A Freeware EDA Framework for Teaching Mixed-Mode Full-Custom VLSI Design

J. Pallarès[1], F. Vila[1], S. Sutula[1], K. Sabine[2], L. Terés[1,3] and F. Serra-Graells[1,3]
paco.serra@imb-cnm.csic.es www.cnm.es/~pserra

[1]Institut de Microelectrònica de Barcelona
IMB-CNM(CSIC)

[2]Peardrop Design Systems Ltd

[3]Dept. of Microelectronics and Electronic Systems
Universitat Autònoma de Barcelona

***Abstract*—This paper presents a freeware EDA framework for teaching mixed-mode full-custom VLSI design. The proposed set of EDA tools and associated physical design kit (PDK) allows students to gain hands-on experience on ASIC design tasks covering schematic entry, both at system and circuit levels, HDL system simulation and block specification, automatic circuit optimization, PCell-based layout, physical verification, parasitics extraction, post-layout simulation and tape-out. A practical design case based on a $\Delta\Sigma$ modulator for A/D conversion in a simple CMOS technology is supplied to illustrate the capabilities of the proposed EDA framework. Students can easily make use of the presented environment both at laboratory and at home, since all EDA tools are available for MS Windows and Linux platforms.**

I. Introduction

Teaching VLSI design imposes real challenges when preparing hands-on laboratory exercises, specially for mixed-mode full-custom ASIC design cases. Professional EDA tools for this purpose usually involve expensive licenses, powerful hardware requirements and complex system administrator tasks. Furthermore, modern deep submicron CMOS technologies are in practice too complex to get familiar with during short laboratory practices, and their process information is often confidential. Last but not least, students usually do not have enough laboratory sessions to truly develop full-custom layout designs.

In order to overcome the above issues, this paper proposes the freeware EDA framework of Fig. 1 for teaching mixed-signal full-custom VLSI design. Although not strictly professional, these alternative EDA tools and associated physical design kit (PDK) cover most design steps and they can be easily customized for academic purposes. Moreover, students can exploit the proposed environment at home, as it is available for both MS Windows and Linux platforms and its installation is straightforward. In particular:

- **gaf** (gschem and friends) includes a customizable schematic editor (*gschem*) together with a programmable netlister (*gnetlist*) all featuring Scheme scripting language [1]. It is part of the open source gEDA (GPL EDA) suite initially developed by Ales Hvezda [2].
- **SpiceOpus** (SPICE with integrated optimization utilities) by the CACD Group at University of Ljubljana [3] is a port of the Berkeley SPICE3F5 electrical simulator featuring Nutmeg scripting language [4], plus a custom optimization tool and the Georgia Tech Research Institute XSpice multi-domain event-driven engine [5]. The resulting simulation suite can perform native mixed-signal circuit and system simulation and optimization.
- **Glade** (GDS, LEF and DEF editor) by Keith Sabine [6] is an IC mask layout editor and physical verification tool featuring Python scripting language [7].

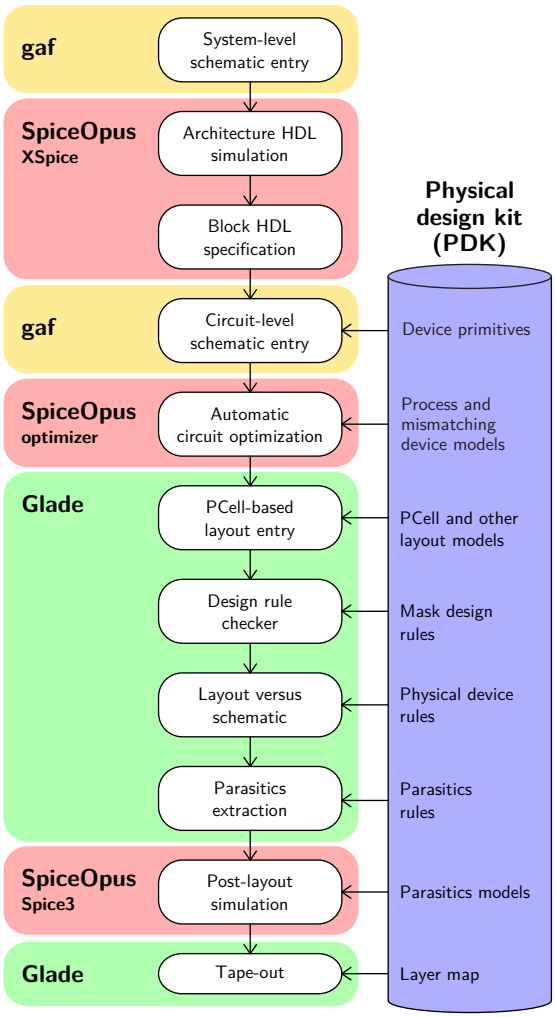

Figure 1. Proposed EDA framework for teaching mixed-signal full-custom VLSI design methodology.

In order to illustrate the capabilities of the proposed EDA framework, the practical design case of Fig. 2 is presented: a delta-sigma modulator ($\Delta\Sigma$M) [8] for a 14-bit 8kHz $2V_{dp}$-input ADC. Apart from dealing with both analog and digital signal domains, the oversampling nature of this exercise also requires modeling at several abstraction levels to speed up its simulation. Concerning the $\Delta\Sigma$M single-loop architecture of Fig. 2(b), a second-order noise shaper with single-bit quantizer is chosen here to avoid stability and distortion issues,

respectively. Furthermore, nested feedforward loops are added to optimize signal full-scale at the cost of adding an extra summer at the input of the quantizer [9].

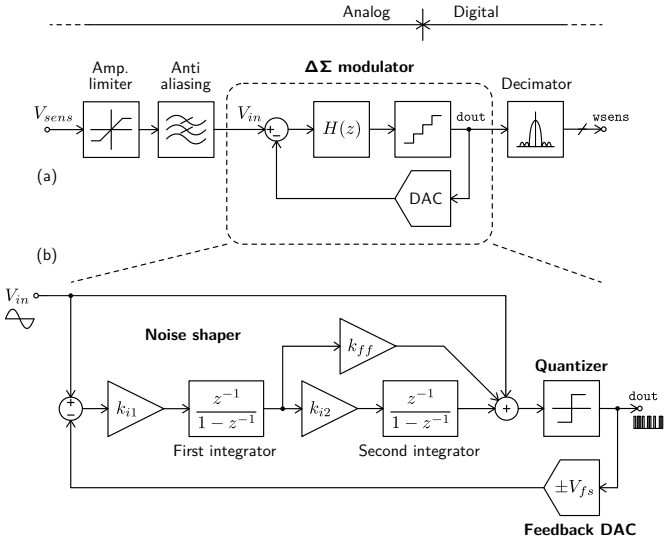

Figure 2. General architecture of a $\Delta\Sigma$ oversampling ADC (a) and differential voltage model in the Z-domain of the proposed $\Delta\Sigma$M design case (b).

Following the academic purposes of this proposal, target technology is chosen to be the IMB-CNM(CSIC) 2.5μm 2P2M CMOS process (CNM25) depicted in Fig. 3. Although very simple, this technology allows students to deal with a limited number of layers and design rules during layout edition and physical verification stages, respectively, together with a reduced set of device model parameters for circuit design, as shown in Fig. 4. Nevertheless, the proposed EDA environment can be extended to modern technologies if required.

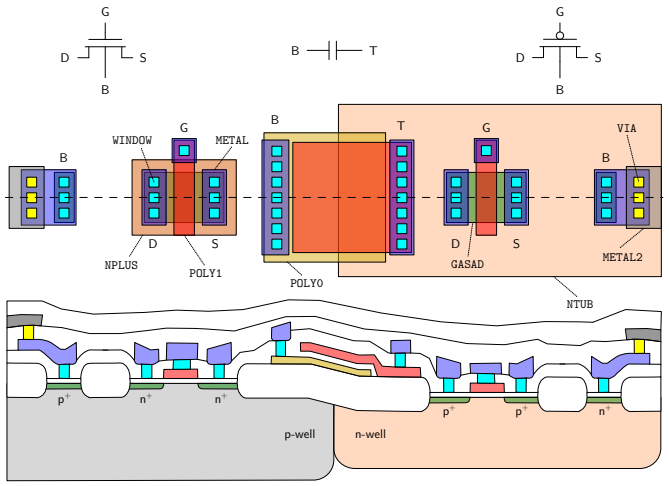

Figure 3. CNM25 technology cross section and device primitives.

II. Schematic Entry

Following the proposal of Fig. 1, all schematic entries at system and circuit levels are performed through *gschem* editor. Despite its simplicity, this tool already supports symbol edition, library browsing, net and pin labeling, hierarchical navigation, instance annotation and automatic rewiring, among other features. As an example, Fig. 5 depicts the schematic of the $\Delta\Sigma$M design case of Fig. 2(b) being

(a) cnm25proc.lib

```
.lib model
.model cnm25modn nmos level=2 vto={vton} uo={uon} tox=...
.model cnm25modp pmos level=2 vto={vtop} uo={uop} tox=...
.model cnm25cpoly c cj={cj} cjsw=0.0
.endl

.lib ttt
.param vton=.942
.param vtop=-1.139
.param uon=648
.param uop=213
.param cj=4.227E-4
.lib 'cnm25proc.lib' model
.endl
```

(b) cnm25match.lib

```
.param avto=30e-9
.param rauo=5e-8
.param rac=7.3e-8

.subckt cnm25modn d g s b param: w=3u l=3u
 + ad=0 as=0 pd=0 ps=0 m=1
.param vton=.942+rndgauss(avto/sqrt(m*w*l))
.param uon=648*(1+rndgauss(rauo/sqrt(m*w*l)))
.model modnlocal nmos level=2 vto={vton} uo={uon} tox=...
mn d g s b modnlocal 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 cj=4.227E-4*(1+rndgauss(rac/sqrt(m*w*l)))
.model cpolylocal c cj={cj} cjsw=0.0
cpip t b cpolylocal w={w} l={l} m={m}
.ends
```

Figure 4. Portion of CNM25 process (a) and mismatching (b) device model files for corner and Montecarlo SPICE simulation.

edited in *gschem* using our own custom symbol library. Taking advantage of Scheme scripting, the accompanying tool *gnetlist* can be programmed to generate the $\Delta\Sigma$M equivalent netlist of Fig. 6 following the particular syntax rules required by XSpice.

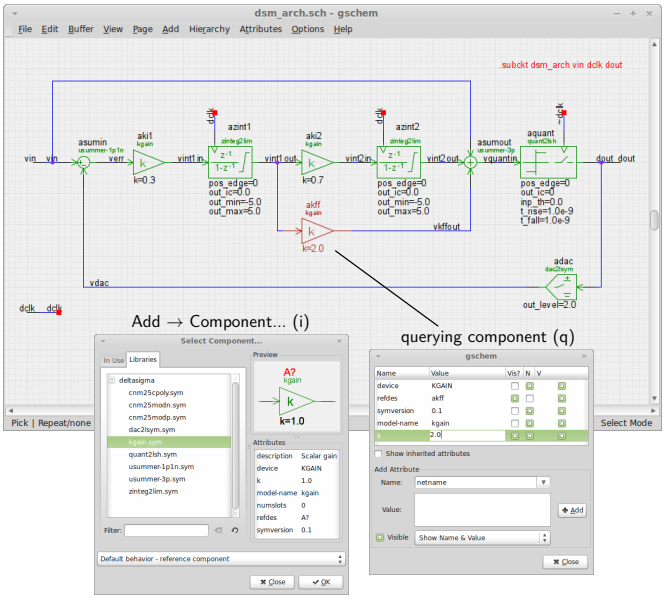

Figure 5. $\Delta\Sigma$M architecture of Fig. 2(b) edited in *gschem*.

dsm-arch.sub

```
.subckt dsm_arch vin dclk dout
asumin [%v(vin) %v(vdac)] %v(verr) msumin
.model msumin usummer(sign=[1.0 -1.0])
aki1 %v(verr) %v(vint1in) mki1
.model mki1 kgain(k=0.3)
azint1 %v(vint1in) %d(dclk) %v(vint1out) mzint1
.model mzint1 zinteg2lim(pos_edge=0 out_ic=0.0
 + out_min=-5.0 out_max=5.0)
aki2 %v(vint1out) %v(vint2in) mki2
.model mki2 kgain(k=0.7)
azint2 %v(vint2in) %d(dclk) %v(vint2out) mzint2
.model mzint2 zinteg2lim(pos_edge=0 out_ic=0.0
 + out_min=-5.0 out_max=5.0)
akff %v(vint1out) %v(vkffout) mkff
.model mkff kgain(k=2.0)
asumout [%v(vint2out) %v(vkffout) %v(vin)]
 + %v(vquantin) msumout
.model msumout usummer(sign=[1.0 1.0 1.0])
aquant %v(vquantin) %d(~dclk) %d(dout) mquant
.model mquant quant2lsh(inp_th=0.0 out_ic=0 pos_edge=0
 + t_rise=1e-9 t_fall=1e-9)
adac %d(dout) %v(vdac) mdac
.model mdac dac2lsym(out_level=2.0)
.ends
```

Figure 6. XSpice netlist generated from the $\Delta\Sigma$M schematic of Fig. 5.

III. Architecture HDL Simulation

The use of hardware description languages (HDLs) for the validation of mixed-mode integrated systems is highly recommended. Not only it simplifies the co-simulation of analog and digital parts of VLSI circuits, but it can also allow strong savings in terms of CPU time. This feature is specially noticeable in oversampled systems, like our $\Delta\Sigma$M case study. For this purpose, the EDA framework proposed in Fig. 1 takes benefit of XSpice mixed-signal code models (CMs), which are compiled separately from the SpiceOpus engine. In this sense, Fig. 7 shows the source code for one of these XSpice custom CMs specifically developed for the $\Delta\Sigma$M architecture of Fig. 5.

zinteg2lim.mod

```
void cm_zinteg2lim(ARGS) {
 inp = INPUT(inp);            /* Retriving input values */
 clk = INPUT_STATE(clk);
 pos_edge = PARAM(pos_edge);  /* Retrieving parameters */
 out_max = PARAM(out_ax);
 ...
 switch (ANALYSIS) {
  case TRANSIENT:
   if ((*clk_mem==ONE)&&(clk==ZERO)) {/* Neg. clk edge */
    if (pos_edge==FALSE)
     action = SAMPLING_INTEGRATION;
   } else {
    if ((*clk_mem==ZERO)&&(clk==ONE)) {/* Pos. clk edge*/
     if (pos_edge==TRUE)
      action = SAMPLING_INTEGRATION;
    } else {                          /* No clock edge */
     action = HOLDING;
   }
 ...
   switch (action) {
    case SAMPLING_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:
     out = *out_mem;
}}}
```

Figure 7. Portion of the custom XSpice CM source for the $\Delta\Sigma$M Z-domain integrator (with limiter) blocks of Fig. 5.

In practice, students can easily get familiar with the C-like syntax of CMs by simple inspection of source codes, like Fig. 7 or the large collection of examples supplied with SpiceOpus. At this stage, the high-speed simulation capabilities of XSpice HDL are exploited to optimize our $\Delta\Sigma$M architecture, as in Fig. 8 and 9.

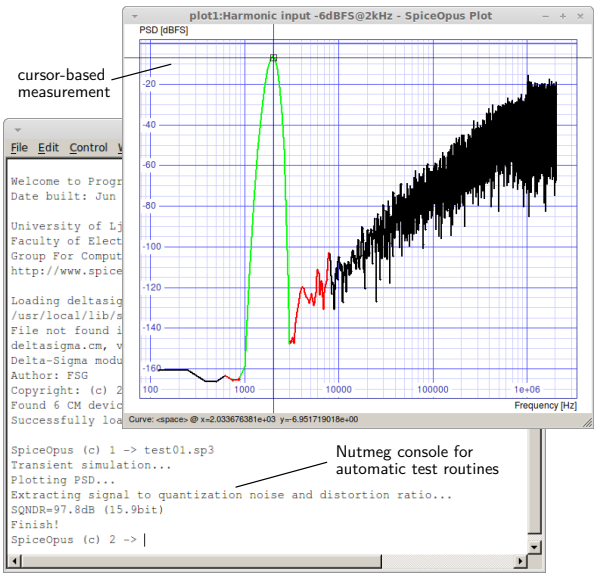

Figure 8. Example of SpiceOpus XSpice automatic SQNDR extraction for the $\Delta\Sigma$M architecture of Fig. 5.

In particular, students can test the required oversampling ratio (OSR) with the automatic computation of the resulting signal to quantization noise and distortion ratio (SQNDR) like in Fig. 8. Based on similar test routines scripted in Nutmeg language, the impact on $\Delta\Sigma$M dynamic range due to limited full-scale in Z-domain integrators is evaluated in Fig. 9(a). Taking into account the target switched capacitor (SC) circuit implementation, students can already observe in Fig. 9(b) the effects caused by deviations in the capacitor k-coefficients of Fig. 5 due to technology mismatching [10]. In this case, students estimate k ranges from kT/C specs and the particular CMOS process and mismatching parameters of Fig. 4.

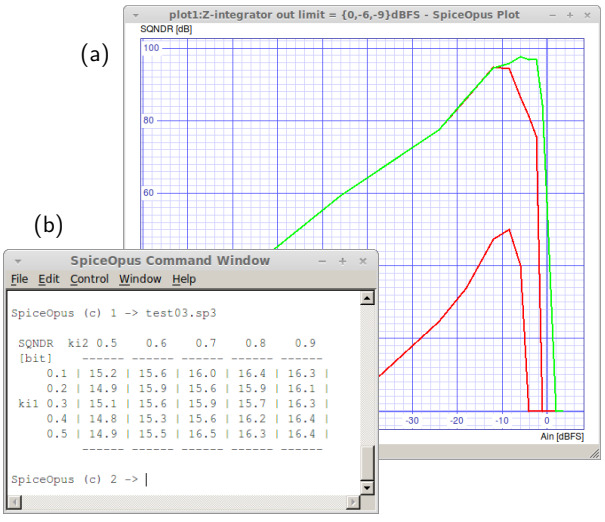

Figure 9. Example of SpiceOpus XSpice simulation of SQNDR vs integrators full-scale (a) and coefficient mismatching (b) for the $\Delta\Sigma$M of Fig. 5.

IV. Block HDL Specification

Once system architecture is fixed, next step in the design methodology depicted in Fig. 1 consists on specifying the required performance for the basic building blocks of the VLSI circuit. In this way, the circuit design and optimization for each block can be done separately following Section V. This strategy does not avoid final simulations at transistor level for the full system, but it speeds up the optimization process for each part. In our design case, the specification of the OpAmp blocks required for the switched-capacitor (SC) circuit implementation of our $\Delta\Sigma$M is chosen. In particular, the impact of OpAmp blocks is modeled through their DC open-loop gain (G), gain bandwidth product (GBW) and slew-rate (SR), as shown in Fig. 10.

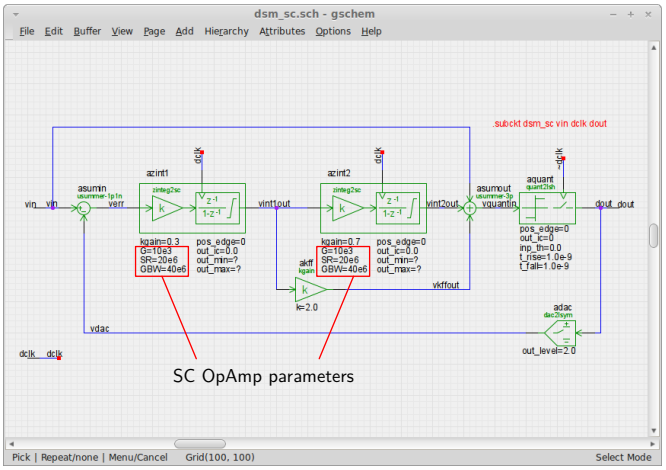

Figure 10. *gschem* schematic of the $\Delta\Sigma$M architecture of Fig. 5 with built-in SC OpAmp parameters.

For this purpose, students are asked first to analyze the CM example of the Z-domain integrator supplied in Fig. 7 and develop their own XSpice custom model by including OpAmp effects. Once this CM is compiled and verified, the SC $\Delta\Sigma$M topology of Fig. 10 is then simulated using Nutmeg test scripts similar to Section III but focused on extracting the minimum G, GBW and SR specs for each OpAmp stage, as depicted in Fig. 11.

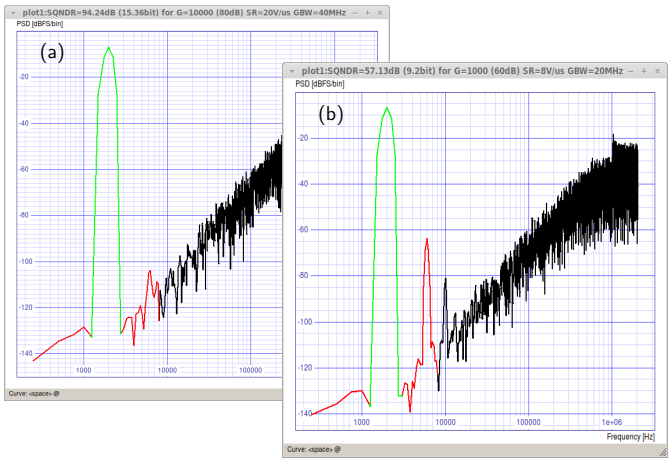

Figure 11. Example of SpiceOpus XSpice simulation of SQNDR vs first stage OpAmp performance {G=80dB, SR=20V/μs, GBW=40MHz} (a) and {G=60dB, SR=8V/μs, GBW=20MHz} (b) for the $\Delta\Sigma$M of Fig. 10.

V. Automatic Circuit Optimization

According to the methodology of Fig. 1, here the design is already split into several circuit pieces for their optimization at transistor level against the target CMOS technology. Typically, this optimization process involves defining design parameters (i.e. size of devices and biasing conditions), figures-of-merit (FoMs) to be measured at each simulation iteration (i.e. performance against power and area resources), implicit rules for discarding solutions, and the cost function to score candidates. In this sense, the EDA environment proposed in Fig. 1 makes extensive use of the SpiceOpus built-in *optimize* command, a Nutmeg extension aimed to manage the full optimization process described above. Following the design case of Fig. 10, students have to optimize the CMOS OpAmp circuit of Fig. 12 for the first stage of our SC $\Delta\Sigma$M according to the specifications and load conditions collected from Section IV. The proposed OpAmp topology has been deeply studied [11], thus programming the *optimize* scripting is straightforward, as illustrated in Fig. 13. An example of optimization results can be seen in Fig. 14.

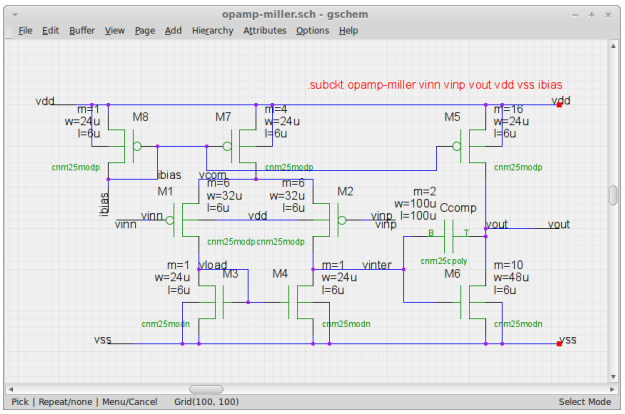

Figure 12. Single-ended two-stage Miller-compensated CMOS OpAmp schematic in *gschem* to be used for the SC $\Delta\Sigma$M topology of Fig. 10.

opamp_optimize.sp3

```
optimize
   parameter 0 @m1:xopamp[w] low 6u high 120u initial 32u
 " parameter 1 @m6:xopamp[m] low 1 high 10 initial 8
 " parameter 3 @ccomp:xopamp[w] low 25u high 250u
...
 " analysis 25 ac dec 50 10 10e6
 " analysis 26 let gmag=20*log10(mag(v(vout)))
 " analysis 27 let gph=phase(v(vout))
 " analysis 28 cursor c right gmag 0
 " analysis 29 let gbw=abs(frequency[%c])/1e6
 " analysis 30 let pm=180+gph[%c]
...
 " analysis 46 tran 1n 5u
 " analysis 47 cursor c right vout 2.1
 " analysis 48 let t1=time[%c]
 " analysis 49 cursor c right vout 2.9
 " analysis 50 let t2=time[%c]
 " analysis 51 let srpos=0.8/(t2-t1)*1e-6
...
 " implicit 0 op2.pd lt 1.5
 " implicit 1 op2.area lt 0.025
 " implicit 4 ac2.pm gt 60
 " implicit 5 tran2.srpos gt 12
 ...
 " cost 1/tran2.srneg+1/tran2.srpos+abs(60-ac2.pm)
 " method genetic elitism yes maxgen 1000
```

Figure 13. Portion of SpiceOpus *optimize* script for the automatic optimization of the SR and phase margin of the CMOS OpAmp circuit of Fig. 12.

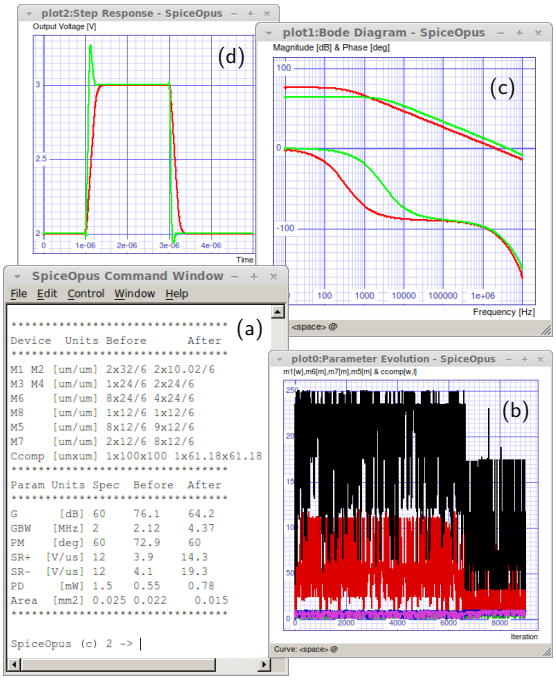

Figure 14. SpiceOpus results from the *optimize* script of Fig. 13: FoM comparison (a), design parameter iteration (b) and performance analysis (c,d).

VI. PCell-Based Layout Entry

With this step, the VLSI design methodology of Fig. 1 enters into the physical domain. The proposed EDA environment relays on Glade for the subsequent design steps. Apart from being a fully featured layout editor, as easily seen from Fig. 15, the internal Python scripting of Glade allows to customize most tasks, from verification rules to automated generation of layout structures. For the later, a complete set of parameterized cells (PCells) for CNM25 transistors and polySi-insulator-polySi (PiP) capacitors are developed, as shown in Fig. 16, so students can learn how to program their own PCells in order to save time during full-custom layout design while preserving analog matching properties [12]. As a side effect of PCell-based design, errors on the subsequent verification steps can be strongly reduced.

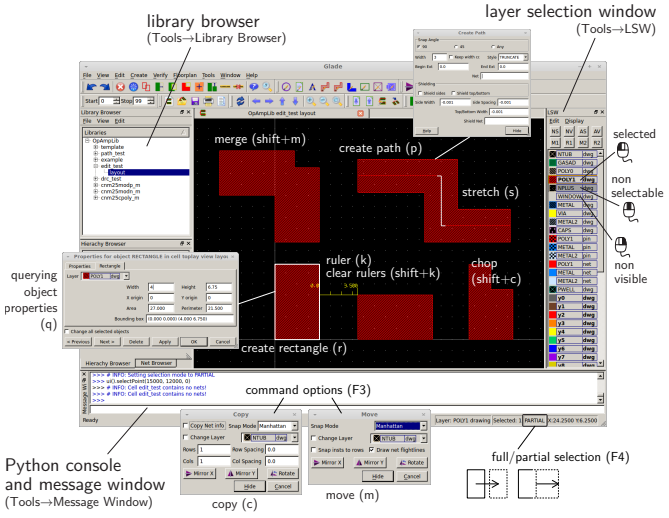

Figure 15. Glade IC layout editor.

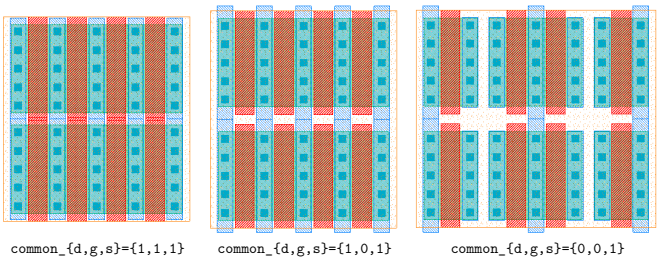

Figure 16. CNM25 NMOSFET layout examples from a custom Python PCells in Glade. In this case, parameters are m=4×2, w=28μm, l=6μm and common terminal options as indicated.

VII. Design Rule Checker

The first physical verification step in Fig. 1 is the design rule checker (DRC), which ensures VLSI layout is compliant from the lithographic point of view only. In this sense, Glade includes the user-friendly interface of Fig. 17 for debugging DRC errors. Furthermore, since the full set of design rules are defined by Python scripting, like in the example of Fig. 19, students can learn the real tasks behind DRC analysis (e.g. derived layers by boolean operations, geometrical measurements), and the complexity of programming a complete rule set to ensure full coverage in CMOS technologies.

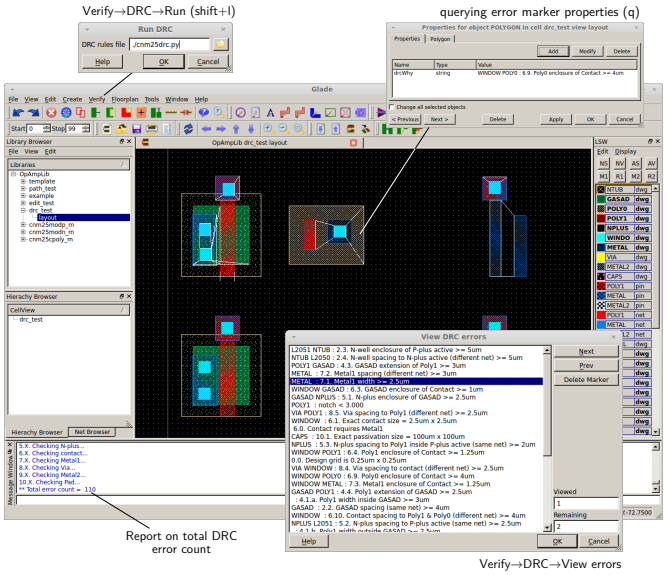

Figure 17. Example of DRC error debugging in Glade.

cnm25drc.py

```
...
active    = geomGetShapes("GASAD", "drawing")
polygate  = geomGetShapes("POLY1", "drawing")
polycap   = geomGetShapes("POLY0", "drawing")
gate      = geomAnd(polygate, active)
cpoly     = geomAnd(polygate, polycap)
geomOffGrid(polygate, 0.25, 1, "0.0. Design grid is ...
geomWidth(gate, 3, "4.1.a. Poly1 width inside GASAD >= ...
geomSpace(polygate, 3, diffnet, "4.2. Poly1 spacing...
geomNotch(polygate, 3, "4.2. Poly1 notch >= 3um")
geomExtension(polygate, active, 2.5, "4.4. Poly1 ext...
geomEnclose(polycap, cpoly, 3, "4.6. Poly0 enclosure...
...
```

Figure 18. Portion of CNM25 DRC rules file for Glade.

VIII. Layout Versus Schematic and Parasitics Extraction

Both layout versus schematic (LVS) and parasitics annotation steps of Fig. 1 require the extraction of the equivalent electrical circuit from its layout. Like in the case of DRC, Glade features circuit extraction driven by Phyton scripting, as shown in Fig. 19. Students can also browse extraction results directly in the layout editor for further electrical rule checking (ERC), following Fig. 20. Concerning LVS, verification is done by the venerable tool Gemini [13], which is already integrated in Glade. Again, students can debug any circuit connectivity or device size mismatching between schematic and layout in the same editor, as depicted in Fig. 21. Finally, Glade can export SPICE netlists with annotated parasitics from extraction, like in Fig. 22, for post-layout simulation with SpiceOpus.

cnm25xtr.py

```
...
geomLabel(polygate, "POLY1", "pin", 1)
geomLabel(polygate, "POLY1", "net", 0)
geomConnect([
  [cont, ndiff, pdiff, polygate, polycap, metal1],
  [via2, metal1, metal2]... ] )
extractMOS("cnm25modn", ngate, polygate, ndiff, pwell)
extractParasitic3(pdiff, metal2, cmetal2diff, 0,
  [metal1, polygate, polycap])
...
```

Figure 19. Portion of CNM25 extraction rules file for Glade.

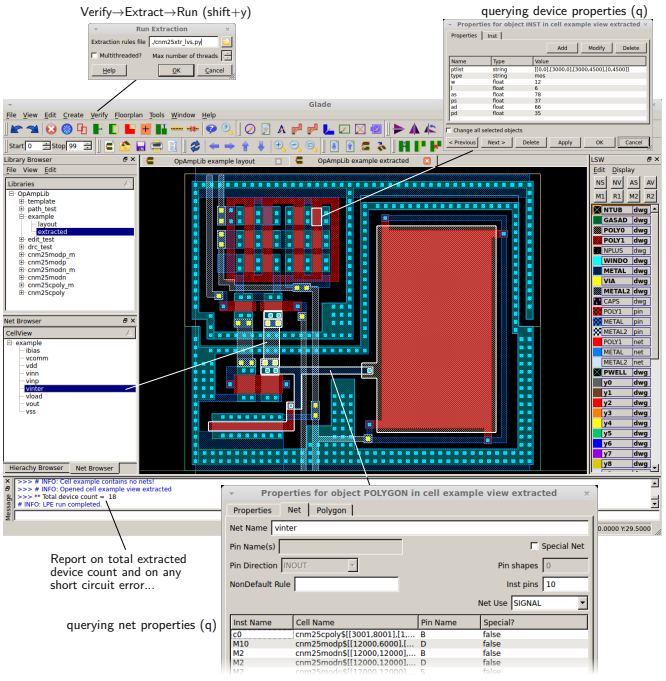

Figure 20. Extraction and ERC browsing in Glade for the OpAmp of Fig. 12.

IX. Conclusions

A complete EDA framework for teaching mixed-mode full-custom VLSI design is presented. The proposed freeware environment allows students to gain hands-on experience on schematic entry, both at system and circuit levels, HDL system simulation and block specification, automatic circuit optimization, PCell-based layout, physical verification, parasitics extraction, post-layout simulation and tape-out.

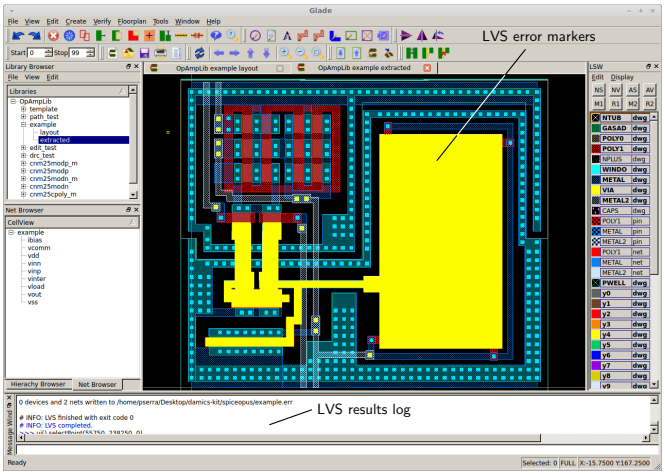

Figure 21. LVS error debugging in Glade for the OpAmp of Fig. 12.

opamp_par.sub

```
.SUBCKT opamp vinn vinp vout vdd vss ibias
MM0 vdd ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=...
MM1 vdd ibias vout vdd cnm25modp w=1.2e-05 l=6e-06 as=...
Cc0 vinter vout cnm25cpoly w=6.42928e-05 l=0.000156207
MM8 vout ibias vdd vdd cnm25modp w=1.2e-05 l=6e-06 as=...
...
CP1 vinter vss  C=3.8582e-13
CP2 vout ibias  C=3.33692e-15
CP3 vinp vss  C=1.85938e-15
CP4 vout vcomm  C=2.0918e-15
...
.ENDS
```

Figure 22. Portion of the SPICE netlist with annotated parasitics generated by Glade from the OpAmp layout of Fig. 20.

In order to illustrate the usage of this EDA framework, a practical design case based on a $\Delta\Sigma$ modulator for A/D conversion in a simple CMOS technology is described step-by-step.

References

[1] "IEEE Standard for the Scheme Programming Language," *IEEE Std 1178-1990*, 1991.

[2] A. Hvezda, *gEDA Project*. http://www.geda-project.org

[3] T. Tuma and Á. Bűrmen, *Circuit Simulation with SPICE OPUS: Theory and Practice*, ser. Modeling and simulation in science, engineering and technology. Birkhäuser Boston, 2009.

[4] T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE3 Version 3F3 User's Manual*, University of California, Berkeley, CA 94720, USA, May 1993.

[5] 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.

[6] K. Sabine, *Glade IC Layout Editor*. http://www.peardrop.co.uk/glade

[7] G. van Rossum, "Python Tutorial," Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Tech. Rep. CS-R9526, May 1995.

[8] J. Candy and G. Temes, *Oversampling Delta-Sigma Data Converters: Theory, Design, and Simulation*. Wiley-IEEE Press, 1991.

[9] J. Silva, U. Moon, J. Steensgaard, and G. C. Temes, "Wideband Low-Distortion Delta-Sigma ADC Topology," *IET Electronics Letters*, vol. 37, no. 12, pp. 737–738, Jun 2001.

[10] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, "Matching Properties of MOS transistors," *IEEE J. Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1440, Oct 1989.

[11] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*. Oxford University Press, 2002.

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

[13] C. Ebeling, N. McKenzie, and L. McMurchie, *The Gemini Users Guide*, University of Washington, Dec 1993.