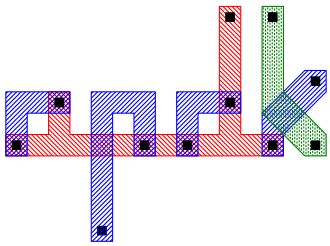


Academic Process Design Kit
CNM25 Edition

Francesc Serra Graells
Integrated Circuits and Systems (ICAS)
Institut de Microelectrònica de Barcelona, IMB-CNM(CSIC)
www.cnm.es/users/pserra/apdk
paco.serra@imb-cnm.csic.es

1 Introduction

1.1 Objectives

The aim of this academic process design kit (APDK) [1] is to introduce circuit designers to the top-down methodology of Fig. 1 for the design of mixed-signal full-custom integrated circuits (ICs) in complementary metal-oxide-semiconductor (CMOS) technologies.

For this purpose, freely available electronic design automation (EDA) tools are proposed for the schematic and the physical IC design stages together with all the technological information required for their use in a simple CMOS process case. Anyway, this APDK can be easily customized to extend its coverage to more complex CMOS technologies.

A detailed documentation scheme in portable document format (PDF) is distributed with this APDK, which includes not only this exhaustive document but also specific manuals for each particular EDA tool and language used along the design flow of Fig. 1.

☞ In order to gain hands-on experience in this APDK, a complete set of **exercises** with practical **questions** are developed step-by-step, including:

- The mixed-signal design at architectural level of a $\Delta\Sigma$ modulator ($\Delta\Sigma$M) for analog-to-digital data conversion.
- The functional specification of its basic building blocks.
- The automatic circuit optimization at transistor level of one of these blocks, an operational amplifier (OpAmp).
- The full-custom analog layout design of the optimized OpAmp circuit.
- The physical verification and post-layout simulation of the OpAmp layout.

☞ At the end of these lab exercises, designers should be able to go from the simulation of the IC architecture at functional level to the tape-out of the IC layout as a graphic database system II (GDSII) file for its CMOS integration at the semiconductor foundry.

Before starting with the use of this APDK, the installation instructions of next section must be followed!

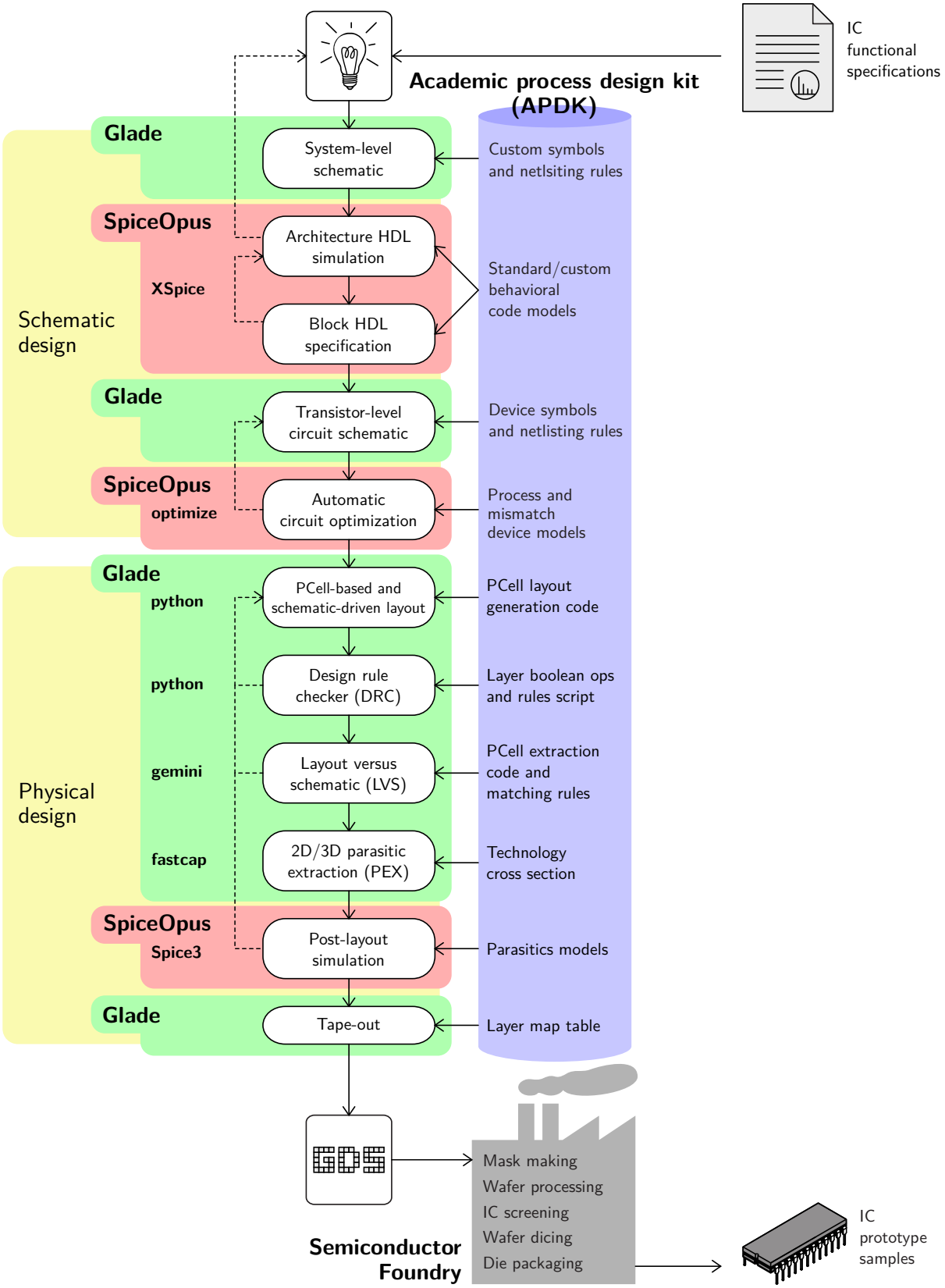


Figure 1 | Mixed-signal full-custom IC design methodology and proposed EDA tools.

1.2 Installation

The EDA tools of Fig. 1 are freely available for both MS Windows and Linux operative systems:

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 in [2] and at `www.peardrop.co.uk`.

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 [3] 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 [4] and at `fides.fe.uni-lj.si/spice`.

☞ Browse to `http://www.cnm.es/users/pserra/apdk`.

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

☞ Download and extract the APDK for your OS:

`apdk/doc`	Documentation
`/glade`	Schematic and layout libraries, technology files and verification rules
`/spiceopus`	Behavioral/electrical simulation models and test scripts

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

- `glade\glade.bat`:

```
set GLADE_HOME=path_to_glade
set PATH=%GLADE_HOME%;%PATH%
set PYTHONHOME=%GLADE_HOME%\Python38
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\cnm25lvs.py
set GLADE_FASTCAP_WORK_DIR=.
rem set GLADE_NO_CHECK_VERSION=1
rem set GLADE_NO_DELETE_TMPFILES=1
rem set GLADE_USE_OPENGL=NO
del .\glade*.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 .
            -o spiceopus.log spinit_local
```

☞ 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/cnm25lvs.py
export GLADE_FASTCAP_WORK_DIR=.
#export GLADE_NO_CHECK_VERSION=1
#export GLADE_NO_DELETE_TMPFILES=1
#export GLADE_USE_OPENGL=NO
export LC_NUMERIC=en_US.UTF-8
rm ./glade*.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 . -o spiceopus.log spinit_local &
```

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

1.3 APDK Files

The complete directory tree structure of this APDK edition is listed in Fig. 2. As it can be seen, each EDA tool has its own working directory, which includes a launching shell script plus all the configuration and data files required to work with the target CMOS technology.

Concerning formats, most of the files distributed with the APDK are OS independent. In fact, only the launching shell scripts, the compiled XSpice code models and the custom signal-processing tool for the computation of power spectral density (PSD) waveforms need to be ported to each OS.

Apart from the necessary data for the APDK itself, some files has been added for the solely purpose of developing the design exercises of Section 3. In particular, these items are the Glade example library (`ExampleLib`), the custom XSpice code models (`xtendedlib.cm`) and their schematic symbols (`XtendedLib`), as well as the collection of SPICE3 NUTMEG test scripts for the simulation of the $\Delta\Sigma$M (`test_dsm_*.sp3`) and OpAmp (`test_oa_*.sp3`) examples.

```
apdk
├── doc
│   ├── apdk_cnm25_v*.pdf      : This manual!
│   └── bib                    : Bibliography (Glade,Gemini,FastCap,SPICE3,BSIM3,XSpice manuals)
│
├── glade
│   ├── CNM25TechLib           : CNM25 technhology library
│   ├── SPICE3Lib              : Standard SPICE3 symbol library
│   ├── XSpiceLib              : Standard XSpice symbol library
│   ├── XtendedLib             : Custom XSpice symbol library
│   ├── ExampleLib             : Example library with lab exercises
│   ├── pcells
│   │   ├── cnm25cpoly_m.py    |
│   │   ├── cnm25cpoly.py      |
│   │   ├── cnm25modn_m.py     | CNM25 layout and extraction Python PCells
│   │   ├── cnm25modn.py       |
│   │   ├── cnm25modp_m.py     |
│   │   └── cnm25modp.py       |
│   ├── verification
│   │   ├── cnm25drc.py        |
│   │   ├── cnm25lvs.py        | CNM25 DRC, LVS and PEX Python scripts
│   │   ├── cnm25pex_2d.py     |
│   │   └── cnm25pex_3d.py     |
│   ├── cnm25.tch              : CNM25 technology file
│   ├── glade_init.py          : Glade init Python script
│   ├── glade_*.log            : Glade output log file
│   ├── gladerc.xml            : Glade configuration file
│   └── glade.bat(.sh)         : Glade launch shell script
│
└── spiceopus
    ├── xspice                 |
    │   ├── analog             |
    │   ├── digital            | XSpice code model source examples
    │   ├── xtendedlib         |
    │   ├── xtradev            |
    │   └── xtraevt            |
    ├── cnm25mod.lib           : CNM25 process and matching device model library
    ├── xtendedlib.cm          : Custom XSpice behavioral code model library
    ├── spiceopus.bat(.sh)     : SpiceOpus launch shell script
    ├── spiceopus.log          : SpiceOpus output log file
    ├── spinit_local           : SpiceOpus init script
    ├── test_dsm_*.sp3         : Delta-Sigma modulator test scripts for lab exercises
    ├── test_oa_*.sp3          : OpAmp test scripts for lab exercises
    └── tran2psd(.exe)         : Custom tool to compute power spectral densities (PSDs)
```

Figure 2 | APDK directory tree (in blue) and file contents (OS dependent in red). The scripts to launch Glade and SpiceOpus EDA tools are highlighted in yellow.

2 CMOS Technology

The current APDK edition targets 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 metal-oxide-semiconductor (MOS) field-effect transistor (MOSFET) and the polySi-insulator-polySi (PiP) capacitor, as shown in Fig. 3. The corresponding physical layers for the full-custom layout design are listed in Table 1.

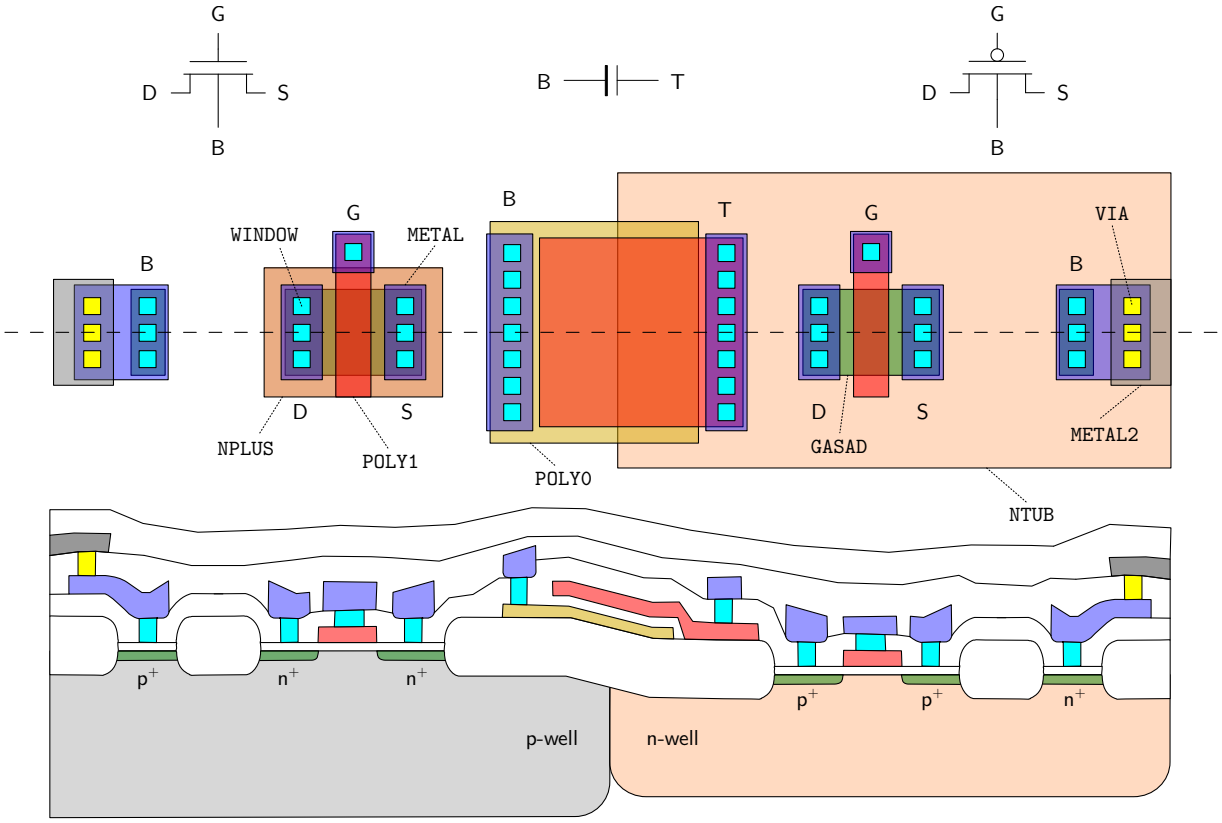


Figure 3 | CNM25 devices (top) and process cross section (bottom). Not to scale.

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 an PiP capacitors
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.

2.1 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

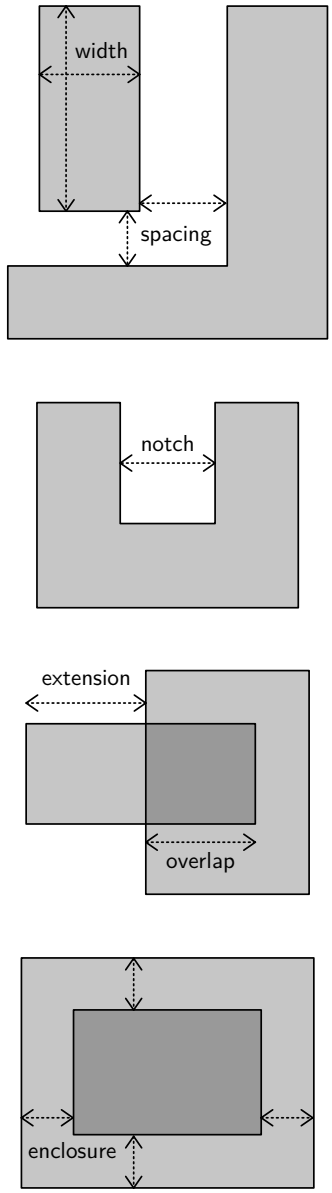


Table 2 | CNM25 design rules (left) and geometrical definitions (right).

This APDK comes with tools for assisting the designer during the edition of the full-custom layout, as explained in Section 3.6. However, due to number of rules present in Table 2 and their layer interdependency, the specific physical verification step of Section 3.7 is also included in the methodology of Fig. 1.

2.2 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 APDK includes `apdk/spiceopus/cnm25mod.lib`, the full set of model parameters for the CNM25 devices of Fig. 3 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 [5]. In general, two types of parameters are given by the foundry to cover its technology deviations:

Process parameters refer to global deviations at wafer level with correlation distances much larger than device dimensions (e.g. drift in gate oxide thickness). 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 with correlation distances comparable to device dimensions (e.g. dopant fluctuations). 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 [6]:

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

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.

apdk/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
```

apdk/spiceopus/cnm25mod.lib

3 Usage by Exercise

In order to explain the capabilities of this APDK, the practical design case of Fig. 4 will be followed. This case study consists on an oversampling $\Delta\Sigma$M analog-to-digital converter (ADC), where $V_{\rm sens}$ stands for the analog differential voltage input to be sensed and `wsens` is the equivalent digital word output. The general scheme usually includes an input amplitude limiter, anti-aliasing filter and digital decimator to finally obtain the digital resolution at the Nyquist sampling rate.

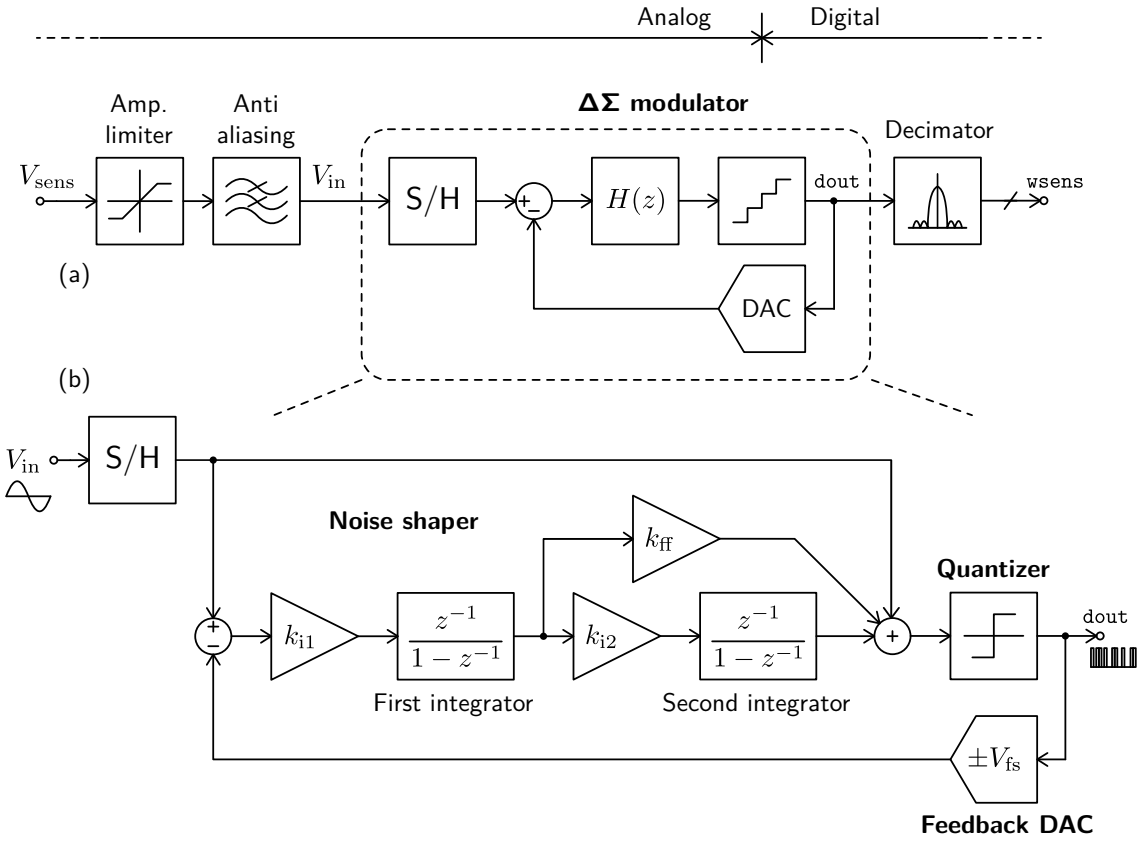


Figure 4 | General scheme of a $\Delta\Sigma$ oversampling ADC (a) and differential voltage signal model of your $\Delta\Sigma$M architecture (b).

The present lab exercises are focused only on the $\Delta\Sigma$M stage and its circuit implementation using switched-capacitor (SC) design techniques. In particular, the architecture chosen for your $\Delta\Sigma$M is the single-loop discrete-time topology of Fig. 4(b), where $V_{\rm in}$ and `dout` are the analog differential voltage input signal and the $\Delta\Sigma$ modulated output bit stream, respectively, while $V_{\rm fs}$ stands for the differential input full scale. A second-order single-bit solution is selected here in order to avoid both, the stability problems of noise shapers with higher order transfer functions $H(z)$, as well as the typical distortion issues due to mismatching of the feedback digital-to-analog converter (DAC) in multi-bit architectures. As shown in the equivalent differential voltage signal model of Fig. 4(b), this noise shaper topology [7] introduces nested feedforward loops to optimize signal full scale and distortion at the cost of adding an extra summer at the input of the quantizer.

The overall design specifications for your $\Delta\Sigma$M are listed in Table 3 in terms of differential input signal full scale (FS) and bandwidth (BW), and output digital dynamic range (DR).

Parameter	Name	Value	Units
Differential input full-scale	V_{fs}	2	$\mathrm{V}_{\mathrm{peak}}$
Input bandwidth at -3 dB	BW	8	kHz
Output dynamic range	DR	14	bit

Table 3 | General design specifications for your $\Delta\Sigma$M.

Based on the above specifications, some preliminary design decisions can be already taken at this level:

Sampling frequency: One of the most important design parameters in oversampled ADCs is the sampling frequency itself, usually referred as oversampling ratio (OSR) respect to the Nyquist rate. For an ideal B-bit N-order single-loop $\Delta\Sigma$M without any feedforward loop and exhibiting unitary gain coefficients (i.e. $k_{\mathrm{i1}} \equiv k_{\mathrm{i2}} \doteq 1$), the theoretical DR considering quantization noise only [8] is found to be:

$$\begin{aligned} \mathrm{DR} &= \tfrac{3\pi}{2}\left(2^B-1\right)^2(2N+1)\left(\tfrac{\mathrm{OSR}}{\pi}\right)^{2N+1} \\ \mathrm{DR[dB]} &= 6.7+20\log\left(2^B-1\right)+10\log(2N+1)+20\,(N+0.5)\log\tfrac{\mathrm{OSR}}{\pi} \end{aligned} \tag{2}$$

Hence, the required OSR can be derived from the resolution specification of Table 3, showing a DR performance improvement of (N+0.5)bit for each octave of OSR upscaling. However, the ideal DR values computed from (2) can not be achieved in practice due to noise and distortion contributions caused by circuit non-idealities, such as limited supply voltage, thermal and flicker device noise, circuit non-linearity, clock jitter and noise from external analog references. In consequence, OSR must be carefully chosen to ensure enough overhead in the ideal DR value.

Sampler capacitance: Since the CMOS implementation of your $\Delta\Sigma$M architecture of Fig. 4(b) will be based on SC circuits, a key design parameter is the capacitance value of the input sampler, which in turn will size the rest of capacitor elements through the corresponding coefficients k_{i1}, k_{i2} and k_{ff}. In this sense, the DR achievable by a differential input SC sampler of single-ended capacitance value C_{s} considering only the thermal noise contributions of the series switch can be expressed as:

$$\mathrm{DR[dB]} = 20\log\left(\frac{V_{\mathrm{fs}}}{v_{\mathrm{neq}}}\right)_{\mathrm{rms}} \qquad v_{\mathrm{neq}}^2 = \frac{2}{\mathrm{OSR}}\frac{kT}{C_{\mathrm{s}}} \tag{3}$$

where k and T are the well known Boltzmann constant and absolute temperature, respectively. In this case, DR increases 0.5 bit every time C_{s} is doubled. Again, it is advised to allocate some DR headroom for the rest of circuit non-idealities.

Q1. Calculate the following design parameters of your $\Delta\Sigma$M according to Table 3 for a dynamic range overhead of **+1 bit** (i.e. $\mathrm{DR} \equiv 15$ bit):

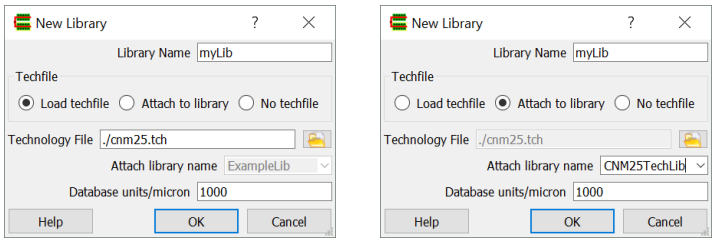
- **a.** **OSR** from (2) and rounded to the next higher power of 2.
- **b.** The minimum single-ended input **sampler capacitance** C_{s} from (3).

3.1 New Design Library

As mentioned in Section 1.2, Glade needs to be launched under a suitable configuration environment to work with CNM25. The same applies for any new design library to be created. Some technology information is stored in each library to assist with the schematic and layout edition in Glade, such as: layer table, layer connectivity, automatic layout structures (e.g. vias, multi-part paths), vertical cross section and layer display properties, among others.

In this sense, the APDK is already shipped with a Glade technology file called `apdk/glade/cnm25.tch`, which includes all the above required information for CNM25. This technology file has been compiled into the technology library `apdk/glade/CNM25TechLib`, so any new design library can be created by compiling either the original technological file or by attaching this technological library, as shown in Fig. 5.

> ☞ The lab exercises of this manual do not require to create any new Glade design library. All examples are collected into the `apdk/glade/ExampleLib` library.



(a) File → New Lib (b)

Figure 5 | Glade dialogues for creating a new design library by compiling the CNM25 technology file (a) or by attaching the technology library `CNM25TechLib` (b).

apdk/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 ;
//
// Manufacturing grid
//
MFGGRID 0.250 ;
//
// 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 0.0 ENDEXT 0.0 ;
MPP nguard LAYER GASAD drawing WIDTH 4.5 BEGEXT 0.0 ENDEXT 0.0 ;
MPP nguard LAYER NPLUS drawing WIDTH 9.5 BEGEXT 0.0 ENDEXT 0.0 ;
MPP nguard LAYER WINDOW drawing WIDTH 2.5 BEGEXT 0.0 ENDEXT 0.0 SPACE 3.0 LENGTH 2.5 ;
MPP nguard LAYER METAL drawing WIDTH 5.0 BEGEXT 0.0 ENDEXT 0.0 ;
MPP pguard LAYER GASAD drawing WIDTH 4.5 BEGEXT 0.0 ENDEXT 0.0 ;
MPP pguard LAYER WINDOW drawing WIDTH 2.5 BEGEXT 0.0 ENDEXT 0.0 SPACE 3.0 LENGTH 2.5 ;
MPP pguard LAYER METAL drawing WIDTH 0.0 BEGEXT 0.0 ENDEXT 0.0 ;
MPP p0m1 LAYER POLY0 drawing WIDTH 10.5 BEGEXT 0.0 ENDEXT 0.0 ;
MPP p0m1 LAYER WINDOW drawing WIDTH 2.5 BEGEXT 0.0 ENDEXT 0.0 SPACE 3.0 LENGTH 2.5 ;
MPP p0m1 LAYER METAL drawing WIDTH 5.0 BEGEXT 0.0 ENDEXT 0.0 ;
MPP p1m1 LAYER POLY1 drawing WIDTH 5.0 BEGEXT 0.0 ENDEXT 0.0 ;
MPP p1m1 LAYER WINDOW drawing WIDTH 2.5 BEGEXT 0.0 ENDEXT 0.0 SPACE 3.0 LENGTH 2.5 ;
MPP p1m1 LAYER METAL drawing WIDTH 5.0 BEGEXT 0.0 ENDEXT 0.0 ;
MPP m1m2 LAYER METAL drawing WIDTH 5.5 BEGEXT 0.0 ENDEXT 0.0 ;
MPP m1m2 LAYER VIA drawing WIDTH 3.0 BEGEXT 0.0 ENDEXT 0.0 SPACE 3.5 LENGTH 3.0 ;
MPP m1m2 LAYER METAL2 drawing WIDTH 5.5 BEGEXT 0.0 ENDEXT 0.0 ;
//
// 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...
```

apdk/glade/cnm25.tch

3.2 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 property annotation, just to mention a few. Fig. 6(a) depicts these capabilities with an schematic test bench for the $\Delta\Sigma$M architecture presented in Fig. 4(b).

Apart from the schematic edition, this APDK also takes advantage of Glade netlisting configurability. For this purpose, specific backend rules have been defined to generate circuit description language (CDL) netlists following Fig. 6(b), which are compatible with both SPICE and XSpice simulation.

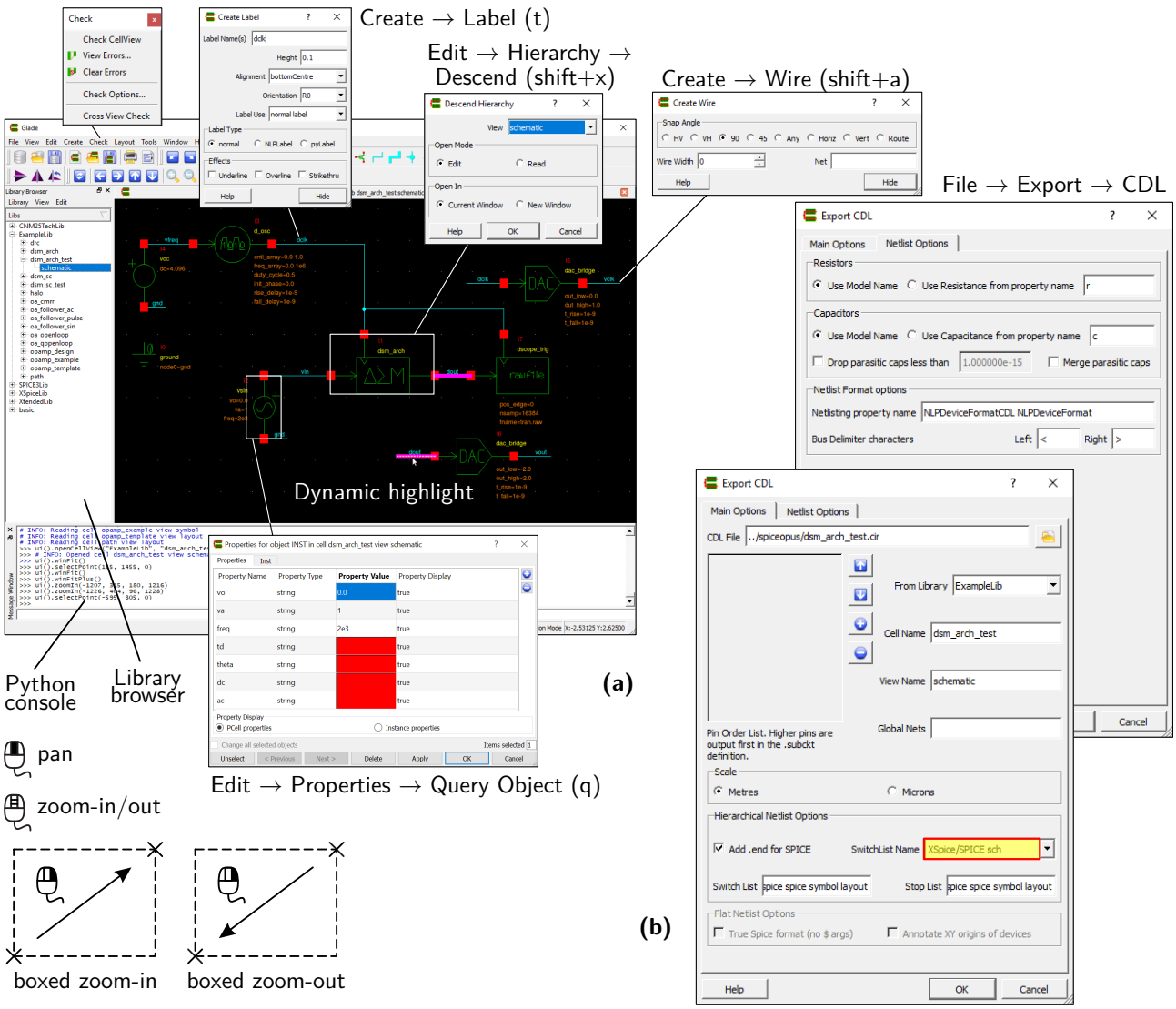
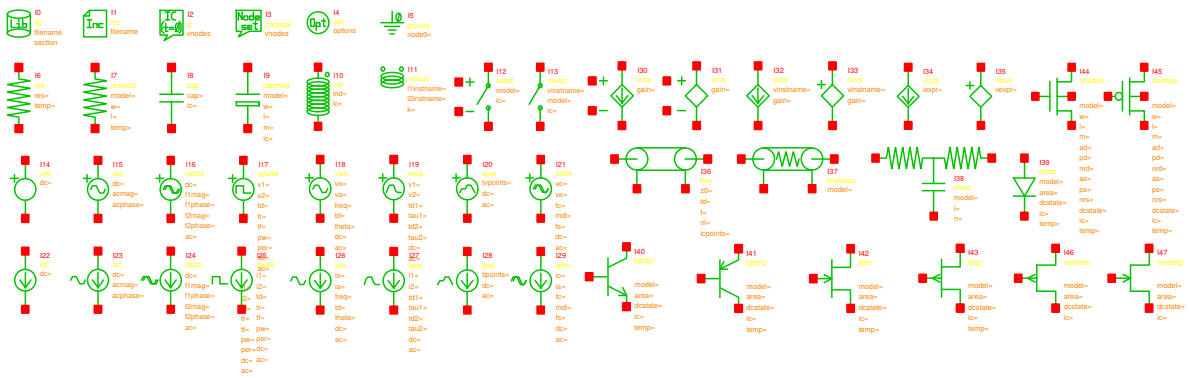


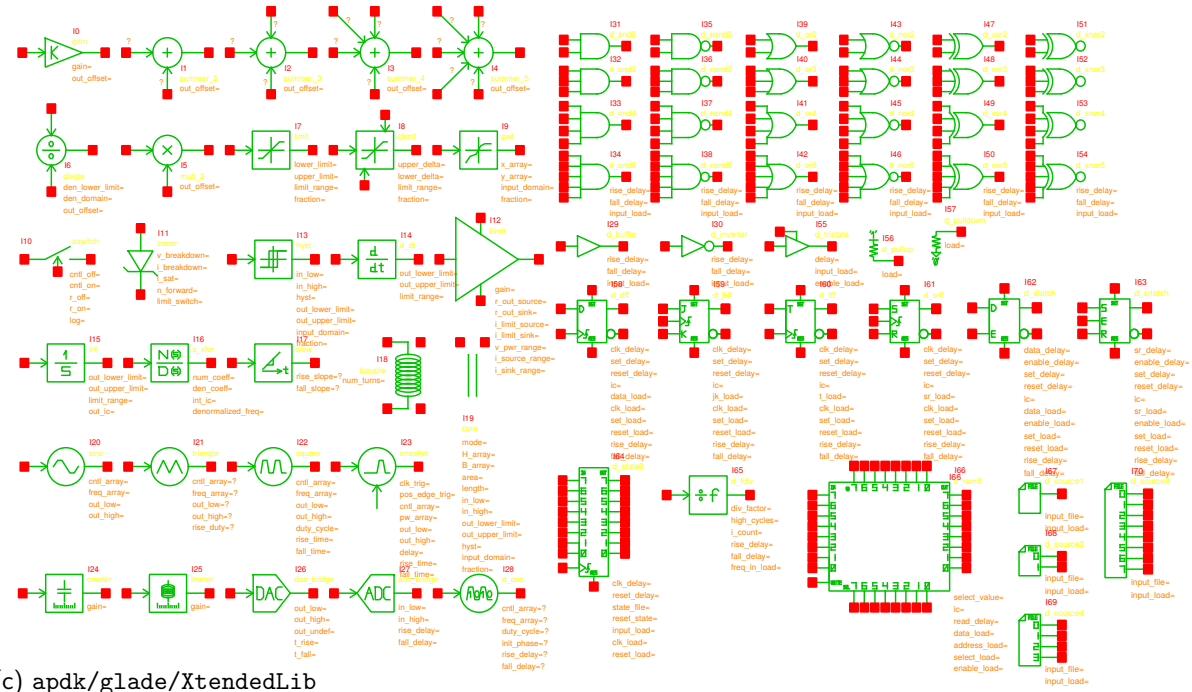
Figure 6 | Main window of the Glade schematic editor (a) and CDL export dialogue configuration (b) for XSpice/SPICE schematics.

In order to fully exploit the Glade schematic editor in conjunction with the SpiceOpus simulation capabilities explained in Sections 3.3 to 3.5, the APDK comes with the reference symbol libraries of Fig. 7(a,b,c) for their usage in general purpose schematics (e.g. test benches). Concerning the target CMOS technology, all circuit schematics to be integrated in CNM25 must be built from the primitive device symbols of Fig. 7(d).

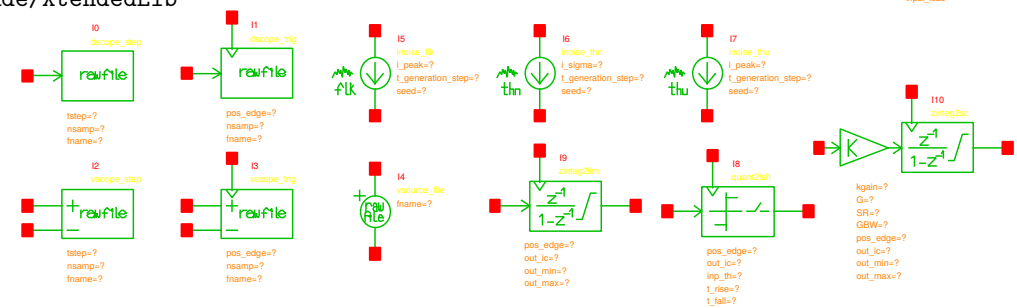
(a) apdk/glade/SPICE3Lib



(b) apdk/glade/XSpiceLib



(c) apdk/glade/XtendedLib



(d) apdk/glade/CNM25TechLib

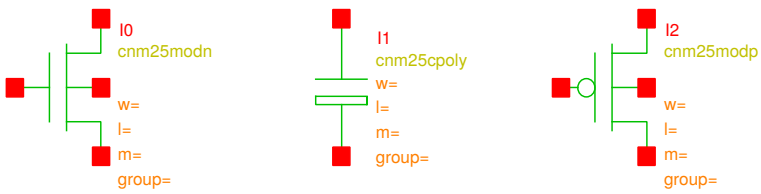


Figure 7 | The full set of standard SPICE3 (a) and XSpice (b), custom XSpice (c) and CNM25 (d) schematic symbols available from the APDK reference libraries.

- ☞ Launch `apdk/glade/glade.bat` (or `.sh`).
- ☞ With the library browser, open the $\Delta\Sigma$M architecture test bench `ExampleLib`→`dsm_arch_test`→`schematic`.
- ☞ Descend into the $\Delta\Sigma$M schematic and **edit** the parameters highlighted in Fig. 8:
  - Integrator and feedforward **coefficients** $k_{\mathrm{i1}} = 0.3$, $k_{\mathrm{i2}} = 0.7$ and $k_{\mathrm{ff}} = 2.0$.
  - Differential output **full-scale** range of the Z-domain integrators `out_min` $= -5.0$ and `out_max` $= 5.0$.
- ☞ **Check** for schematic connectivity errors with Check→Check Cellview.
- ☞ Ascend back to the test bench and **check** the top schematic.
- ☞ Using the CDL export dialogue configuration of Fig. 6(b), generate the equivalent XSpice **circuit netlist** `apdk/spiceopus/dsm_arch_test.cir`, which should be similar to the example shown below.

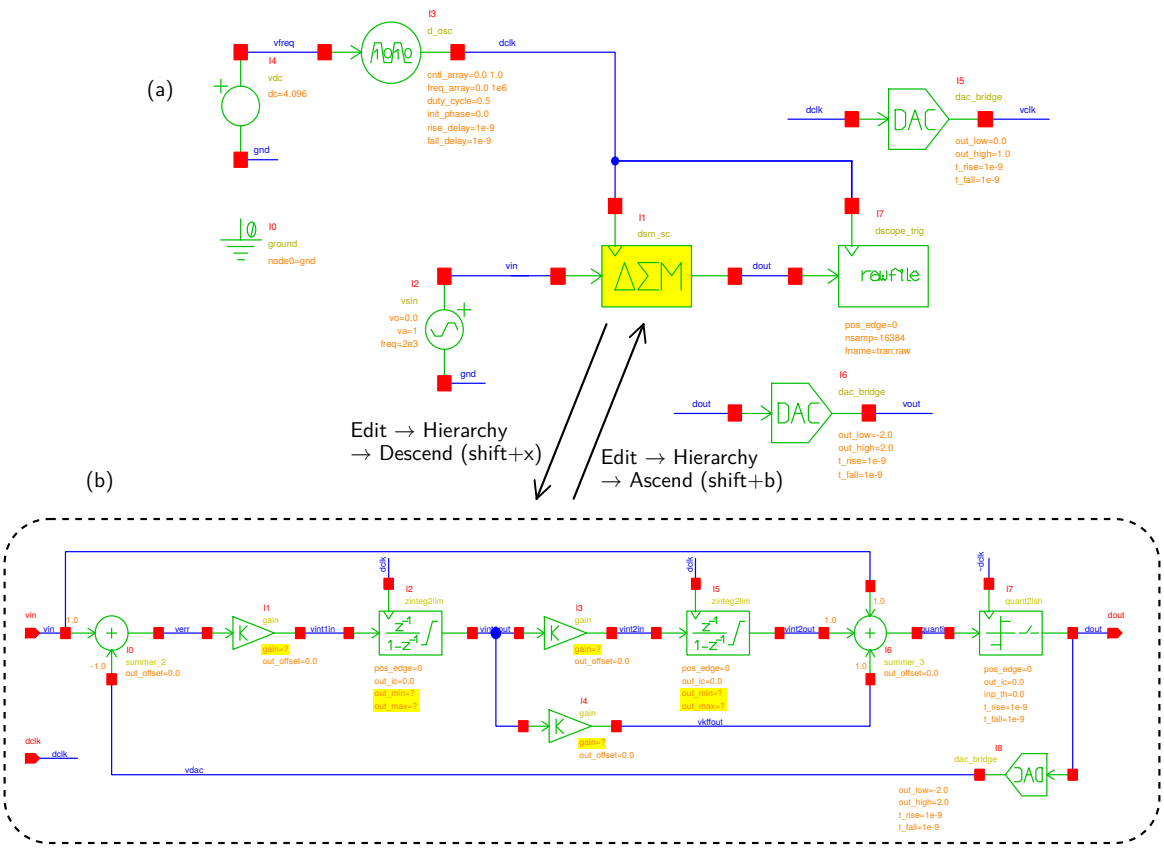


Figure 8 Test bench `ExampleLib`→`dsm_arch_test`→`schematic` (a) and subcircuit `dsm_arch`→`schematic` (b) for the differential-voltage $\Delta\Sigma$M architecture of Fig. 4(b). Block parameters to be set are highlighted in yellow.

dsm_arch_test.cir

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

*******************************************************************************
* Library Name: ExampleLib
* Cell Name: dsm_arch
* View Name: schematic
*******************************************************************************

.SUBCKT dsm_arch vin dclk dout

   aI7 %v(vquantin) %d(~dclk) %d(dout) mI7
      .model mI7 quant2lsh(pos_edge=0 out_ic=0.0 inp_th=0.0 t_rise=1.0e-9 t_fall=1.0e-9)
   aI3 %v(vint1out) %v(vint2in) mI3
      .model mI3 gain( gain=0.7  out_offset=0.0)
   aI1 %v(verr) %v(vint1in) mI1
      .model mI1 gain( gain=0.3  out_offset=0.0)
   aI4 %v(vint1out) %v(vkffout) mI4
      .model mI4 gain( gain=2.0  out_offset=0.0)
   aI2 %v(vint1in) %d(dclk) %v(vint1out) mI2
      .model mI2 zinteg2lim(pos_edge=0 out_ic=0.0  out_min=-5.0 out_max=5.0 )
   aI5 %v(vint2in) %d(dclk) %v(vint2out) mI5
      .model mI5 zinteg2lim(pos_edge=0 out_ic=0.0  out_min=-5.0 out_max=5.0 )
   aI8 %d(dout) %v(vdac) mI8
      .model mI8 dac_bridge(out_low=-2.0 out_high=2.0 t_rise=1e-9 t_fall=1e-9)
   aI0 [%v(vin) %v(vdac)] %v(verr) mI0
      .model mI0 summer(in_gain=[-1.0 1.0] out_offset=0.0)
   aI6 [%v(vin) %v(vint2out) %v(vkffout)] %v(vquantin) mI6
      .model mI6 summer(in_gain=[1.0 1.0 1.0] out_offset=0.0)

.ENDS

aI6 [%d(dout)] [%v(vout)] mI6
   .model mI6 dac_bridge(out_low=-2.0 out_high=2.0 t_rise=1e-9 t_fall=1e-9)

vI0 gnd 0 0

aI7 %d(dout) %d(dclk) mI7
   .model mI7 dscope_trig(pos_edge=0 nsamp=16384 fname="tran.raw")

vI4 vfreq gnd dc 4.096

xI1 vin dclk dout dsm_arch

aI5 [%d(dclk)] [%v(vclk)] mI5
   .model mI5 dac_bridge(out_low=0.0 out_high=1.0 t_rise=1e-9 t_fall=1e-9)

aI3 %v(vfreq) %d(dclk) mI3
   .model mI3 d_osc(cntl_array=[0.0 1.0] freq_array=[0.0 1e6] duty_cycle=0.5
                    init_phase=0.0 rise_delay=1e-9 fall_delay=1e-9)

vI2 vin gnd sin(0.0 1 2e3  )

.END
```

3.3 Architectural HDL Simulation

The use of a hardware description language (HDL) for the validation of mixed-mode integrated systems is highly recommended. Not only it simplifies the co-simulation of the analog and digital parts of integrated circuits, but it can also allow strong savings in terms of simulation time when replacing transistor-level blocks by their equivalent functional-level counterparts. This feature is specially noticeable in oversampled systems, like the $\Delta\Sigma$M case study. For this purpose, the EDA framework proposed in Fig. 1 takes benefit of XSpice mixed-signal code model (CM) concept, which introduces a new type of programmable SPICE elements easily identifiable in circuit netlists by their a-suffixed notation.

According to Fig. 9, SpiceOpus implements a dual engine including SPICE3 [9] for electrical circuit simulation and XSpice [10] for event-driven circuit parts. The latter is based on custom models written in a C-language, which can be added to the simulator by compiling them separately, without requiring any modification of the event-driven simulator engine. XSpice already comes with an extensive library of CMs covering common analog, digital and mixed-signal functions.

As an example, the custom CM library `apdk/spiceopus/xtendedlib.cm` has been developed to simulate the $\Delta\Sigma$M architecture of Fig. 8 at functional level. Furthermore, an external program called `tran2psd` is also distributed with the APDK due to the lack of accurate enough algorithms in NUTMEG for the computation of PSD figures from transient simulations. In this sense, `apdk/spiceopus/tran2psd` is a patched version of the PSD C-source code available from the PhysioToolkit library [11], which has been modified here to read and write SPICE raw data format.

Finally, SpiceOpus also introduces a new NUTMEG command named `optimize`, which will be exploited later on in Section 3.5 for the automatic optimization of circuit blocks.

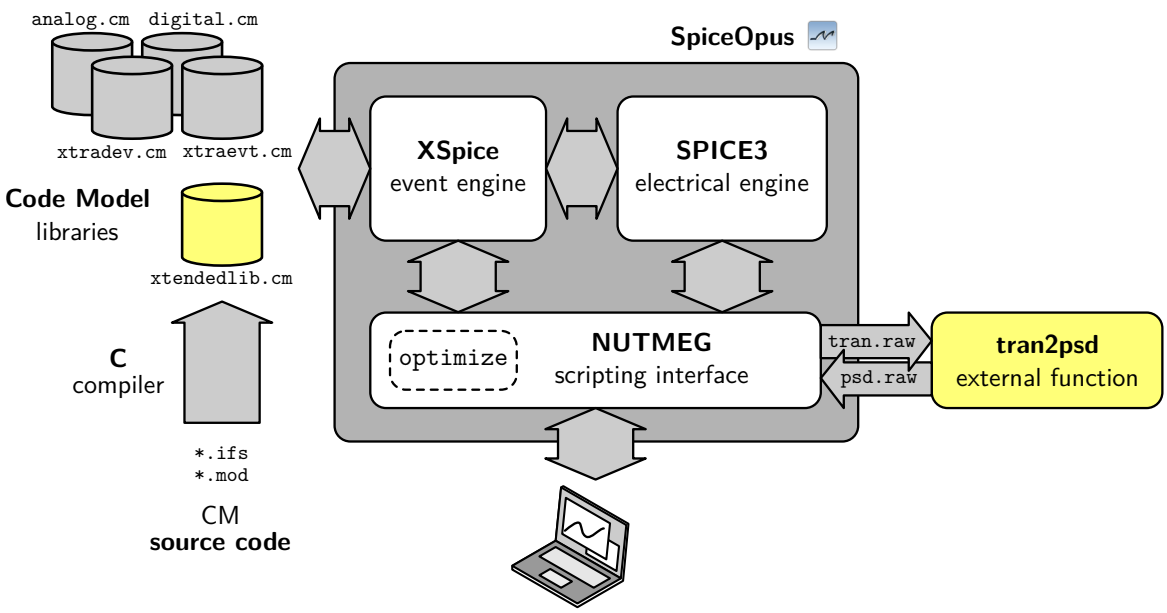


Figure 9 | Main components of the SpiceOpus mixed-signal simulation environment. Custom parts developed in the frame of this APDK are highlighted in yellow.

Coming back to the ΔΣM architecture circuit of Fig. 8, its equivalent netlist `dsm_arch_test.cir` is clearly based on XSpice a-suffixed elements. In fact, three different CMs from the custom library `xtendedlib.cm` are employed here: `quant2lsh`, `dac2lsym` and `zinteg2lim`. For your reference, the corresponding interface specification (IFS) and model definition (MOD) source code files are fully listed in Appendix A.

Concerning simulation environment, the strategy of Fig. 10 is followed in all SpiceOpus simulation cases of this APDK. Basically, self-contained circuit netlists (`*.cir`) are generated by Glade editor. They include not only the complete test bench around the device under test (DUT) but also the call to CNM25 device models (`cnm25mod.lib`) when required. No extra file inclusions are needed to use the custom CMs from `xtendedlib.cm`, since the entire collection is loaded at SpiceOpus startup, as displayed in Fig. 10.

Test scripts written in NUTMEG (`*.sp3`) are launched simply by invoking their name from the SpiceOpus shell following Fig. 10. 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.

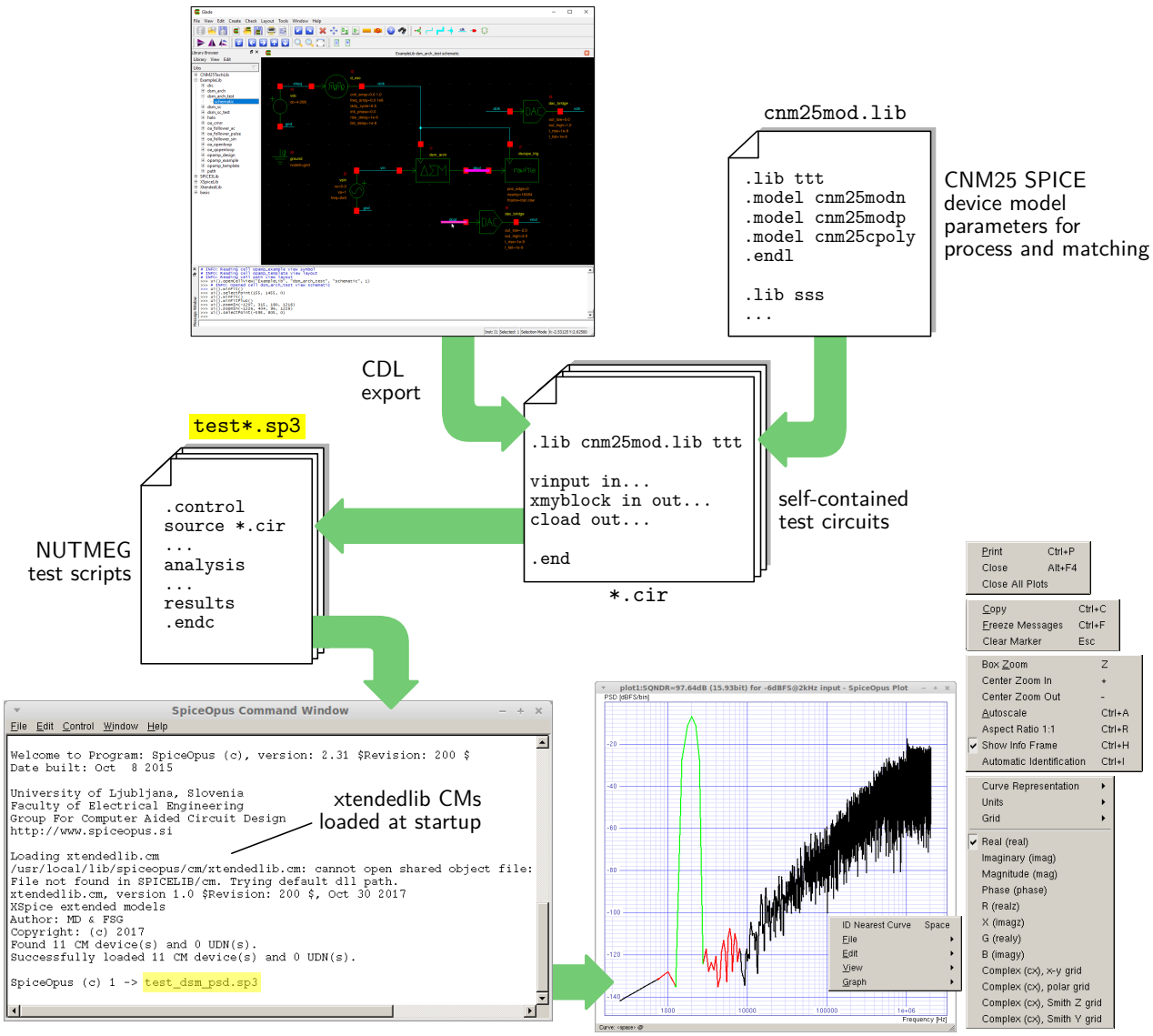


Figure 10 | SpiceOpus simulation interface used in the APDK.

In the case of our $\Delta\Sigma$M architecture, the test bench of Fig. 8(a) contains all the required auxiliary blocks for input stimulation, clock generation and waveform recording.

Q2. Before starting with the NUTMEG test scripts of this section, answer the following questions about `dsm_arch_test.cir`:

a. Classify all signal nodes (including inside `dsm_arch` subcircuit) according to **analog** and **digital** domains.

b. Which amplitude [dB_{FS}] and frequency [kHz] is programmed for the input harmonic **stimulus**?

c. Calculate the effective **sampling frequency** of this test setup.

d. Is the resulting OSR high enough according to your calculus in **Q1.a**?

e. What is the purpose of the **`dscope_trig`** element (see also Appendix A)?

Taking benefit of the simulation hierarchy of Fig. 10, five different test scripts are programmed in NUTMEG for the validation and optimization of the $\Delta\Sigma$M architecture. At this abstraction level, no circuit effects are considered but only quantization noise, distortion caused by internal full-scale limitation, and coefficient deviations due to technology mismatching:

- **`test_dsm_dc.sp3`**: This initial script changes the signal source in `dsm_arch_test.cir` from harmonic waveform to simple constant direct current (DC) input type, and it executes a short transient analysis in order to evaluate the resulting $\Delta\Sigma$ modulation in the output bitstream from the temporal viewpoint only.
- **`test_dsm_sin.sp3`**: Similar to `test_dsm_dc.sp3`, but for harmonic stimulus at 8 kHz.
- **`test_dsm_psd.sp3`**: In this case, a transient analysis is computed for a total length of 8 input harmonic cycles. Based on the modulated digital output bitstream, the computation of the equivalent PSD is performed with the help of the external program `tran2psd`. Finally, the spectral components are plotted in different colors and the overall SQNDR is extracted and echoed.
- **`test_dsm_sndr.sp3`**: This script involves the same analysis as `test_dsm_psd.sp3` but parameterized against input amplitude. After iteratively repeating the steps described above for each input amplitude value, SQNDR results are collected and plotted for measuring both $\mathrm{SQNDR}_{\mathrm{max}}$ and DR figures.
- **`test_dsm_coeff_a|b.sp3`**: Here, the transient and PSD simulation routine is parameterized against integrators and feedforward loop coefficients. Hence, results are reported in this case in terms of SQNDR maps. In particular, both $k_{\mathrm{i1}} : k_{\mathrm{i2}}$ and $k_{\mathrm{i1}} : k_{\mathrm{ff}}$ mapping studies are built.

apdk/spiceopus/test_dsm_dc.sp3

```
test_dsm_dc.sp3
* Transient test of dsm_arch_test.cir
                                with DC input
* Requires cmload xtendedlib.cm

.control

delcirc all
destroy all
delete all

source dsm_arch_test.cir
save v(vin) v(vout)
set reltol=1e-6
set abstol=1e-15
set vntol=1e-9
ic v(vdac:xi1)=-2.0
ic v(vint1out:xi1)=0.0
ic v(vint2out:xi1)=0.0

let vindc=0.75
echo "Changing to Vin={vindc}VDC and fs=4MHz"
let @vi2[sin]=({vindc};0;0)
let @vi4[dc]=4

echo "Starting transient simulation!"
echo "Please wait..."
tran 1e-9 20e-6
echo "Simulation completed!"

plot create plot1 v(vout) v(vin) vs time*1e6
plot append plot1 xlabel "Time [us]" ylabel
                              "Vin and dout [V]"

linearize 0.25e-6 v(vout)
let vindc=@vi2[sin][0]
let doutdc=mean(v(vout))

plot append plot1 title
             "dout(DC)={round(doutdc*1e6)/1e6}V
                              for Vin={vindc}VDC"
plot print plot1 file test_dsm_dc.pdf

echo " "
echo "dout(DC)={round(doutdc*1e6)/1e6}V
                              for Vin={vindc}VDC"
echo " "

.endc
.end
```

apdk/spiceopus/test_dsm_dc.sp3

☞ Launch apdk/spiceopus/spiceopus.bat (or .sh).

Q3. Simply type test_dsm_dc.sp3 to execute this NUTMEG test script, which stimulates the $\Delta\Sigma$M with $V_{\text{in}} = 0.75$ V and it computes the equivalent DC level of the modulated digital output in terms of input signal. Initial results should be similar to Fig. 11:

a. How is this equivalent **DC output level** computed? What is its value [V] at 20 μs?

b. Repeat the previous analysis for **200** μ**s**, **2 ms** and **4 ms** by changing stop time value marked in red, and report the corresponding DC output levels.

c. In case of increasing simulation time indefinitely, would **quantization noise** limit the resolution of the equivalent DC reading? Why?

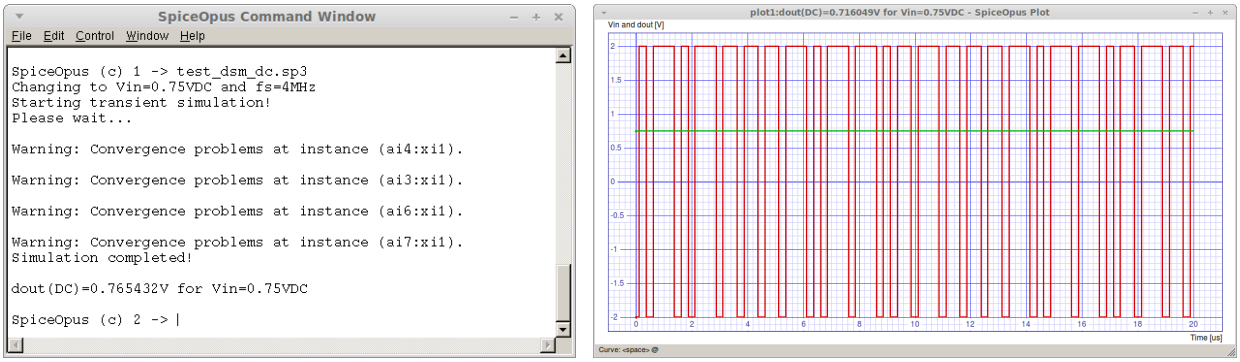


Figure 11 | Example of SpiceOpus results obtained from the NUTMEG test script apdk/spiceopus/test_dsm_dc.sp3.

apdk/spiceopus/test_dsm_sin.sp3

```
test_dsm_sin.sp3
* Transient test of dsm_arch_test.cir
                          with harmonic input
* Requires cmload xtendedlib.cm

.control

delcirc all
destroy all
delete all

source dsm_arch_test.cir
save all
set reltol=1e-6
set abstol=1e-15
set vntol=1e-9
ic v(vdac:xi1)=-2.0
ic v(vint1out:xi1)=0.0
ic v(vint2out:xi1)=0.0

let fin=8e3
echo "Changing to fin={fin}Hz"
let @vi2[sin]=(0;1;{fin})

echo "Starting transient simulation!"
echo "Please wait..."
tran 1e-9 250u
echo "Simulation completed!"

plot create plot1 v(vout) v(vin) vs time*1e6
plot append plot1 xlabel "Time [us]"
                 ylabel "Vin [V] and dout [Veq]"
plot print plot1 file test_dsm_sin_a.pdf
plot create plot2 v(vquantin:xi1) v(vint2out:xi1)
              v(vint1out:xi1) v(vin) vs time*1e6
plot append plot2 xlabel "Time [us]" ylabel
       "Vin, Vint1out, Vint2out and Vquantin [V]"
plot print plot2 file test_dsm_sin_b.pdf

.endc
.end
```

apdk/spiceopus/test_dsm_sin.sp3

Q4. Execute test script `test_dsm_sin.sp3` to stimulate the $\Delta\Sigma$M with a pure harmonic at 8 kHz, like in Fig. 12:

a. Describe the transient $\Delta\Sigma$ **modulation** in the output bitstream respect to the input stimulus.

b. Looking at the state signals of your $\Delta\Sigma$M, what is their **internal full-scale occupancy** respect to the range limits programmed in Fig. 8?

c. Are these internal state variables **periodic** like the input stimulus applied?

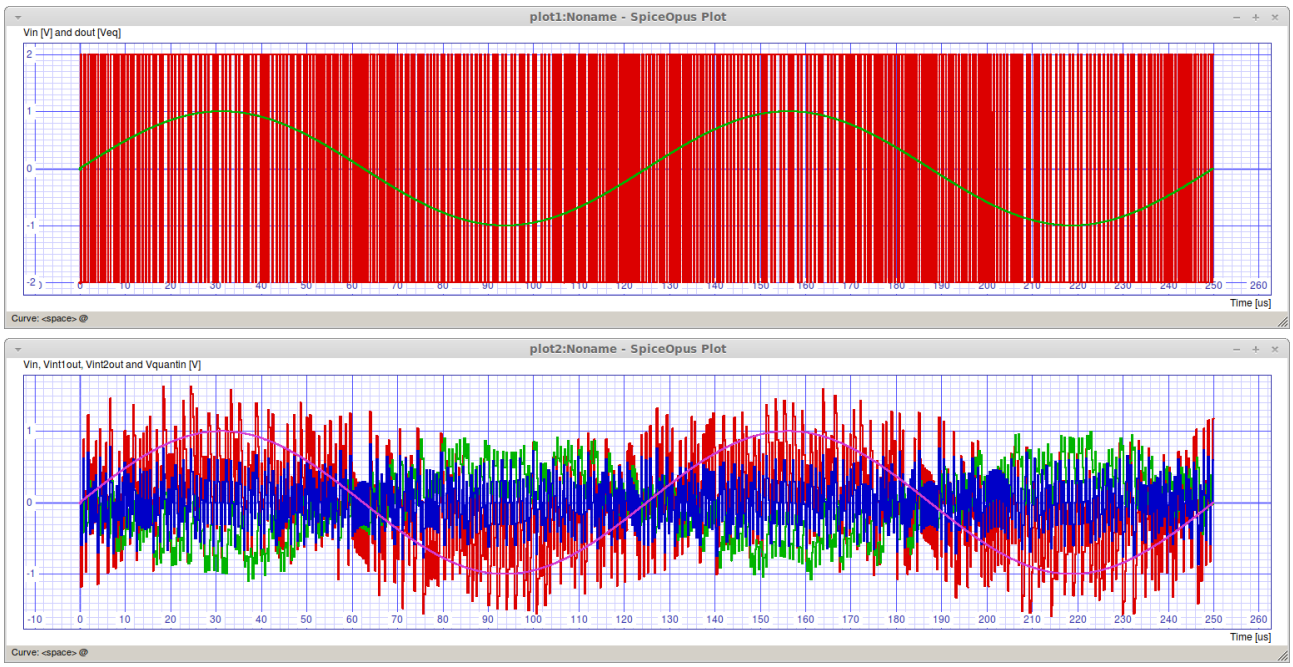


Figure 12 | Example of SpiceOpus results obtained from the NUTMEG test script `apdk/spiceopus/test_dsm_sin.sp3`.

apdk/spiceopus/test_dsm_psd.sp3

```
test_dsm_psd.sp3
* PSD profile test for dsm_arch_test.cir
* Requires cmload xtendedlib.cm

.control

delcirc all
destroy all
delete all

source dsm_arch_test.cir
save v(vout)
set reltol=1e-6
set abstol=1e-15
set vntol=1e-9
ic v(vdac:xi1)=-2.0
ic v(vint1out:xi1)=0.0
ic v(vint2out:xi1)=0.0

echo "Starting transient simulation!"
echo "Please wait..."
tran 1e-9 4.1e-3
echo "Simulation completed!"

shell "tran2psd -f 4.096e6 -w Blackman
                        -z tran.raw > psd.raw"
set filetype=ascii
load "psd.raw"

let vfs=@@mi8:xi1[out_high]
let a=@vi2[sin]
let ain=a[1]/vfs
let p=(mag(v(vpsd))*vfs*2)^2

let kbw=0
cursor kbw right frequency 8e3
let kleft=0
cursor kleft right frequency 2e3
let kright=kleft
while ((p[kleft-1] lt p[kleft])&
                            (kleft gt 0))
 kleft=kleft-1
 end
while ((p[kright+1] lt p[kright])&
                              (kright lt kbw))
 kright=kright+1
 end

let S=sum(p[kleft,kright])
let SQND=sum(p[3,kbw])
let SQNDRdB=10*log(S/(SQND-S))
let SQNDRb=((SQNDRdB-1.76)/6.02)
let PdB=10*log(p)
let AINdB=20*log(ain)

set color2 = r000g000b000
set color3 = r255g000b000
set color4 = r000g255b000
plot create plot1 xlog PdB vs frequency
          PdB[3,kbw] vs frequency[3,kbw]
   PdB[kleft,kright] vs frequency[kleft,kright]
plot append plot1 xlog xlabel "Frequency [Hz]"
                        ylabel "PSD [dBFS/bin]"
plot append plot1 xlog title
            "SQNDR={round(SQNDRdB*100)/100}dB
             ({round(SQNDRb*100)/100}bit) for
          {round(AINdB*10)/10}dBFS@2kHz input"
plot print plot1 file test_dsm_psd.pdf

echo " "
echo "SQNDR={round(SQNDRdB*100)/100}dB
        ({round(SQNDRb*100)/100}bit) for
       {round(AINdB*10)/10}dBFS@2kHz input"
echo " "

.endc

.end
```

apdk/spiceopus/test_dsm_psd.sp3

Q5. Execute test script `test_dsm_psd.sp3`. Results should be like Fig. 13:

a. What is the meaning of the **green** and **red** marked areas in the output spectrum?

b. Measure signal input **amplitude** [dB_{FS}]. Does it agree with **Q2.b**?

c. Extract the **noise shaping** slope [dB/dec]. Is it consistent with the order of the $\Delta\Sigma$M of Fig. 8?

d. What is the reported SQNDR [bit] at this particular input amplitude?

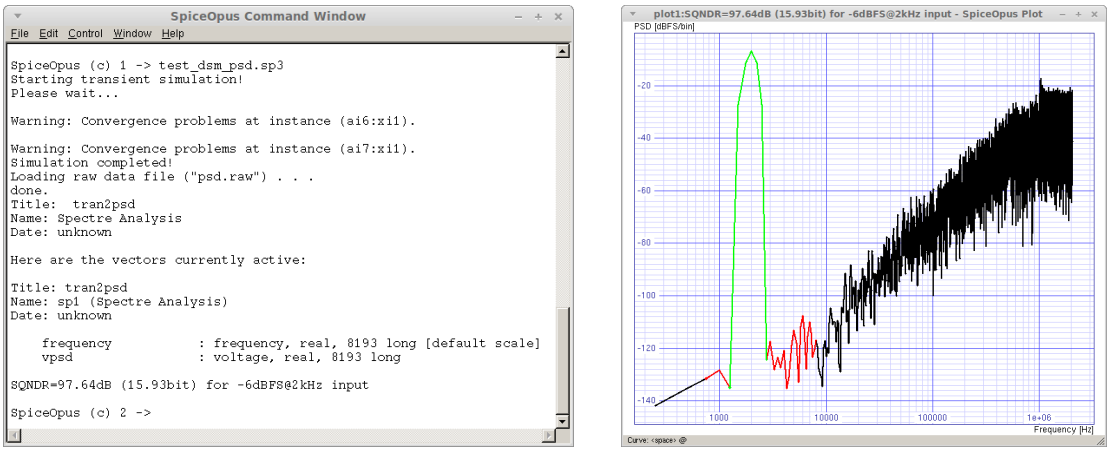


Figure 13 | Example of SpiceOpus results obtained from the NUTMEG test script `apdk/spiceopus/test_dsm_psd.sp3`.

apdk-kit/spiceopus/test_dsm_sndr.sp3

```
test_dsm_sndr.sp3
* SQNDR curve test for dsm_arch_test.cir
* Requires cmload xtendedlib.cm

.control

delcirc all
destroy all
delete all
set color2=r000g000b000
set color3=r255g000b000
set color4=r000g255b000

setplot new
nameplot myvars
let aindBFS=(-80;-60;-40;-20;-15;-10;-6;-3;-2;
                        -1.5;-1;-0.5;0;1.5;3)
let nsim=length(aindBFS)
let SQNDRdB=vector(nsim)*0

let k=0
while k le nsim-1

 source dsm_arch_test.cir
 let vfs=@@mi8:xi1[out_high]
 let vin=10^(aindBFS[k]/20)*vfs
 let @vi2[sin]=(0;vin;2e3)
 let out1max=@@mi2:xi1[out_max]
 let out1min=@@mi2:xi1[out_min]
 let out2max=@@mi5:xi1[out_max]
 let out2min=@@mi5:xi1[out_min]

 save v(vout)
 set reltol=1e-6
 set abstol=1e-15
 set vntol=1e-9
 ic v(vdac:xi1)=-2.0
 ic v(vint1out:xi1)=0.0
ic v(vint2out:xi1)=0.0

echo "Simulating for
             {round(aindBFS[k]*10)/10}dBFS
                          ({k+1}/{nsim}) ..."
tran 1e-9 4.1e-3

shell "tran2psd -f 4.096e6 -w Blackman
                     -z tran.raw > psd.raw"
set filetype=ascii
load "psd.raw"
let p=(mag(v(vpsd))*myvars.vfs*2)^2

let kbw=0
cursor kbw right frequency 8e3
let kleft=0
cursor kleft right frequency 2e3
let kright=kleft
while ((p[kleft-1] lt p[kleft])&(kleft gt 0))
 kleft=kleft-1
 end
while ((p[kright+1] lt p[kright])&
                            (kright lt kbw))
 kright=kright+1
 end

let S=sum(p[kleft,kright])
let SQND=sum(p[3,kbw])
let SQNDRdB=10*log(S/(SQND-S))
let SQNDRb=((SQNDRdB-1.76)/6.02)

let PdB=10*log(p)
let AINdB=myvars.aindBFS[myvars.k]
plot create plot1 xlog PdB vs frequency
          PdB[3,kbw] vs frequency[3,kbw]
   PdB[kleft,kright] vs frequency[kleft,kright]
plot append plot1 xlog xlabel "Frequency [Hz]"
                       ylabel "PSD [dBFS/bin]"
plot append plot1 xlog title
            "SQNDR={round(SQNDRdB*100)/100}dB
```

```
            ({round(SQNDRb*100)/100}bit) for
          {round(AINdB*10)/10}dBFS@2kHz input"

 setplot myvars
 let SQNDRdB[k]=sp1.SQNDRdB

 destroy sp1
 destroy tran1
 delcirc
 let k=k+1

 end


set filetype=ascii
unset appendwrite
write "test_dsm_sndr.raw" aindBFS SQNDRdB

plot create plot2 SQNDRdB vs aindBFS
plot append plot2 xlabel "Input amplitude [dBFS]"
                  ylabel "SQNDR [dB]"
plot append plot2 title "Dynamic range for
             {out1min}<zint1out<{out1max} and
                 {out2min}<zint2out<{out2max}"
plot append plot2 xlimit -100 5 ylimit 0 100
plot print plot2 file test_dsm_sndr.pdf

.endc

.end
```

apdk-kit/spiceopus/test_dsm_sndr.sp3

Q6. Execute test script `test_dsm_sndr.sp3`. Results should be similar to Fig. 14(a).

a. Explain the frequency **tones** exhibited by the PSD profile.

b. Which **slope** value [dB/dB$_{\mathrm{FS}}$] shows the SQNDR versus input amplitude curve?

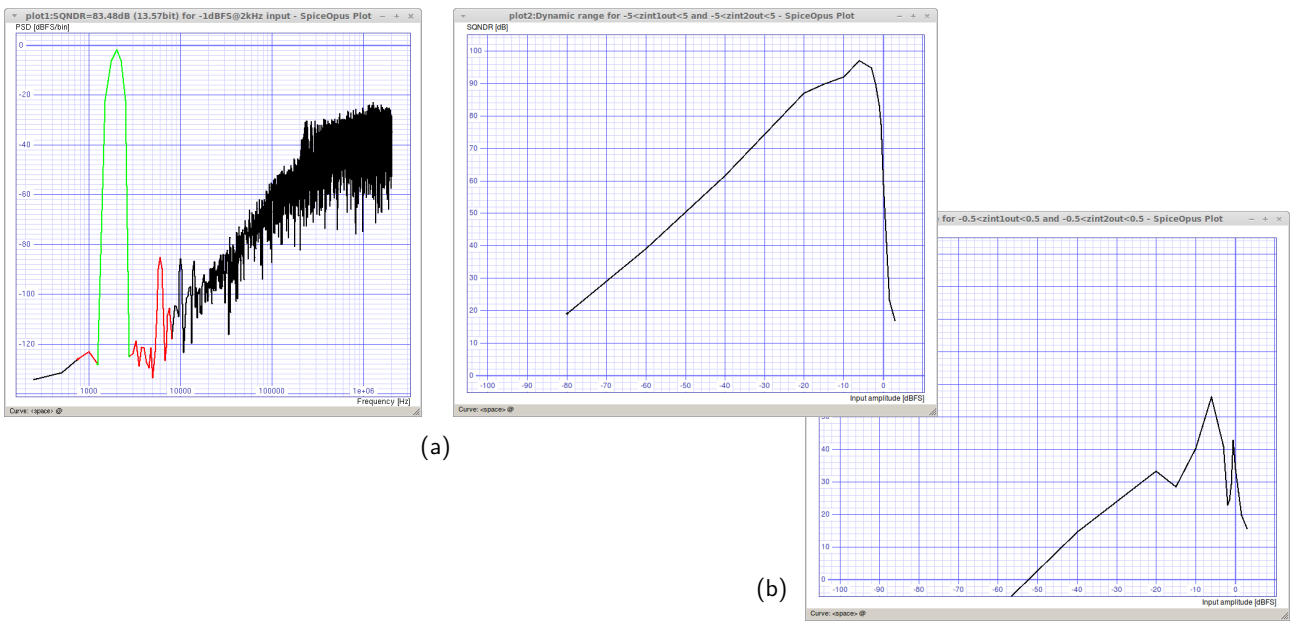
c. Estimate DR and $\mathrm{SQNDR}_{\mathrm{max}}$.

☞ Use Glade to reduce the **output range** of the Z-domain integrators (`out_min/max` params.) of Fig. 8.

☞ **Regenerate** the circuit netlist `apdk/spiceopus/dsm_arch_test.cir`.

☞ **Re-execute** `test_dsm_sndr.sp3` and check if any **lost** of DR nor $\mathrm{SQNDR}_{\mathrm{max}}$ is noted, like in the extreme case of Fig. 14(b).

Q7. Iterate the above steps to find the **minimum** full scale allowed for the Z-domain integrators of the $\Delta\Sigma$M.



(a)

(b)

Figure 14 Example of SpiceOpus results obtained from the NUTMEG test script `apdk/spiceopus/test_dsm_sndr.sp3` for `out_max/min`$=\pm5$ (a) and `out_max/min`$=\pm0.5$ (b).

apdk/spiceopus/test_dsm_coeff_a.sp3

```

test_dsm_coeff_a.sp3
* Coefficient mapping test for dsm_arch_test.cir
* Requires cmload xtendedlib.cm          (Part A)

.control

delcirc all
destroy all
delete all

setplot new
nameplot myvars
let ki1=(0.1;0.2;0.3;0.4;0.5)
let ki2=(0.5;0.6;0.7;0.8;0.9)
let nsim=length(ki1)*length(ki2)
let SQNDRb=vector(nsim)*0

let k=0
while k le length(ki1)-1
 let m=0
 while m le length(ki2)-1

  source dsm_arch_test.cir
  let vfs=@@mi8:xi1[out_high]
  let @@mi1:xi1[gain]=ki1[k]
  let @@mi3:xi1[gain]=ki2[m]

  save v(vout)
  set reltol=1e-6
  set abstol=1e-15
  set vntol=1e-9
  ic v(vdac:xi1)=-2.0
  ic v(vint1out:xi1)=0.0
  ic v(vint2out:xi1)=0.0

  echo "Simulating for ki1={round(ki1[k]*10)/10}
                  and ki2={round(ki2[m]*10)/10}
              ({m+k*length(ki1)+1}/{nsim}) ..."
  tran 1e-9 4.1e-3

  shell "tran2psd -f 4.096e6 -w Blackman
                      -z tran.raw > psd.raw"
  set filetype=ascii
  load "psd.raw"
  let p=(mag(v(vpsd))*myvars.vfs*2)^2

  let kbw=0
  cursor kbw right frequency 8e3
  let kleft=0
  cursor kleft right frequency 2e3
  let kright=kleft
  while ((p[kleft-1] lt p[kleft])&(kleft gt 0))
   kleft=kleft-1
   end
  while ((p[kright+1] lt p[kright])&
                              (kright lt kbw))
   kright=kright+1
   end
  let S=sum(p[kleft,kright])
  let SQND=sum(p[3,kbw])
  let SQNDRdB=10*log(S/(SQND-S))
  let SQNDRb=((SQNDRdB-1.76)/6.02)

  setplot myvars
  let SQNDRb[m+k*length(ki1)]=sp1.SQNDRb

  destroy sp1
  destroy tran1
  delcirc
  let m=m+1
  end
 let k=k+1
 end

let ki1all=vector(nsim)*0
let ki2all=vector(nsim)*0

echo " "
echo " SQNDR[bit]"
set label=( "  ki2  " )
let m=0
while m le length(ki2)-1
 set label=( $(label)
                 {round(10*ki2[m])/10} "  " )
 let m=m+1
 end
echo $(label)
echo " ki1   ------- ------- ------- -------
                                     -------"
let k=0
while k le length(ki1)-1
 set label=( " " {round(10*ki1[k])/10} "|")
 let m=0
 while m le length(ki2)-1
  set label=( $(label)
     {round(10*SQNDRb[m+k*length(ki1)])/10} "|")
  let ki1all[m+k*length(ki1)]=ki1[k]
  let ki2all[m+k*length(ki1)]=ki2[m]
  let m=m+1
  end
 echo $(label)
 let k=k+1
 end
echo "       ------- ------- ------- -------
                                     -------"
echo " "

set filetype=ascii
unset appendwrite
write "test_dsm_coeff_a.raw" ki1all ki2all SQNDRb

.endc

.end
```

apdk/spiceopus/test_dsm_coeff_a.sp3

☞ Execute test scripts `test_dsm_coeff_a.sp3` and `test_dsm_coeff_b.sp3`. SQNDR results for $k_{\mathrm{i1}} : k_{\mathrm{i2}}$ and $k_{\mathrm{i1}} : k_{\mathrm{ff}}$ deviation maps should be similar to Fig. 15.

Q8. The default range of coefficient deviations need to be readjusted according to the expected **technology mismatching** between SC elements:

a. Select a capacitance value C_{s} for the input sampler above the lower bound obtained from **Q1.b** (e.g. 1pF). Calculate the required PiP capacitor **area** WL for the `cnm25cpoly` device from the typical case (`ttt`) process parameters values listed in `cnm25mod.lib` file (starting on page 8).

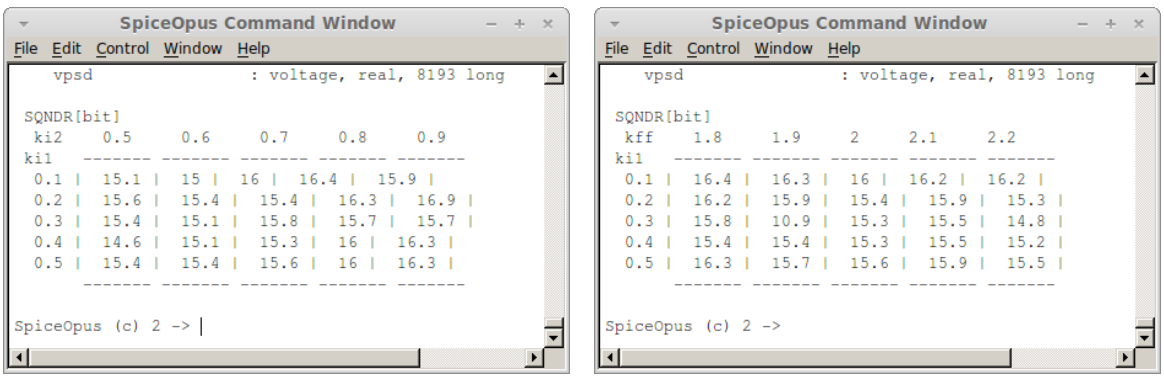
b. Based on this capacitor area (WL), estimate the **relative standard deviation** of C_{s} capacitance ratios (so of k-coefficients) according to Pelgrom's law [6]:

$$\sigma\left(\frac{\Delta k}{k}\right) \equiv \sigma\left(\frac{\Delta C_{\mathrm{s}}}{C_{\mathrm{s}}}\right) = \frac{A_C/C}{\sqrt{WL}} \tag{4}$$

where A_C/C can be obtained from the CNM25 matching section (`ttt_mc`) of the same `cnm25mod.lib` file.

☞ Edit `test_dsm_coeff_a.sp3` and `test_dsm_coeff_b.sp3` scripts in order to **adjust** coefficient ranges to $\pm 3\sigma(\Delta C_{\mathrm{s}}/C_{\mathrm{s}})$ around default values ($k_{\mathrm{i1}} = 0.3$, $k_{\mathrm{i2}} = 0.7$ and $k_{\mathrm{ff}} = 2.0$). In the case of `test_dsm_coeff_a.sp3`, the range vectors to be updated are marked in red on page 28 (code lines 14 and 15). Once completed, re-execute both test scripts.

Q9. What will be the maximum $\Delta\mathrm{SQNDR}$ **loss** [bit] for 99.7% ($\pm 3\sigma$) of the fabricated chip samples due to technology mismatching?



(a) (b)

Figure 15 Example of SpiceOpus results obtained from the NUTMEG test scripts `apdk/spiceopus/test_dsm_coeff_a|b.sp3` (a|b) for the default range of coefficient deviations.

3.4 HDL Blocks Specifications

Once your system top architecture is frozen, the next step in the schematic design methodology of Fig. 1 consists on finding the required performance for each basic building block of your IC. In this way, the circuit design of the overall system is simplified by splitting it into several smaller circuit parts, which can be designed separately. This strategy does not avoid final simulations at transistor level of the full system, but it speeds up the optimization process for each part. During this process, it is common to execute intermediate simulations of the whole system combining different abstraction levels for each block.

In these lab exercises, the above strategy is applied to the SC circuit implementation of the $\Delta\Sigma$M proposed in Fig. 16(b). In particular, the specifications required for the two OpAmp blocks of the Z-domain integrators will be found before designing them at transistor level.

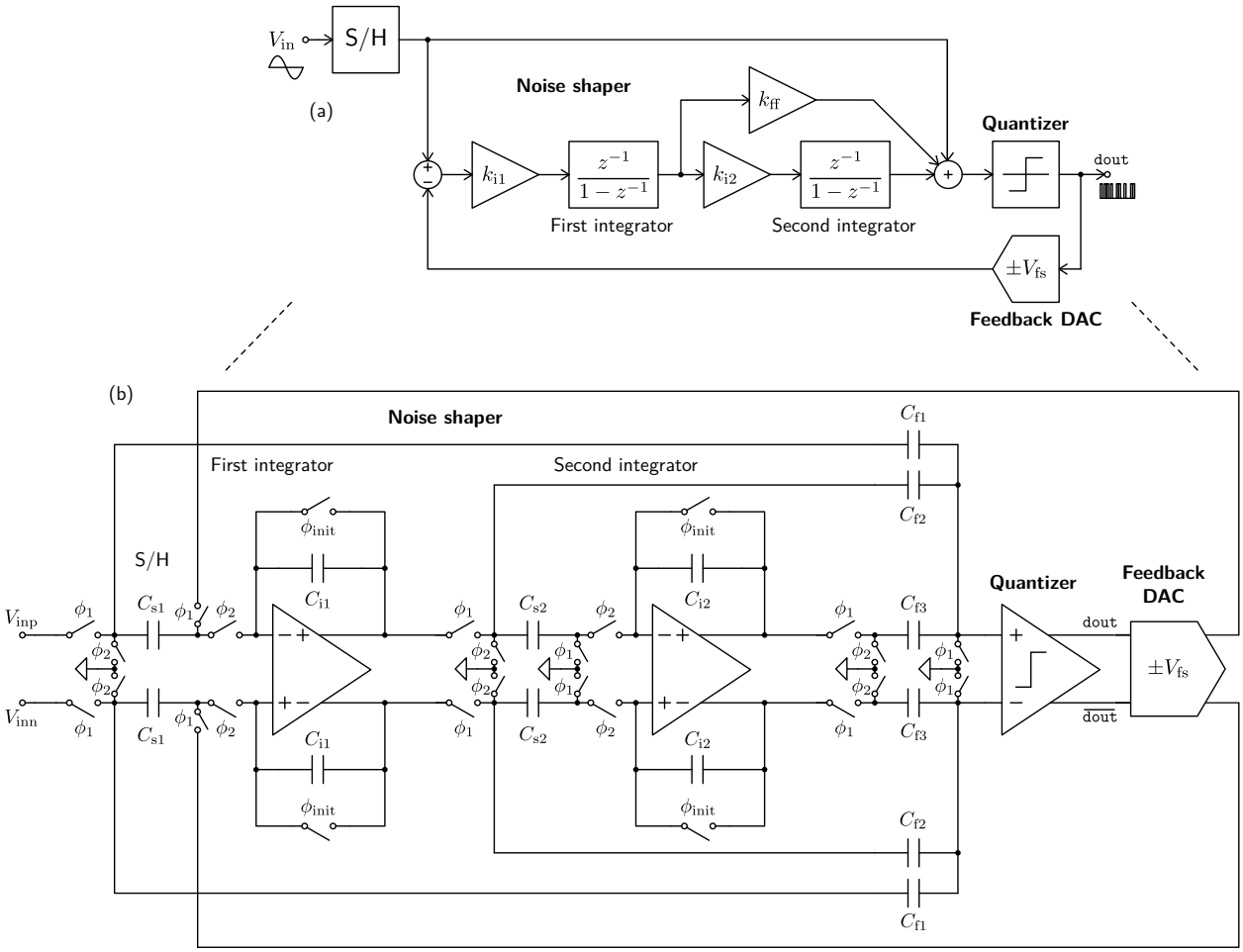


Figure 16 | $\Delta\Sigma$M Z-domain architecture (a) and fully-differential SC circuit implementation proposal (b).

In order to study the impact of these OpAmp blocks on the overall $\Delta\Sigma$M performance, a black-box description is chosen based on the following circuit-level parameters: open loop gain (G), slew-rate (SR) and gain bandwidth product (GBW). Such an OpAmp functional model allows us to study its effects on the SC integrators through the mathematical analysis of in Fig. 17. Once completed, the new $\Delta\Sigma$M model of Fig. 18 can be developed to include OpAmp circuit non-idealities without requiring to simulate the full SC of Fig. 16(b).

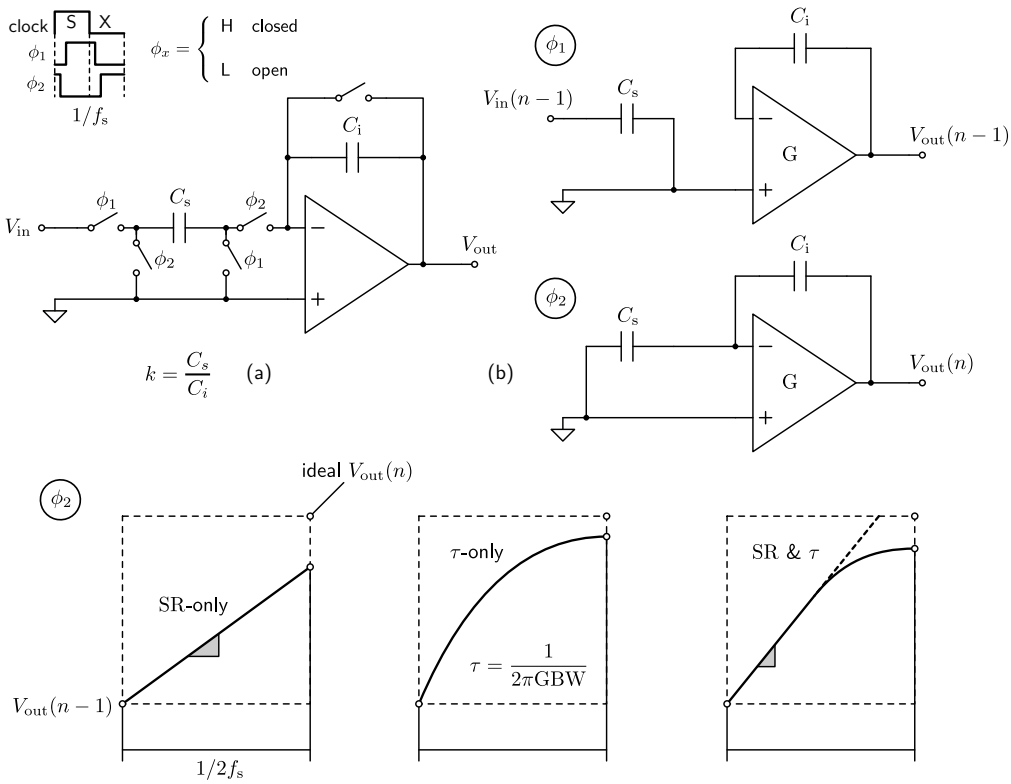


Figure 17 | Single-ended circuit (a) and operation (b) of the SC integrators of Fig. 16(b).

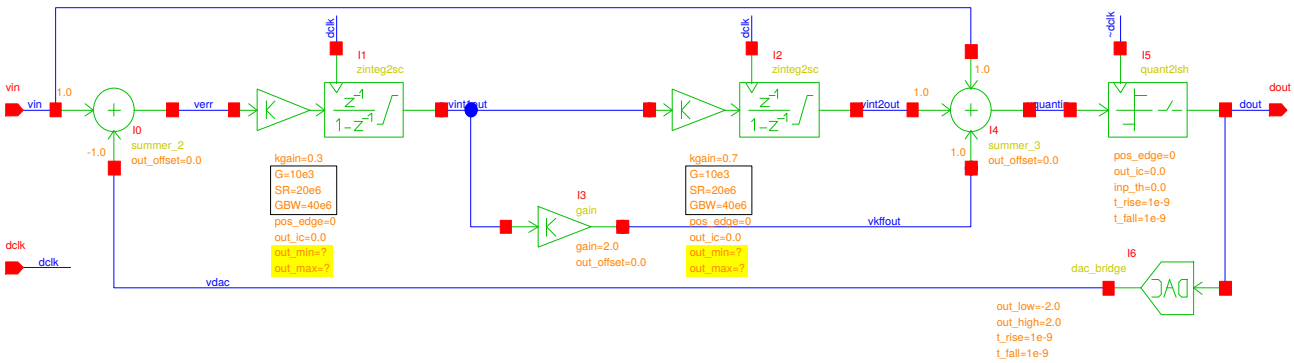


Figure 18 | Glade cell view `ExampleLib`→`dsm_sc`→`schematic` for the ΔΣM SC model of Fig. 16(b). Block parameters to be set are highlighted in yellow.

Q10. **Analyze** the operation of the SC circuits in Fig. 17 in order to obtain the general expressions of the Z-domain integrator output $V_{out}(n)$ as a function of $V_{in}(n-1)$, $V_{out}(n-1)$, k and the OpAmp parameters G, SR and GBW.

Q11. Translate the analytical expressions from **Q10** into the corresponding **C source code** to fill the missing section of the corresponding XSpice CM `apdk/spiceopus/xspice/xtendedlib/zinteg2sc.mod` file highlighted in **red** (see line 98 of code below). All the required XSpice syntax information, practical source code **examples** and further manual references are supplied in Appendix A.

apdk/spiceopus/xspice/xtendedlib/zinteg2sc.ifs

```
NAME_TABLE:

Spice_Model_Name: zinteg2sc
C_Function_Name:  cm_zinteg2sc
Description:      "Z-domain SC integrator"

PORT_TABLE:

Port_Name:        inp       clk       out
Description:      "input"   "clock"   "output"
Direction:        in        in        out
Default_Type:     v         d         v
Allowed_Types:    [v]       [d]       [v]
Vector:           no        no        no
Vector_Bounds:    -         -         -
Null_Allowed:     no        no        no

PARAMETER_TABLE:

Parameter_Name:   kgain                pos_edge                  out_ic
Description:      "input gain factor"  "L->H edge output sync?"  "output initial condition"
Data_Type:        real                 int                       real
Default_Value:    1.0                  0                         0.0
Limits:           [0 -]                [0 1]                     -
Vector:           no                   no                        no
Vector_Bounds:    -                    -                         -
Null_Allowed:     no                   no                        no

PARAMETER_TABLE:

Parameter_Name:   out_min               out_max
Description:      "lower output limit"  "upper output limit"
Data_Type:        real                  real
Default_Value:    -1.0                  1.0
Limits:           -                     -
Vector:           no                    no
Vector_Bounds:    -                     -
Null_Allowed:     no                    no

PARAMETER_TABLE:

Parameter_Name:   G                 GBW                       SR
Description:      "open loop gain"  "gain bandwidth product"  "slew-rate"
Data_Type:        real              real                      real
Default_Value:    1e3               1e6                       1e6
Limits:           [0 -]             [0 -]                     [0 -]
Vector:           no                no                        no
Vector_Bounds:    -                 -                         -
Null_Allowed:     no                no                        no
```

apdk/spiceopus/xspice/xtendedlib/zinteg2sc.ifs

apdk/spiceopus/xspice/xtendedlib/zinteg2sc.mod

```

#include <math.h>

#define SAMPLING_INTEGRATION 1
#define HOLDING 0
#define sign(x) (( x > 0 ) - ( x < 0 ))
```

```

void cm_zinteg2sc(ARGS)
{
           double inp,       /* analog voltage input */
                  out,       /* analog voltage output */
                  *inp_mem,  /* sampled input */
                  *out_mem,  /* integrated output */
                  *time_mem, /* previous integration time */
                  delta_v,   /* output voltage increment */
                  delta_t,   /* time increment */
                  out_ic,    /* output initial condition */
                  kgain,     /* input gain factor */
                  out_min,   /* minimum output limit */
                  out_max,   /* maximum output limit */
                  G,         /* open loop gain */
                  GBW,       /* gain bandwidth product */
                  SR,        /* slew-rate */
                  tau;       /* settling time constant */
  Digital_State_t clk,       /* current clock level */
                  *clk_mem,  /* previous clock level*/
                  pos_edge;  /* L->H edge clock output? */
              int action;    /* action type */
             char *error;    /* error message */

  inp = INPUT(inp);            /* Retriving input values */
  clk = INPUT_STATE(clk);
  kgain = PARAM(kgain);        /* Retriving parameters */
  pos_edge = PARAM(pos_edge);
  out_ic = PARAM(out_ic);
  out_min = PARAM(out_min);
  out_max = PARAM(out_max);
  G = PARAM(G);
  GBW = PARAM(GBW);
  SR = PARAM(SR);
  tau = 1/(2*M_PI*GBW);

  if (INIT==1) { /* Static storage allocation and checking */

    cm_analog_alloc(1,sizeof(double));
    cm_analog_alloc(2,sizeof(double));
    cm_analog_alloc(3,sizeof(double));
    cm_event_alloc(4,sizeof(Digital_State_t));

    if (out_min>out_max) {
      error = "\n*** zinteg2lim error: out_min>out_max !\n";
      cm_message_send(error);
    }
    if ((out_ic>out_max)||(out_ic<out_min)) {
      error = "\n*** zinteg2lim error: out_ic exceeds out_min,out_max !\n";
      cm_message_send(error);
    }

  }

  switch (ANALYSIS) {

    case TRANSIENT: /* Transient analysis */

      inp_mem = cm_analog_get_ptr(1,0); /* Retriving previous state */
      out_mem = cm_analog_get_ptr(2,0);
```

```
      time_mem = cm_analog_get_ptr(3,0);
      clk_mem = cm_event_get_ptr(4,0);

      if (TIME==0) { /* Initialization */

        *inp_mem = inp;
        *out_mem = out_ic;
        *time_mem = 0;
        out = out_ic;

      } else { /* Regular operation */

        if ((*clk_mem==ONE)&(clk==ZERO)) { /* Negative clk edge */
          if (pos_edge==FALSE)
            action = SAMPLING_INTEGRATION;

        } else {
          if ((*clk_mem==ZERO)&(clk==ONE)) { /* Positive clk edge */
            if (pos_edge==TRUE)
              action = SAMPLING_INTEGRATION;

          } else {  /* No clock edge */
              action = HOLDING;
          }
        }

        switch (action) {
           case SAMPLING_INTEGRATION: /* Sampling and integration */
            *inp_mem = inp;
            delta_t = (TIME-*time_mem)/2;

            /********** TO BE COMPLETED **********/

            if (out<out_min) { out = out_min; } /* Limiter */
            if (out>out_max) { out = out_max; }

            *out_mem = out;
            *time_mem = TIME;
            break;
          case HOLDING:    /* Holding */
            out = *out_mem;
        }
      }

      *clk_mem = clk;
      OUTPUT(out) = out;

      break;

    case DC:        /* DC analysis */
      OUTPUT(out) = out_ic;
      break;

    default:        /* Analysis not supported */
      error = "\n*** zinteg2sc error: analysis not supported !\n";
      cm_message_send(error);
  }
}
```

apdk/spiceopus/xspice/xtendedlib/zinteg2sc.mod

For your convenience, the APDK comes with the XSpice model `zinteg2sc.mod`, and the rest of custom models used in these lab exercises, already compiled in the `xtendedlib.cm` binary library according to your OS choice. Anyway, detailed instructions about how to compile XSpice code models are explained by the developers of SpiceOpus in `fides.fe.uni-lj.si/spice/xspice.html`.

☞ Open the `ExampleLib`→`dsm_sc`→`schematic` view of Fig. 18 with Glade, and **fill** the highlighted **full-scale** limits obtained in **Q7** for both Z-domain integrators. Check for schematic connectivity errors with Check→Check Cellview.

☞ Open the test bench `ExampleLib`→`dsm_sc_test`→`schematic` view of Fig. 19 and generate the equivalent XSpice netlist `apdk/spiceopus/dsm_sc_test.cir`.

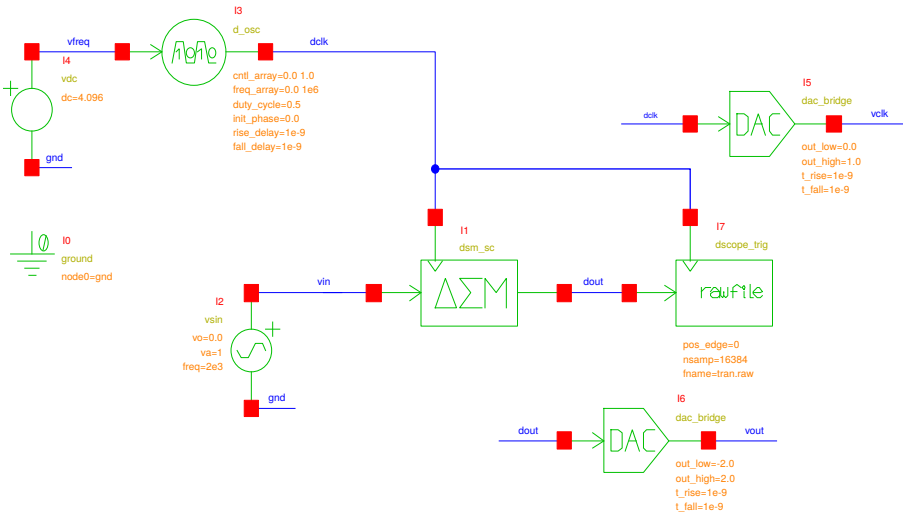


Figure 19 | Glade cell view `ExampleLib`→`dsm_sc_test`→`schematic` for the test of the $\Delta\Sigma$M SC model of Fig. 18.

Q12. Execute test script `apdk/spiceopus/test_dsm_oa.sp3` to simulate the $\Delta\Sigma$M SC model with **default specs** ($\mathrm{G} = 10000$, $\mathrm{SR} = 20\ \mathrm{V}/\mu\mathrm{s}$ and $\mathrm{GBW} = 40\ \mathrm{MHz}$) for both OpAmps of Fig. 16(b). Results should be similar to Fig. 20(a).

a. What is the expected SNDR at -6 $\mathrm{dB_{FS}}$ input? Is it still above 15-bit specs?

b. Relax **first integrator OpAmp** parameters G, SR and GBW in `dsm_sc_test.cir` and resimulate in order to find the minimum circuit specs to keep the overall $\Delta\Sigma$M performance just above 15-bit, unlike in the extreme case of Fig. 20(b).

c. Repeat the same procedure for the **second integrator OpAmp** parameters and **compare** the required performance figures between these two circuit blocks.

apdk/spiceopus/test_dsm_oa.sp3

```
test_dsm_oa.sp3
* PSD profile test for dsm_sc_test.cir
* Requires cmload xtendedlib.cm

.control
delcirc all
destroy all
delete all
source dsm_sc_test.cir
save v(vout)
set reltol=1e-6
set abstol=1e-15
set vntol=1e-9
ic v(vdac:xi1)=-2.0
ic v(vint1out:xi1)=0.0
ic v(vint2out:xi1)=0.0
echo "Starting transient simulation!"
echo "Please wait..."
tran 1e-9 4.1e-3
echo "Simulation completed!"
shell "tran2psd -f 4.096e6 -w Blackman
                    -z tran.raw > psd.raw"
set filetype=ascii
load "psd.raw"
let vfs=@@mi6:xi1[out_high]
let a=@vi2[sin]
let ain=a[1]/vfs
let p=(mag(v(vpsd))*vfs*2)^2

let kbw=0
cursor kbw right frequency 8e3
let kleft=0
cursor kleft right frequency 2e3
let kright=kleft
while ((p[kleft-1] lt p[kleft])&(kleft gt 0))
 kleft=kleft-1
 end
while ((p[kright+1] lt p[kright])&
                              (kright lt kbw))
 kright=kright+1
 end
let S=sum(p[kleft,kright])
let SQND=sum(p[3,kbw])
let SQNDRdB=10*log(S/(SQND-S))
let SQNDRb=((SQNDRdB-1.76)/6.02)
let PdB=10*log(p)
let AINdB=20*log(ain)

let Glin=@@mi1:xi1[g]
let GdB=20*log(Glin)
let SR=@@mi1:xi1[sr]/1e6
let GBW=@@mi1:xi1[gbw]/1e6
set color2 = r000g000b000
set color3 = r255g000b000
set color4 = r000g255b000
plot create plot1 xlog PdB vs frequency
          PdB[3,kbw] vs frequency[3,kbw]
  PdB[kleft,kright] vs frequency[kleft,kright]
plot append plot1 xlog xlabel "Frequency [Hz]"
                       ylabel "PSD [dBFS/bin]"
plot append plot1 xlog title
            "SQNDR={round(SQNDRdB*100)/100}dB
             ({round(SQNDRb*100)/100}bit) for
             G={round(Glin)} ({round(GdB)}dB)
                     SR={round(10*SR)/10}V/us
                   GBW={round(gbw*10)/10}MHz"
plot print plot1 file test_dsm_oa.pdf
echo " "
echo "SQNDR={round(SQNDRdB*100)/100}dB
              ({round(SQNDRb*100)/100}bit) at
          {round(AINdB*10)/10}dBFS@2kHz input"
echo "for G={round(Glin)} ({round(GdB)}dB)
                     SR={round(10*SR)/10}V/us
                   GBW={round(gbw*10)/10}MHz"
echo " "
.endc
.end
```

apdk/spiceopus/test_dsm_oa.sp3

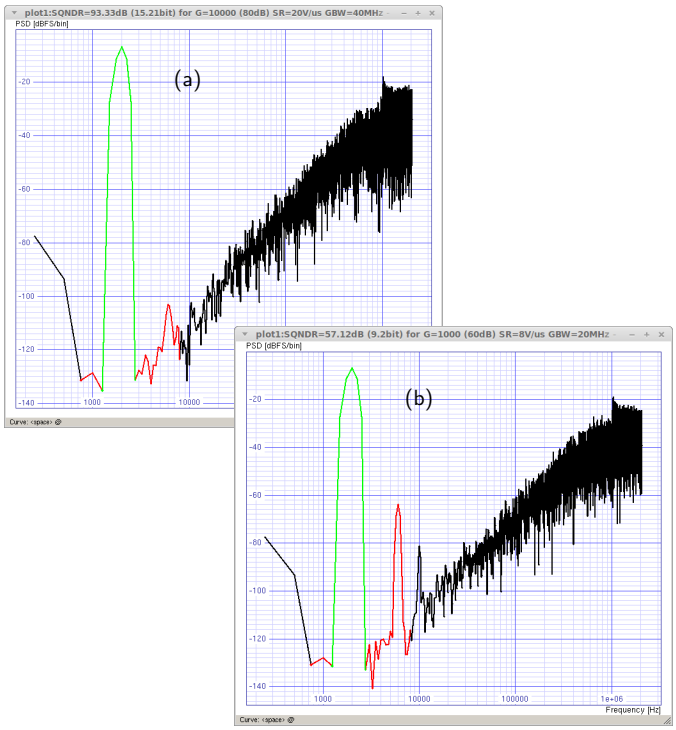


Figure 20 Example of SpiceOpus results obtained from the NUTMEG test script apdk/spiceopus/test_dsm_oa.sp3 when changing the first integrator OpAmp specs from $G = 10000$, $SR = 20\ V/\mu s$ and $GBW = 40\ MHz$ (a) to $G = 1000$, $SR = 8\ V/\mu s$ and $GBW = 20\ MHz$ (b).

3.5 Automatic Circuit Optimization

At this point of the EDA schematic methodology of Fig. 1, your IC design is already split into several circuit blocks for their independent optimization at transistor level against the target CMOS technology. This key step typically involves the definition of: design parameters (e.g. size of devices and biasing conditions), one (or more) figure of merit (FOM) to be measured at each simulation iteration (e.g. performance parameters, power and area resources), implicit rules for discarding unacceptable solutions, and the cost function in terms of the previous FOMs to be minimized when scoring all circuit candidates. In this sense, the simulation environment proposed in Fig. 9 makes extensive use of the SpiceOpus built-in `optimize` tool, a NUTMEG extension command aimed to manage the full optimization process described above.

Following the $\Delta\Sigma$M SC design case of previous section, and due to the limited scope of the current lab exercises, the automatic circuit optimization process described in this section is only applied to the first-integrator OpAmp block of Fig. 16(b). For simplification purposes, the single-ended two-stage Miller-compensated circuit of Fig. 21 is proposed. This OpAmp topology has been deeply studied [12], thus translating its design equations into the `optimize` scripting should be straightforward. The Glade equivalent schematic view with the initial device sizing is depicted in Fig. 22.

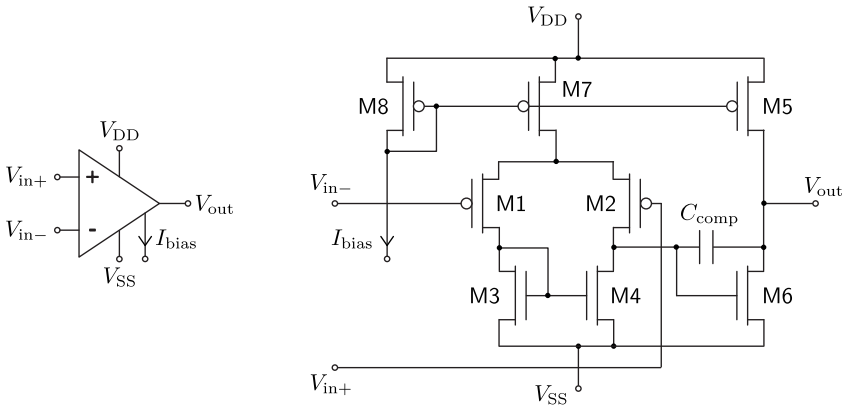


Figure 21 | Single-ended two-stage Miller-compensated CMOS OpAmp topology [12] proposed for the first Z-domain integrator of the SC $\Delta\Sigma$M.

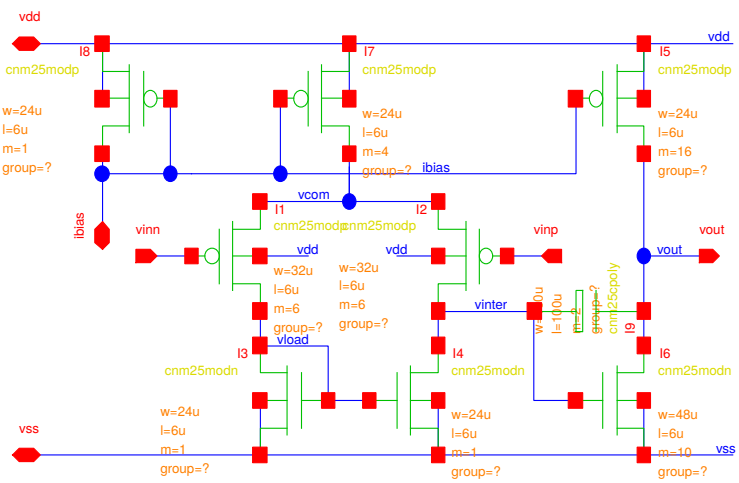


Figure 22 | Glade cell view `ExampleLib`→`opamp_design`→`schematic` with initial device sizing for the CMOS OpAmp of Fig. 21. The unset device property `group` is for the physical design stage of Section 3.6.

The first step towards any automated circuit optimization is to build a suitable test bench for the analysis of the circuit performance under optimization. In this sense, the APDK features a set of test-bench examples oriented to the characterization of OpAmps, as summarized in Fig. 23. 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`).

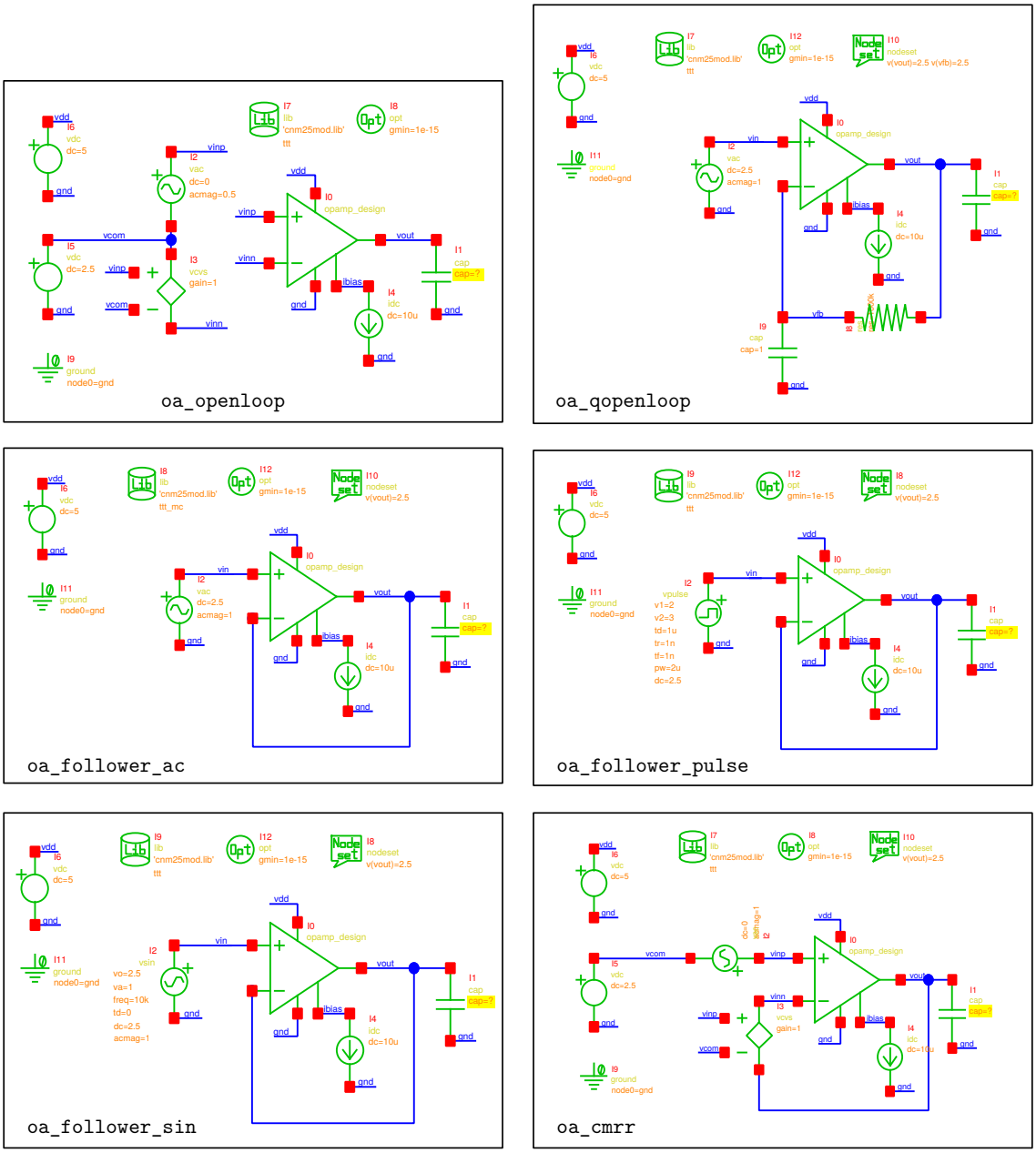


Figure 23 Test-bench schematics available in `ExampleLib` for the characterization of the `opamp_design` circuit of Fig. 22. Load capacitance conditions to be set are highlighted in yellow.

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=?
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 simulation, apart from the circuit under test itself, like: CNM25 models to use (`.lib`), convergence aids (`.nodeset`), initial conditions (`.ic`) or simulation options (`.options`).

Regarding CNM25 primitive devices, 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 2.2.

Q13. For the $\Delta\Sigma$M SC circuit proposal of Fig. 16(b):

a. Taking the input sampler capacitance value from **Q8.a** and the default coefficient configuration ($k_{\mathrm{i1}} = 0.3$, $k_{\mathrm{i2}} = 0.7$ and $k_{\mathrm{ff}} = 2.0$), resolve for all capacitors.

b. Find the **load condition** of the first-integrator OpAmp as $C_{\mathrm{load}} = C_{\mathrm{i1}} + C_{\mathrm{s2}} + C_{\mathrm{f2}}$.

☞ Using the Glade schematic editor, annotate this C_{load} value to each test bench of Fig. 23. Export the corresponding CDL netlists (`*.cir`) to `apdk/spiceopus`.

Q14. Execute the associated NUTMEG test scripts shown below to manually characterize the performance of the OpAmp of Fig. 22 (e.g. through the SpiceOpus cursor tool).

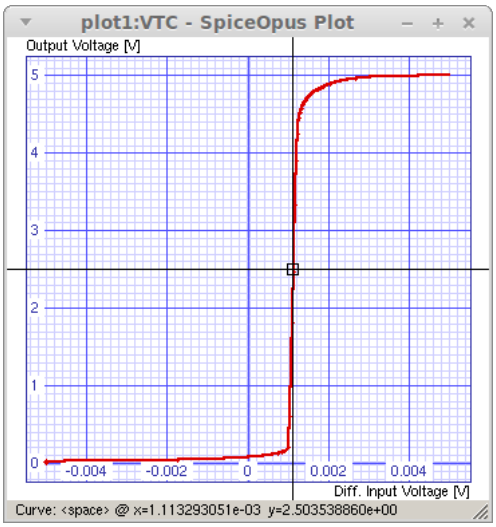
a. What is the biasing current level (I_{bias}) used in all tests?

b. Why is test bench `qopenloop` used for the Bode analysis instead of `openloop`?

c. What is the relation between the results of `test_oa_vtc` and `test_oa_offset`?

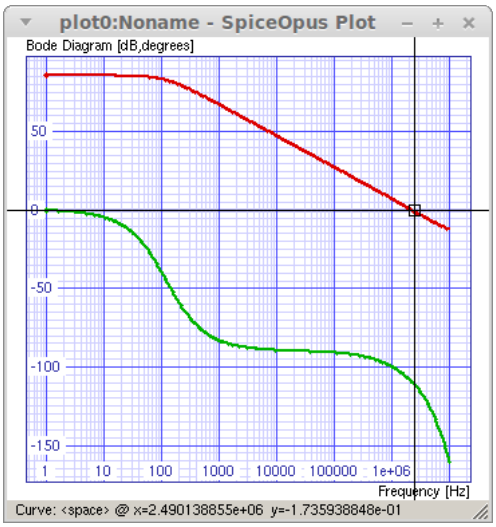
apdk/spiceopus/test_oa_vtc.sp3

```
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 create plot1 vout vs vdin xlabel 'Diff...
plot print plot1 file test_oa_vtc_a.pdf
dc vi2 -2.5m 2.5m 5u
...
.endc
.end
```



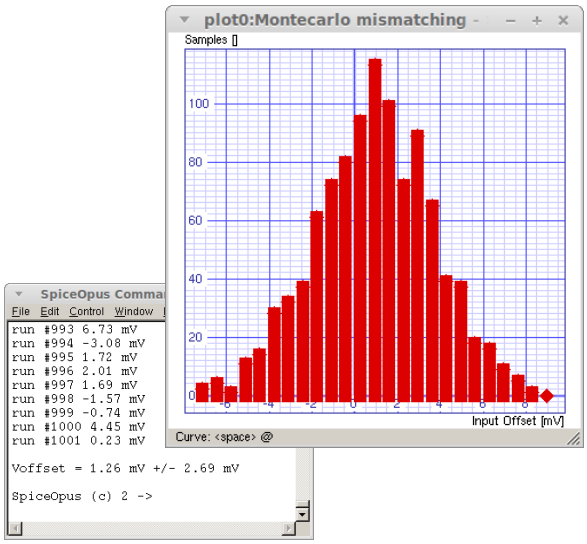
apdk/spiceopus/test_oa_bode.sp3

```
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 create plot1 Gmag Gph xlog xlabel 'Freq...
plot print plot1 file test_oa_bode.pdf
.endc
.end
```



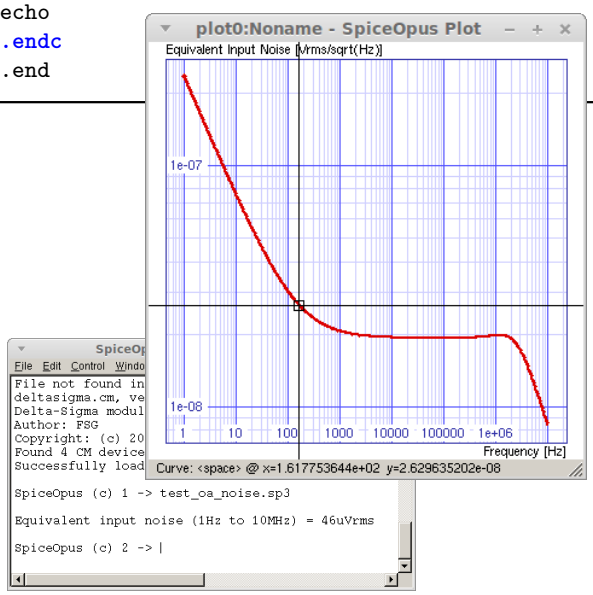
apdk/spiceopus/test_oa_offset.sp3

```
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 create plot1 ybin vs xbin*1e3 xlabel
         'Input Offset [mV]' ylabel 'Samples []'
                  title 'Montecarlo mismatching'
set plottype=line
set linewidth=1
plot print plot1 file test_oa_offset.pdf
.endc
.end
```



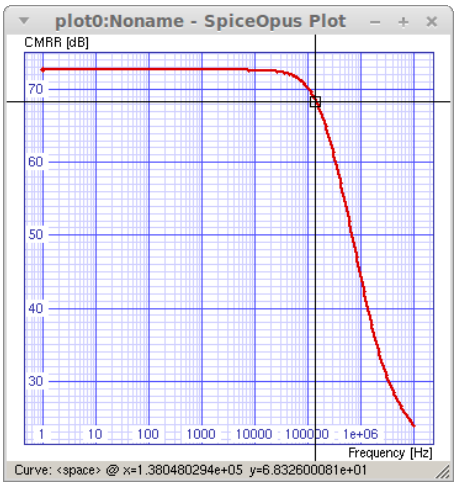
apdk/spiceopus/test_oa_noise.sp3

```
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 create plot1 sqrt(noise1.onoise_spectrum)
                      vs noise1.frequency ylog
                        xlabel 'Frequency [Hz]'
 ylabel 'Equivalent Input Noise [Vrms/sqrt(Hz)]'
plot print plot1 file test_oa_noise.pdf
let vnin=sqrt(noise2.onoise_total)*1e6
echo
echo Equivalent input noise (1Hz to 10MHz) =
                              {round(vnin)}uVrms
echo
.endc
.end
```



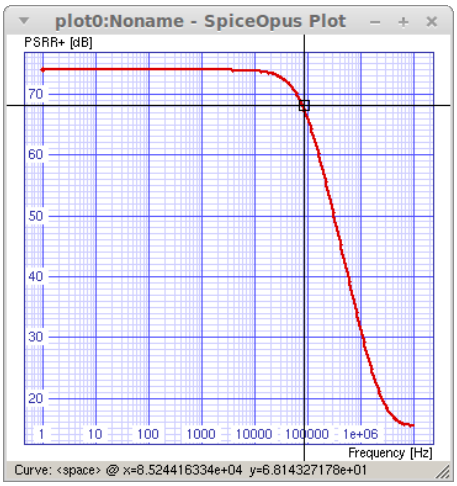
apdk/spiceopus/test_oa_cmrr.sp3

```
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 create plot1 CMRR xlog
    xlabel 'Frequency [Hz]' ylabel 'CMRR [dB]'
plot print plot1 file test_oa_cmrr.pdf
.endc
.end
```



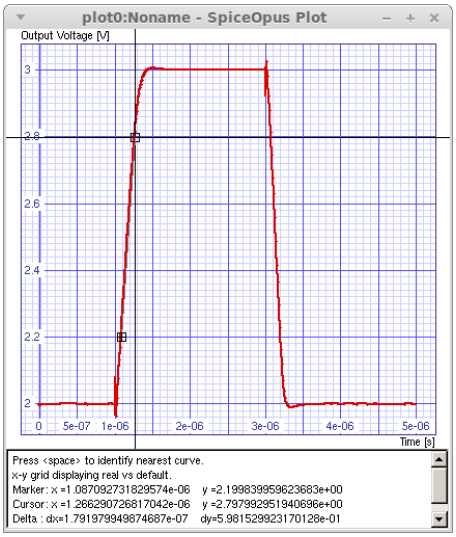
apdk/spiceopus/test_oa_psrr.sp3

```
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 create plot1 PSRRP xlog
    xlabel 'Frequency [Hz]' ylabel 'PSRR+ [dB]'
plot print plot1 file test_oa_psrr.pdf
.endc
.end
```



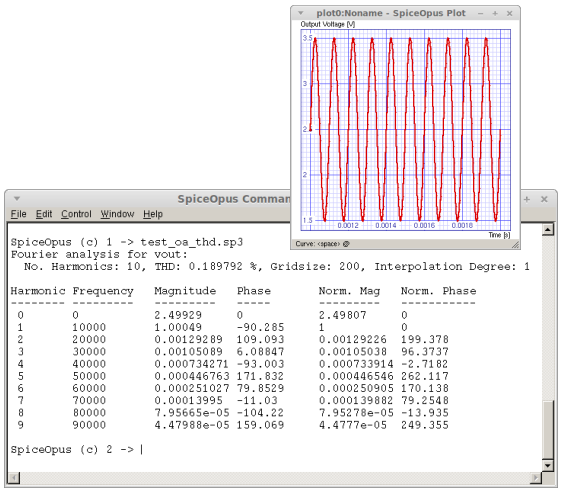
apdk/spiceopus/test_oa_sr.sp3

```
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 create plot1 vout xlabel 'Time [s]'
             ylabel 'Output Voltage [V]'
plot print plot1 file test_oa_sr.pdf
.endc
.end
```



apdk/spiceopus/test_oa_thd.sp3

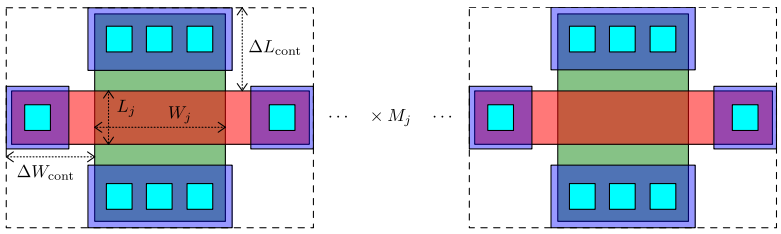
```
test_oa_thd.sp3
* Harmonic distortion analysis
.control
delcirc all
destroy all
delete all
save all
source oa_follower_sin.cir
tran 1u 2m 1m 1u
let vout=v(vout)
plot vout xlabel 'Time [s]'
          ylabel 'Output Voltage [V]'
fourier 10k vout
.endc
.end
```



Once the key test benches have been chosen for characterizing the circuit under optimization, the next step consists on the automation of their evaluation. Instead of doing manual measurements as above, a NUTMEG test script can be programmed to extract the key parameters automatically. Apart from the desired performance parameters, it is also recommended to monitor the design resources (i.e. circuit power consumption and Silicon area) as part of this automatic process. In the case of the CMOS OpAmp of Fig. 22, such performance parameters are chosen to be its open-loop gain (G), gain-bandwidth product (GBW), phase margin (PM) and slew-rate ($\mathrm{SR}\pm$), while resource parameters are defined as static power consumption (P_D) and device area.

Q15. For the `apdk/spiceopus/test_oa_datasheet.sp3` OpAmp extraction script:

a. In the case of total **device area**, extra contributions are estimated according to the following bounding-box rule:



(5)

$$\mathrm{area}_j \doteq (W_j + 2\Delta W_{\mathrm{cont}})(L_j + 2\Delta L_{\mathrm{cont}})M_j$$

Complete the **red-marked** missing values in the script for MOSFETs (`dwm` and `dlm`) and PiP capacitors (`dwc` and `dwc`) by applying the CNM25 **design rules** of Table 2.

b. Execute the NUTMEG script to **extract** the performance parameters from the OpAmp initial device sizing, like in the example of Fig. 24.

c. For the purpose of evaluating the impact of CMOS **technology deviations**, change the typical case (`ttt`) by the slow corner (`sss`) in test circuits `oa_qopenloop.cir` and `oa_follower_pulse.cir`, and repeat the previous point. Repeat again for the fast corner (`fff`) and annotate the minimum and maximum values of the OpAmp datasheet. Are all they compliant with the SC $\Delta\Sigma$M specifications from **Q12.b**?

apdk/spiceopus/test_oa_datasheet.sp3

```
test_oa_datasheet.sp3
* OpAmp datasheet extraction
.control
delcirc all
destroy all
delete all
save all
set units=degrees
source oa_qopenloop.cir
source oa_follower_pulse.cir

setcirc ckt1
op
let pd=-i(vi6)*v(vdd)*1e3
let dwm=?
let dlm=?
let dwc=?
let dlc=?
let aream1=(@xi1:xi0[w]+2*{dwm})*
           (@xi1:xi0[l]+2*{dlm})*(@xi1:xi0[m])
let aream2=(@xi2:xi0[w]+2*{dwm})*
           (@xi2:xi0[l]+2*{dlm})*(@xi2:xi0[m])
let aream3=(@xi3:xi0[w]+2*{dwm})*
           (@xi3:xi0[l]+2*{dlm})*(@xi3:xi0[m])
let aream4=(@xi4:xi0[w]+2*{dwm})*
           (@xi4:xi0[l]+2*{dlm})*(@xi4:xi0[m])
let aream5=(@xi5:xi0[w]+2*{dwm})*
           (@xi5:xi0[l]+2*{dlm})*(@xi5:xi0[m])
let aream6=(@xi6:xi0[w]+2*{dwm})*
           (@xi6:xi0[l]+2*{dlm})*(@xi6:xi0[m])
let aream7=(@xi7:xi0[w]+2*{dwm})*
           (@xi7:xi0[l]+2*{dlm})*(@xi7:xi0[m])
let aream8=(@xi8:xi0[w]+2*{dwm})*
           (@xi8:xi0[l]+2*{dlm})*(@xi8:xi0[m])
let areaccomp=(@xi9:xi0[w]+{dwc})*
           (@xi9:xi0[l]+{dlc})*(@xi9:xi0[m])
let area=(aream1+aream2+aream3+aream4+aream5+
          aream6+aream7+aream8+areaccomp)*1e6
ac dec 50 10 100e6
let gmag=20*log10(mag(v(vout)))
let gph=phase(v(vout))
let gdc=gmag[0]
let c=0
cursor c right gmag 0
let gbw=abs(frequency[%c])/1e6
let pm=180+gph[%c]
plot create plot1 gmag gph vs frequency xlog
 xlabel 'Frequency[Hz]' ylabel 'Magnitude[dB] &
 Phase[deg]' title 'Bode Diagram' ylimit -180 100
plot print plot1 file test_oa_datasheet_a.pd

setcirc ckt2
tran 1n 5u
let vout=v(vout)
let c=0
cursor c right vout 2.1
let t1=time[%c]
cursor c right vout 2.9
let t2=time[%c]
let srpos=0.8/(t2-t1)*1e-6
cursor c right time 3u
cursor c right vout 2.9
let t1=time[%c]
cursor c right vout 2.1
let t2=time[%c]
let srneg=0.8/(t2-t1)*1e-6
plot create plot2 vout vs time xlabel 'Time [s]'
 ylabel 'Output Voltage[V]' title 'Step Response'
plot print plot2 file test_oa_datasheet_b.pdf
echo " "
echo "******************"
echo "Param Units Value"
echo "******************"
echo "G       [dB] {round(ac1.gdc*10)/10}"
echo "GBW    [MHz] {round(ac1.gbw*100)/100}"
echo "PM     [deg] {round(ac1.pm*10)/10}"
echo "SR+   [V/us] {round(tran1.srpos*10)/10}"
echo "SR-   [V/us] {round(tran1.srneg*10)/10}"
echo "PD      [mW] {round(op1.pd*100)/100}"
echo "Area   [mm2] {round(op1.area*1000)/1000}"
echo "******************"
echo " "
.endc
.end
```

apdk/spiceopus/test_oa_datasheet.sp3

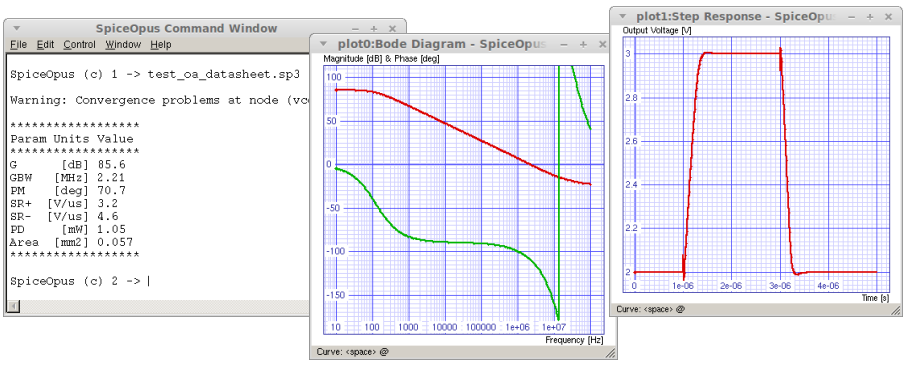


Figure 24 | Example of SpiceOpus results from `apdk/spiceopus/test_oa_datasheet.sp3` for the OpAmp of Fig. 22 and CNM25 typical-case (`ttt`) conditions.

After the performance extraction of a particular IC block has been automated, its optimization routine can be developed based on the SpiceOpus `optimze` command. In general, this procedure involves the definition of the script sections listed in Table 4.

Section	Code examples
Optimization **variables**	`optimize parameter 0 @xi1:xi0[w] low 6u high 250u initial 32u`
to be changed	`optimize parameter 1 @xi6:xi0[m] low 5 high 50 initial 10`
Simulation **analysis**	`optimize analysis 25 ac dec 50 10 100e6`
and **FOM** evaluation	`optimize analysis 29 let gdc=gmag[0]`
Implicit rules to	`optimize implicit 0 ac2.gdc gt 60`
filter candidates	`optimize implicit 1 ac2.pm gt 60`
Cost function	`optimize cost 1/abs(ac2.gbw)`
to be **minimized**	`optimize cost 1/abs(tran2.srneg)+1/abs(tran2.srpos)`
Algorithm	`optimize method genetic elitism yes maxgen 100`
selection	`optimize method axis_search method quadratic`
Optimization	`optimize options number_of_iterations 1000`
options	`optimize options initial_guess 0`

Table 4 Basic syntax examples for the SpiceOpus `optimize` command. Further information can be found in `fides.fe.uni-lj.si/spice/optimize.html`.

Q16. Based on the `apdk/spiceopus/test_oa_optimize.sp3` script for the optimization of the OpAmp circuit block of Fig. 22:

a. How many and which **design variables** are defined?

b. List the **implicit constraints** used to filter optimization solutions.

d. Define the **cost function** expression in line 139 using the FOMs available from optimization analysis: $|\mathrm{G(DC)}|$ (`ac2.gdc` [dB]), $\mathrm{SR}\pm$ (`tran2.srpos|neg` [V/μs]), GBW (`ac2.gbw` [MHz]), phase margin (`ac2.pm` [deg]), static power consumption (`op2.pd` [mW]) and device area (`op2.area` [mm^2]).

e. Once completed, run the optimization script. Result examples are given in Fig. 25. Does your OpAmp optimization meet the SC $\Delta\Sigma$M specifications from **Q12.b**?

f. Try to improve your OpAmp performance or resource savings by refining: cost function **expression** (code line 139), design variable **ranges** (67-71), optimization **method** (140) or **options** (143).

g. In case minimum specifications can not be reached, annotate the best OpAmp results in terms of $|\mathrm{G(DC)}|$, min($\mathrm{SR}\pm$) and GBW, and repeat **Q12.b** in order to quantify the expected **degradation** of your SC $\Delta\Sigma$M dynamic range.

apdk/spiceopus/test_oa_optimize.sp3

```
test_oa_optimize.sp3
*OpAmp circuit optimization
.control
delcirc all
destroy all
delete all
save all
set units=degrees
source oa_qopenloop.cir
source oa_follower_pulse.cir

**** Initial Values
setcirc ckt1
op
let pd=-i(vi6)*v(vdd)*1e3
let w1i=@xi1:xi0[w]*1e6
let l1i=@xi1:xi0[l]*1e6
let m1i=@xi1:xi0[m]
let w6i=@xi6:xi0[w]*1e6
let l6i=@xi6:xi0[l]*1e6
let m6i=@xi6:xi0[m]
let w3i=@xi3:xi0[w]*1e6
let l3i=@xi3:xi0[l]*1e6
let m3i=@xi3:xi0[m]
let w8i=@xi8:xi0[w]*1e6
let l8i=@xi8:xi0[l]*1e6
let m8i=@xi8:xi0[m]
let m5i=@xi5:xi0[m]
let m7i=@xi7:xi0[m]
let wccompi=@xi9:xi0[w]*1e6
let lccompi=@xi9:xi0[l]*1e6
let mccompi=@xi9:xi0[m]
let aream1=m1i*(w1i+12.5)*(l1i+11)
let aream6=m6i*(w6i+12.5)*(l6i+11)
let aream3=m3i*(w3i+12.5)*(l3i+11)
let aream8=m8i*(w8i+12.5)*(l8i+11)
let areaccomp=mccompi*(wccompi+6.25)*(lccompi+10.5)
let area=(2*aream1+aream6+2*aream3+(1+(m7i+m5i)/m8i)*aream8+areaccomp)*1e-6
ac dec 50 10 100e6
let gmag=20*log10(mag(v(vout)))
let gph=phase(v(vout))
let c=0
let gdc=gmag[c]
cursor c right gmag 0
let gbw=abs(frequency[%c])/1e6
let pm=180+gph[%c]
if pm ge 90
 pm=pm-360
end
setcirc ckt2
tran 1n 5u
let vout=v(vout)
let c=0
cursor c right vout 2.1
let t1=time[%c]
cursor c right vout 2.9
let t2=time[%c]
let srpos=0.8/(t2-t1)*1e-6
cursor c right time 3u
```

```
cursor c right vout 2.9
let t1=time[%c]
cursor c right vout 2.1
let t2=time[%c]
let srneg=0.8/(t2-t1)*1e-6

**** Optimization Process
optimize parameter 0 @xi1:xi0[w] low 6u high 250u initial 32u
optimize parameter 1 @xi6:xi0[m] low 5 high 50 initial 10
optimize parameter 2 @xi5:xi0[m] low 1 high 50 initial 16
optimize parameter 3 @xi7:xi0[m] low 1 high 50 initial 4
optimize parameter 4 @xi9:xi0[w] low 50u high 250u initial 100u

optimize analysis 0 setplot new
optimize analysis 1 nameplot myvalues
optimize analysis 2 let m1_w=@xi1:xi0[w]
optimize analysis 3 let m6_m=@xi6:xi0[m]
optimize analysis 4 let m5_m=@xi5:xi0[m]
optimize analysis 5 let m7_m=@xi7:xi0[m]
optimize analysis 6 let ccomp_w=@xi9:xi0[w]
optimize analysis 7 setcirc ckt1
optimize analysis 8 let @xi1:xi0[w]=myvalues.m1_w
optimize analysis 9 let @xi6:xi0[m]=myvalues.m6_m
optimize analysis 10 let @xi5:xi0[m]=myvalues.m5_m
optimize analysis 11 let @xi7:xi0[m]=myvalues.m7_m
optimize analysis 12 let @xi9:xi0[w]=myvalues.ccomp_w
optimize analysis 13 let @xi2:xi0[w]=@xi1:xi0[w]
optimize analysis 14 let @xi3:xi0[m]=ceil(@xi7:xi0[m]/@xi5:xi0[m]/2*@xi6:xi0[m])
optimize analysis 15 let @xi4:xi0[m]=@xi3:xi0[m]
optimize analysis 16 let @xi9:xi0[l]=@xi9:xi0[w]
optimize analysis 17 op
optimize analysis 18 let pd=-i(vi6)*v(vdd)*1e3
optimize analysis 19 let aream12=2*(@xi1:xi0[m])*(@xi1:xi0[w]+12.5u)*(@xi1:xi0[l]+11u)
optimize analysis 20 let aream6=(@xi6:xi0[m])*(@xi6:xi0[w]+12.5u)*(@xi6:xi0[l]+11u)
optimize analysis 21 let aream34=2*(@xi3:xi0[m])*(@xi3:xi0[w]+12.5u)*(@xi3:xi0[l]+11u)
optimize analysis 22 let aream875=(@xi8:xi0[m]+@xi7:xi0[m]+@xi5:xi0[m])*(@xi8:xi0[w]+12.5u)*
                                                                        (@xi8:xi0[l]+11u)
optimize analysis 23 let areaccomp=(@xi9:xi0[m])*(@xi9:xi0[w]+6.25u)*(@xi9:xi0[l]+10.5u)
optimize analysis 24 let area=(aream12+aream6+aream34+aream875+areaccomp)*1e6
optimize analysis 25 ac dec 50 10 100e6
optimize analysis 26 let gmag=20*log10(mag(v(vout)))
optimize analysis 27 let gph=phase(v(vout))
optimize analysis 28 let c=0
optimize analysis 29 let gdc=gmag[c]
optimize analysis 30 cursor c right gmag 0
optimize analysis 31 let gbw=abs(frequency[%c])/1e6
optimize analysis 32 let pm=180+gph[%c]
optimize analysis 33 if pm ge 90
optimize analysis 34  pm=pm-360
optimize analysis 35 end
optimize analysis 36 setcirc ckt2
optimize analysis 37 let @xi1:xi0[w]=myvalues.m1_w
optimize analysis 38 let @xi6:xi0[m]=myvalues.m6_m
optimize analysis 39 let @xi5:xi0[m]=myvalues.m5_m
optimize analysis 40 let @xi7:xi0[m]=myvalues.m7_m
optimize analysis 41 let @xi9:xi0[w]=myvalues.ccomp_w
optimize analysis 41 let @xi2:xi0[w]=@xi1:xi0[w]
optimize analysis 43 let @xi3:xi0[m]=ceil(@xi7:xi0[m]/@xi5:xi0[m]/2*@xi6:xi0[m])
optimize analysis 44 let @xi4:xi0[m]=@xi3:xi0[m]
optimize analysis 45 let @xi9:xi0[l]=@xi9:xi0[w]
```

```
optimize analysis 46 tran 1n 5u
optimize analysis 47 let vout=v(vout)
optimize analysis 48 let c=0
optimize analysis 49 cursor c right vout 2.1
optimize analysis 50 let t1=time[%c]
optimize analysis 51 cursor c right vout 2.9
optimize analysis 52 let t2=time[%c]
optimize analysis 53 let srpos=0.8/(t2-t1)*1e-6
optimize analysis 54 cursor c right time 3u
optimize analysis 55 cursor c right vout 2.9
optimize analysis 56 let t1=time[%c]
optimize analysis 57 cursor c right vout 2.1
optimize analysis 58 let t2=time[%c]
optimize analysis 59 let srneg=0.8/(t2-t1)*1e-6

optimize implicit 0 ac2.gdc gt 60
optimize implicit 1 ac2.pm gt 60
optimize implicit 3 op2.pd lt 2.5

optimize cost ?
optimize method genetic elitism yes maxgen 100
*optimize method monte_carlo
*optimize method complex oscillation_detection yes k 10 alpha 1.3 size 2.5u
optimize options number_of_iterations 1000

rusage
optimize
rusage
**** Final Solution
setplot optimize1
setcirc ckt1
let @xi1:xi0[w]=parameter[0]
let @xi6:xi0[m]=parameter[1]
let @xi5:xi0[m]=parameter[2]
let @xi7:xi0[m]=parameter[3]
let @xi9:xi0[w]=parameter[4]
let @xi2:xi0[w]=@xi1:xi0[w]
let @xi3:xi0[m]=ceil(@xi7:xi0[m]/@xi5:xi0[m]/2*@xi6:xi0[m])
let @xi4:xi0[m]=@xi3:xi0[m]
let @xi9:xi0[l]=@xi9:xi0[w]
op
let pd=-i(vi6)*v(vdd)*1e3
let w1f=@xi1:xi0[w]*1e6
let l1f=@xi1:xi0[l]*1e6
let m1f=@xi1:xi0[m]
let w6f=@xi6:xi0[w]*1e6
let l6f=@xi6:xi0[l]*1e6
let m6f=@xi6:xi0[m]
let w3f=@xi3:xi0[w]*1e6
let l3f=@xi3:xi0[l]*1e6
let m3f=@xi3:xi0[m]
let w8f=@xi8:xi0[w]*1e6
let l8f=@xi8:xi0[l]*1e6
let m8f=@xi8:xi0[m]
let m5f=@xi5:xi0[m]
let m7f=@xi7:xi0[m]
let wccompf=@xi9:xi0[w]*1e6
let lccompf=@xi9:xi0[l]*1e6
let mccompf=@xi9:xi0[m]
let aream1=m1f*(w1f+12.5)*(l1f+11)
```

```
let aream6=m6f*(w6f+12.5)*(l6f+11)
let aream3=m3f*(w3f+12.5)*(l3f+11)
let aream8=m8f*(w8f+12.5)*(l8f+11)
let areaccomp=mccompf*(wccompf+6.25)*(lccompf+10.5)
let area=(2*aream1+aream6+2*aream3+(1+(m7f+m5f)/m8f)*aream8+areaccomp)*1e-6
ac dec 50 10 100e6
let gmag=20*log10(mag(v(vout)))
let gph=phase(v(vout))
let c=0
let gdc=gmag[c]
cursor c right gmag 0
let gbw=abs(frequency[%c])/1e6
let pm=180+gph[%c]
if pm ge 90
 pm=pm-360
end
setcirc ckt2
setplot op2
let @xi1:xi0[w]=w1f*1e-6
let @xi6:xi0[m]=m6f
let @xi5:xi0[m]=m5f
let @xi7:xi0[m]=m7f
let @xi9:xi0[w]=wccompf*1e-6
let @xi2:xi0[w]=@xi1:xi0[w]
let @xi3:xi0[m]=ceil(@xi7:xi0[m]/@xi5:xi0[m]/2*@xi6:xi0[m])
let @xi4:xi0[m]=@xi3:xi0[m]
let @xi9:xi0[l]=@xi9:xi0[w]
tran 1n 5u
let vout=v(vout)
let c=0
cursor c right vout 2.1
let t1=time[%c]
cursor c right vout 2.9
let t2=time[%c]
let srpos=0.8/(t2-t1)*1e-6
cursor c right time 3u
cursor c right vout 2.9
let t1=time[%c]
cursor c right vout 2.1
let t2=time[%c]
let srneg=0.8/(t2-t1)*1e-6

**** Comparative Table
echo " "
echo "********************************"
echo "Device  Units Before      After"
echo "********************************"
echo "M1 M2 [um/um] {round(op1.m1i)}x{round(op1.w1i*100)/100}/{round(op1.l1i*100)/100}
                    {round(op2.m1f)}x{round(op2.w1f*100)/100}/{round(op2.l1f*100)/100}"
echo "M3 M4 [um/um] {round(op1.m3i)}x{round(op1.w3i*100)/100}/{round(op1.l3i*100)/100}
                    {round(op2.m3f)}x{round(op2.w3f*100)/100}/{round(op2.l3f*100)/100}"
echo "M6    [um/um] {round(op1.m6i)}x{round(op1.w6i*100)/100}/{round(op1.l6i*100)/100}
                    {round(op2.m6f)}x{round(op2.w6f*100)/100}/{round(op2.l6f*100)/100}"
echo "M8    [um/um] 1x{round(op1.w8i*100)/100}/{round(op1.l8i*100)/100}
                    1x{round(op2.w8f*100)/100}/{round(op2.l8f*100)/100}"
echo "M5    [um/um] {round(op1.m5i)}x{round(op1.w8i*100)/100}/{round(op1.l8i*100)/100}
                    {round(op2.m5f)}x{round(op2.w8f*100)/100}/{round(op2.l8f*100)/100}"
echo "M7    [um/um] {round(op1.m7i)}x{round(op1.w8i*100)/100}/{round(op1.l8i*100)/100}
                    {round(op2.m7f)}x{round(op2.w8f*100)/100}/{round(op2.l8f*100)/100}"
echo "Ccomp [umxum] {round(op1.mccompi)}x{round(op1.wccompi*100)/100}x{round(op1.lccompi*100)/100}
```

```
                    {round(op2.mccompf)}x{round(op2.wccompf*100)/100}x{round(op2.lccompf*100)/100}"
echo "*******************************"
echo "Param Units Before  After"
echo "*******************************"
echo "G       [dB] {round(ac1.gdc*10)/10}    {round(ac2.gdc*10)/10}"
echo "GBW    [MHz] {round(ac1.gbw*100)/100}    {round(ac2.gbw*100)/100}"
echo "PM     [deg] {round(ac1.pm*10)/10}    {round(ac2.pm*10)/10}"
echo "SR+   [V/us] {round(tran1.srpos*10)/10}    {round(tran2.srpos*10)/10}"
echo "SR-   [V/us] {round(tran1.srneg*10)/10}    {round(tran2.srneg*10)/10}"
echo "PD      [mW] {round(op1.pd*100)/100}    {round(op2.pd*100)/100}"
echo "Area   [mm2] {round(op1.area*1000)/1000}    {round(op2.area*1000)/1000}"
echo "*******************************"
echo " "
setplot optimize1
plot create plot1 xi1:xi0_w*1e6 xi6:xi0_m xi5:xi0_m xi7:xi0_m xi9:xi0_w*1e6 vs iteration ylabel
          'm1[w],m6[m],m7[m],m5[m]  ccomp[w,l]' xlabel 'Iteration' title 'Parameter Evolution'
plot print plot1 file test_oa_optimize_a.pdf
set color2 = r255g000b000
set color3 = r255g000b000
set color4 = r000g255b000
set color5 = r000g255b000
plot create plot2 ac1.gmag ac1.gph vs ac1.frequency ac2.gmag ac2.gph vs ac2.frequency xlabel
     'Frequency [Hz]' ylabel 'Magnitude [dB]  Phase [deg]' title 'Bode Diagram' ylimit -180 100
plot print plot2 file test_oa_optimize_b.pdf
set color2 = r255g000b000
set color3 = r000g255b000
plot tran1.vout vs tran1.time tran2.vout vs tran2.time xlabel 'Time [s]'
plot create plot3 tran1.vout vs tran1.time tran2.vout vs tran2.time xlabel 'Time [s]' ylabel
                                                 'Output Voltage [V]' title 'Step Response'
plot print plot3 file test_oa_optimize_c.pdf
.endc
.end
```

apdk/spiceopus/test_oa_optimize.sp3

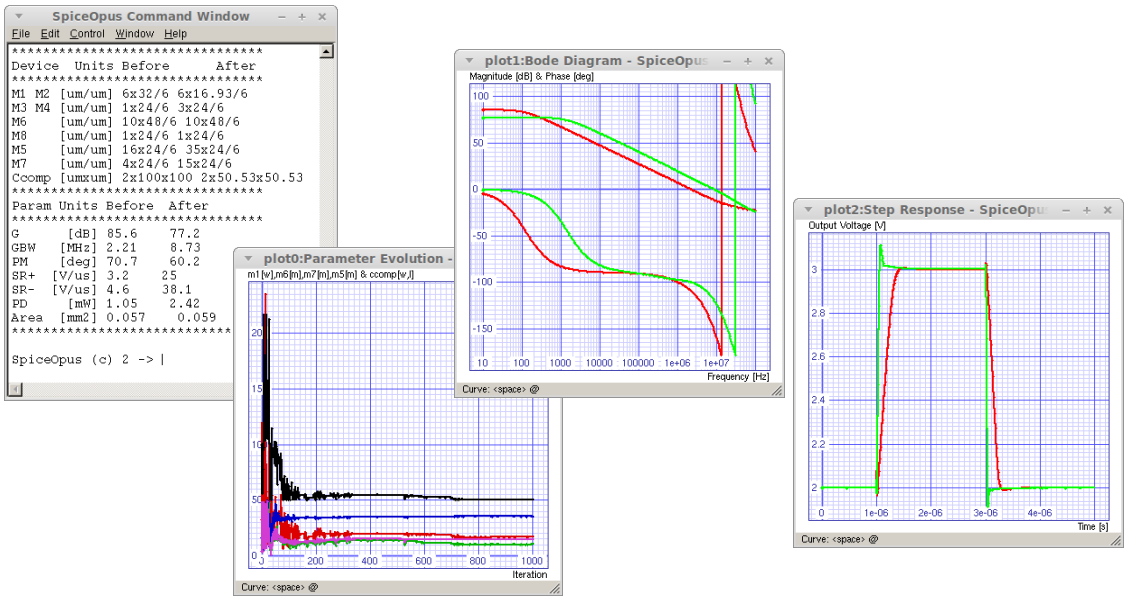


Figure 25 Example of SpiceOpus results obtained from the optimization script example appdk/spiceopus/test_oa_optimize.sp3 after having declared a cost function equal to 1/abs(tran2.srneg)+1/abs(tran2.srpos) and selected the complex method.

3.6 PCell-Based Schematic-Driven Layout Design

Once the optimal device sizing of all the circuit blocks has been obtained, the physical CMOS design of the IC can start according to the methodology of Fig. 1. Unlike in semi-custom digital circuits, where compactness and speed are the major design targets, the main goals in full-custom analog and mixed-signal 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. Regarding device matching, the layout guidelines of Table 5 are strongly recommended.

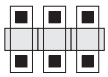
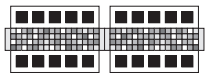
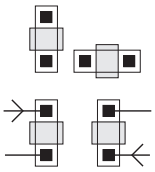
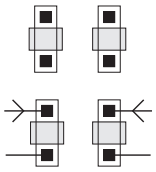
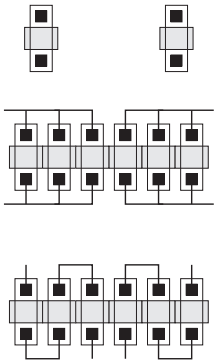
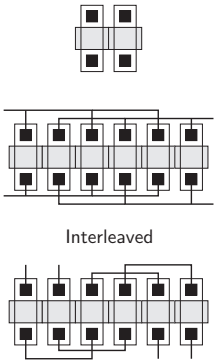
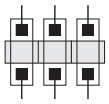
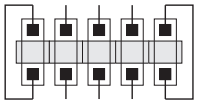
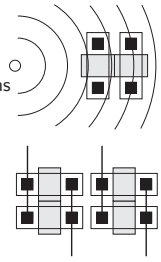
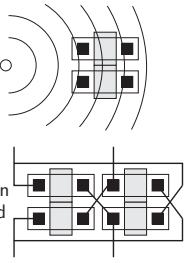
Layout Rule	Bad	Good
Unitary Elements		
Large Area	 Process Resolution	
Same Orientation		
Minimum Distance $(W/L)>>1$ $(W/L)<<1$		 Interleaved
Same Sorround		 Dummy Dummy
Same Symmetry	 Iso Therms	 Common Centroid

Table 5 | Analog layout style guide for best device matching [13].

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. 26. 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. 26, this operation mode displays the design rules on the fly, during object edition.

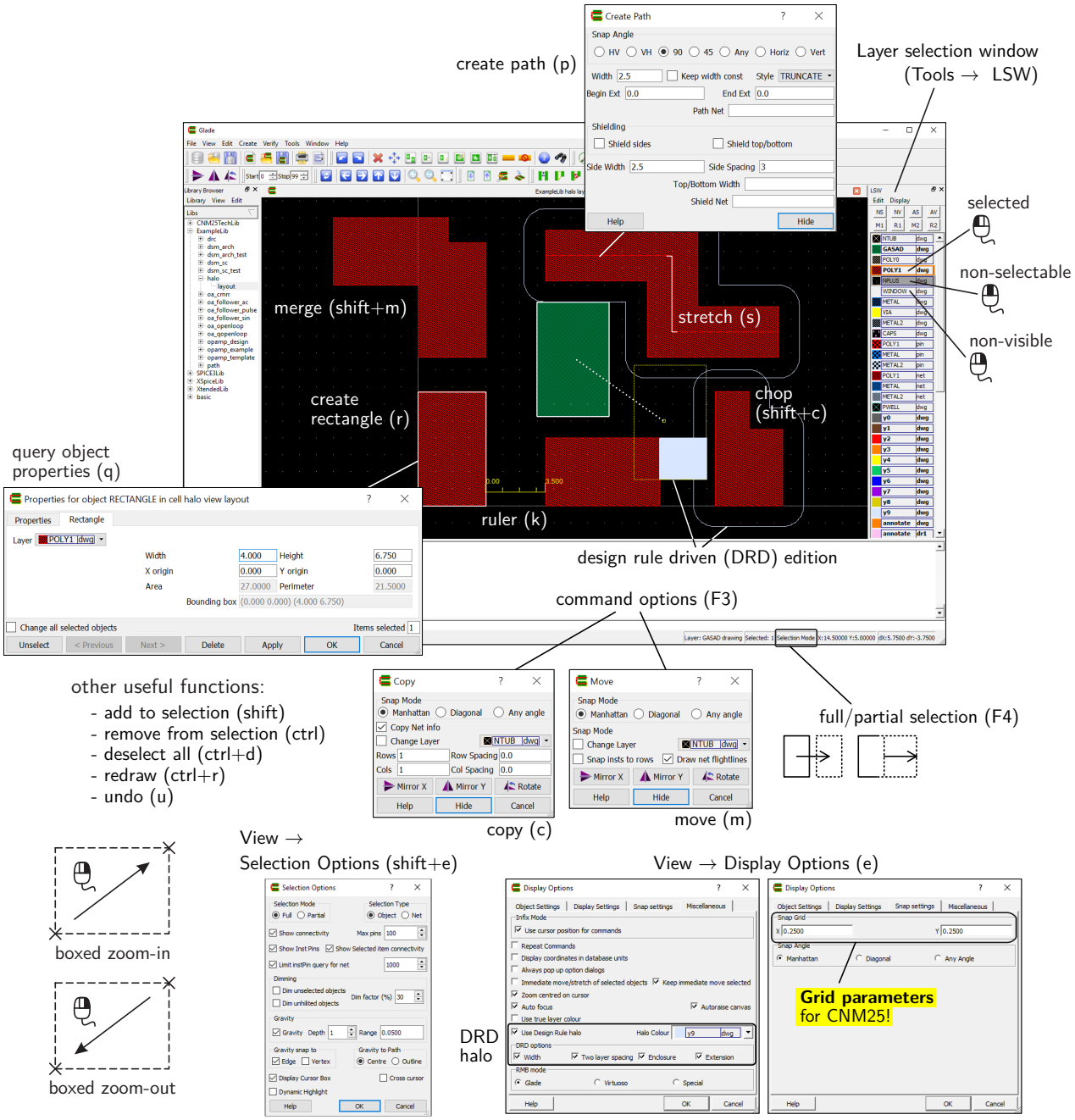


Figure 26 | Glade layout editor main window and associated tools.

In order to speed up the full-custom edition of the IC layout, this APDK supports the use of parameterized cells (PCells) 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. 27. In Glade, PCells can be programmed for each CMOS technology using Python language, like the code example shown below.

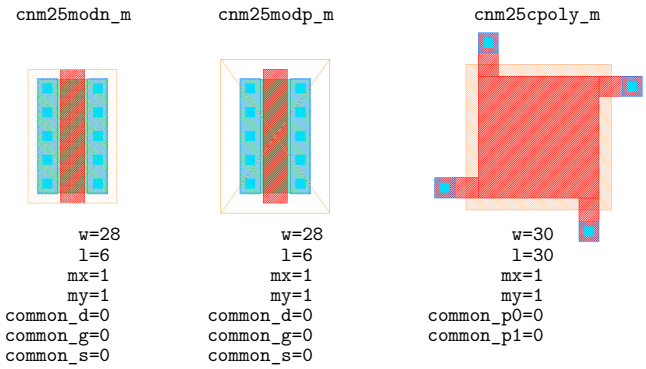


Figure 27 | Glade layout PCells available in the CNM25 APDK and sizing parameters. PCell devices can be called through Create→Instance (i).

apdk/glade/pcells/cnm25modn_m.py

```
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(int(-(cont_space_poly + cont_size + gasad_ov_cont)), 0,
                int(length + cont_space_poly + cont_size + gasad_ov_cont), int(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, int(-poly_ext_one_side), int(length), int(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(int(-cont_space_poly - cont_size), int(gasad_ov_cont + n *
                    (cont_size + s_cont)), int(-cont_space_poly), int(gasad_ov_cont +
                    cont_size + n * (cont_size + s_cont)))
                r.offset(xoffset, yoffset)
                contact = cv.dbCreateRect(r, layer)
                r = Rect(int(length + cont_space_poly), int(gasad_ov_cont + n * (cont_size +
                    s_cont)), int(length + cont_space_poly + cont_size), int(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(int(-(cont_space_poly + cont_size + metal_ov_cont)), int(-metal_ext_one_side),
                int(-cont_space_poly + metal_ov_cont), int(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(int(length + cont_space_poly - metal_ov_cont), int(-metal_ext_one_side), int(
             length + cont_space_poly + cont_size + metal_ov_cont), int(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(int(-nplus_x_ext), int(-nplus_ov_gasad), int(length + nplus_x_ext + xoffset
        - dxoffset_alt[x%2!=0]), int(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(int(-ntub_x_ext), int(-ntub_ov_gasad), int(length + ntub_x_ext + xoffset
        - dxoffset_alt[x%2!=0]), int(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()
```

apdk/glade/pcells/cnm25modn_m.py

Examples of usage for CNM25 PCells can be found in Figs. 28 and 29. 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.

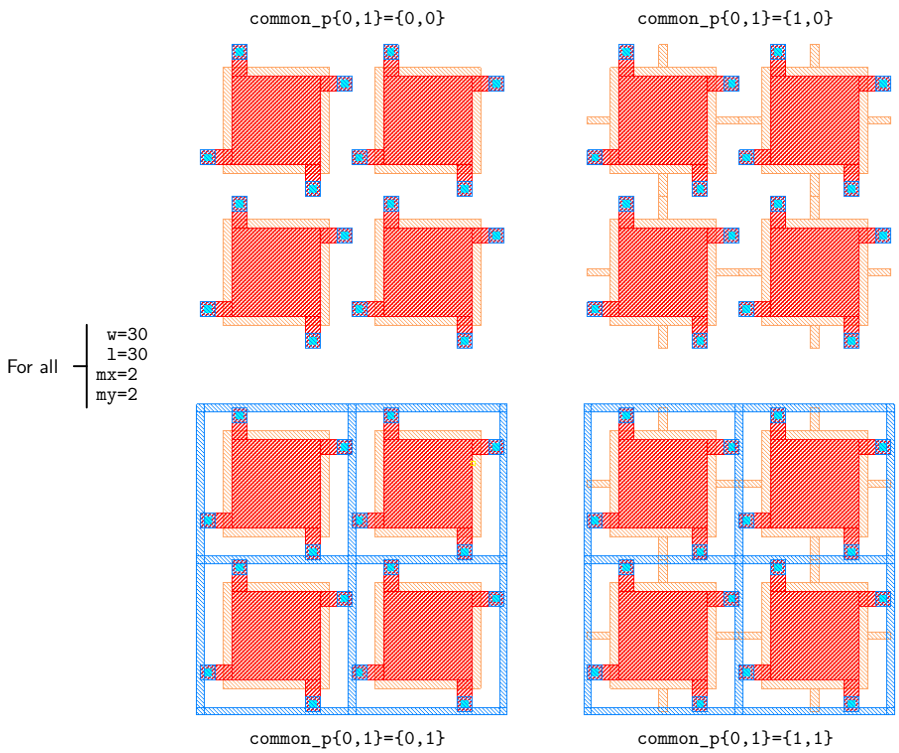


Figure 28 Example of automatic PiP capacitor generation using CNM25 PCell `cnm25cpoly_m`. Parameters for this PCell are: element width (`w`) and length (`l`), number of vertical (`mx`) and horizontal (`my`) elements, and conditional options for common bottom (`common_p0`) and/or top (`common_p1`) plates.

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. 30.

Finally, the Glade multi-part path (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. 31, 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. 32. 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.

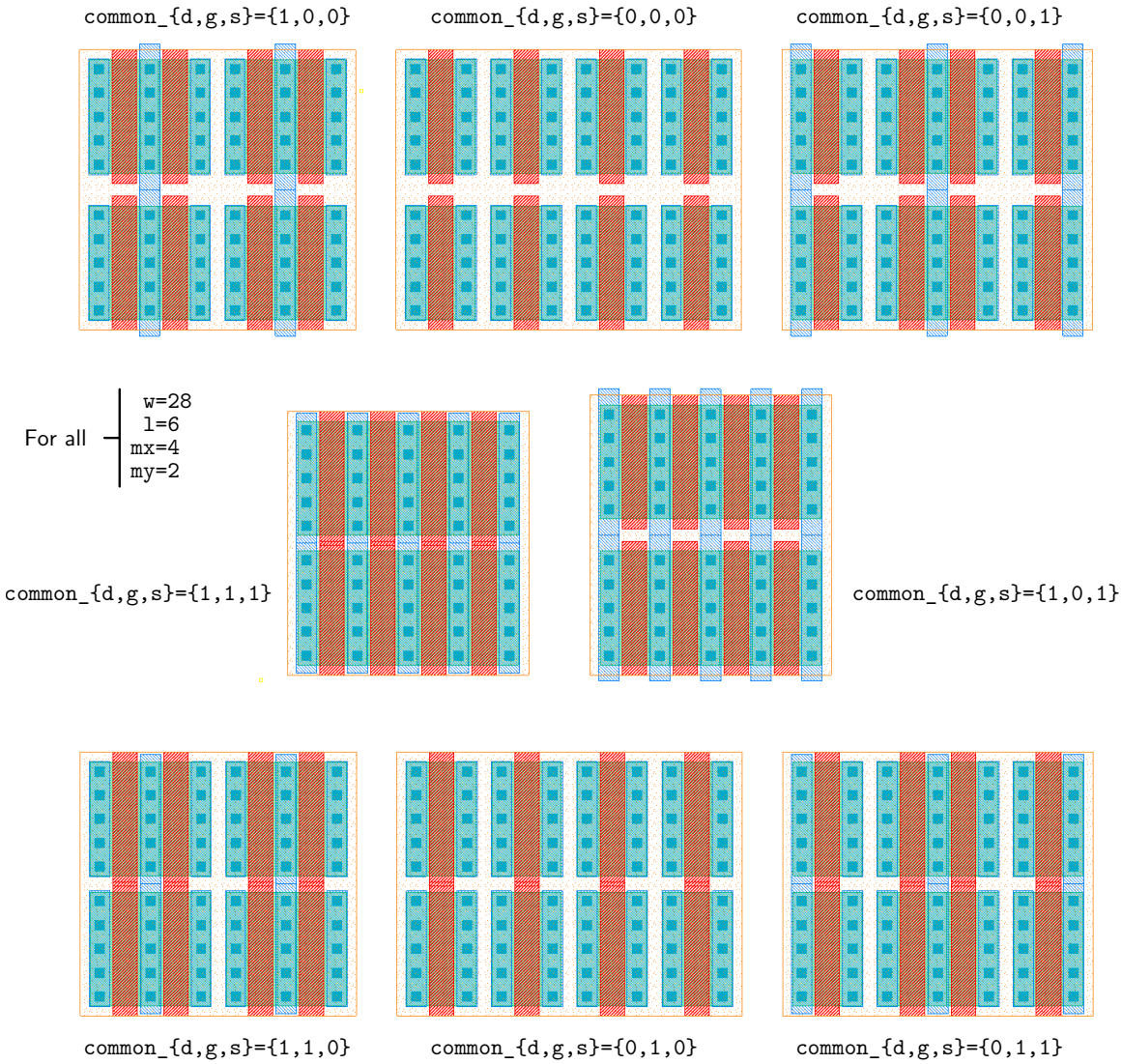


Figure 29 Example of automatic NMOS device generation using CNM25 PCell `cnm25modn_m`. Parameters for this PCell are: element width (`w`) and length (`l`), vertical (`mx`) and horizontal (`my`) multiplicity, and conditional options for common drain (`common_d`), gate (`common_g`), and/or source (`common_s`).

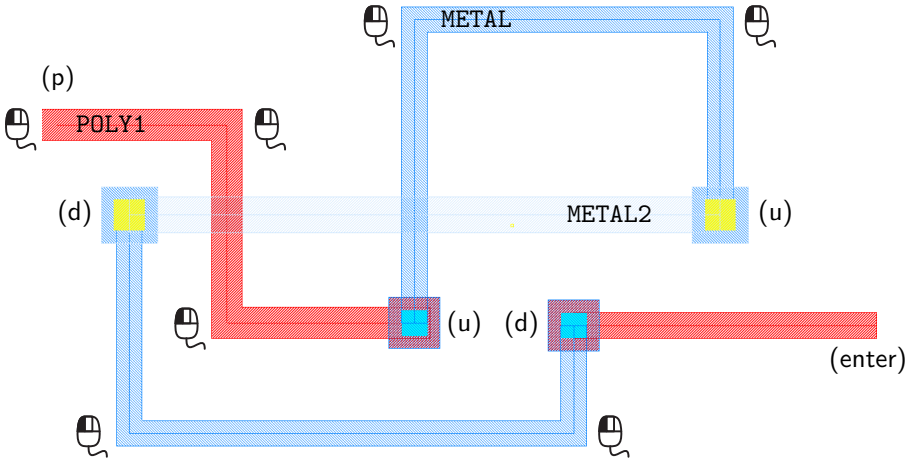
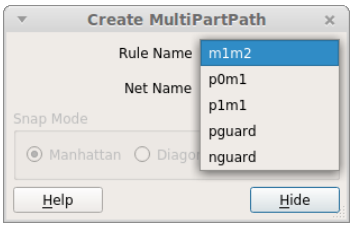


Figure 30 | Example of Glade Create→Path (p) command usage for CNM25 routing.




```
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)
```

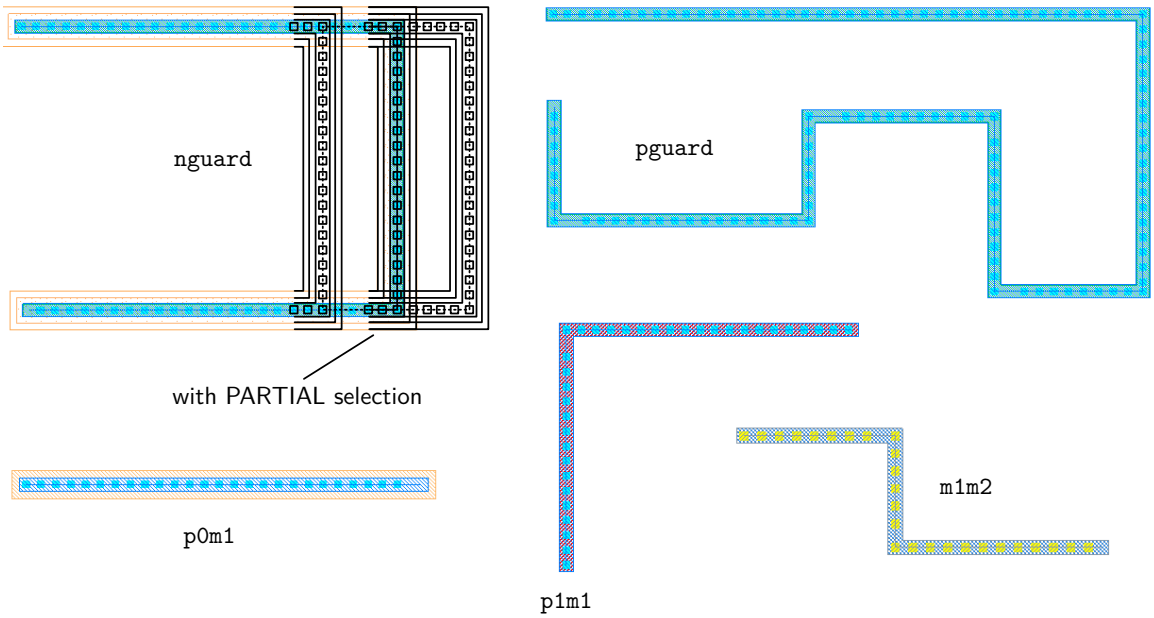


Figure 31 | CNM25 automatic and stretchable MPP structures available through the Glade Create→MultiPartPath command.

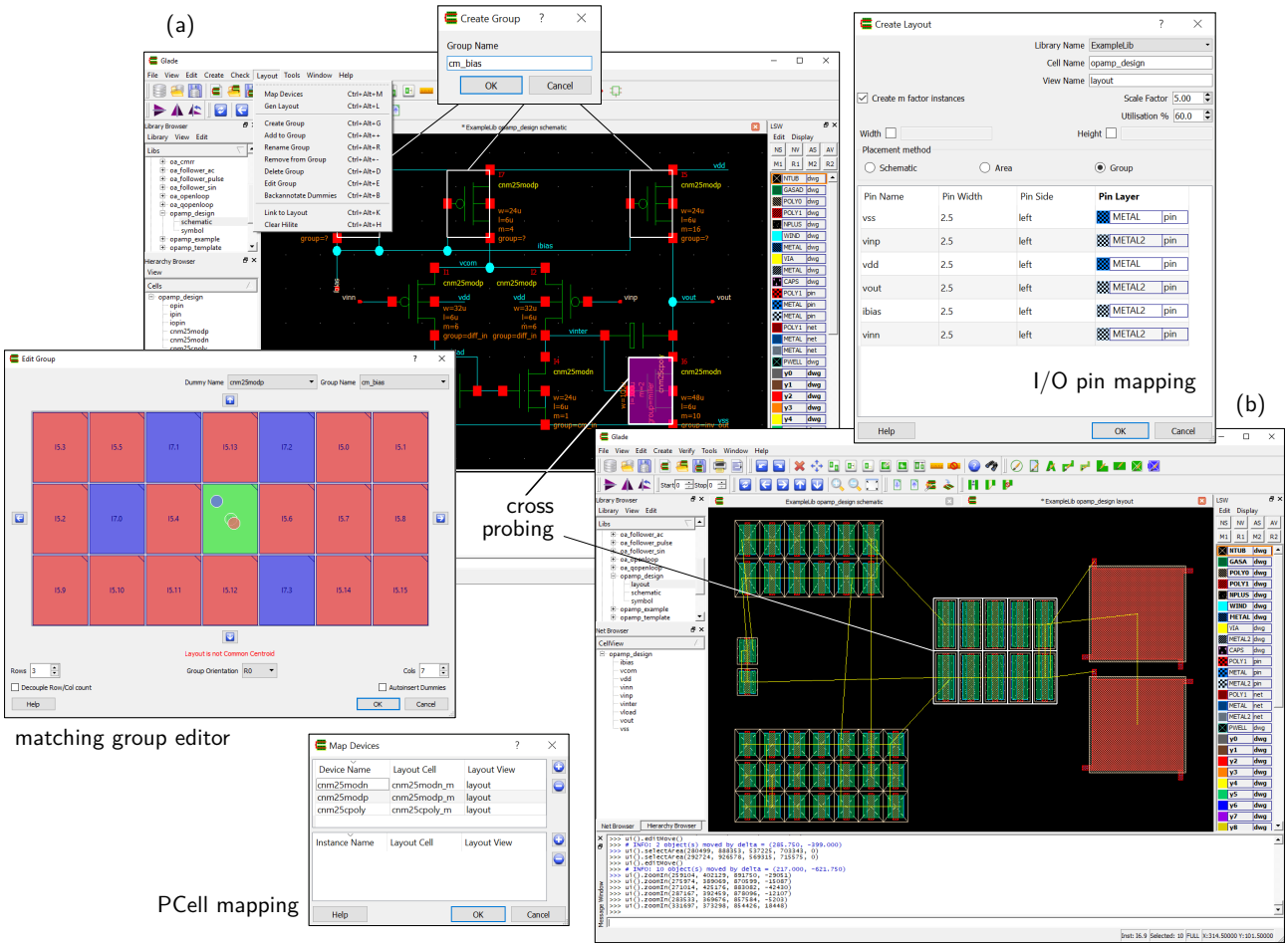


Figure 32 | Usage of Glade layout generation tool for managing matching groups (a) and for automated PCell placement (b) with the OpAmp example of Fig. 22.

☞ Update device dimensions of `ExampleLib`→`opamp_design`→`schematic` according to the **optimized** values returned by the NUTMEG script `test_oa_optimze.sp3`!

☞ Use the Glade layout generation tool of Fig. 32 to define the **matching groups** at schematic level and to **place** all PCell devices of your OpAmp at layout level.

☞ Check grid and snap parameters are set to **X=0.25** and **Y=0.25**, as in Fig. 26.

Q17. Design the **full-custom layout** of your optimized OpAmp:

- Exploit the advantage of **paths** and **MPPs** as in Figs. 30 and 31.
- Follow the matching design **guidelines** of Table 5.
- Ensure layout compatibility with the cell **template** of Fig. 33.

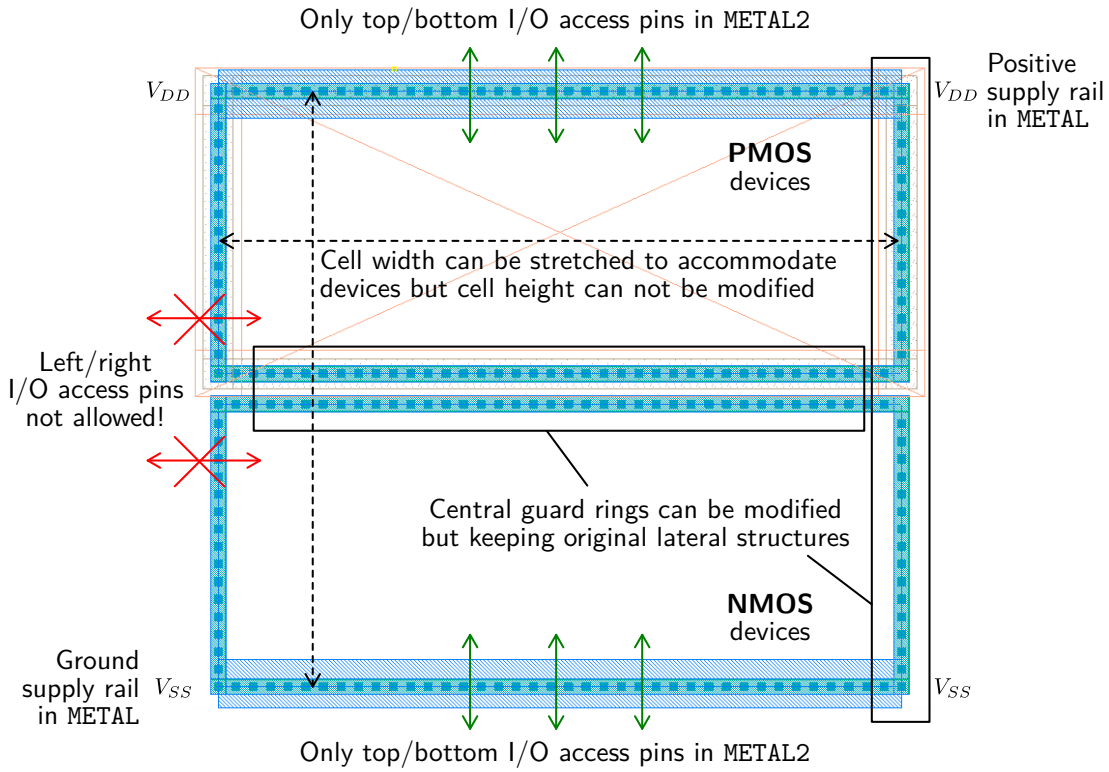


Figure 33 Glade cell view `ExampleLib`→`opamp_template`→`layout` to be used for the design boundaries of your OpAmp layout. Standard-cell height is 200 μm.

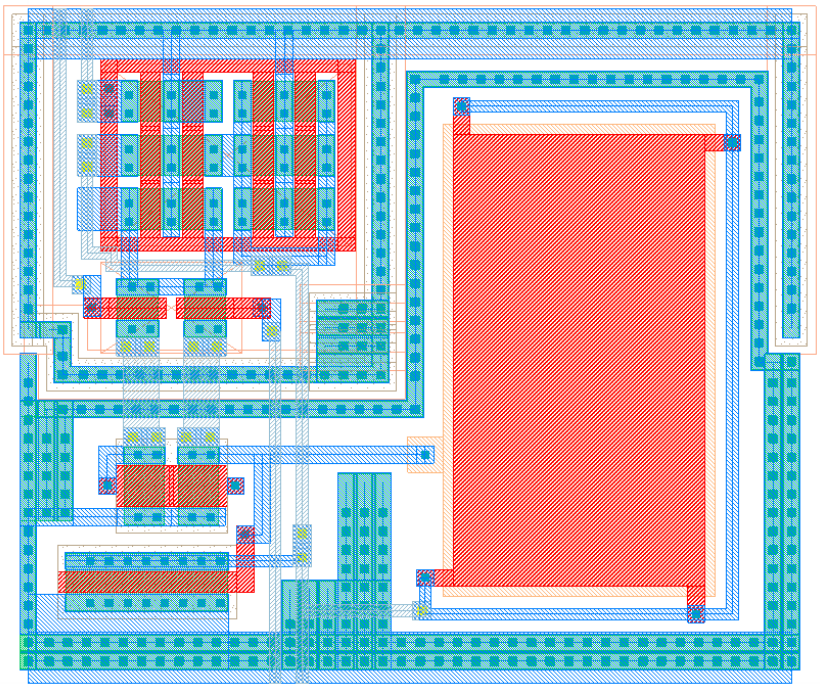


Figure 34 Glade cell view `ExampleLib`→`opamp_example`→`layout` to illustrate the use of the cell template of Fig. 33 and the matching guidelines of Table 5. Overall cell width is 226 μm.

3.7 Design Rule Checker

When 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 fabricability 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 APDK is already shipped with Python script `apdk/glade/verification/cnm25drc.py`, 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. 35.

apdk/glade/verification/cnm25drc.py

```
# CNM25 2M DRC deck

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

# Checking off-grid...
print("0.0. Checking off-grid...")
geomOffGrid(nwell, 0.25, 1.0, "Error: N-well grid not multiple of 0.25um x 0.25um")
geomOffGrid(active, 0.25, 1.0, "Error: GASAD grid not multiple of 0.25um x 0.25um")
geomOffGrid(polygate, 0.25, 1.0, "Error: Poly1 grid not multiple of 0.25um x 0.25um")
geomOffGrid(polycap, 0.25, 1.0, "Error: Poly0 grid not multiple of 0.25um x 0.25um")
geomOffGrid(nimp, 0.25, 1.0, "Error: N-plus grid not multiple of 0.25um x 0.25um")
geomOffGrid(cont, 0.25, 1.0, "Error: Contact grid not multiple of 0.25um x 0.25um")
geomOffGrid(metal1, 0.25, 1.0, "Error: Metal1 grid not multiple of 0.25um x 0.25um")
geomOffGrid(via12, 0.25, 1.0, "Error: Via grid not multiple of 0.25um x 0.25um")
geomOffGrid(metal2, 0.25, 1.0, "Error: Metal2 grid not multiple of 0.25um x 0.25um")
geomOffGrid(pad, 0.25, 1.0, "Error: Pad grid not multiple of 0.25um x 0.25um")

# Checking N-well...
print("1.X. Checking N-well...")
geomWidth(nwell, 8.0, "Error: N-well width < 8um (see rule 1.1)")
geomSpace(nwell, 8.0, "Error: N-well spacing < 8um (see rule 1.2)")
geomNotch(nwell, 8.0, "Error: N-well notch < 8um (see rule 1.2)"

# Checking GASAD...
print("2.X. Checking GASAD...")
geomWidth(active, 2.0, "Error: GASAD width < 2um (see rule 2.1)")
geomSpace(active, 4.0, "Error: GASAD spacing < 4um (see rule 2.2)")
geomNotch(active, 4.0, "Error: GASAD notch < 4um (see rule 2.2)")
geomEnclose(nwell, pdiff, 5.0, "Error: N-well enclosure of P-plus active < 5um (see rule ...
geomSpace(nwell, ndiff, 5.0, "Error: N-well spacing to N-plus active < 5um (see rule 2.4)")

# Checking Poly0...
print("3.X. Checking Poly0...")
geomWidth(polycap, 2.5, "Error: Poly0 width < 2.5um (see rule 3.1)")
geomSpace(polycap, 6.0, "Error: Poly0 spacing < 6um (see rule 3.2)")
geomNotch(polycap, 6.0, "Error: Poly0 notch < 6um (see rule 3.2)")
geomSpace(polycap, active, 6.0, "Error: Poly0 spacing to GASAD < 6um (see rule 3.3)")

# Checking Poly1...
print("4.X. Checking Poly1...")
geomWidth(gate, 3.0, "Error: Poly1 width inside GASAD < 3um (see rule 4.1.a)")
geomWidth(geomAndNot(polygate, gate), 2.5, "Error: Poly1 width outside GASAD < 2.5um (see rule ...
geomSpace(polygate, 3.0, "Error: Poly1 spacing < 3um (see rule 4.2)")
geomNotch(polygate, 3.0, "Error: Poly1 notch < 3um (see rule 4.2)")
geomExtension(active, polygate, 3.0 , "Error: GASAD extension of Poly1 < 3um (see rule 4.3)")
geomExtension(polygate, active, 2.5, "Error: Poly1 extension of GASAD < 2.5um (see rule 4.4)")
geomSpace(polygate, active, 1.25, "Error: Poly1 spacing to GASAD < 1.25um (see rule 4.5)")
geomEnclose(polycap, polygate, 3.0, "Error: Poly0 enclosure of Poly1 < 3um (see rule 4.6)"

# Checking N-plus...
print("5.X. Checking N-plus...")
geomEnclose(nimp, active, 2.5, "Error: N-plus enclosure of GASAD < 2.5um (see rule 5.1)")
geomSpace(nimp, pdiff, 2.5, "Error: N-plus spacing to P-plus active < 2.5um (see rule 5.2)")
geomSpace(nimp, pgate, 2.0, "Error: N-plus spacing to Poly1 inside P-plus active < 2um (see ...
geomExtension(nimp, ngate, 1.5, "Error: N-plus extension of Poly1 inside N-plus active < 1.5um ...
geomWidth(nimp, 2.5, "Error: N-plus width < 2.5um (see rule 5.5)")
geomSpace(nimp, 2.5, "Error: N-plus spacing < 2.5um (see rule 5.6)")
geomNotch(nimp, 2.5, "Error: N-plus notch < 2.5um (see rule 5.6)")

# Checking Contact...
print("6.X. Checking Contact...")
```

```
saveDerived(badcon, "Error: Contact without Metal1")
geomWidth(cont, 2.5, not_equal, "Error: Contact size not equal to 2.5um x 2.5um (see rule 6.1)")
geomSpace(cont, 3.0, "Error: Contact spacing < 3um (see rule 6.2)")
geomNotch(cont, 3.0, "Error: Contact notch < 3um (see rule 6.2)")
geomEnclose(active, cont, 1.0, "Error: GASAD enclosure of Contact < 1um (see rule 6.3)")
geomEnclose(polygate, cont, 1.25, "Error: Poly1 enclosure of Contact < 1.25um (see rule 6.4)")
geomSpace(polygatecont, 2.5, "Error: Poly1 Contact spacing to GASAD < 2.5um (see rule 6.5)")
geomSpace(cont, gate, 2.0, "Error: Contact spacing to Poly1 inside GASAD < 2um (see rule 6.6)")
# 6.7 and 6.8 not implemented!
geomEnclose(polycap, cont, 4.0, "Error: Poly0 enclosure of Contact < 4um (see rule 6.9)")
geomSpace(cont, cpoly, 4.0, "Error: Contact spacing to Poly1 & Poly0 < 4um (see rule 6.10)")

# Checking Metal1...
print("7.X. Checking Metal1...")
geomWidth(metal1, 2.5, "Error: Metal1 width < 2.5um (see rule 7.1)")
geomSpace(metal1, 3.0, "Error: Metal1 spacing < 3um (see rule 7.2)")
geomNotch(metal1, 3.0, "Error: Metal1 notch < 3um (see rule 7.2)")
geomEnclose(metal1, cont, 1.25, "Error: Metal1 enclosure of Contact < 1.25um (see rule 7.3)")

# Checking Via...
print("8.X. Checking Via...")
saveDerived(badvia, "Error: Via without Metal2")
geomWidth(via12, 3.0, not_equal, "Error: Via size not equal to 3um x 3um (see rule 8.1)")
geomSpace(via12, 3.5, "Error: Via spacing < 3.5um (see rule 8.2)")
geomNotch(via12, 3.5, "Error: Via notch < 3.5um (see rule 8.2)")
geomEnclose(metal1, via12, 1.25, "Error: Metal1 enclosure of Via < 1.25um (see rule 8.3)")
geomSpace(via12, cont, 2.5, "Error: Via spacing to contact < 2.5um (see rule 8.4)")
geomSpace(via12, polygate, 2.5, "Error: Via spacing to Poly1 < 2.5um (see rule 8.5)"

# Checking Metal2...
print("9.X. Checking Metal2...")
geomWidth(metal2, 3.5, "Error: Metal2 width < 3.5um (see rule 9.1)")
geomSpace(metal2, 3.5, "Error: Metal2 spacing < 3.5um (see rule 9.2)")
geomNotch(metal2, 3.5, "Error: Metal2 notch < 3.5um (see rule 9.2)")
geomEnclose(metal2, via12, 1.25, "Error: Metal2 enclosure of Via < 1.25um (see rule 9.3)")

# Checking Pad...
print("10.X. Checking Pad...")
geomWidth(pad, 100.0, not_equal, "Error: Pad size not equal to 100um x 100um (see rule 10.1)")

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

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

apdk/glade/verification/cnm25drc.py

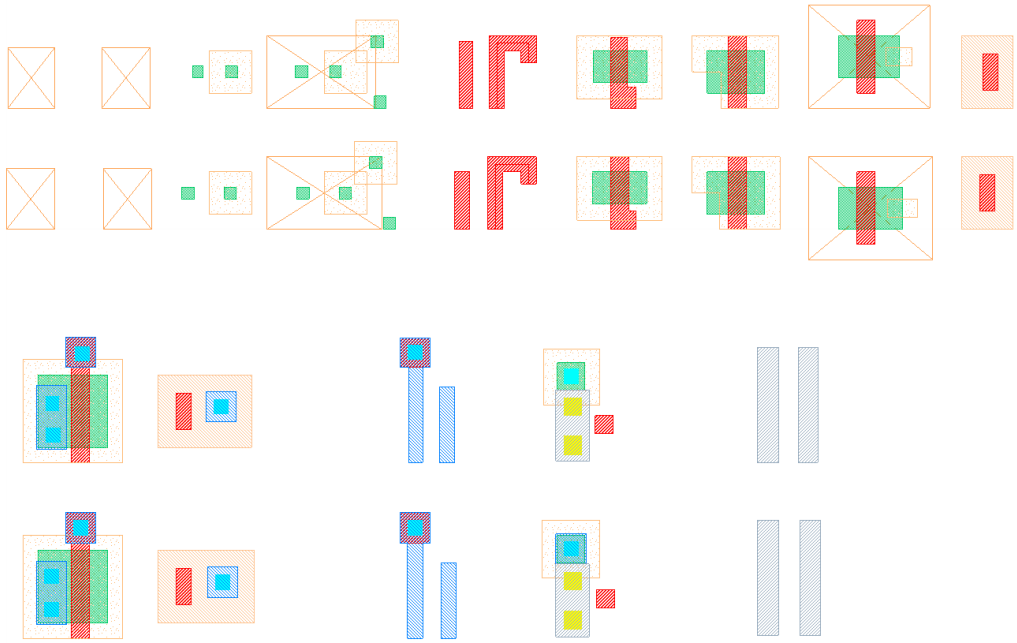


Figure 35 | Glade cell view `ExampleLib→drc→layout` includes examples of CNM25 layout structures with (top) and without (bottom) DRC errors.

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 `apdk/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. 36.
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`.

Q18. 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?

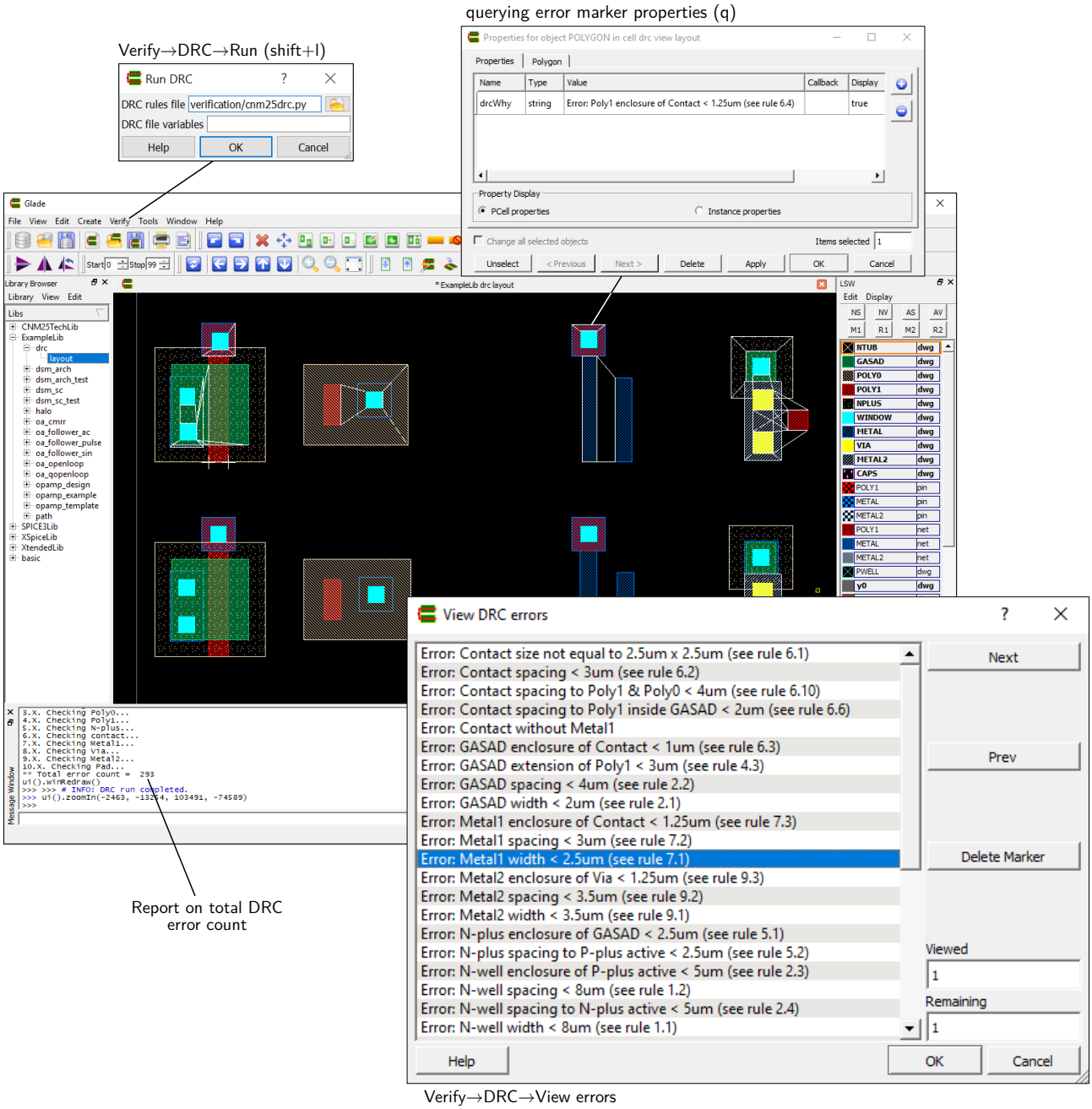


Figure 36 | Example of Glade DRC error browsing for `ExampleLib`→`drc`→`layout`.

3.8 Layout 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. 37:

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. 22.
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 the verification step of Section 3.9. Again, Fig. 37 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. 22.
4. Iterate above for each internal net of the OpAmp (i.e. `vcomm`, `vinter` and `vload`).

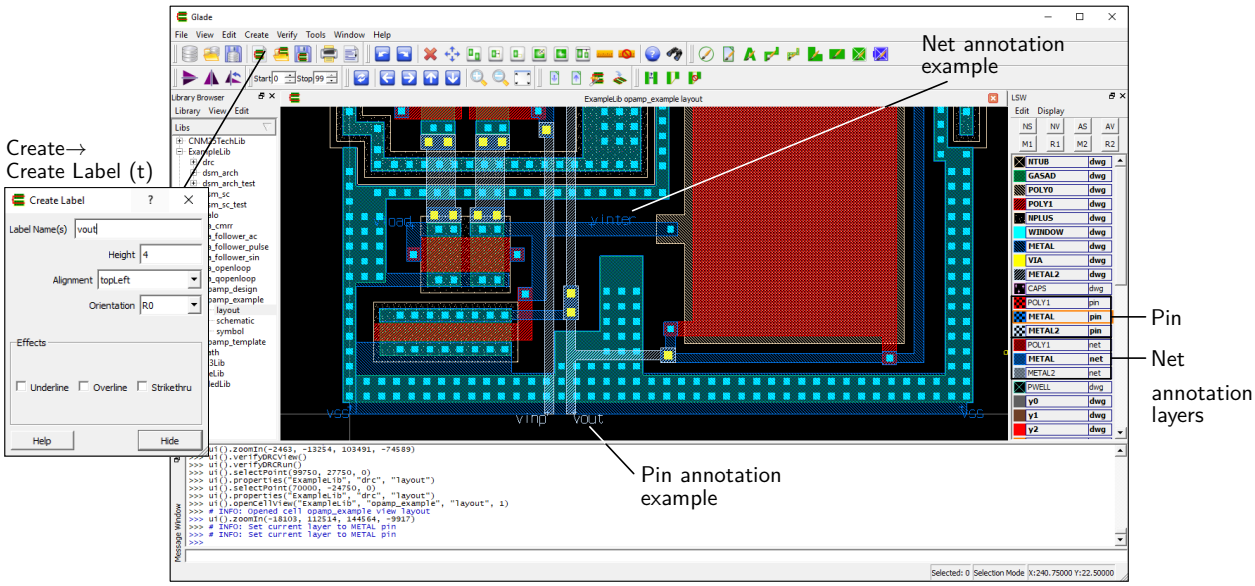


Figure 37 | Glade cell view `ExampleLib→opamp_example→layout` with pin and net annotation examples.

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 APDK comes with an specific Python code to cover such functionalities, which is called `apdk/glade/verification/cnm25lvs.py` and is shown below.

apdk/glade/verification/cnm25lvs.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", True)
geomLabel(metal1, "METAL", "pin", True)
geomLabel(metal2, "METAL2", "pin", True)
geomLabel(polygate, "POLY1", "net", False)
geomLabel(metal1, "METAL", "net", False)
geomLabel(metal2, "METAL2", "net", False)

# 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("# Extracting NMOS transistors...")
        extractMOS("cnm25modn", ngate, polygate, ndiff, pwell)

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

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

print "# End of circuit extraction"
geomEnd()

# Reporting results
cv_ex = libxtrlvs.dbFindCellViewByName(cv.cellName(), "extracted")
box = cv_ex.bBox()
objs = cv_ex.dbGetOverlaps(box, 0, True, True, True)
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)

```

apdk/glade/verification/cnm25lvs.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. 3. These Python PCells are in charge of extracting device sizing properties and annotating device terminal locations, like in `apdk/glade/pcells/cnm25cpoly.py`.

apdk/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()
```

apdk/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. 38 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 `apdk/glade/verification/cnm25lvs.py` and launch the extraction process.
4. Inspect for any noticeable ERC error, as depicted in Fig. 38.
5. If present, correct ERC errors in the layout and iterate Section 3.7 and 3.8.

Q19. Execute the above verification procedure in your optimized OpAmp layout to obtain an **ERC error-free extracted view**. Did you find any connectivity error?

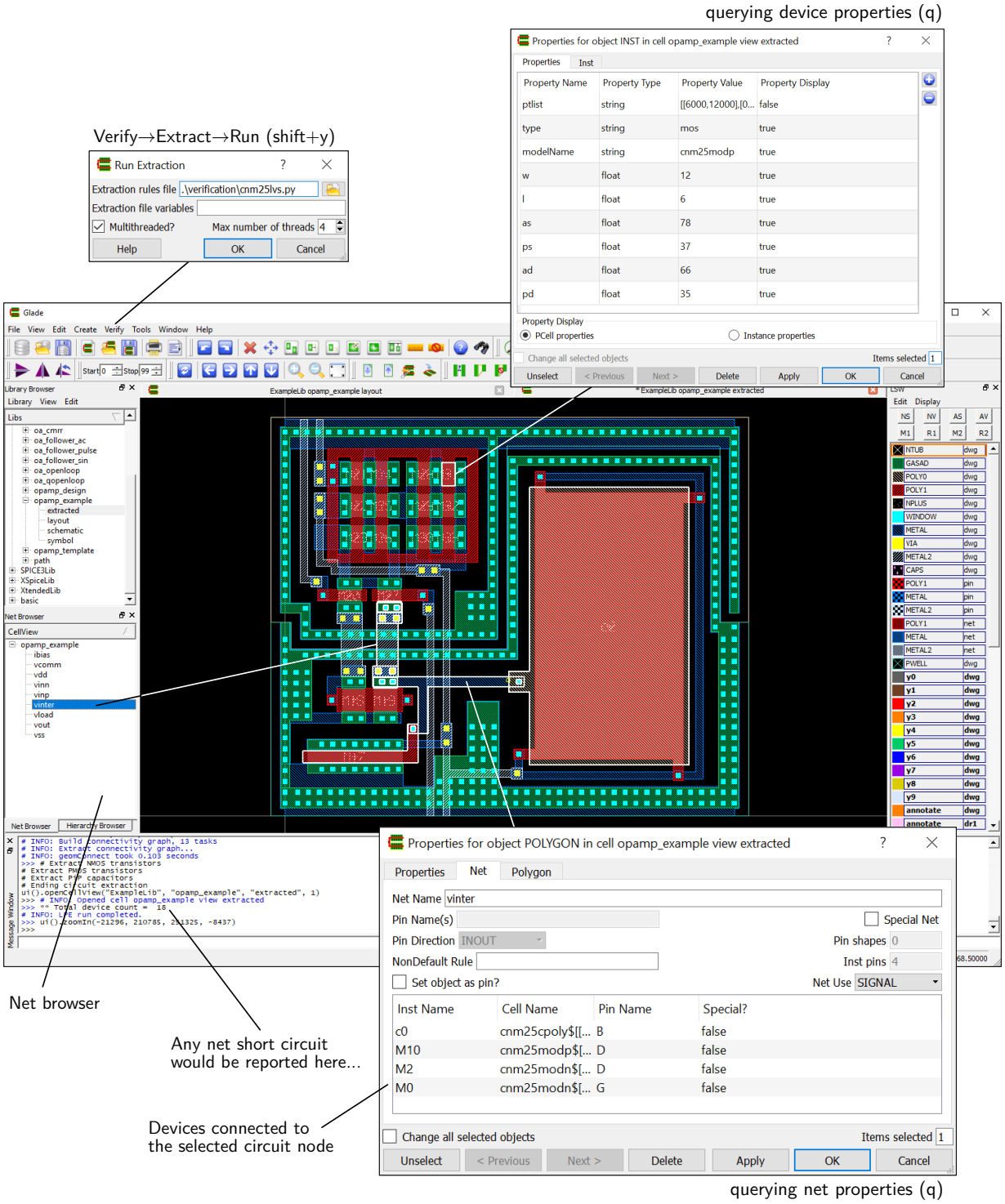


Figure 38 | Glade ERC inspection for `ExampleLib`→`opamp_example`→`extracted`.

3.9 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 3.8 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 each specific device, like in the example of Fig. 39.

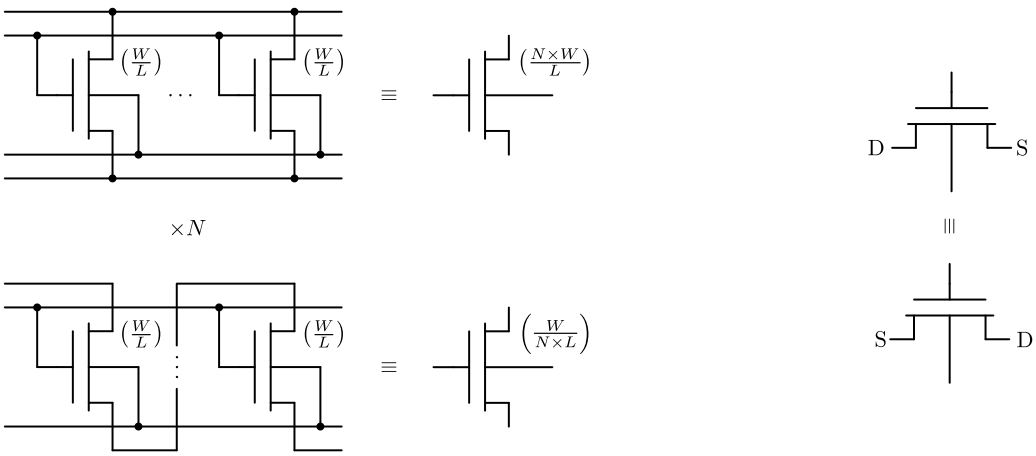


Figure 39 | Typical reduction (left) and permutation (right) LVS rules for the case of MOS transistors.

In our context, Glade relies on the venerable tool Gemini[1] [14] 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 5 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 at `www.cs.washington.edu/research/gemini-netlist-comparison-project`.

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. 40 and run the LVS process.
5. Once completed, review the results in the message window and in the extracted view.

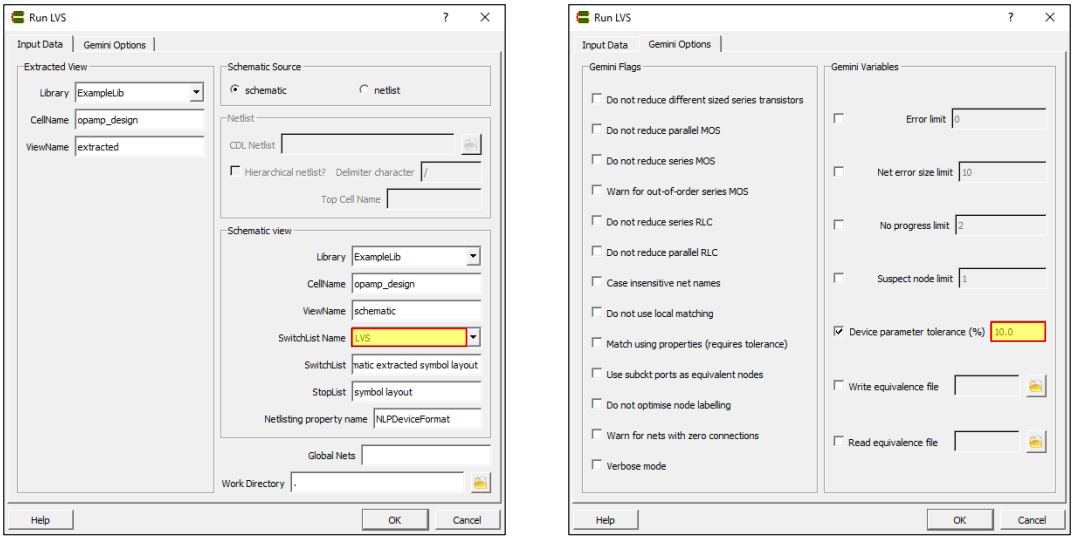


Figure 40 | Gemini configuration for the LVS verification of your `ExampleLib`→`opamp_design`→`extracted` view.

For illustration purposes, the APDK comes with the schematic of Fig. 41, which is the equivalent circuit of the OpAmp layout example used in the previous section and presented in Fig. 34. In this case, one PMOS dummy device needs to be added to match the physical transistor array of current mirrors.

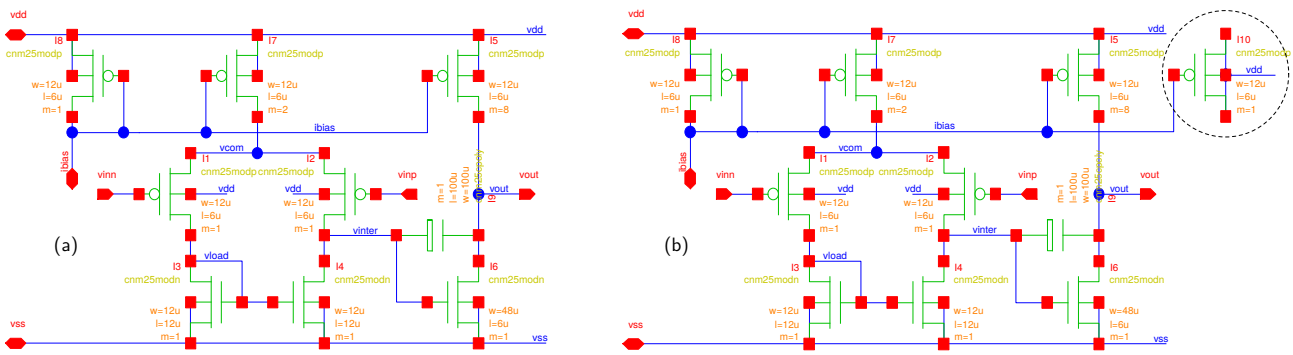


Figure 41 | Glade `ExampleLib`→`opamp_example`→`schematic` view of the OpAmp layout example used in Fig. 38 before (a) and after (b) including the corresponding layout dummy devices.

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

apdk/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
```

apdk/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:

apdk/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. 41(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. 42.

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
```

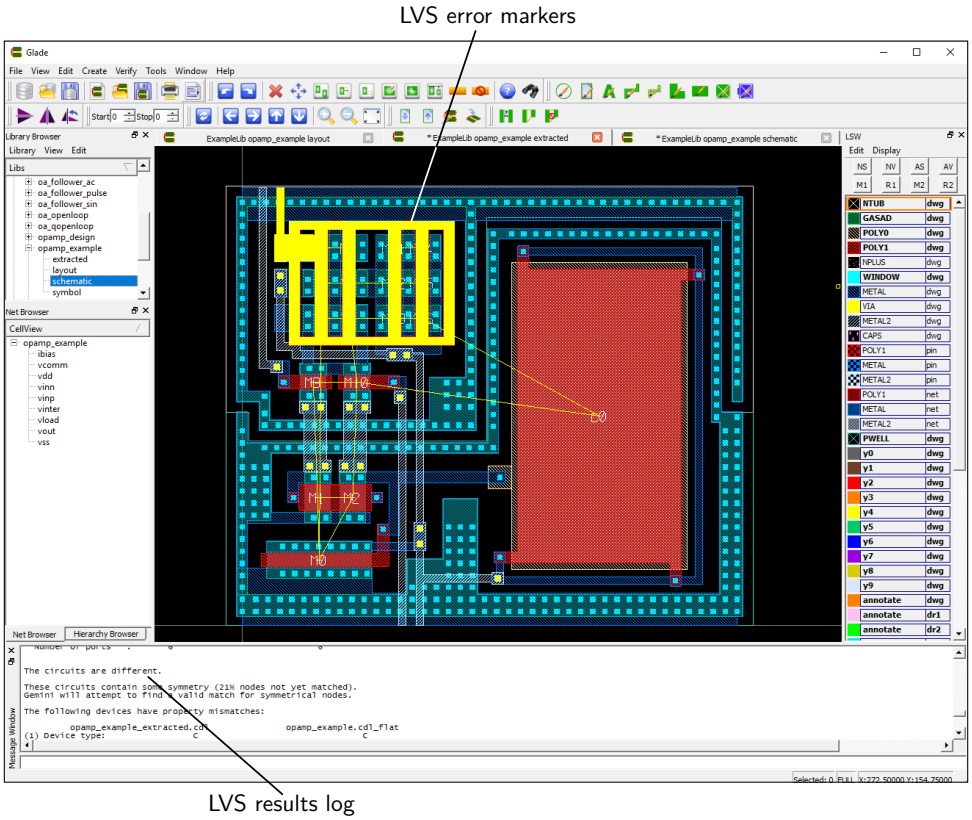


Figure 42 | Glade LVS results for cell view `ExampleLib`→`opamp_example`→`extracted` before updating the schematic view with layout dummy devices.

On the contrary, if the same LVS verification process is repeated **after** introducing the equivalent layout dummy devices into the schematic view of Fig. 41(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. 39. 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 also performs a comparative audit of device properties according to the 10% tolerance specified in Fig. 40. 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.

Q20. Execute the above **LVS verification** to your optimized OpAmp layout, review any error from Gemini output and **correct the layout** accordingly.

3.10 2D and 3D Parasitic 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 parasitic extraction (PEX), 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. 43.

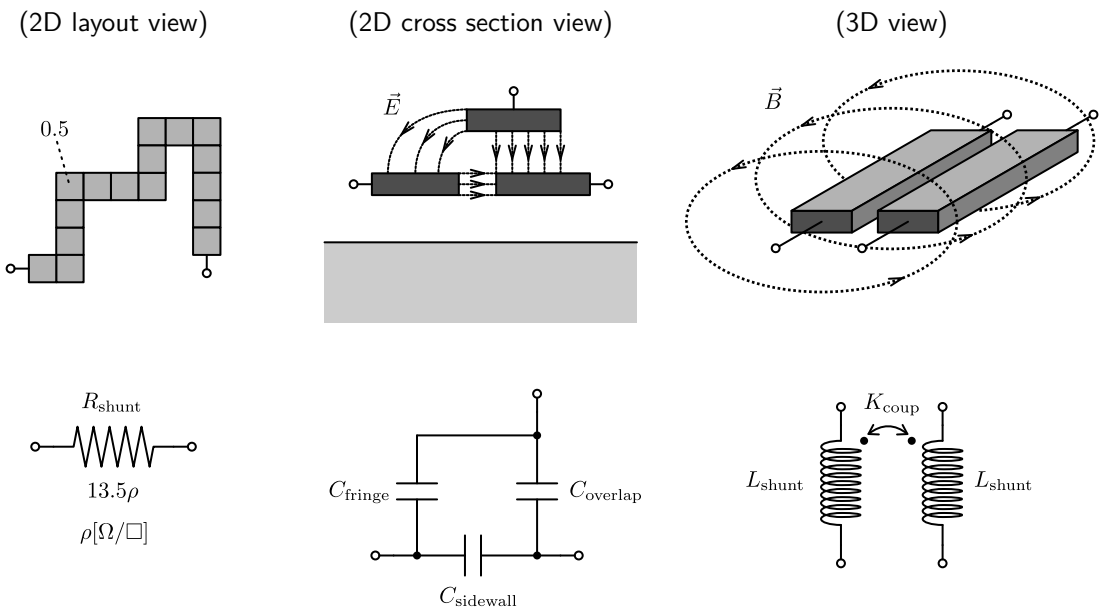


Figure 43 | Typical RLC circuit interconnectivity parasitics present in CMOS planar technologies.

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

$$C_{\text{par}} = A\frac{\epsilon_{\text{o}}\epsilon_{\text{ox}}}{t_{\text{ox}}} \tag{6}$$

where A stands for the parallel-plate area, t_{ox} and ϵ_{ox} are the insulator thickness and its relative permittivity ($\sim$3.9 for SiO_2), and ϵ_o is the well-known vacuum permittivity (about 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 can be directly extracted from the specific layout pattern, the APDK includes also numerical data about the fixed vertical spacing between routing layers. For the CNM25 case, and under the assumption of ideal CMOS process planarization and thin metal layers, insulator thickness values are depicted in Fig. 44. Basically, the field oxide is 1060-nm thick, while the inter-metal oxide insulator height is around 1300 nm.

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. 45, where the 3D exploded view of the OpAmp layout example of Fig. 34 is rendered. This section will evaluate both 2D and 3D extraction techniques for estimating the capacitive parasitics of your CMOS circuit.

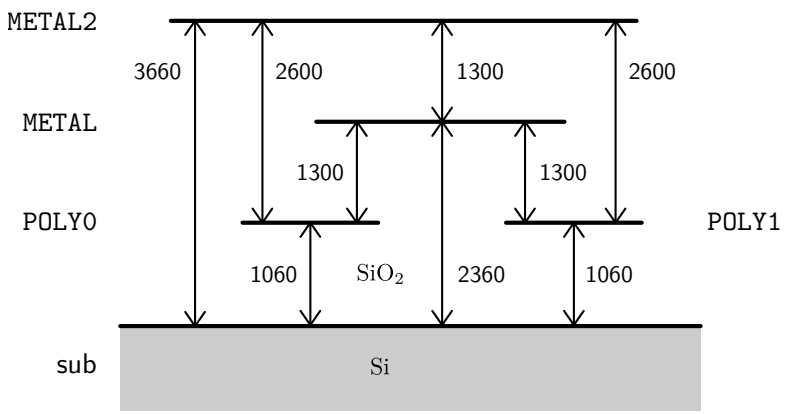


Figure 44 | Simplified CNM25 cross section used for the interconnectivity parasitic capacitance model. All units are in nm. Drawing not to scale.

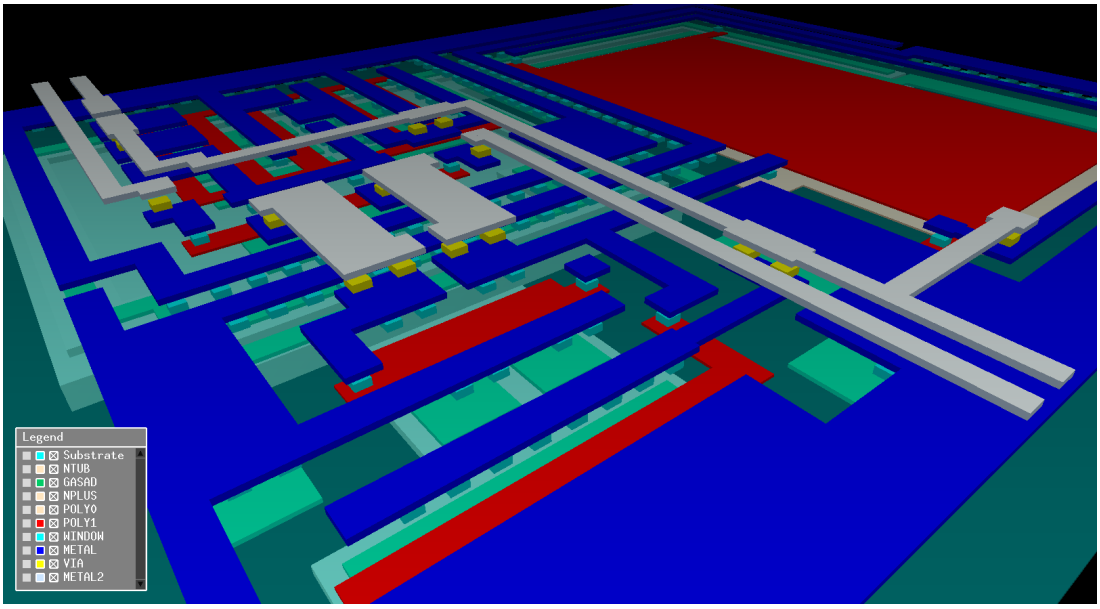


Figure 45 | 3D exploded view of `ExampleLib`→`opamp_example`→`layout` based on the simplified CNM25 cross section model of Fig. 44. Rendered by the EDA tool GDS3D from the University of Twente. More information about GDS3D can be found at `github.com/trilomix/GDS3D`.

The first approach relies on 2D analysis, much like the method used in Section 3.8 for the extraction of native CNM25 devices (i.e. `cnm25modn`, `cnm25modp` and `cnm25cpoly`). For this reason, the APDK includes the Python script `apdk/glade/verification/cnm25pex_2d.py`, which is very similar to the regular extraction code `apdk/glade/verification/cnm25lvs.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 `apdk/glade/verification/cnm25pex_2d.py` and launch the extraction process.

4. Regenerate the OpAmp test-bench netlists (`*.cir`) of Fig. 23 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.

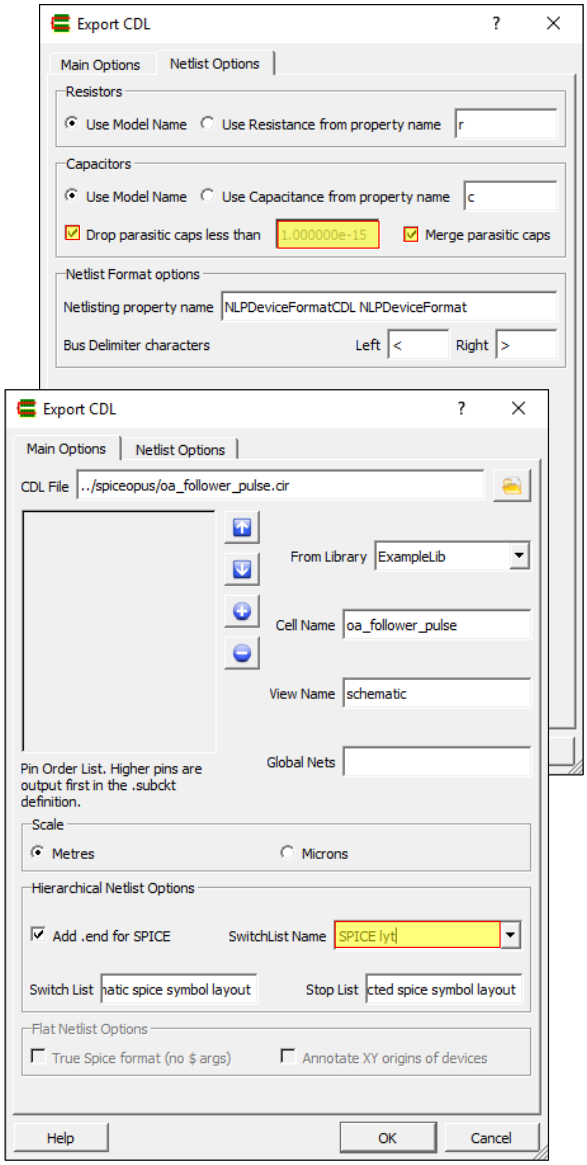


Figure 46 | Glade CDL export options for extracted views with parasitics.

apdk/glade/verification/cnm25pex_2d.py

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

# Initialise boolean package & Loading pcells
See apdk/glade/verification/cnm25lvs.py

# Parasitic capacitance density [F/m2]
e0 = 8.854187817e-12 # [F/m]
er = 3.9             # SiO2
c0 = e0 * er * 1e6  # 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 apdk/glade/verification/cnm25lvs.py

# Extracting parasitics
print("# Extracting Poly0 parasitic caps...")
extractParasitic2(pwell, polycap, cpoly0sub, 0.0)
extractParasitic2(nwell, polycap, cpoly0sub, 0.0)

print("# Extracting Poly1 parasitic caps...")
extractParasitic2(pwell, polygate, cpoly1sub, 0.0)
extractParasitic2(nwell, polygate, cpoly1sub, 0.0)

print("# Extracting Metal1 parasitic caps...")
extractParasitic2(polygate, metal1, cmetal1poly1, 0.0)
extractParasitic2(polycap, metal1, cmetal1poly0, 0.0)
extractParasitic3(pdiff, metal1, cmetal1diff, 0.0, [polygate, polycap])
extractParasitic3(ndiff, metal1, cmetal1diff, 0.0, [polygate, polycap])
extractParasitic3(pwell, metal1, cmetal1sub, 0.0, [polygate, polycap, pdiff, ndiff])
extractParasitic3(nwell, metal1, cmetal1sub, 0.0, [polygate, polycap, pdiff, ndiff])

print("# Extracting Metal2 parasitic caps...")
extractParasitic2(metal1, metal2, cmetal2metal1, 0.0)
extractParasitic3(polygate, metal2, cmetal2poly1, 0.0, [metal1])
extractParasitic3(polycap, metal2, cmetal2poly0, 0.0, [metal1, polygate])
extractParasitic3(pdiff, metal2, cmetal2diff, 0.0, [metal1, polygate, polycap])
extractParasitic3(ndiff, metal2, cmetal2diff, 0.0, [metal1, polygate, polycap])
extractParasitic3(pwell, metal2, cmetal2sub, 0.0, [metal1, polygate, polycap, pdiff, ndiff])
extractParasitic3(nwell, metal2, cmetal2sub, 0.0, [metal1, polygate, polycap, pdiff, ndiff])

print("# End of circuit extraction")
geomEnd()
```

apdk/glade/verification/cnm25pex_2d.py

For instance, the 2D parasitics for the OpAmp example of Fig. 34 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] [15], the classic 3D finite-element tool capable of extracting self and mutual capacitances between ideal conductors of arbitrary shapes, orientations and sizes. The CNM25 APDK 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 80 but selecting the script file `apdk/glade/verification/cnm25pex_3d.py` instead. In this sense, Fig. 47 presents its usage for the same OpAmp layout example.

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

It is important to note that this APDK 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. 38.

apdk/glade/verification/cnm25pex_3d.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 apdk/glade/verification/cnm2lvs.py

# Extracting devices

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

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

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

# Extracting 3D parasitics with Fastcap

print "# Extracting 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 "# End of circuit extraction"
geomEnd()
```

apdk/glade/verification/cnm25pex_3d.py

For instance, the 3D parasitics for the OpAmp example of Fig. 34 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
```

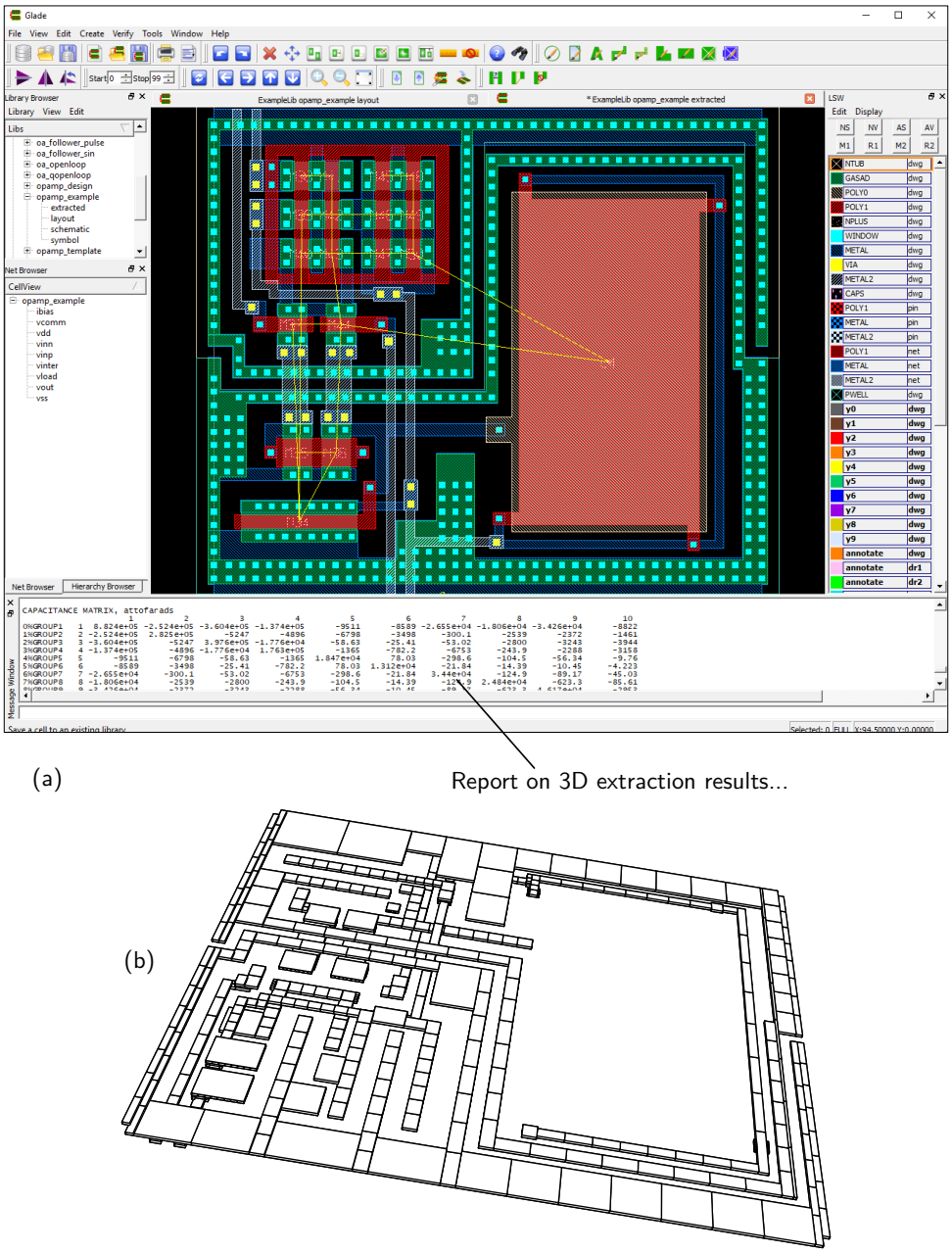


Figure 47 | Glade 3D extraction results for `ExampleLib`→`opamp_example`→`extracted` cell view (a) and FastCap equivalent finite-element mesh (b).

Q21. Execute the 2D and 3D parasitic extraction procedure to your optimized OpAmp layout following Fig. 46:

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?

3.11 Post-Layout Simulation

According to Fig. 1, the last physical verification step of your full-custom IC layout consists on the electrical post-layout simulation of the extracted circuits in order to evaluate the effects of parasitics. Thanks to the test-bench hierarchy of Fig. 23, this post-layout simulation only requires re-exporting the CDL netlists as in Fig. 46 and re-executing the NUTMEG test scripts of Section 3.5.

Q22. Redo **Q15.b** twice for the new circuit netlists with 2D and 3D parasitics. Build a **summary datasheet** of your OpAmp 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?

Q23. Repeat **Q12.b** in order to quantify the expected **degradation** of your SC $\Delta\Sigma$M dynamic range according to the 3D parasitics results.

3.12 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. 48.

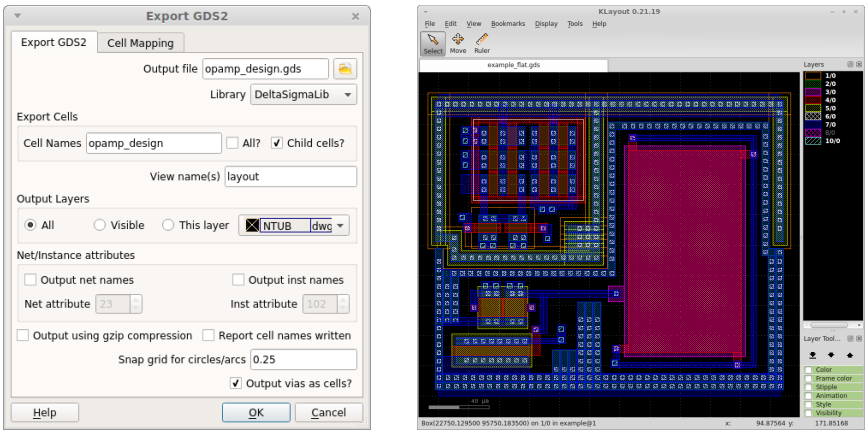


Figure 48 Glade GDSII export options and `ExampleLib`→`opamp_example`→`layout` GDSII file example viewed with KLayout. More information about KLayout can be found at `www.klayout.de`.

Q24. Export your OpAmp layout to `apdk/glade/opamp_design.gds`.

A XSpice Code Model Reference

This appendix includes the source code for the CM blocks of `xtendedlib.cm`, the custom XSpice library specifically developed for this APDK. The MOD files shown here can be taken as practical examples for the C programming of the `zinteg2sc` block of Section 3.4. In this sense, the complete sources for the `analog`, `digital`, `xtradev` and `xtraevt` standard XSpice libraries are also supplied in `apdk/spiceopus/xspice`. Finally, the macro definitions of Table 6 to 8 together with the functions declared in Table 10 to 11 are given for reference purposes. Further details can be found in [10].

apdk/spiceopus/xspice/xtendedlib/zinteg2lim.ifs

```
NAME_TABLE:

Spice_Model_Name: zinteg2lim
C_Function_Name:  cm_zinteg2lim
Description:      "Z-domain integrator with limited output"

PORT_TABLE:

Port_Name:        inp       clk       out
Description:      "input"   "clock"   "output"
Direction:        in        in        out
Default_Type:     v         d         v
Allowed_Types:    [v]       [d]       [v]
Vector:           no        no        no
Vector_Bounds:    -         -         -
Null_Allowed:     no        no        no

PARAMETER_TABLE:

Parameter_Name:   pos_edge                  out_ic
Description:      "L->H edge output sync?"  "output initial condition"
Data_Type:        int                       real
Default_Value:    0                         0.0
Limits:           [0 1]                     -
Vector:           no                        no
Vector_Bounds:    -                         -
Null_Allowed:     no                        no

PARAMETER_TABLE:

Parameter_Name:   out_min               out_max
Description:      "lower output limit"  "upper output limit"
Data_Type:        real                  real
Default_Value:    -1.0                  1.0
Limits:           -                     -
Vector:           no                    no
Vector_Bounds:    -                     -
Null_Allowed:     no                    no
```

apdk/spiceopus/xspice/xtendedlib/zinteg2lim.ifs

apdk/spiceopus/xspice/xtendedlib/zinteg2lim.mod

```
#define SAMPLING_INTEGRATION 1
#define HOLDING 0

void cm_zinteg2lim(ARGS)
{
           double inp,      /* analog voltage input */
                  out,      /* analog voltage output */
                  *inp_mem, /* sampled input */
                  *out_mem, /* integrated output */
                  out_ic,   /* output initial condition */
                  out_min,  /* minimum output limit */
                  out_max;  /* maximum output limit */
  Digital_State_t clk,      /* current clock level */
                  *clk_mem, /* previous clock level*/
                  pos_edge; /* L->H edge clock output? */
              int action;   /* action type */
             char *error;   /* error message */

  inp = INPUT(inp);            /* Retriving input values */
  clk = INPUT_STATE(clk);
  pos_edge = PARAM(pos_edge);  /* Retriving parameters */
  out_ic = PARAM(out_ic);
  out_min = PARAM(out_min);
  out_max = PARAM(out_max);

  if (INIT==1) { /* Static storage allocation and checking */

    cm_analog_alloc(1,sizeof(double));
    cm_analog_alloc(2,sizeof(double));
    cm_event_alloc(3,sizeof(Digital_State_t));

    if (out_min>out_max) {
      error = "\n*** zinteg2lim error: out_min>out_max !\n";
      cm_message_send(error);
    }
    if ((out_ic>out_max)||(out_ic<out_min)) {
      error = "\n*** zinteg2lim error: out_ic exceeds out_min,out_max !\n";
      cm_message_send(error);
    }
  }

  switch (ANALYSIS) {

    case TRANSIENT: /* Transient analysis */

      inp_mem = cm_analog_get_ptr(1,0); /* Retriving previous state */
      out_mem = cm_analog_get_ptr(2,0);
      clk_mem = cm_event_get_ptr(3,0);

      if (TIME==0) { /* Initialization */

        *inp_mem = inp;
        *out_mem = out_ic;
        out = out_ic;

      } else { /* Regular operation */

        if ((*clk_mem==ONE)&(clk==ZERO)) { /* Negative clk edge */
```

```
          if (pos_edge==FALSE)
            action = SAMPLING_INTEGRATION;

        } else {
          if ((*clk_mem==ZERO)&(clk==ONE)) { /* Positive clk edge */
            if (pos_edge==TRUE)
              action = SAMPLING_INTEGRATION;

          } else {  /* No clock edge */
              action = HOLDING;
          }
        }

        switch (action) {
           case SAMPLING_INTEGRATION: /* Sampling and integration */
            *inp_mem = inp;
            out = *out_mem+*inp_mem;
            if (out<out_min) { out = out_min; } /* Limiter */
            if (out>out_max) { out = out_max; }
            *out_mem = out;
            break;
          case HOLDING:    /* Holding */
            out = *out_mem;
        }
      }

      *clk_mem = clk;
      OUTPUT(out) = out;
      break;

    case DC:        /* DC analysis */
      OUTPUT(out) = out_ic;
      break;

    default:        /* Analysis not supported */
      error = "\n*** zinteg2lim error: analysis not supported !\n";
      cm_message_send(error);
  }
}
```

apdk/spiceopus/xspice/xtendedlib/zinteg2lim.mod

apdk/spiceopus/xspice/xtendedlib/quant2lsh.ifs

```
NAME_TABLE:

Spice_Model_Name: quant2lsh
C_Function_Name:  cm_quant2lsh
Description:      "2-level quantizer with S/H"

PORT_TABLE:

Port_Name:        inp       clk       out
Description:      "input"   "clock"   "output"
Direction:        in        in        out
Default_Type:     v         d         d
Allowed_Types:    [v]       [d]       [d]
Vector:           no        no        no
Vector_Bounds:    -         -         -
Null_Allowed:     no        no        no
```

```
PARAMETER_TABLE:

Parameter_Name:     inp_th             out_ic
Description:        "input threshold"  "output initial condition"
Data_Type:          real               int
Default_Value:      0.0                0
Limits:             -                  [0 1]
Vector:             no                 no
Vector_Bounds:      -                  -
Null_Allowed:       no                 no

PARAMETER_TABLE:

Parameter_Name:     pos_edge          t_rise        t_fall
Description:        "L->H edge out?"  "rise delay"  "fall delay"
Data_Type:          int               real          real
Default_Value:      0                 1.0e-9        1.0e-9
Limits:             [0 1]             [1e-12 -]     [1e-12 -]
Vector:             no                no            no
Vector_Bounds:      -                 -             -
Null_Allowed:       no                no            no
```

apdk/spiceopus/xspice/xtendedlib/quant2lsh.mod

```
#define SAMPLING_QUANTIZATION 1
#define HOLDING 0

void cm_quant2lsh(ARGS)
{
           double inp,      /* analog voltage input */
                  *inp_mem, /* sampled input */
                  inp_th,   /* input threshold */
                  t_rise,   /* output rise time */
                  t_fall;   /* output fall time */

  Digital_State_t out,      /* digital output */
                  *out_mem, /* holded output */
                  clk,      /* current clock level */
                  *clk_mem, /* previous clock level*/
                  out_ic,   /* output initial condition */
                  pos_edge; /* L->H edge clock output? */
              int action;   /* action type */
             char *error;   /* error message */

  inp = INPUT(inp);            /* Retriving input values */
  clk = INPUT_STATE(clk);
  inp_th = PARAM(inp_th);      /* Retrieving parameters */
  t_rise = PARAM(t_rise);
  t_fall = PARAM(t_fall);
  out_ic = PARAM(out_ic);
  pos_edge = PARAM(pos_edge);

  if (INIT==1) { /* Static storage allocation and checking */

    cm_analog_alloc(1,sizeof(double));
    cm_event_alloc(2,sizeof(Digital_State_t));
    cm_event_alloc(3,sizeof(Digital_State_t));
```

```
  if (t_rise<1e-12) {
    error = "\n*** quant2lsh error: t_rise<1ps !\n";
    cm_message_send(error);
  }
  if (t_fall<1e-12) {
    error = "\n*** quant2lsh error: t_fall<1ps !\n";
    cm_message_send(error);
  }

}

switch (ANALYSIS) {

  case TRANSIENT: /* Transient analysis */

    inp_mem = cm_analog_get_ptr(1,0); /* Retriving previous state */
    out_mem = cm_event_get_ptr(2,0);
    clk_mem = cm_event_get_ptr(3,0);

    if (TIME==0) { /* Initialization */

      *inp_mem = inp;
      *out_mem = out_ic;
      out = out_ic;
      OUTPUT_CHANGED(out) = TRUE;
      OUTPUT_STATE(out) = out;
      OUTPUT_STRENGTH(out) = STRONG;

    } else { /* Regular operation */

      if ((*clk_mem==ONE)&&(clk==ZERO)) { /* Negative clk edge */
        if (pos_edge==FALSE)
        {
          action = SAMPLING_QUANTIZATION;
        }
      } else {
        if ((*clk_mem==ZERO)&&(clk==ONE)) { /* Positive clk edge */
          if (pos_edge==TRUE)
          {
            action = SAMPLING_QUANTIZATION;
          }
        } else {  /* No clock edge */
            action = HOLDING;
        }
      }

      switch (action) {
        case SAMPLING_QUANTIZATION: /* Quantization action */
          OUTPUT_CHANGED(out) = TRUE;
          *inp_mem = inp;
          if (*inp_mem>inp_th) {
            out = ONE;
            OUTPUT_DELAY(out) = t_rise;
          } else {
            out = ZERO;
            OUTPUT_DELAY(out) = t_fall;
          }
          OUTPUT_STATE(out) = out;
          OUTPUT_STRENGTH(out) = STRONG;
          *out_mem = out;
```

```
            break;
          case HOLDING:            /* Holding action */
            OUTPUT_CHANGED(out) = FALSE;
        }
      }

      *clk_mem = clk;

      break;

    case DC:        /* DC analysis */
      OUTPUT_STATE(out) = out_ic;
      OUTPUT_STRENGTH(out) = STRONG;
      break;

    default:        /* Analysis not supported */
      error = "\n*** quant2lsh error: analysis not supported !\n";
      cm_message_send(error);
  }
}
```

apdk/spiceopus/xspice/xtendedlib/dac2lsym.ifs

```
NAME_TABLE:

Spice_Model_Name: dac2lsym
C_Function_Name:  cm_dac2lsym
Description:      "2-level symmetrical DAC"

PORT_TABLE:

Port_Name:        inp       out
Description:      "input"   "output"
Direction:        in        out
Default_Type:     d         v
Allowed_Types:    [d]       [v]
Vector:           no        no
Vector_Bounds:    -         -
Null_Allowed:     no        no

PARAMETER_TABLE:

Parameter_Name:     out_level
Description:        "output level"
Data_Type:          real
Default_Value:      1.0
Limits:             [0 -]
Vector:             no
Vector_Bounds:      -
Null_Allowed:       no
```

apdk/spiceopus/xspice/xtendedlib/dac2lsym.mod

```
void cm_dac2lsym(ARGS)
{
  Digital_State_t inp;       /* digital input */
           double out,       /* analog voltage o
                  out_level; /* output level */
             char *error;    /* error message */

  inp = INPUT_STATE(inp);       /* Retriving inp
  out_level = PARAM(out_level); /* Retrieving pa

  if (INIT==1) { /* Initial checking */
    if (out_level<0) {
      error = "\n*** dac2lsym error: out_level m
      cm_message_send(error);
    }
  }

  if (ANALYSIS!=AC) { /* DC and Transient anlysi
    switch (inp) {
        case ZERO:
          out = -out_level;
          break;
        case ONE:
          out = out_level;
    }
    OUTPUT(out) = out;
  } else {            /* Other analysis */
    error = "\n*** dac2lsym error: analysis not
    cm_message_send(error);
  }

}
```

apdk/spiceopus/xspice/xtendedlib/dscope_trig.ifs

```
NAME_TABLE:

Spice_Model_Name: dscope_trig
C_Function_Name:  cm_dscope_trig
Description:      "digital scope by digital trigger to SPICE3 raw file"

PORT_TABLE:

Port_Name:        inp       strb
Description:      "input"   "strobe"
Direction:        in        in
Default_Type:     d         d
Allowed_Types:    [d]       [d]
Vector:           no        no
Vector_Bounds:    -         -
Null_Allowed:     no        no

PARAMETER_TABLE:

Parameter_Name:     pos_edge          nsamp               fname
Description:        "L->H edge out?"  "number of samples" "filename"
Data_Type:          int               int                 string
Default_Value:      0                 1                   "tran.raw"
Limits:             [0 1]             [1 -]               -
Vector:             no                no                  no
Vector_Bounds:      -                 -                   -
Null_Allowed:       no                no                  no

STATIC_VAR_TABLE:

Static_Var_Name:       ncount
Data_Type:             pointer
Description:           "current sample index"
```

apdk/spiceopus/xspice/xtendedlib/dscope_trig.mod

```
#include <stdlib.h>
#include <stdio.h>
#define SAMPLING_WRITING 1
#define NONE 0

void cm_dscope_trig(ARGS)
{
  Digital_State_t inp,            /* digital input signal */
                  strb,           /* digital strobe */
                  *strb_mem,      /* previous strobe level */
                  pos_edge;       /* L->H edge strobe? */

            double nsamp,         /* number of samples */
                   *ncount;       /* sample counter */
               int action;        /* action type */
              FILE *outfile;      /* output file pointer */
              char *filename;     /* output file name */

  inp = INPUT_STATE(inp);     /* Retriving input values */
  strb = INPUT_STATE(strb);
  pos_edge = PARAM(pos_edge); /* Retrieving parameters */
  nsamp = PARAM(nsamp);
  filename = PARAM(fname);
```

```
  if (INIT==1) { /* Static storage allocation and file header */
    cm_event_alloc(1,sizeof(Digital_State_t));
    ncount = malloc(sizeof(double));
    STATIC_VAR(ncount) = ncount;
    outfile = fopen(filename,"w");
    if (outfile!=NULL) {
      fprintf(outfile,"Title: %s\n",filename);
      fprintf(outfile,"Date: unknown\n");
      fprintf(outfile,"Plotname: Transient Analysis\n");
      fprintf(outfile,"Flags: real\n");
      fprintf(outfile,"Sckt. naming: from bottom to top\n");
      fprintf(outfile,"No. Variables: 2\n");
      fprintf(outfile,"No. Points: %.0f\n",nsamp);
      fprintf(outfile,"Variables:\n");
      fprintf(outfile,"\t0\ttime\ttime\n");
      fprintf(outfile,"\t1\tvout\tvoltage\n");
      fprintf(outfile,"Values:\n");
      fclose(outfile);
    }
  } else {
    switch (ANALYSIS) {
      case TRANSIENT: /* Transient analysis */
        strb_mem = cm_event_get_ptr(1,0); /* Retriving previous state */
        ncount = STATIC_VAR(ncount);
        if (TIME==0) { /* Initialization */
          *strb_mem = strb;
          *ncount = 0;
         } else { /* Regular operation */

          if ((*strb_mem==ONE)&&(strb==ZERO)) { /* Negative strobe edge */
            if (pos_edge==FALSE)
            {
              action = SAMPLING_WRITING;
            }
          } else {
            if ((*strb_mem==ZERO)&&(strb==ONE)) { /* Positive strobe edge */
              if (pos_edge==TRUE)
              {
                action = SAMPLING_WRITING;
              }
            } else {  /* No strobe edge */
                action = NONE;
            }
          }
          if ((action==SAMPLING_WRITING)&&(*ncount<nsamp)) { /* Writing action */
            outfile = fopen(filename,"a");
            if (outfile!=NULL) {
              fprintf(outfile,"\t%.0f\t%.15e\t%d\n",*ncount,TIME,inp);
              fclose(outfile);
              *ncount = *ncount+1;
            }
          }
          *strb_mem = strb;
          STATIC_VAR(ncount) = ncount;
        }
        break;
    }
  }
}
```

Name	Description	Example
ARGS	Passing arguments to the CM.	`void dsm_opamp(ARGS)`
CALL_TYPE	Returns the simulator type used for the CM (EVENT or ANALOG).	`if (CALL_TYPE==ANALOG) {...}`
INIT	Returns 1 when first call of the CM.	`if (INIT==1) {...}`
ANALYSIS	Returns the current analysis type (AC, DC or TRANSIENT).	`if (ANALYSIS!=AC) {...}`
FIRST_TIMEPOINT	Returns 1 when first call of CM during the current analysis step.	`if (FIRST_TIMEPOINT==0) {...}`
TIME	Double returning current time point of TRANSIENT analysis in s.	`t2 = TIME-t1;`
T(n)	Double vector returning [current previous] time points of TRANSIENT analysis.	`dt = T(0)-T(1);`
RAD_FREQ	Double returning current frequency of AC analysis in rad/s.	`f = RAD_FREQ/(2*pi);`
TEMPERATURE	Double vector returning current analysis temperature in °C.	`TK = TEMPERATURE+273;`

Table 6 | XSpice macro definitions for circuit data.

Name	Description	Example
PARAM(param)	Returns CM parameter value.	`k = PARAM(gain);`
PARAM_SIZE(param)	Returns CM parameter vector size.	`num_coeff = PARAM_SIZE(coeff);`
PARAM_NULL(param)	Returns 1 when no value specified.	`if (PARAM_NULL(gain)==1) {...}`
PORT_SIZE(a)	Returns port size.	`num_out = PORT_SIZE(out);`
PORT_NULL(a)	Returns 1 when port is not connected.	`if (PORT_NULL(inp)==1) {...}`
LOAD(a)	Adds load capacitance in F to digital port.	`LOAD(inp) = 1e-12;`
TOTAL_LOAD(a)	Reads total load capacitance in F at digital port due to all attached CMs.	`delay = TOTAL_LOAD(out)*...`

Table 7 | XSpice macro definitions for parameter and port data.

Name	Description	Example
INPUT(x)	Reads value from input port.	signal = INPUT(inp);
INPUT_STATE(x)	Reads the state of a digital input port (ZERO, ONE or UNKNOWN).	if (INPUT_STATE(inp)!=ONE) {...}
INPUT_STRENGTH(x)	Reads the strength with which a digital input port is externally driven (STRONG, RESISTIVE, HI_IMPEDANCE or UNDETERMINATED).	if (INPUT_STRENGTH(inp)!= HI_IMPEDANCE) {...}
OUTPUT(y)	Writes value to output port.	OUTPUT(out) = result;
OUTPUT_CHANGED(y)	Flags a digital output port as modified. If TRUE (default) then state, strength and delay need to be defined.	OUTPUT_CHANGED(out) = FALSE;
OUTPUT_DELAY(y)	Defines the delay in s (>0) of a digital output port.	OUTPUT_DELAY(out) = 1e-9;
OUTPUT_STATE(y)	Writes the state of a digital output port (ZERO, ONE or UNKNOWN).	OUTPUT_STATE(out) = ZERO;
OUTPUT_STRENGTH(y)	Writes the strength with which a digital output port is internally driven (STRONG, RESISTIVE, HI_IMPEDANCE or UNDETERMINATED).	OUTPUT_STRENGTH(out) == STRONG;

Table 8 | XSpice macro definitions for I/O data.

Name	Description	Example
PARTIAL(y,x)	Sets the partial derivative of output port y with respect to input port x. Needed by the simulator to solve non-linear equations. The cm_analog_auto_partial() function of Table 10 may be used instead.	PARTIAL(out,in) = 1;
AC_GAIN(y,x)	Sets the gain from input port x to output port y in AC analysis. Gain follows the complex data structure Complex_t defined in Table 11.	AC_GAIN(out,in) = gain_complex;
STATIC_VAR(a)	Provides access to the static variables defined in the IFS file.	last_x = STATIC_VAR(x); STATIC_VAR(x) = x;

Table 9 | XSpice macro definitions for partial derivatives, gains and static variables.

```
void cm_smooth_corner(       /* Quadratic smoothing between two intersecting lines */
    double x_input,          /* The value of the x input */
    double x_center,         /* The x intercept of the two slopes */
    double y_center,         /* The y intercept of the two slopes */
    double domain,           /* The smoothing domain */
    double lower_slope,      /* The lower slope */
    double upper_slope,      /* The upper slope */
    double *y_output,        /* The smoothed y output */
    double *dy_dx)           /* The partial of y wrt x */

void cm_smooth_discontinuity(/* Quadratic smoothing between two discontinuity points */
    double x_input,          /* The x value at which to compute y */
    double x_lower,          /* The x value of the lower corner */
    double y_lower,          /* The y value of the lower corner */
    double x_upper,          /* The x value of the upper corner */
    double y_upper,          /* The y value of the upper corner */
    double *y_output,        /* The computed smoothed y value */
    double *dy_dx)           /* The partial of y wrt x */

double cm_smooth_pwl(        /* Piecewise linear interpolation or extrapolation */
    double x_input,          /* The x input value */
    double *x,               /* The vector of x values */
    double *y,               /* The vector of y values */
    int size,                /* The size of the xy vectors */
    double input_domain,     /* The smoothing domain */
    double *dout_din)        /* The partial of the output wrt the input */

void *cm_analog_alloc(  /* Allocates storage space for analog state information */
    int tag,            /* The user-specified tag for this block of memory */
    int bytes)          /* The number of bytes to allocate */

void *cm_event_alloc(   /* Allocates storage space for event state information */
    int tag,            /* The user-specified tag for the memory block */
    int bytes)          /* The number of bytes to be allocated */

void *cm_analog_get_ptr(/* Returns pointer to pre-allocated analog state */
    int tag,            /* The user-specified tag for this block of memory */
    int timepoint)      /* The timepoint of interest - 0=current 1=previous */

void *cm_event_get_ptr( /* Returns pointer to pre-allocated event state */
    int tag,            /* The user-specified tag for the memory block */
    int timepoint)      /* The timepoint - 0=current, 1=previous */

int cm_analog_integrate(/* Computes analog integral over time in TRANSIENT analysis */
    double integrand,   /* The integrand */
    double *integral,   /* The (pre-allocated) current and returned value of integral */
    double *partial)    /* The partial derivative of integral wrt integrand */

int cm_analog_converge( /* Notifies analog simulator to converge state variable */
    double *state)      /* The (pre-allocated) state to be converged */

void cm_analog_not_converged() /* Forces analog simulator recall model for further convergence */

void cm_analog_auto_partial() /* Forces analog simulator to compute all PARTIAL automatically */

double cm_ramp_factor() /* Returns fraction of RAMPTIME achieved by the analog simulator */
```

Table 10 | XSpice smoothing, storage, integration and convergence function headers.

```
char *cm_message_get_errmsg() /* Returns address of error message string */

int cm_message_send(          /* Sends message to standard output */
    char *msg)                /* The message to output. */

int cm_analog_set_perm_bkpt(  /* Forces analog simulator to solve for that time stamp */
    double time)              /* The time of the breakpoint to be set */

int cm_analog_set_temp_bkpt(  /* Same as cm_analog_set_perm_bkpt but for the next time step only */
    double time)              /* The time of the breakpoint to be set */

int cm_event_queue(           /* Same as cm_analog_set_perm_bkpt but for event models */
    double time)              /* The time of the event to be queued */

void cm_climit_fcn(                 /* Limits analog output and computes partial derivative */
    double in,                      /* The input value */
    double in_offset,               /* The input offset */
    double cntl_upper,              /* The upper control input value */
    double cntl_lower,              /* The lower control input value */
    double lower_delta,             /* The delta from control to limit value */
    double upper_delta,             /* The delta from control to limit value */
    double limit_range,             /* The limiting range */
    double gain,                    /* The gain from input to output */
    int    percent,                 /* The fraction vs. absolute range flag */
    double *out_final,              /* The output value */
    double *pout_pin_final,         /* The partial of output wrt input */
    double *pout_pcntl_lower_final, /* The partial of output wrt lower control input */
    double *pout_pcntl_upper_final) /* The partial of output wrt upper control input */

typedef struct Complex_s {
    double real;               /* The real part of the complex number */
    double imag;               /* The imaginary part of the complex number */
} Complex_t;                  /* Complex number type */

Complex_t cm_complex_set(      /* Costructs a complex number */
    double real,               /* The real part of the complex number */
    double imag)               /* The imaginary part of the complex number */

Complex_t cm_complex_add(      /* Returns complex addition */
    Complex_t x,               /* The first operand of x + y */
    Complex_t y)               /* The second operand of x + y */

Complex_t cm_complex_subtract( /* Returns complex substraction */
    Complex_t x,               /* The first operand of x - y */
    Complex_t y)               /* The second operand of x - y */

Complex_t cm_complex_multiply( /* Returns complex multiplication */
    Complex_t x,               /* The first operand of x * y */
    Complex_t y)               /* The second operand of x * y */

Complex_t cm_complex_divide(   /* Returns complex division */
    Complex_t x,               /* The first operand of x / y */
    Complex_t y)               /* The second operand of x / y */
```

Table 11 | XSpice message, breakpoint, special and complex math function headers.

Glossary

ADC analog-to-digital converter

APDK academic process design kit

BSIM3v3 Berkeley Short-channel IGFET Model version 3.3

BW bandwidth

CDL circuit description language

CM code model

CMRR common-mode rejection ratio

CNM25 2.5μm 2-polySi 2-metal CMOS technology from IMB-CNM(CSIC)

CMOS complementary metal-oxide-semiconductor

DAC digital-to-analog converter

DC direct current

DR dynamic range

DRC design rule checker

DRD design rule driven

$\Delta\Sigma$M $\Delta\Sigma$ modulator

DUT device under test

EDA electronic design automation

ERC electrical rule checker

FOM figure of merit

FS full scale

GDSII graphic database system II

HDL hardware description language

IC integrated circuit

IFS interface specification

I/O input/output

LSW layer selection window

LVS layout versus schematic

MPP multi-part path

MOD model definition

MOS metal-oxide-semiconductor

MOSFET MOS field-effect transistor

NMOS N-type MOS

OpAmp operational amplifier

OS operative system

OSR oversampling ratio

PEX parasitic extraction

PCell parameterized cell

PDF portable document format

PiP polySi-insulator-polySi

PMOS P-type MOS

PSD power spectral density

PVT process, supply voltage and temperature

RF radio frequency

SC switched-capacitor

SPICE Simulation Program with Integrated Circuit Emphasis

References

[1] J. Pallarès, K. Sabine, L. Terés, and F. Serra-Graells. An Academic EDA Suite for the Full-Custom Design of Mixed-Mode Integrated Circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. `https://doi.org/10.1109/ISCAS.2017.8050519`.

[2] Peardrop Design Systems. *Glade Reference Manual*. `apdk/doc/bib/Glade_Reference.pdf`.

[3] F.L. Cox, W.B. Kuhn, J.P. Murray, and S.D. Tynor. Code-level Modeling in XSPICE. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 871–874, May 1992. `https://doi.org/10.1109/ISCAS.2017.8050519`.

[4] 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.

[5] 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. `apdk/doc/bib/BSIM3v3_Manual.pdf`.

[6] 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`.

[7] J. Silva, U. Moon, J. Steensgaard, and C. Temes G. Wideband Low-Distortion Delta-Sigma ADC Topology. *IEE Electronics Letters*, 37(12):737–738, Jun 2001. `https://doi.org/10.1049/el:20010542`.

[8] V. Peluso, M. Steyaert, and W. M. C. Sansen. *Design of Low-Voltage Low-Power CMOS Delta-Sigma A/D Converters*, volume 493 of *The Springer International Series in Engineering and Computer Science*. Springer, 1999. `https://doi.org/10.1007/978-1-4757-2978-8`.

[9] 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. `apdk/doc/bib/Spice3f3_Users_Manual.pdf`.

[10] F. L. Cox, W. B. Kuhn, H. W. Li, J. P. Murray, S. D. Tynor, and M. J. Willis. *XSpice Software User's Manual*. Georgia Tech Research Institute, Atlanta, GA 30332, Dec 1992. `apdk/doc/bib/Xspice_Users_Manual.pdf`.

[11] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation*, 101(23):e215–e220, 2000. `https://physionet.org`.

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

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

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

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