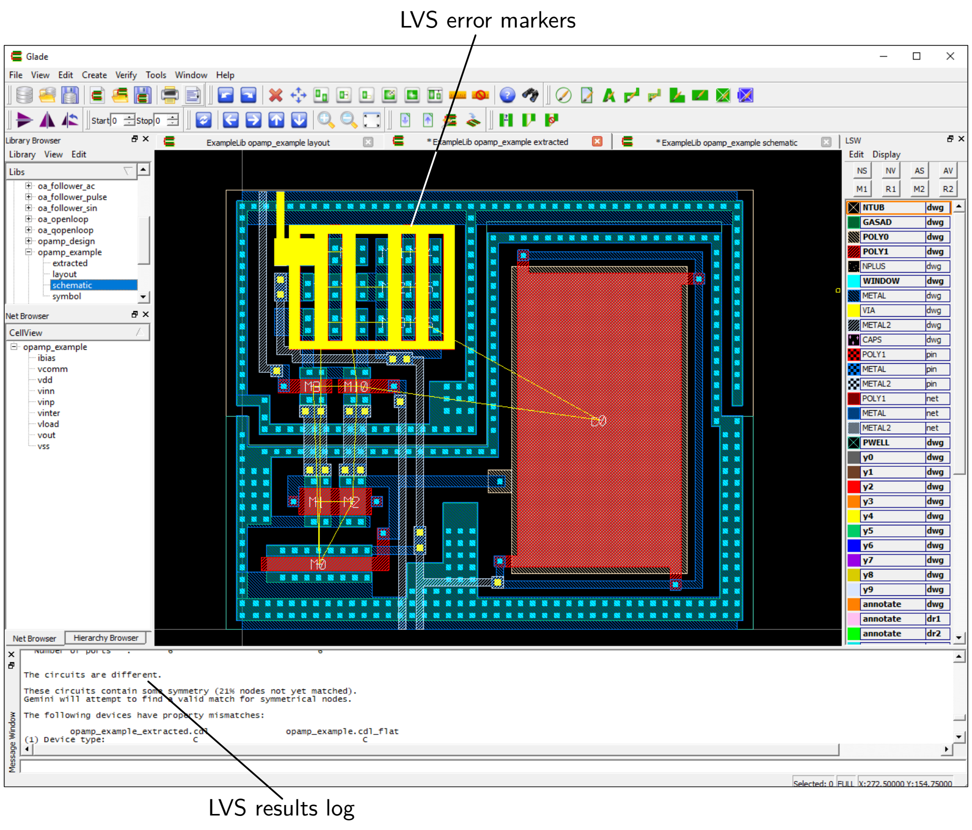
damics-kit/glade/opamp_example_extracted.cdl

```
******************************************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_example
* View Name:    extracted
******************************************************************************

.SUBCKT opamp_example
*.PININFO vss:B vinp:B vcomm:B vdd:B vout:B ibias:B vinn:B vinter:B vload:B

MM6 vdd ibias vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=3.325e-05 ...
MM9 vcomm ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=4.575e-05 ...
MM4 vdd ibias ibias vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=3.325e-05 ...
MM7 vdd ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=4.575e-05 ...
MM13 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=6.675e-05 ...
MM5 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=3.325e-05 ...
MM2 vinter vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05 $X=4.45 ...
MM15 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=7.925e-05 ...
MM14 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=7.925e-05 ...
MM8 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=4.575e-05 ...
MM3 vload vinn vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05 $X=2.65e-05 ...
Cc0 vinter vout w=6.42928e-05 l=0.000156207 $X=0.000122999 $Y=2.5749e-05
MM16 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=7.925e-05 ...
MM12 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=6.675e-05 ...
MM11 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05 $X=6.675e-05 ...
MM0 vout vinter vss vss cnm25modn w=4.8e-05 l=6e-06 as=2.64e-10 ps=0.000107 ad=2.64e-10 pd=0.000107 $X=1.125 ...
MM1 vload vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05 $X=2.85 ...
MM10 vinter vinp vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05 $X=4.65 ...
.ENDS
```

Before updating the schematic with the layout dummy devices as in Fig. 24(a), the LVS process clearly returns a negative match between the schematic and extracted circuits due to the absence of these elements in the former. Not only this result is reported in the Glade message window, but the physical location of the specific LVS errors is highlighted in the extracted view itself, as shown in Fig. 25.

Glade log file BEFORE schematic dummy correction

```
-----------------------------------------------------------------------------
        Netlist summary : opamp_example_extracted.cdl
-----------------------------------------------------------------------------
  Number of devices before reduction: 18
  Number of nets before reduction: 9

  Number of devices after reduction: 10
  Number of nets after reduction: 9

-----------------------------------------------------------------------------
        Netlist summary : opamp_example_lvs_err.sub
-----------------------------------------------------------------------------
  Number of devices before reduction: 9
  Number of nets before reduction: 9

  Number of devices after reduction: 9
  Number of nets after reduction: 9

The circuits are different.

The following netlist mismatches occurred:

-----------------------------------------------------------------------------
        Netlist errors : opamp_example_extracted.cdl
-----------------------------------------------------------------------------
2 NETS do not match:
NET "vdd" 11 connections
NET "ibias" 5 connections
  N: (inst MM11) [g] ibias :: [s,d,sub] vdd, vout, vdd
  N: (inst MM7) [g] ibias :: [s,d,sub] vdd, vdd, vdd
  N: (inst MM9) [g] ibias :: [s,d,sub] vcomm, vdd, vdd
  N: (inst MM4) [g] ibias :: [s,d,sub] vdd, ibias, vdd
  N: (inst MM4) [g] ibias :: [s,d,sub] vdd, ibias, vdd
2 DEVICES could not be matched, possibly because of other unmatched devices:
DEVICE N: (inst MM4) [g] ibias :: [s,d,sub] vdd, ibias, vdd
DEVICE N: (inst MM7) [g] ibias :: [s,d,sub] vdd, vdd, vdd

-----------------------------------------------------------------------------
        Netlist errors : opamp_example_lvs_err.sub
-----------------------------------------------------------------------------
2 NETS do not match:
NET "vdd" 8 connections
  N: (inst MI8) [g] ibias :: [s,d,sub] vdd, ibias, vdd
  N: (inst MI5) [g] ibias :: [s,d,sub] vdd, vout, vdd
  N: (inst MI7) [g] ibias :: [s,d,sub] vdd, vcom, vdd
  N: (inst MI2) [g] vinp :: [s,d,sub] vcom, vinter, vdd
  N: (inst MI1) [g] vinn :: [s,d,sub] vcom, vload, vdd
  N: (inst MI7) [g] ibias :: [s,d,sub] vdd, vcom, vdd
  N: (inst MI5) [g] ibias :: [s,d,sub] vdd, vout, vdd
  N: (inst MI8) [g] ibias :: [s,d,sub] vdd, ibias, vdd
NET "ibias" 4 connections
  N: (inst MI8) [g] ibias :: [s,d,sub] vdd, ibias, vdd
  N: (inst MI5) [g] ibias :: [s,d,sub] vdd, vout, vdd
  N: (inst MI7) [g] ibias :: [s,d,sub] vdd, vcom, vdd
  N: (inst MI8) [g] ibias :: [s,d,sub] vdd, ibias, vdd

1 DEVICES do not match:
DEVICE N: (inst MI8) [g] ibias :: [s,d,sub] vdd, ibias, vdd
```



```
  Number of ports   :       6                          6

The circuits are different.

These circuits contain some symmetry (21% nodes not yet matched).
Gemini will attempt to find a valid match for symmetrical nodes.

The following devices have property mismatches:

         opamp_example_extracted.cdl              opamp_example.cdl_flat
(1) Device type:                 C                                C
```

Figure 25 | Glade LVS results for cell view `ExampleLib`→`opamp_example`→`extracted` before updating the schematic view with layout dummy devices.

On the contrary, if the same LVS verification process is repeated **after** introducing the equivalent layout dummy devices into the schematic view of Fig. 24(b), then Gemini returns a positive matching:

Glade log file AFTER schematic dummy correction

```
------------------------------------------------------------------------------
        Netlist summary before reduction : opamp_example_extracted.cdl
------------------------------------------------------------------------------
  Number of devices :        18
  Number of nets    :        9
  Number of ports   :        6
------------------------------------------------------------------------------
        Netlist summary before reduction : opamp_example.cdl_flat
------------------------------------------------------------------------------
  Number of devices :        10
  Number of nets    :        9
  Number of ports   :        6
------------------------------------------------------------------------------
        Netlist summary after reduction :
------------------------------------------------------------------------------
                opamp_example_extracted.cdl     opamp_example.cdl_flat
  Number of devices :        10                   10
  Number of nets    :        9                    9
  Number of ports   :        6                    6

The following devices have property mismatches:
         opamp_example_extracted.cdl               opamp_example.cdl_flat
(1) Device type:                    C                                C
    Inst name   :                 Cc0                              CI9
    Model       :                   C                                C
    Terminals   :              vinter                             vout
                                 vout                           vinter
    Value (farad):                  0                                0
    W/L (um)    :             64.293/156.207                   100.000/100.000

1 device property error.
12 (63%) matches were found by local matching.
All nodes were matched in 3 passes.

The netlists match.
```

Two comments arise from this last example. First, Gemini effectively applies both collapsing and permutation rules like the ones illustrated in Fig. 22. Collapsing can be easily noticed by comparing the number of MOS devices between schematic and extracted netlists before reduction. Permutation allows for example to match these circuits even with the terminals of the compensation capacitor flipped between `vinter` and `vload` nets, as highlighted in each netlist.

Second, although netslits may match topologically, Gemini still performs a comparative audit of device properties according to the 10-% tolerance specified in Fig. 23. In this case, LVS output reports a mismatch in the compensation capacitor size between schematic (100 μm $\times$ 100 μm) and layout ($\simeq$ 64 μm $\times$ 156 μm), which have been also highlighted in both netlists.

Q7. Execute the above **LVS verification** to your optimized OpAmp layout, review any error from Gemini output and **correct the layout** accordingly.

4.8 2D and 3D Parasitics Extraction

After validating the correct matching between the optimized schematic and the full-custom layout topologies, the next physical verification step from Fig. 1 consists on the parasitics extraction, which is required for the final electrical re-simulation of the circuit. In general, CMOS planar technologies introduce RLC parasitic elements in the interconnectivity between devices following Fig. 26.

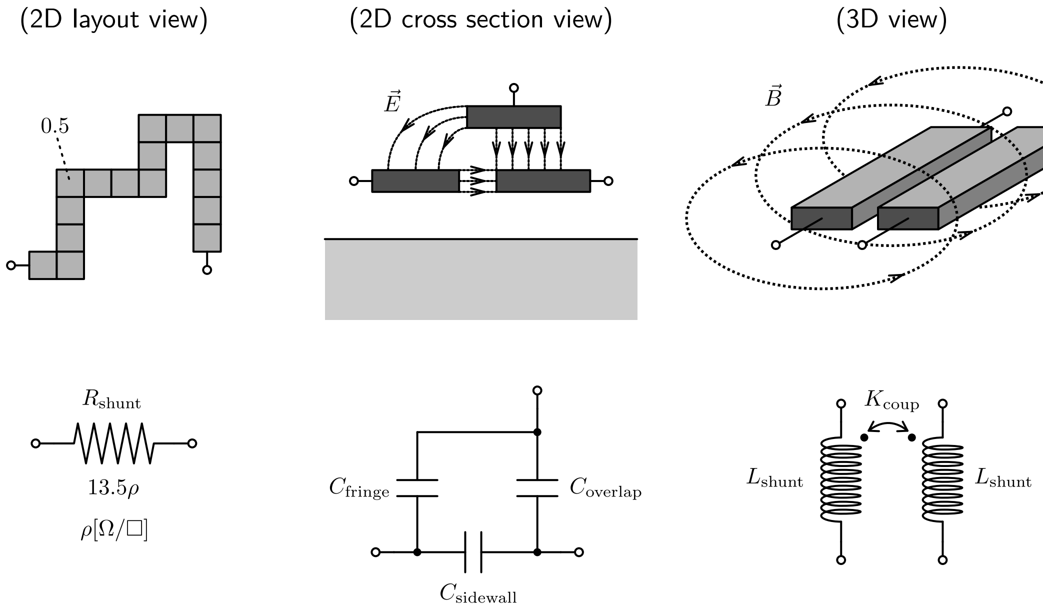


Figure 26 | Typical RLC circuit interconnectivity parasitics present in CMOS planar technologies.

For simplification purposes, only capacitive parasitics will be considered here. For this purpose, the simple parallel-plate capacitor model can be taken:

$$C_{\text{par}} = A\frac{\epsilon_{\text{o}}\epsilon_{\text{ox}}}{t_{\text{ox}}} \tag{3}$$

where A stands for the plate area, t_{ox} and ϵ_{ox} are the insulator thickness and its relative permittivity (~3.9 for SiO_2), and ϵ_o is the well-known permittivity of free space (i.e. 8.8542×10^{-12} F/m). Even with this very simplistic capacitive model, it is clear that some technology information is needed regarding the spacing (t_{ox}) between conductors. While horizontal spacing must be directly extracted from the specific layout pattern, the PDK includes also numerical data about the fixed vertical spacing between routing layers. In the particular case of CNM25, and under the assumption of ideal CMOS process planarization and thin metal layers, insulator thickness values are depicted in Fig. 27. Basically, the field oxide is 1060nm thick, while the inter-metal oxide insulator height is around 1300nm.

In practice, the extraction of parasitic elements is probably one of the most EDA time consuming steps of the whole physical verification part of Fig. 1, even considering only an small layout block like your OpAmp. The complexity of this process can be clearly understood with Fig. 28, where the 3D exploded view of the OpAmp layout example of Fig. 17 is rendered. This section will evaluate both 2D and 3D extraction techniques for estimating the capacitive parasitics of your CMOS circuit.

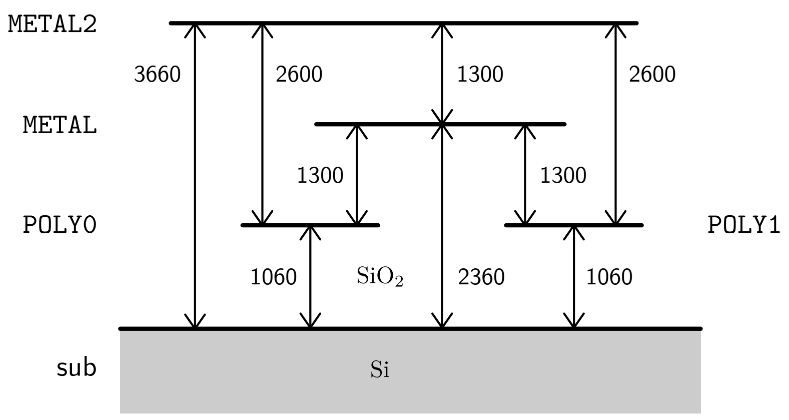


Figure 27 Simplified CNM25 cross section used for the interconnectivity parasitic capacitance model. Units in nm. Not to scale.

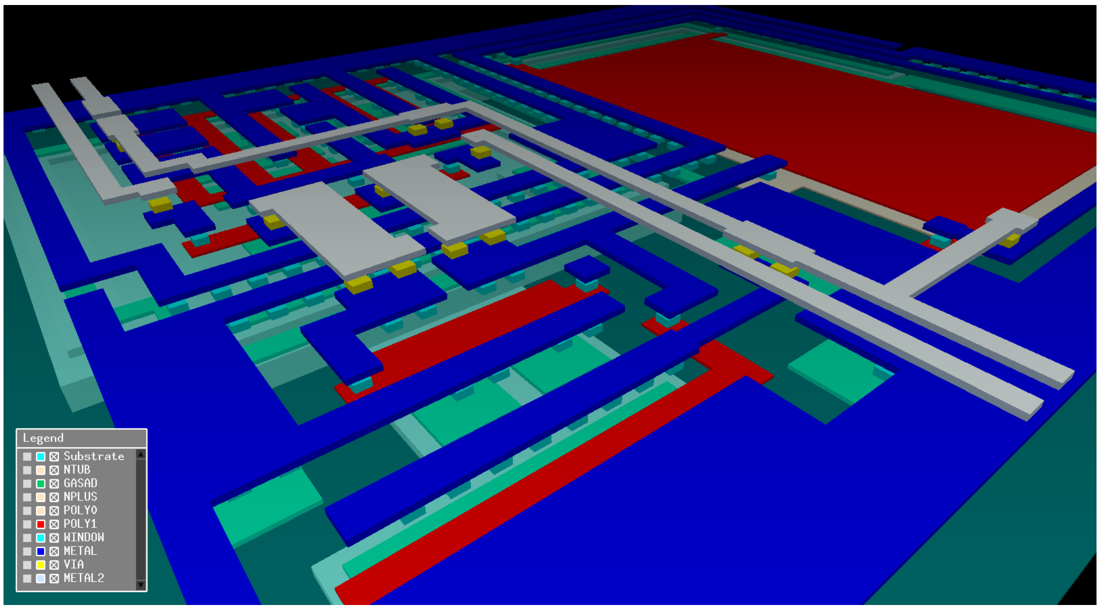
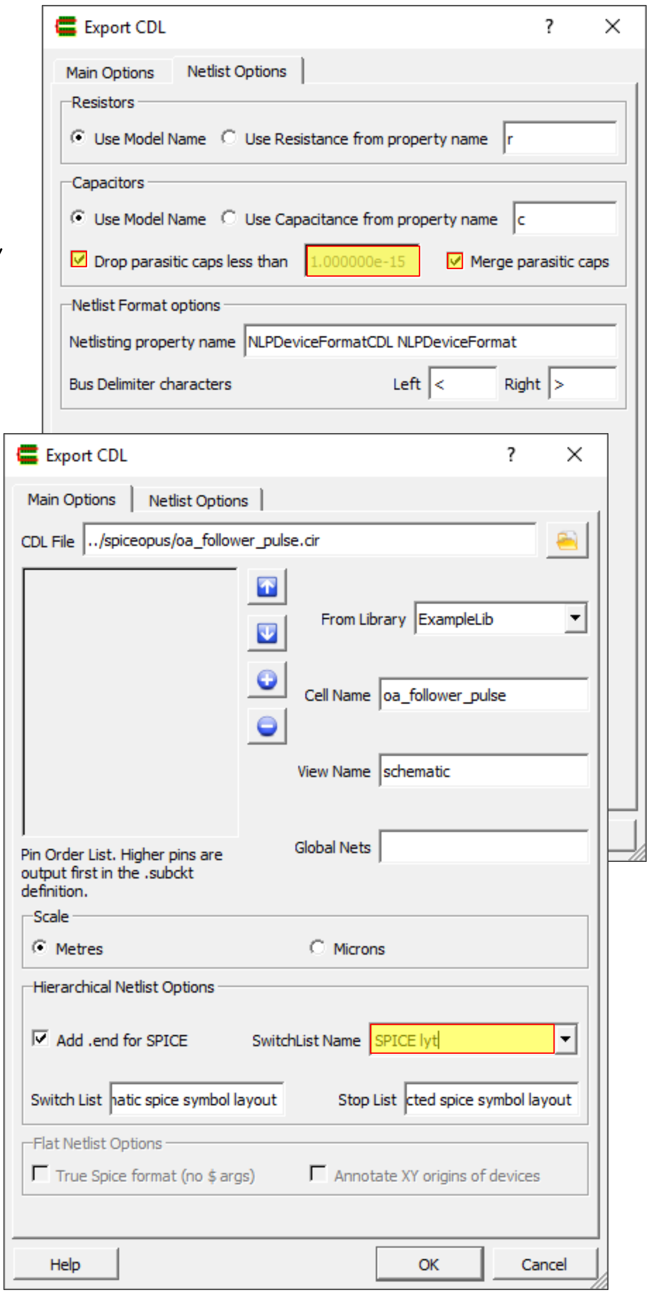


Figure 28 3D exploded view of `ExampleLib`→`opamp_example`→`layout` based on the simplified CNM25 cross section model of Fig. 27. Rendered by the EDA tool GDS3D from the University of Twente. More information about GDS3D can be found at `http://sourceforge.net/projects/gds3d`.

The first approach relies on 2D analysis, much like the method used in Section 4.6 for the extraction of native CNM25 devices (i.e. `cnm25modn`, `cnm25modp` and `cnm25cpoly`). For this reason, the PDK includes the Python script `damics-kit/glade/verification/cnm25xtr_par2d.py`, which is very similar to the regular extraction code `damics-kit/glade/verification/cnm25xtr_lvs.py` but adding here the specific functions for the computation of overlapping capacitance. In fact, Glade can also extract the perimeter of these overlapping regions in order to estimate fringing capacitance effects.

The sequential procedure to generate 2D parasitics extraction netlists is as follows:

1. Open your OpAmp layout `ExampleLib`→`opamp_design`→`layout`.
2. Execute Verify→Extract→Run (shift+y).
3. Select the extraction script `damics-kit/glade/verification/cnm25xtr_par2d.py` and launch the extraction process.
4. Regenerate the OpAmp test-bench netlists (`*.cir`) of Fig. 6 but using the following CDL export configuration:
   (a) Select *Use Model Name* option for passive devices.
   (b) Set the parasitic capacitance threshold to 1 fF.
   (c) Choose to *Merge parasitic caps*.
   (d) Select *SPICE lyt* switch-list name.



Export CDL — Netlist Options

Resistors: [✓ Use Model Name] Use Resistance from property name: r

Capacitors: [✓ Use Model Name] Use Capacitance from property name: c

[✓ Drop parasitic caps less than] 1.000000e-15 [✓ Merge parasitic caps]

Netlist Format options: Netlisting property name: NLPDeviceFormatCDL NLPDeviceFormat

Bus Delimiter characters: Left < Right >

Export CDL — Main Options

CDL File: ../spiceopus/oa_follower_pulse.cir

From Library: ExampleLib

Cell Name: oa_follower_pulse

View Name: schematic

Global Nets:

Pin Order List. Higher pins are output first in the .subckt definition.

Scale: [✓ Metres] Microns

Hierarchical Netlist Options: [✓ Add .end for SPICE] SwitchList Name: SPICE lyt

Switch List: atic spice symbol layout Stop List: cted spice symbol layout

Flat Netlist Options: True Spice format (no $ args) Annotate XY origins of devices

Figure 29 | Glade CDL export options for extracted views with parasitics.

damics-kit/glade/verification/cnm25xtr_par2d.py

```
# CNM25 2M extraction deck
# with parasitic capacitances

# Initialise boolean package & Loading pcells
See damics-kit/glade/verification/cnm25xtr_lvs.py

# Parasitic capacitance density [F/um2]
e0 = 8.854187817e-12 # [F/m]
er = 3.9             # SiO2
c0 = e0 * er * 1e-6  # Normalized to 1um thickness
cpoly0sub     = c0 / 1.060
cpoly1sub     = c0 / 1.060
cmetal1poly1  = c0 / 1.300
cmetal1poly0  = c0 / 1.300
cmetal1diff   = c0 / 1.300
cmetal1sub    = c0 / 2.360
cmetal2metal1 = c0 / 1.300
cmetal2poly1  = c0 / 2.600
cmetal2poly0  = c0 / 2.600
cmetal2diff   = c0 / 3.660
cmetal2sub    = c0 / 3.660

# Get raw layers & Form derived layers & Extract pin and net names before geomConnect
# Form connectivity & Save interconnect & Extracting devices
See damics-kit/glade/verification/cnm25xtr_lvs.py

# Extracting parasitics
print "# Extract Poly0 parasitics"
extractParasitic2(pwell, polycap, cpoly0sub, 0)
extractParasitic2(nwell, polycap, cpoly0sub, 0)

print "# Extract Poly1 parasitics"
extractParasitic2(pwell, polygate, cpoly1sub, 0)
extractParasitic2(nwell, polygate, cpoly1sub, 0)

print "# Extract Metal1 parasitics"
extractParasitic2(polygate, metal1, cmetal1poly1, 0)
extractParasitic2(polycap, metal1, cmetal1poly0, 0)
extractParasitic3(pdiff, metal1, cmetal1diff, 0, [polygate, polycap])
extractParasitic3(ndiff, metal1, cmetal1diff, 0, [polygate, polycap])
extractParasitic3(pwell, metal1, cmetal1sub, 0, [polygate, polycap, pdiff, ndiff])
extractParasitic3(nwell, metal1, cmetal1sub, 0, [polygate, polycap, pdiff, ndiff])

print "# Extract Metal2 parasitics"
extractParasitic2(metal1, metal2, cmetal2metal1, 0)
extractParasitic3(polygate, metal2, cmetal2poly1, 0, [metal1])
extractParasitic3(polycap, metal2, cmetal2poly0, 0, [metal1, polygate])
extractParasitic3(pdiff, metal2, cmetal2diff, 0, [metal1, polygate, polycap])
extractParasitic3(ndiff, metal2, cmetal2diff, 0, [metal1, polygate, polycap])
extractParasitic3(pwell, metal2, cmetal2sub, 0, [metal1, polygate, polycap, pdiff, ndiff])
extractParasitic3(nwell, metal2, cmetal2sub, 0, [metal1, polygate, polycap, pdiff, ndiff])

print "# Ending circuit extraction"
geomEnd()

# Opening extracted view
ui.openCellView(libxtrpar.libName(), cv.cellName(), "extracted")

```

damics-kit/glade/verification/cnm25xtr_par2d.py

For instance, the 2D parasitics for the OpAmp example of Fig. 17 are listed as follows:

```
********************************************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_example
* View Name:    extracted
********************************************************************************

.SUBCKT opamp_example vinn vinp vout vdd vss ibias
*.PININFO vss:B vinp:B vdd:B vinn:B ibias:B vout:B

MM2 vinter vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM0 vout vinter vss vss cnm25modn w=4.8e-05 l=6e-06 as=2.64e-10 ps=0.000107 ad=2.64e-10 pd=0.000107
MM14 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM12 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM4 vdd ibias ibias vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM6 vdd ibias vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
Cc0 vinter vout cnm25cpoly w=6.42928e-05 l=0.000156207
MM11 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM16 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM9 vcomm ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM15 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM5 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM3 vload vinn vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM8 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM7 vdd ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM13 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM1 vload vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM10 vinter vinp vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
CP0 vinter vdd  C=1.39453e-15
CP1 vinn vdd  C=6.18331e-15
CP2 vload vdd  C=1.39453e-15
CP3 vinter vss  C=3.8582e-13
CP5 vout ibias  C=3.33692e-15
CP7 ibias vdd  C=7.53437e-14
CP8 vinp vss  C=1.85938e-15
CP9 vinp vdd  C=5.88887e-15
CP11 vcomm ibias  C=2.72266e-15
CP12 vload vss  C=1.59238e-14
CP13 vdd ibias  C=3.05469e-15
CP14 vout vcomm  C=2.0918e-15
CP16 vout vss  C=3.37819e-13
.ENDS
```

Depending on the particular IC application, like radio frequency (RF) communications, more accurate parasitics estimations may be required. For such cases, Glade integrates FastCap[2] [8], the classic 3D finite-element tool capable of extracting self and mutual capacitances between ideal conductors of arbitrary shapes, orientations and sizes. The CNM25 PDK already incorporates the geometrical cross-section information in the technological file to exploit this fast but accurate extraction tool. Indeed, the procedure to perform the 3D capacitance extraction and generate the corresponding netlist is exactly the same as for the 2D case of page 53 but selecting the script file `damics-kit/glade/verification/cnm25xtr_par3d.py` instead. In this sense, Fig. 30 presents its usage for the same OpAmp layout example.

[2]More information can be found at `http://www.rle.mit.edu/cpg/research_codes.htm`.

It is important to note that this PDK is configured to annotate all the FastCap coupling contributions to bulk and to infinite boundaries into a predefined node named vss. This preset and the rest of FastCap configuration parameters highlighted in green below can be easily changed by listing them as switch variables in the extraction dialogue Verify→Extract→Run of Fig. 21.

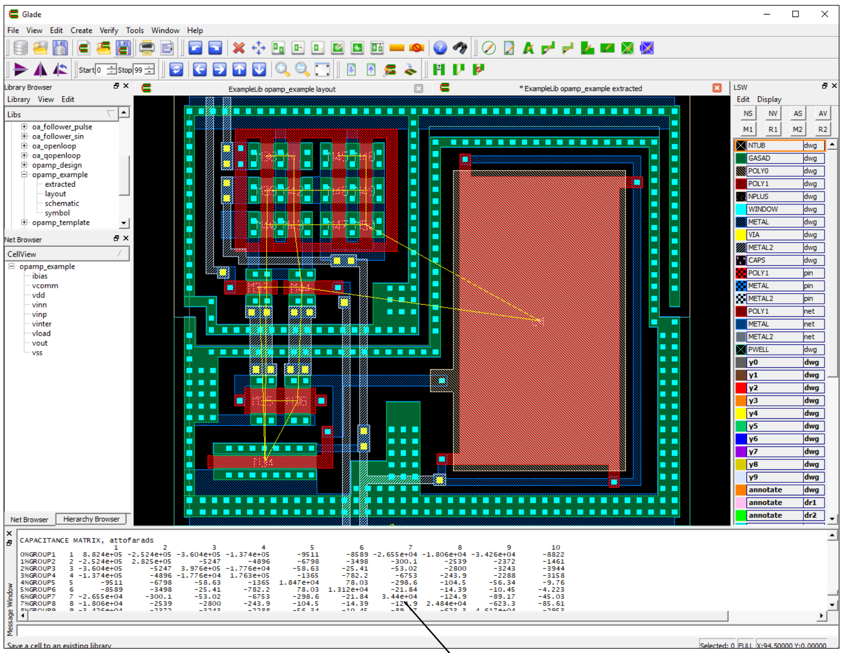
damics-kit/glade/verification/cnm25xtr_par3d.py

```
# CNM25 2M extraction deck
# with 3D parasitic capacitances

# Initialise boolean package & Loading pcells
# Get raw layers & Form derived layers
# Extract pin and net names before geomConnect
# Form connectivity & Save interconnect & Extracting devices

See damics-kit/glade/verification/cnm25xtr_lvs.py

# Extracting devices

if geomNumShapes(ngate) > 0 :
        print "# Extract NMOS transistors"
        extractMOS("cnm25modn", ngate, polygate, ndiff, pwell)

if geomNumShapes(pgate) > 0 :
        print "# Extract PMOS transistors"
        extractMOS("cnm25modp", pgate, polygate, pdiff, nwell)

if geomNumShapes(cpoly) > 0 :
        print "# Extract PiP capacitors"
        extractDevice("cnm25cpoly", cpoly, [[polygate, "T"], [polycap, "B"]])

# Extracting 3D parasitics with Fastcap

print "# Extract 3D cap parasitics using switch values:"

if 'bulk_name' not in globals() :
        bulk_name = "vss"
print "  bulk_name = ", bulk_name

if 'ref_name' not in globals() :
        ref_name = "vss"
print "  ref_name = ", ref_name

if 'fastcap_tol' not in globals() :
        fastcap_tol = 0.01
print "  fastcap_tol = ", fastcap_tol

if 'fastcap_order' not in globals() :
        fastcap_order = 3
print "  fastcap_order = ", fastcap_order

extractParasitic3D(bulk_name, ref_name, fastcap_tol, fastcap_order)

print "# Ending circuit extraction"
geomEnd()

# Opening extracted view
ui.openCellView(libxtrpar.libName(), cv.cellName(), "extracted")
```

damics-kit/glade/verification/cnm25xtr_par3d.py

For instance, the 3D parasitics for the OpAmp example of Fig. 17 are listed as follows:

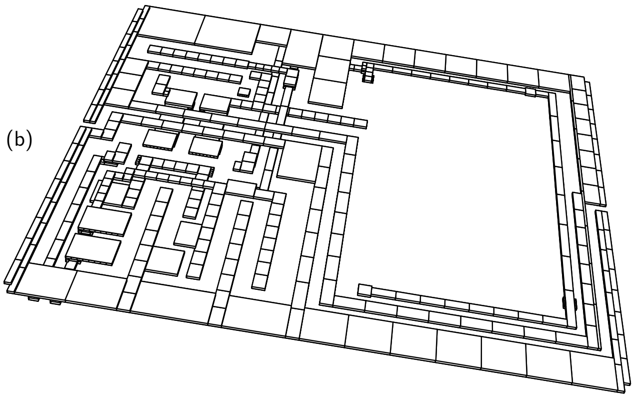
```
********************************************************************************
* Library Name: ExampleLib
* Cell Name:    opamp_example
* View Name:    extracted
********************************************************************************

.SUBCKT opamp_example vinn vinp vout vdd vss ibias
*.PININFO vss:B vinp:B vdd:B vinn:B ibias:B vout:B

MM2 vinter vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM0 vout vinter vss vss cnm25modn w=4.8e-05 l=6e-06 as=2.64e-10 ps=0.000107 ad=2.64e-10 pd=0.000107
MM14 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM12 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM4 vdd ibias ibias vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM6 vdd ibias vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
Cc0 vinter vout cnm25cpoly w=6.42928e-05 l=0.000156207
MM11 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM16 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM9 vcomm ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM15 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM5 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM3 vload vinn vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM8 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM7 vdd ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM13 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=7.8e-11 ps=3.7e-05 ad=6.6e-11 pd=3.5e-05
MM1 vload vload vss vss cnm25modn w=1.2e-05 l=1.2e-05 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
MM10 vinter vinp vcomm vdd cnm25modp w=1.2e-05 l=6e-06 as=6.6e-11 ps=3.5e-05 ad=6.6e-11 pd=3.5e-05
CP1 vinn vdd  C=5.378e-15
CP2 vinn vout  C=1.433e-15
CP4 vout vss  C=1.6182e-13
CP8 vss vdd  C=2.9485e-15
CP9 vss vss  C=4.10208e-13
CP10 vload vss  C=2.164e-14
CP11 vss vout  C=1.6651e-15
CP14 vinter vdd  C=2.313e-15
CP16 vinp vout  C=3.227e-15
CP18 vinp vdd  C=1.327e-15
CP27 vinter vout  C=2.122e-15
CP30 vinter vss  C=3.7989e-14
CP32 ibias vdd  C=2.687e-15
CP33 vdd vss  C=2.78947e-13
CP34 vinp vss  C=1.2754e-14
CP37 vout vdd  C=4.388e-15
CP38 vinn vss  C=9.51276e-15
CP39 vinp vinter  C=3.258e-15
CP40 vcomm vout  C=8.066e-15
CP41 vcomm vss  C=2.45168e-14
CP42 vload vdd  C=2.399e-15
CP44 ibias vss  C=1.0712e-14
.ENDS
```



```
CAPACITANCE MATRIX, attofarads
                    1          2          3          4          5          6          7          8          9         10
0%GROUP1   1  8.824e+05 -2.524e+05 -3.604e+05 -1.374e+05      -9511      -8589 -2.655e+04 -1.806e+04 -3.426e+04      -8822
1%GROUP2   2 -2.524e+05  2.825e+05      -5247      -4896      -6798      -3498     -300.1      -2539      -2372      -1461
2%GROUP3   3 -3.604e+05      -5247  3.976e+05 -1.776e+04     -58.63     -25.41     -53.02      -2800      -3243      -3944
3%GROUP4   4 -1.374e+05      -4896 -1.776e+04  1.763e+05      -1365     -782.2      -6753     -243.9      -2288      -3158
4%GROUP5   5      -9511      -6798     -58.63      -1365  1.847e+04      78.03     -298.6     -104.5     -56.34      -9.76
5%GROUP6   6      -8589      -3498     -25.41     -782.2      78.03  1.312e+04     -21.84     -14.39     -10.45     -4.223
6%GROUP7   7 -2.655e+04     -300.1     -53.02      -6753     -298.6     -21.84   3.44e+04     -124.9     -89.17     -45.03
7%GROUP8   8 -1.806e+04      -2539      -2800     -243.9     -104.5     -14.39     -124.9  2.484e+04     -623.3     -85.61
```

(a)

Report on 3D extraction results...



(b)

Figure 30 | Glade 3D extraction results for `ExampleLib`→`opamp_example`→`extracted` cell view (a) and FastCap equivalent finite-element mesh (b).

Q8. Execute the 2D and 3D parasitics extraction procedure to your optimized OpAmp layout following Fig. 29:

a. Which are the **top 3** parasitic caps of your layout?

b. What are the quantitative differences between **2D** and **3D** cap values?

c. Qualitatively speaking, what **impact** do you expect in OpAmp performance?

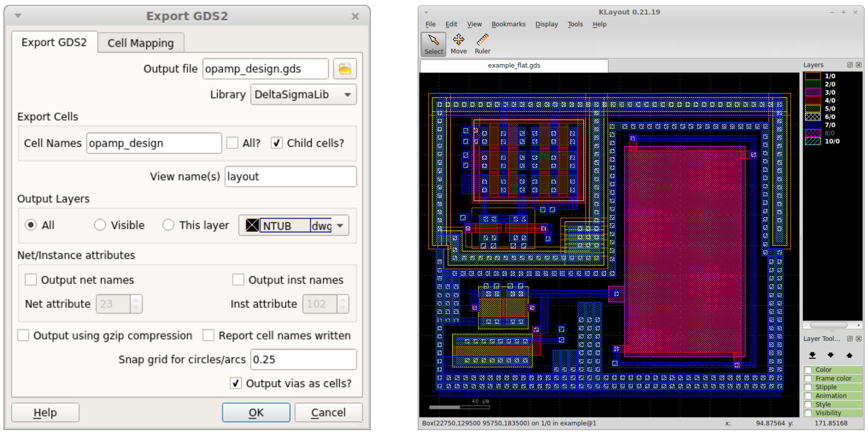
4.9 Post-Layout Simulation

According to Fig. 1, the last physical verification step of your full-custom layout consists on the electrical re-simulation of the circuit extracted in the previous section to evaluate the effects of parasitics. Thanks to the SpiceOpus netlist hierarchy of Fig. 8, this post-layout simulation only requires the substitution of the subcircuit netlist file before redoing the OpAmp datasheet.

Q9. Build a **summary datasheet** of your OpAmp following Table 5 with 3 performance columns: optimized schematic, extracted layout with 2D and 3D parasitics. Which OpAmp figures are the most affected by the layout parasitics? What are the main differences between 2D and 3D post-layout simulation results?

4.10 Tape-Out

Finally, your full-custom CMOS layout design is ready for fabrication at the IC foundry! The process of transferring the layout from the design house to the semiconductor manufacturer is called tape-out, from the time when the physical media support used for sending the EDA database was a magnetic tape itself. Although several database standards exist specifically for IC mask design transfer, the commonly accepted choice is the GDSII file format. Glade can generate this file format through File→Export→Export GDS2 and choosing the options of Fig. 31.



Export GDS2

Output file: opamp_design.gds
Library: DeltaSigmaLib
Export Cells
Cell Names: opamp_design [ ] All? [✓] Child cells?
View name(s): layout
Output Layers: [● All] ○ Visible ○ This layer NTUB dwg
Net/Instance attributes
[ ] Output net names [ ] Output inst names
Net attribute: 23 Inst attribute: 102
[ ] Output using gzip compression [ ] Report cell names written
Snap grid for circles/arcs: 0.25
[✓] Output vias as cells?

Figure 31 Glade GDSII export options and `ExampleLib→opamp_example→layout` GDSII file example viewed with KLayout. More information about KLayout can be found at `http://www.klayout.de`.

Q10. Export your OpAmp layout to `damics-kit/glade/opamp_design.gds`.

Glossary

BSIM3v3	Berkeley Short-channel IGFET Model version 3.3
CDL	circuit description language
CNM25	2.5μm 2-polySi 2-metal CMOS technology from IMB-CNM(CSIC)
CMOS	complementary metal-oxide-semiconductor
CMRR	common-mode rejection ratio
DRC	design rule checker
DRD	design rule driven
EDA	electronic design automation
ERC	electrical rule checker
GDSII	graphic database system II
MPP	MultiPartPath
MOS	metal-oxide-semiconductor
MOSFET	MOS field-effect transistor
NMOS	N-type MOS
PDK	physical design kit
PMOS	P-type MOS
IC	integrated circuit
I/O	input/output
LSW	layer selection window
LVS	layout versus schematic
OpAmp	operational amplifier
OS	operative system
PCell	parameterized cell
PDF	portable document format
PiP	polySi-insulator-polySi
PVT	process, supply voltage and temperature
RF	radio frequency
SPICE	Simulation Program with Integrated Circuit Emphasis

References

[1] T. Tuma and Á. Bűrmen. *Circuit Simulation with SPICE OPUS: Theory and Practice*. Modeling and Simulation in Science, Engineering and Technology. Birkhäuser Boston, 2009. ISBN 978-0-8176-4867.

[2] Yuhua Cheng, Mansun Chan, Kelvin Hui, Min chie Jeng, Zhihong Liu, Jianhui Huang, Kai Chen, James Chen, Robert Tu, Ping K. Ko, and Chenming Hu. BSIM3v3 Manual. Technical report, University of California, Berkeley, CA 94720, USA, 1996. `damics-kit/doc/bib/BSIM3v3_Manual.pdf`.

[3] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. Matching Properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1440, Oct 1989. `https://doi.org/10.1109/JSSC.1989.572629`.

[4] P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*. Oxford University Press, 2002. `http://www.aicdesign.org`.

[5] T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE3 Version 3f3 User's Manual*. University of California, Berkeley CA 94720, May 1993. `damics-kit/doc/bib/Spice3f3_Users_Manual.pdf`.

[6] A. Hastings. *The Art of Analog Layout*. Prentice Hall, 2005.

[7] C. Ebeling, N. McKenzie, and L. McMurchie. *The Gemini Users Guide*. University of Washington, Dec 1993. `damics-kit/doc/bib/The_Gemini_Users_Guide.pdf`.

[8] K. Nabors, S. Kim, J. White, and S. Senturia. *FastCap User's Guide*. Massachusetts Institute of Technology, Sep 1992. `damics-kit/doc/bib/FastCap_Users_Guide.pdf`.