TUTORIAL

CADENCE DESIGN ENVIRONMENT

Antonio J. Lopez Martin
alopmart@gauss.nmsu.edu

Klipsch School of Electrical and Computer Engineering
New Mexico State University
October 2002

SCHEDULE – CADENCE SEMINAR

MONDAY, OCTOBER 21

9:00H-9:30H. **Lecture**

Introduction to Cadence. Basic Features

9:30H-11:00H: **Lecture**

Schematic Edition and Circuit Simulation with Cadence DFWII

11:00H-11:15H: Break

11:15H-13:00H: **Lab session**

Schematic Edition and Simulation of an OTA

TUESDAY, OCTOBER 22

9:00H-11:00H. **Lecture**

Layout Edition and Verification with Cadence Virtuoso and Diva.

11:00H-11:15H: Break

11:15H-13:00H: **Lab session**

Layout of an OTA. Verification: DRC, LVS, post-layout simulation (First session)

WEDNESDAY, OCTOBER 23

9:00H-11:00H. **Lecture**

Advanced Layout Design
Transfer to foundry
Case study: a commercial IC designed with Cadence.

11:00H-11:15H: Break

11:15H-13:00H: **Lab session**

Layout of an OTA. Verification: DRC, LVS, post-layout simulation (Second session)

CONTENTS

APPENDIX: ADVANCED TOPICS

1. INTRODUCTION

This manual is intended to introduce microelectronic designers to the Cadence Design Environment, and to describe all the steps necessary for running the Cadence tools at the Klipsch School of Electrical and Computer Engineering.

Cadence is an Electronic Design Automation (EDA) environment that allows integrating in a single framework different applications and tools (both proprietary and from other vendors), allowing to support all the stages of IC design and verification from a single environment. These tools are completely general, supporting different fabrication technologies. When a particular technology is selected, a set of configuration and technology-related files are employed for customizing the Cadence environment. This set of files is commonly referred as a *design kit*.

It is not the objective of this manual to provide an in-depth coverage of all the applications and tools available in Cadence. Instead, a detailed introduction to those required for an analog designer, from the conception of the circuit to its physical implementation, is provided. References to other manuals and information sources with a deeper treatment of these and other Cadence tools are also provided.

2. ANALOG IC DESIGN FLOW AND REQUIRED TOOLS

Fig. 1 shows the basic design flow of an analog IC design, together with the Cadence tools required in each step.

First, a schematic view of the circuit is created using the Cadence *Composer Schematic Editor*. Alternatively, a text netlist input can be employed.

Then, the circuit is simulated using the Cadence *Affirma* analog simulation environment. Different simulators can be employed, some sold with the Cadence software (e.g., Spectre) some from other vendors (e.g., HSPICE) if they are installed and licensed.

Once circuit specifications are fulfilled in simulation, the circuit layout is created using the *Virtuoso Layout Editor*.

The resulting layout must verify some geometric rules dependent on the technology (design rules). For enforcing it, a *Design Rule Check* (DRC) is performed. Optionally, some electrical errors (e.g. shorts) can also be detected using an *Electrical Rule Check* (ERC). Then, the layout should be compared to the circuit schematic to ensure that the intended functionality is implemented. This can be done with a *Layout Versus Schematic* (LVS) check. All these verification tools are included in the *Diva* software in Cadence (more powerful Cadence tools can also be available, like *Dracula,* or *Assura* in deep submicron technologies).

Finally, a netlist including all layout parasitics should be extracted, and a final simulation of this netlist should be made. This is called a *Post-Layout simulation*, and is performed with the same Cadence simulation tools.

Once verified the layout functionality, the final layout is converted to a certain standard file format depending on the foundry (GDSII, CIF, etc.) using the Cadence conversion tools.

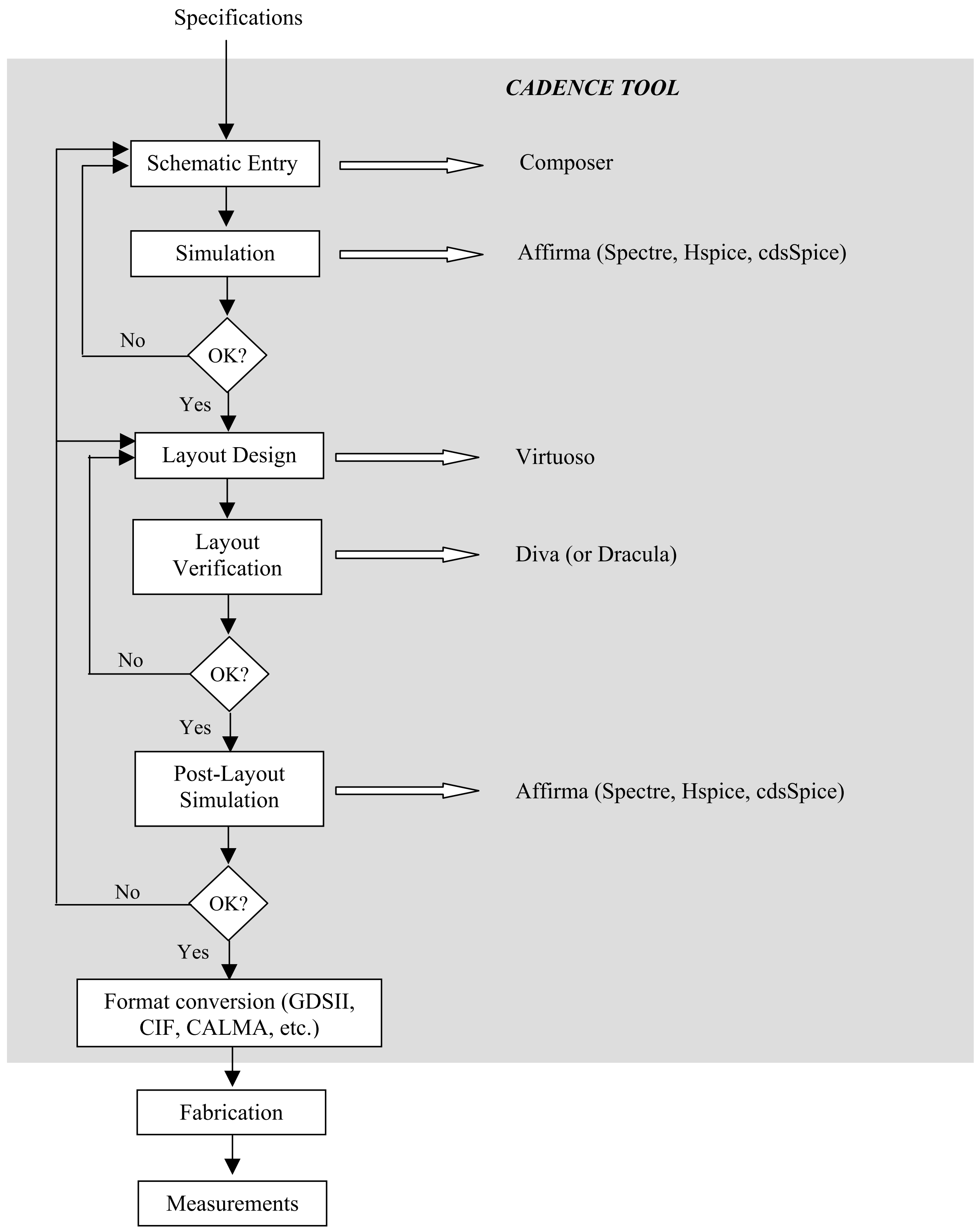


Figure 1. Analog IC design flow and Cadence tools involved

3. SETTING YOUR UNIX ENVIRONMENT

Cadence can be run only on Unix terminals or PCs loaded with Linux (or Unix terminal emulators) and *X* Windows servers like Exceed, X-Win32, or Xfree (Linux). Also the VNC software can be employed (see the accompanying document about VNC tools). You need to have a **tesla** account. If you are using Unix emulators, you can remote logon to the host **tesla.nmsu.edu** using the SSH protocols. It your emulator does not support SSH connections, you can first start a telnet session to **gauss**, and then connect to **tesla** with SSH. Before you can start Cadence, there are a few configuration files that are needed in your home directory. These files determine the environment in which Cadence runs, what libraries are to be included in your current session, etc. No doubt these files can be edited to suit personal preferences. The setup given below is tailored for **tesla** (the machine hosting the Cadence environment at the Department), and for the **NCSU Design Kit**. This kit can be freely obtained from the North Carolina State University, and provides all the necessary technology files and simulation models for using several technologies available through MOSIS.

Once in **tesla**, if this is the first time that you are going to use Cadence, copy the required setup files (**.cdsinit, .cdsenv, .cdsplotinit, .simrc**) and scripts (**runNCSU**) from the directory /**Kits/NCSU/newuser**. You can use the following commands, from your home directory (remember that the Unix commands are case sensitive):

```
Bash-2.05a$ cp /Kits/NCSU/newuser/.cds* .
Bash-2.05a$ cp /Kits/NCSU/newuser/.simrc .
Bash-2.05a$ cp /Kits/NCSU/newuser/runNCSU .
```

It is also recommended that this first time you create a new directory for working with Cadence, so that all the files generated by Cadence will be in that directory. This will be your Cadence working directory in all subsequent sessions, and you will start Cadence from there. E.g., you can type:

```
Bash-2.05a$ mkdir cadence
```

The next step is to enter the directory where you work with Cadence, for instance:

```
Bash-2.05a$ cd cadence
```

Before running Cadence, you have to instruct **tesla** so that the Cadence windows can be displayed in your PC. You can type:

```
Bash-2.05a$ DISPLAY=yourIPaddress:0
Bash-2.05a$ export DISPLAY
```

where **yourIPaddress** means the IP address of your PC (if you don't know it, you can get it, e.g., from http://www.accessam.com/myip.shtml). You can also use the DNS address.

4. RUNNING CADENCE

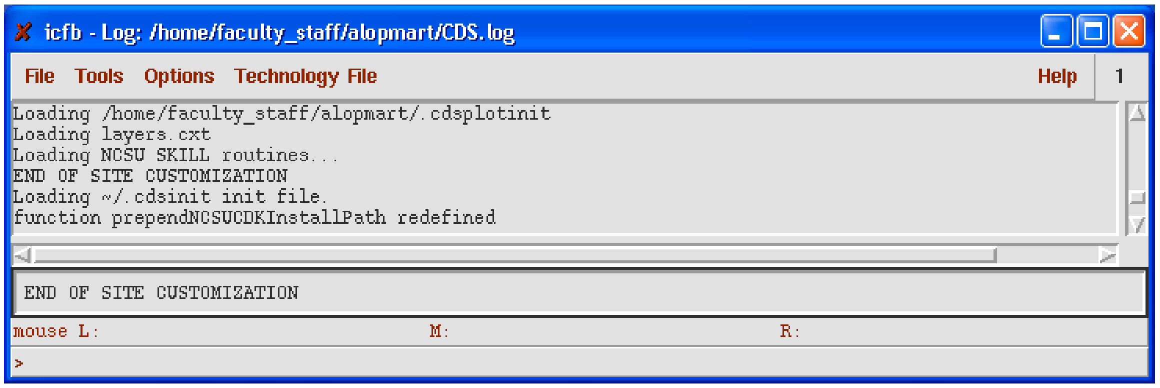
Once your Cadence environment setup you can start working with Cadence. You can run Cadence from your working directory by typing:

```
Bash-2.05a$ ~/runNCSU
```

or, equivalently, if the parent directory is your home directory,

```
Bash-2.05a$ ../runNCSU
```

The Cadence main window (**Common Interface Window, CIW**) and the **Library Manager Window** are opened. From the CIW menus, all Cadence main tools, online help and options can be accessed. In the window area, all kind of messages (info, errors, warnings, etc) generated by the different Cadence tools appear. You can also introduce commands. Fig. 2 shows this window.


icfb - Log: /home/faculty_staff/alopmart/CDS.log
File Tools Options Technology File Help 1

```
Loading /home/faculty_staff/alopmart/.cdsplotinit
Loading layers.cxt
Loading NCSU SKILL routines...
END OF SITE CUSTOMIZATION
Loading ~/.cdsinit init file.
function prependNCSUCDKInstallPath redefined
```

```
END OF SITE CUSTOMIZATION
```

```
mouse L:                         M:                          R:
>
```

Figure 2. CIW window

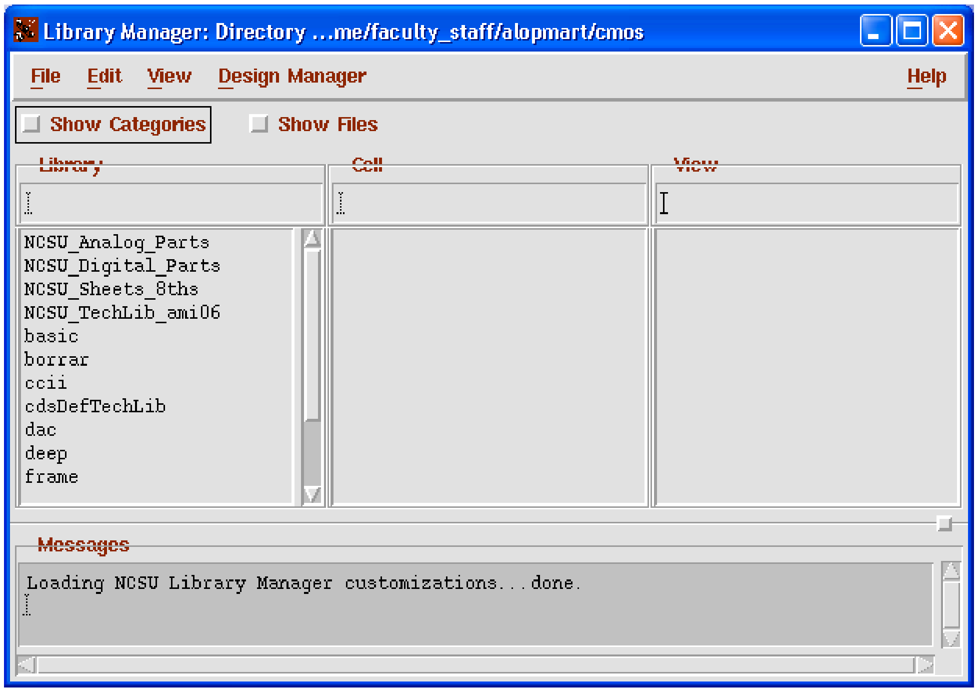
All the entities in Cadence are managed using **libraries**, and each library contains **cells**. Each cell contains different design **views** (the structure is similar –and physically corresponds - to a directory (library) containing subdirectories (cells), each one containing files (views). Thus, for instance, a certain circuit (e.g. an ADC) can be stored in a library, and such library can contain the different ADC blocks (comparators, registers, resistor strings, etc) stored as cells. Each block (cell) contains different views (schematic, layout, etc.).

There are usually three types of libraries:

- A set of common Cadence libraries that come with the Cadence software (containing basic components, such as voltage and current sources, R, L, C, etc).
- Libraries that come with a certain design kit and that are related to a certain technology (e.g. transistors with a certain model attached, etc). In the NCSU design kit, some general Cadence libraries are customized and converted into design kit libraries.
- User libraries; where the user stores its designs. These designs employ components from the Cadence/design kit libraries.

All the libraries are managed from the Library Manager Window (shown in Fig. 3). It appears by default at Cadence start, and can be opened at any time by selecting ***Tools>Library Manager...*** from the CIW. Libraries with names starting by NCSU are design kit libraries, and contain basic components for building designs. In Fig. 3, they are:

- *NCSU_Analog_Parts*: contains all the required blocks for designing an analog schematic (sources, GND and VDD terminals, transistors, R, L, C, diodes, etc.)
- *NCSU_Digital_Parts*: similarly, it contains all parts required for digital design (logic gates, muxes, etc.)
- *NCSU_Sheets_8ths:* it contains informative sheet borders for the schematics, in different sheet sizes. Its use is optional, for design documentation.
- *NCSU_TechLibAMI06:* it is a technology-specific library that is created when the user attaches this particular technology (AMI06) to a user library.



NCSU_Analog_Parts
NCSU_Digital_Parts
NCSU_Sheets_8ths
NCSU_TechLib_ami06
basic
borrar
ccii
cdsDefTechLib
dac
deep
frame

```
Loading NCSU Library Manager customizations...done.
```

Figure 3. Library Manager window

5. ANALOG DESIGN WITH CADENCE DESIGN FRAMEWORK II

Now we are going to illustrate how to carry out the complete design flow shown in Fig. 1 using the Cadence tools. A simple Operational Transconductance Amplifier (OTA) will be designed in the AMI 0.5μm CMOS technology. However, the same procedures apply to complete chip designs.

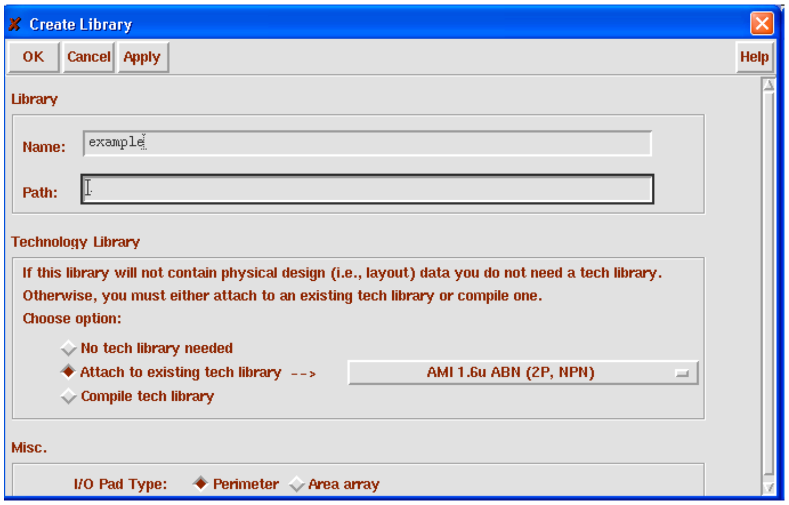
5.1. Library creation and selection of technology

It is recommended that you use a library to store related cell views; e.g., use a library to hold all the cell views for a single project (that can involve a complete chip design). In our example, we are going to create a new library for our design. From the CIW or from the Library Manager window,

a) Select ***File -> New -> Library***. A new window appears (see Fig. 4).
b) Enter a library name, e.g., **example**.
c) Enter the absolute path name if you want the library created somewhere else than the working directory.
d) Choose the **Attach to an existing techfile** option.
e) Choose your technology; for instance, for AMI 0.5μm in MOSIS, choose **AMI 0.6u C5N**

NOTE: The NCSU Kit 1.6u and 0.6u AMI processes correspond to MOSIS' 1.5μ and 0.5μ. For consistency, the NCSU kit names its tech libs based on the drawn length of devices, so this sometimes runs afoul of MOSIS' naming scheme.

This will be the technology chosen for your design (that you will employ eventually for fabrication). Now all the designs made in this library are technology-dependent (e.g., the schematic MOS symbols have by default the model for this technology, the available layout layers correspond to this technology, etc.).



Library

Name: example

Path:

Technology Library

If this library will not contain physical design (i.e., layout) data you do not need a tech library.
Otherwise, you must either attach to an existing tech library or compile one.
Choose option:

No tech library needed [✓ Attach to existing tech library] --> AMI 1.6u ABN (2P, NPN) Compile tech library

Misc.

I/O Pad Type: [✓ Perimeter] Area array

Figure 4. Create Library window

5.2. Schematic Entry with Composer

5.2.1. Transistor-level schematic

The traditional method for capturing (i.e. describing) your transistor-level or gate-level design is via the Composer schematic editor. Schematic editors provide simple, intuitive means to draw, to place and to connect individual components that make up your design. The resulting schematic drawing must accurately describe the main electrical properties of all components and their interconnections. Also included in the schematic are the power supply and ground connections, as well as all "pins" for the input and output signals of your circuit. This information is crucial for generating the corresponding netlist, which is used in later stages of the design. The generation of a complete circuit schematic is therefore the first important step of the former design flow. Usually, some properties of the components (e.g. transistor dimensions) and/or the interconnections between the devices are subsequently modified as a result of iterative optimization steps. These later modifications and improvements on the circuit structure must also be accurately reflected in the most current version of the corresponding schematic.

Now you are going to create the OTA schematic. From the CIW or from the Library Manager window,

a) Select the library name that you just created, e.g., **example**.
b) Select ***File -> New -> Cellview***
c) Enter a cell name, for instance, **OTA**
d) Choose **Composer - Schematic** as the **Tool**. View name should be *schematic*.
e) Click **OK**.

An empty blank Composer - schematic window should open. In this window you will create your schematic. The final schematic is shown in Fig. 5 (the icon actions are also shown). To create it, you can employ the window top menus or left icons (or also shortkeys). This multiple access to actions is common to all the Cadence tools. A detailed information concerning the use of the schematic editor can be obtained by selecting **Help** from this window. Basically, you can:

- *Create components:* by selecting the **Instance** icon and browsing in the pop-up window through the different libraries. Most components (transistors, R, L, C, sources, rail terminals, etc.) are in the *NCSU_Analog_Parts* library. When you create an instance of a certain component (e.g., transistor, R, C, etc.) a window appears where you can select the properties of this element. *Note that for transistors, the model is automatically set according to the technology you have selected, and that some parameters (drain and source area and perimeter, etc) are automatically calculated.*

NOTE: There are three-terminal (nmos, pmos) and four-terminal (nmos4, pmos4) MOS devices available. The three-terminal devices have a hidden fourth (bulk) terminal, which is connected by default to "gnd!" for nmos and "vdd!" for pmos. This means that you need to have nets named "vdd!"and "gnd!" somewhere in your schematic. (The easiest way to do this is to drop in the vdd and gnd pins from NCSU_Analog_Parts->Supply_Nets.) If you don't have these nets in your schematic somewhere, you'll get complaints about unknown nets either in the netlister or in LVS.
(You can also change the bulk node in the three-terminal device if you want to. Just select the device in the schematic, and bring up the "Edit Object Properties" form (Edit->Object->Properties...). Change the "Bulk node connection" field to whichever net you want.

- *Wire components:* select the **Wire (narrow)** icon, click to the first terminal and drag until the other terminal, then click again.

  In complex designs, for avoiding excessive wiring, labels can be employed. When two wire ends are labeled with the same name, they are effectively connected. Labels are created, e.g., with the **Label** icon.

- *Set instance properties*: you can modify at any time the properties of a certain component instance (resistance value, transistor dimensions, etc), e.g., by selecting the instance (click on it) and then clicking on the **Properties** icon. You can also change a group of instances of the same component simultaneously, by first selecting this group, then clicking in Properties and modifying the parameters, selecting **Apply to > All selected**.

Most of the commands in Composer will start a mode (the default mode is selection), and as long as you do not choose a new mode (by clicking an icon, pressing a shortkey or selecting a menu item) you will remain in that mode. To quit from any mode and return to the default selection mode, the *"Esc"* key can be used.

There are basically two ways of creating schematics:

a) **Non-Hierarchical schematic**. You introduce all the circuit schematic at the same (transistor) level, including the required sources. This is only viable for small designs.

b) **Hierarchical schematic**. If your design is very large, or if you want to reuse your design in other designs (e.g. an OpAmp to be employed in other circuits), you should create basic blocks at a low level, then create symbols for them and then use these symbols as basic components at a higher hierarchical level of the design. It is the same concept employed in SPICE netlists with the .SUBCKT command.

When a certain circuit design consists of smaller hierarchical components (or modules), it is usually very beneficial to use this approach, first identifying such modules early in the design process and then assigning each such module a corresponding symbol (or icon) to represent that circuit module. This step largely simplifies the schematic representation of the overall system. The "symbol" view of a circuit module is an icon that stands for the collection of all components within the module.

A symbol view of the circuit is also recommended for some of the subsequent simulation steps; thus, the schematic capture of the circuit topology is usually followed by the creation of a symbol to represent the entire circuit. The shape of the icon to be used for the symbol may suggest the function of the module (e.g. logic gates - AND, OR, NAND, NOR), but the default symbol icon is a simple rectangular box with input and output pins. Note that this icon can now be used as the building block of another module, and so on, allowing the circuit designer to create a system-level design consisting of multiple hierarchy levels.

This is the approach that will be followed in our example. For doing this, we have to create a symbol view of our OTA.

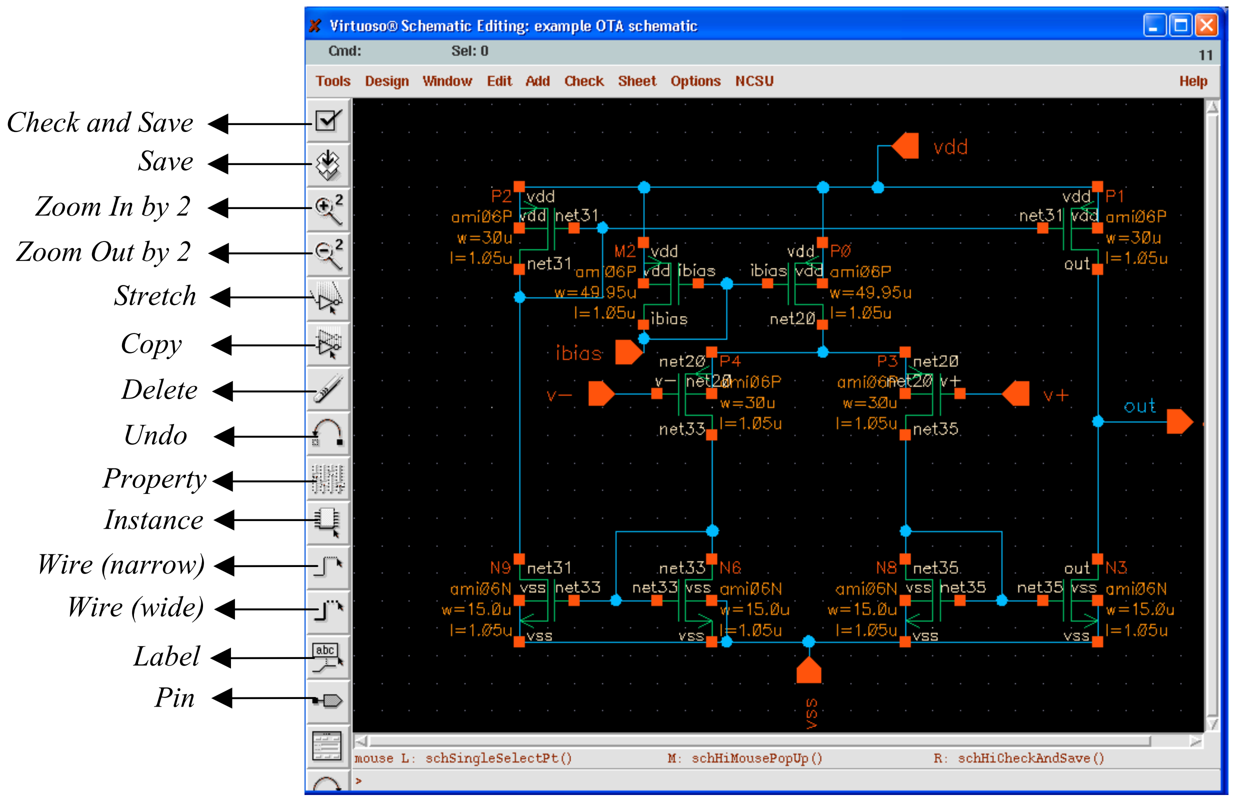
5.2.2. Symbol Creation

a) First, you have to create pins in your schematic in those non-global nets (inputs, outputs, supplies perhaps) that have to be accessible outside the symbol. For doing this, you can select the **Create Pin** icon in the Composer schematic window, select the pin name, its input or output configuration, etc, and then click at the end of the wire when the pin has to be placed. Six pin icons have been placed in the example of Fig. 5: **v+, v-, ibias, vdd, vss** (input pins) and **out** (output pin).

b) Now, select ***Design > Create Cellview > From cellview***. A window appears that by default creates a symbol view from the schematic view. Just click OK. A black window pops up with the symbol. It can be edited if desired (changing shape, pin distribution, etc.). Fig. 6 shows the resulting symbol (after some editing) and the actions associated to the left icons.

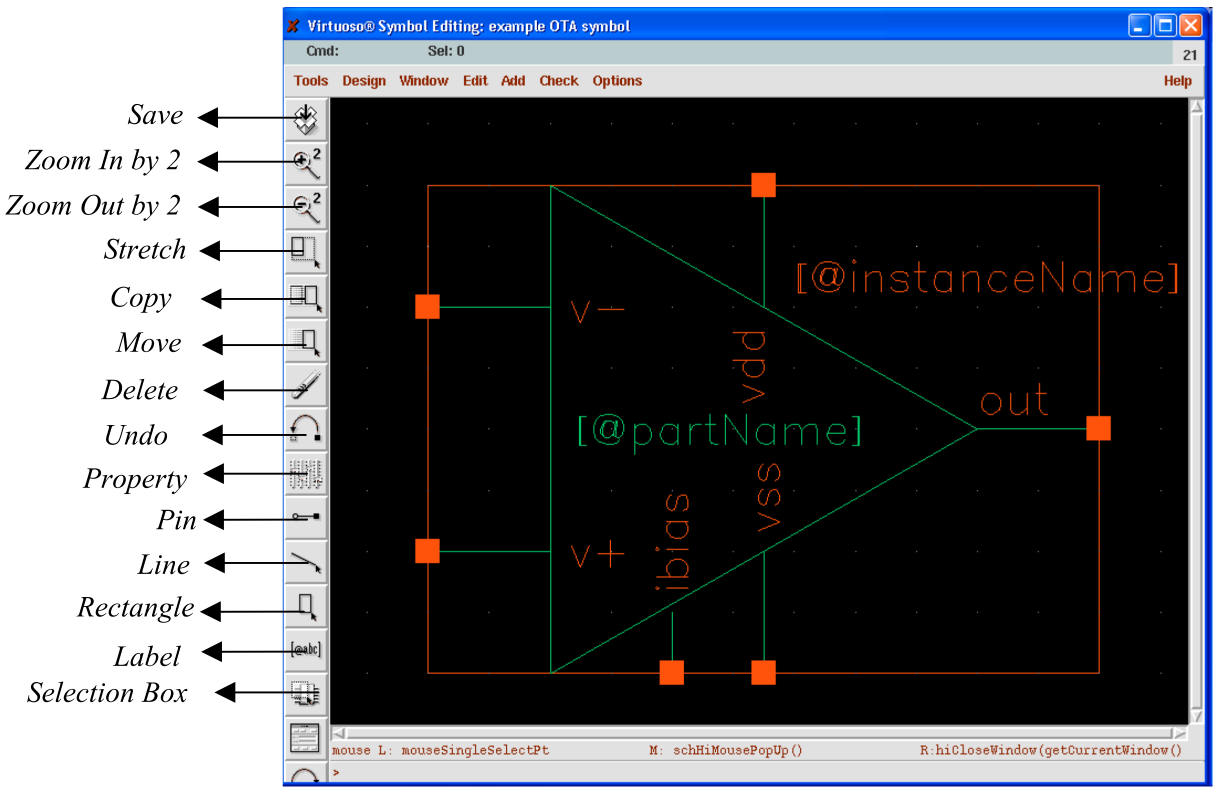
Now you can use your design in a higher hierarchical schematic. For instance, in Fig. 7 a new schematic is created where the OTA symbol is instantiated and some sources are included, forming a voltage follower ready to be simulated. This will be the design that we will simulate subsequently.

For instantiating the OTA symbol, the procedure is identical than for any other component. You can select the **Instance** icon, go to your library and select the OTA.



```
mouse L: schSingleSelectPt()          M: schHiMousePopUp()          R: schHiCheckAndSave()
```

Figure 5. Composer Schematic Editor window



```
mouse L: mouseSingleSelectPt          M: schHiMousePopUp()          R:hiCloseWindow(getCurrentWindow()
```

Figure 6. Symbol Editor window

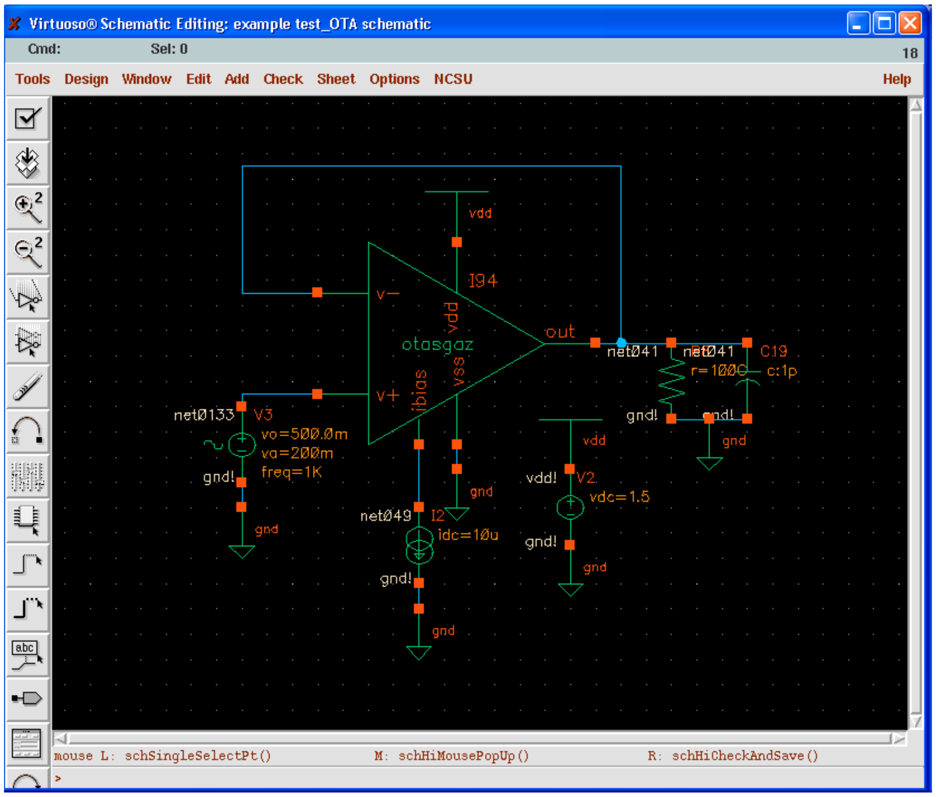


Figure 7. Composer Schematic Editor window, hierarchical design

We can go up and down in the hierarchy with menu options or with shortkeys **X** (or **x**, to descend) and **b** (to ascend). Parameters for voltage and current sources are set as for any other component, by selecting the instance and editing its properties.

5.3. Simulation

After the transistor-level description of a circuit is completed using the Schematic Editor, the electrical performance and the functionality of the circuit must be verified using a Simulation tool. The detailed transistor-level simulation of your design will be the first in-depth validation of its operation, hence, it is extremely important to complete this step before proceeding with the subsequent design optimization steps. Based on simulation results, the designer usually modifies some of the device properties (such as transistor width-to-length ratio) in order to optimize the performance.

The initial simulation phase also serves to detect some of the design errors that may have been created during the schematic entry step. It is quite common to discover errors such as a missing connection or an unintended crossing of two signals in the schematic. Some of these errors (e.g., floating nodes) can be detected even before simulation, by pressing the ***Check and Save*** icon in the schematic window.

The second simulation phase will follow the "extraction" of a mask layout (post-layout simulation), to accurately assess the electrical performance of the completed design.

Like in other simulation environments, it is the netlist text file *extracted* from the schematic (or layout) what is actually simulated.

In order to start simulations, from the Composer window that contains the schematic you want to simulate (in our example, *test_OTA*, shown in Fig. 7), choose ***Tools > Analog Environment***. The simulation window appears (Fig. 8).

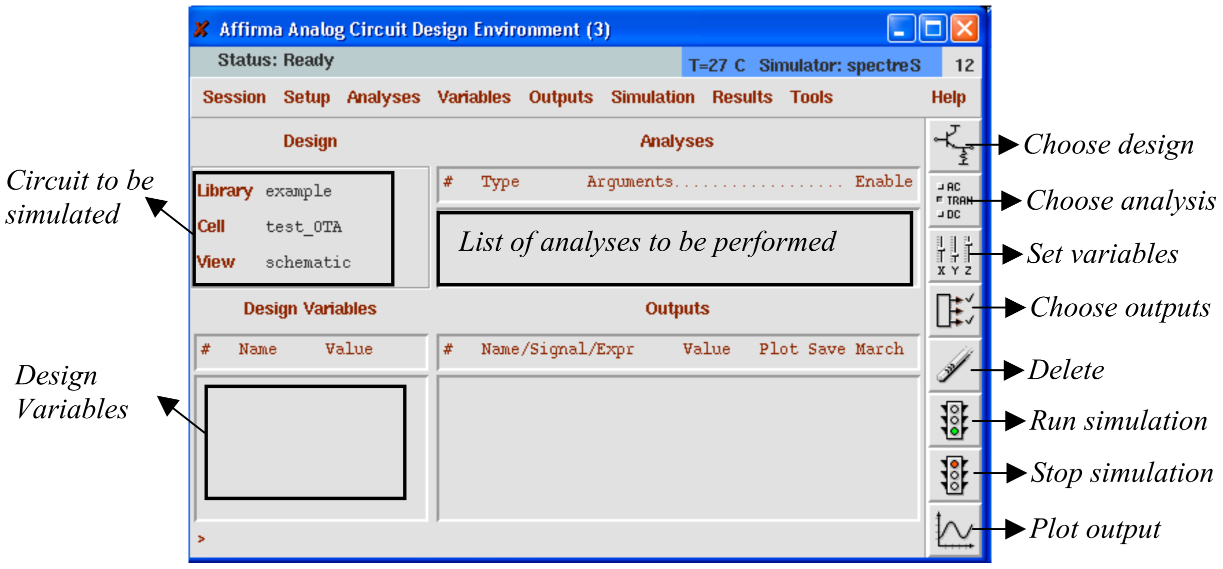

Affirma Analog Circuit Design Environment (3)
Status: Ready
T=27 C Simulator: spectreS
12
Session Setup Analyses Variables Outputs Simulation Results Tools Help
Design
Analyses
Library example
Cell test_OTA
View schematic
# Type Arguments.................. Enable
List of analyses to be performed
Design Variables
Outputs
# Name Value
# Name/Signal/Expr Value Plot Save March
Circuit to be simulated
Design Variables
Choose design
Choose analysis
Set variables
Choose outputs
Delete
Run simulation
Stop simulation
Plot output

Figure 8. Simulation window

By default, the design from which we have launched the window can be simulated using Spectre.

5.3.1. Setting simulator

We can change the design with the corresponding icon or using the menus. We can also change the simulator by choosing ***Setup->Simulator/Directory/Host*** and setting e.g., HSPICE. Obviously, the chosen simulator must be installed and licensed.

5.3.2. Setting models

Also by default the component models for the library technology (in our example AMI 0.5µ) will be used. If we want to use other models, we can store them in some directory and choose ***Setup->Model Path***, typing the full path (including filename) of the model(s) needed for simulation, before the path for the default models (list position means precedence in the search for models). This procedure assumes that your models have the same name as the default models.

5.3.3. Setting design variables

We can use in our schematic design variables for the component parameters, so that their values can be assigned just before simulation. For instance, we can name some (or all) of the transistor lengths as ‘L’. Then, in the simulation window we assign the desired value to L. This way, we can make several simulations just changing the parameter(s) without having to edit the schematic. For editing design variables, choose ***Variables >Edit*** (or select the corresponding icon).

5.3.4. Selecting the analysis

Several analyses can be performed (DC, AC, transient, etc.). For selecting the required analysis, choose ***Analysis > Choose*** (or the corresponding icon) and complete the settings in the window that appears.

Example: Transient analysis, choose 'tran', 'conservative' and then type in the total time you wish to run (ex. 40n or 40e-9)

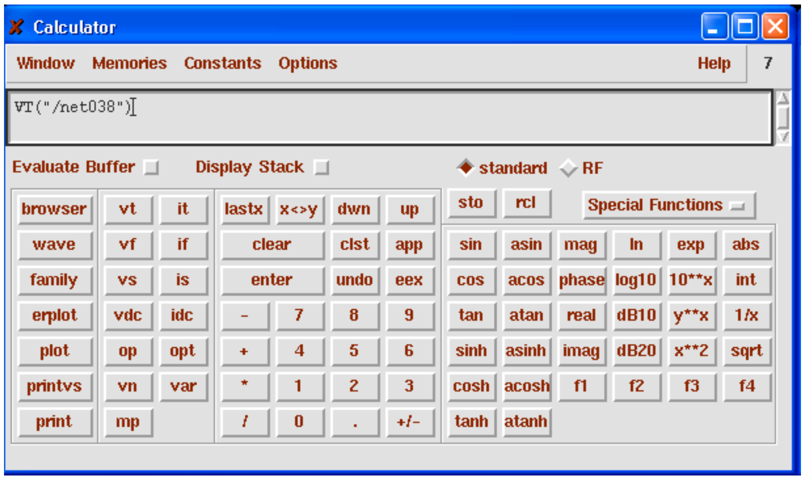
Currents cannot be plotted by default. If you are interested in viewing currents in your circuit, choose*:* ***Output > Save all ....> Select all DC/transient terminal currents*** (for DC and transient analyses) or ***Select all AC terminal currents*** (for AC analysis).

5.3.5. Running the simulation

When the former steps are performed, simulation can start. Click the Green Traffic Light Icon (bottom right corner of the simulation window) or choose ***Run > Simulation*** to run the simulation. If you want to interrupt the simulation at any instant, click the Red Traffic Light Icon or choose ***Run > Interrupt.***

5.3.6. Plotting the simulation results

There are different ways of plotting the results of a simulation. Here we are going to use the **Calculator** tool. After running the simulation, choose ***Tools > Calculator***. A calculator window appears (see Fig. 9).


Calculator
Window Memories Constants Options Help 7
Evaluate Buffer Display Stack standard RF
browser vt it lastx x<>y dwn up sto rcl Special Functions
wave vf if clear clst app sin asin mag ln exp abs
family vs is enter undo eex cos acos phase log10 10**x int
erplot vdc idc - 7 8 9 tan atan real dB10 y**x 1/x
plot op opt + 4 5 6 sinh asinh imag dB20 x**2 sqrt
printvs vn var * 1 2 3 cosh acosh f1 f2 f3 f4
print mp / 0 . +/- tanh atanh

```
VT("/net038")
```

Figure 9. Calculator

With this tool we can, among other things:

- Plot in a waveform window the different currents and voltages
- Print (to a printer or a file) the selected waveforms
- Perform different functions on the selected waveform (multiply, divide, dB calculation, DFT, THD calculation, bandwidth, maximum and minimum calculation, etc).
- Use as a normal calculator for making calculations

You can select the **Help** option for details.

When we want to display a certain waveform, we first select the button corresponding to the type of waveform. The most common are:

Vt: nodal voltage (transient analysis)

It: terminal current (transient analysis)

Vf: nodal voltage (AC analysis)

If: terminal current (AC analysis)

Vs: nodal voltage (DC sweep)

Is: terminal current (DC sweep)

Vdc: nodal voltage (quiescent value)

Idc: terminal current (quiescent value)

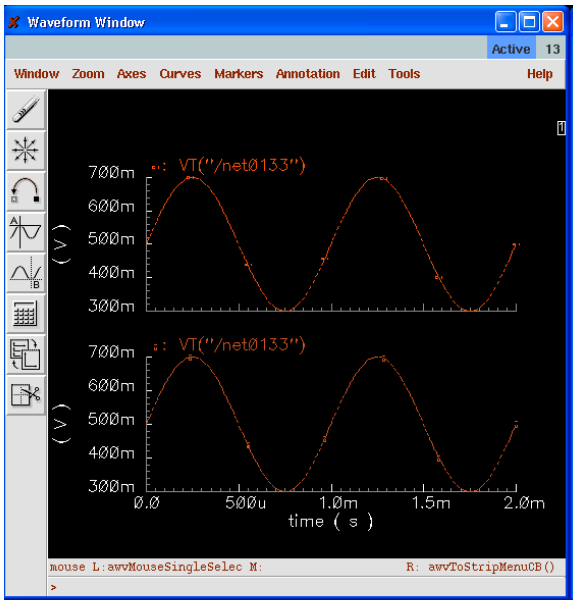
Then, we just click on the corresponding wire (voltages) or terminal (currents) in the schematic. Finally, we select in the calculator:

Plot: To plot the waveform without removing already displayed waveforms
Erplot: To remove displayed waveforms and plot the selected one.

The waveform is plotted in the Waveform Window. We can continue selecting circuit nodes and selecting **plot** or **erplot**. Finally, press *Esc* for disabling the current action.

The Waveform Window, showing the input and output voltages of our voltage buffer, is shown in Fig. 10. Different configurations can be chosen (various graphs or axes, cursors, etc.). Press the **Help** button for details.

If you want to modify and re-simulate your circuit, just change your schematic, design variables or simulation command, run again the simulation and select ***Window > Update results*** in the Waveform Window.



mouse L: awvMouseSingleSelec M: R: awvToStripMenuCB()

Figure 10. Waveform window

5.4. Layout

The creation of the mask layout is one of the most important steps in the full-custom (bottom-up) design flow, where the designer describes the detailed geometry and the relative positioning of each mask layer to be used in actual fabrication, using a **Layout Editor**. Physical layout design is very tightly linked to overall circuit performance (area, speed and power dissipation) since the physical structure determines the transconductances of the transistors, the parasitic capacitances and resistances, and obviously, the silicon area that is used to realize a certain function.

The physical (mask layout) design is an iterative process, which starts with the circuit topology and the initial sizing of the transistors. It is extremely important that the layout design must not violate any of the **Layout Design Rules** of the fabrication process, in order to ensure a high probability of defect-free fabrication of all features described in the mask layout. It is also important to **Extract** the netlist underlying the layout view, for two main purposes:

- This allows comparing it with the netlist extracted from the schematic. This **Layout versus Schematic (LVS)** comparison ensures that the layout actually implements the required functionality.
- If the extraction program allows to extract also parasitics from the layout view, a more accurate, **Post-Layout Simulation,** can be performed taking into account the geometry of the circuit.

The detailed mask layout requires a very intensive and time-consuming design effort, so that automated tools are employed as much as possible. Typically, the design of digital circuits based on single-clock synchronous logic is completely automated. First, a circuit description (typically in a HDL language as VHDL or Verilog) is synthesized, leading to a gate-level circuit description. From this gate-level netlist, Place&Route programs generate automatically the layout.

Unfortunately, analog circuits are very sensitive to the layout style, so that an automated procedure is difficult to implement. Usually a tedious, full-custom approach is followed, where the designer builds manually the layout, basically drawing rectangles of different layers. Nevertheless, there are some aids to the full-custom designer:

- *Parametrized Cells (Pcells)*. They are complete layout cells (e.g. a transistor or a row of contacts) already available, containing all the required layers, and whose characteristics can be set by the designer by means of some parameters (e.g., transistor length or width, etc.). They can be implemented in Cadence using the SKILL language, and some of them come usually with a certain Design Kit.
- *Standard Cells*. Cells already designed that can be placed in our layout. In case of digital cells, they are usually automatically placed with a Place&Route program.

  The availability of parametrized and standard cells in a certain Design Kit is an important factor for its quality (and price!).
- *"Semi-automated" layout*. The layout of the components in the schematic is automatically generated from the schematic, with its corresponding sizes. The designer just places and routes such components. Even for assisting place and route tasks there are additional features available. This can be done in Cadence with the *Virtuoso Accelerator (Virtuoso XL)*.

We will now make the layout of our OTA. From the CIW or from the Library Manager window,

a) Select the library name that you just created, e.g., **example**.
b) Select ***File -> New -> Cellview***

c) Enter the cell name, e.g., **OTA**
d) Choose **Virtuoso** as the **Tool**. View name should be layout.
f) Click **OK**. Two design windows will pop up (see Fig. 11), LSW and Virtuoso Editor.

LSW

The **Layer Selection Window (LSW)** lets the user select different layers of the mask layout. Virtuoso will always use the layer currently selected in the LSW for editing. The LSW can also be used to restrict the type of layers that are visible or selectable. To select a layer, simply click on the desired layer within the LSW.

WARNING: Only layers with the dg property will be fabricated. The rest are basically for labeling, highlighting errors, and documenting!!

- **Common Layers**. The following layers are common to all MOSIS CMOS processes:

NCSU Kit layer name	Description	Stream layer number	CIF layer abbreviation
nwell	N-well	42	CWN
pwell	P-well	41	CWP
active	active (diffusion)	43	CAA
nactive*	N-active	43	CAA
pactive*	P-active	43	CAA
nselect	N-select (ion implant)	45	CSN
pselect	P-select (ion implant)	44	CSP
poly	polysilicon	46	CPG
metal1	first-layer metal	49	CMF
ca	metal1-active contact (obsolete; use cc instead)	48	CCA
cp	metal1-polysilicon contact (obsolete; use cc instead)	47	CCP
cc	generic contact (metal1 to active or polysilicon)	25	CCC
via	metal1-metal2 contact	50	CVA
metal2	second-layer metal	51	CMS
pad	wirebond pad marker	26	XP
glass	overglass cut	52	COG
cap_id	capacitor marker	**NA**	**NA**
res_id	resistor marker	**NA**	**NA**
dio_id	diode marker	**NA**	**NA**

**nactive and pactive are simply convenience layers for the user, not mask layers, and are treated as ``active" for purposes of streaming out, DRC, and extraction.*

Table I. Common layers in all MOSIS processes

- **Layers corresponding to optional technology features.** Some specific additional layers can be included for implementing optional features of the technology. Table II shows which active processes support which optional technology features. Table III shows the mask layers that correspond to the technology features.

Technology Feature	AMI 1.6um	AMI 0.6um	HP 0.6um	HP 0.4um	TSMC 0.4um (4M)	TSMC 0.4um (4M2P)	TSMC 0.3um	TSMC 0.2um
electrode	■	■				■		
poly capacitor								
high-res implant		■						
npn	■							
third-layer metal		■	■	■	■	■	■	■
fourth-layer metal				■	■	■	■	■
fifth-layer metal							■	■
sixth-layer metal								■
metal-metal cap							■	■
silicide block			■		■		■	■
thin-oxide cap			■					
high-voltage FETs					■	■	■	■

Table II. Optional features in some MOSIS processes

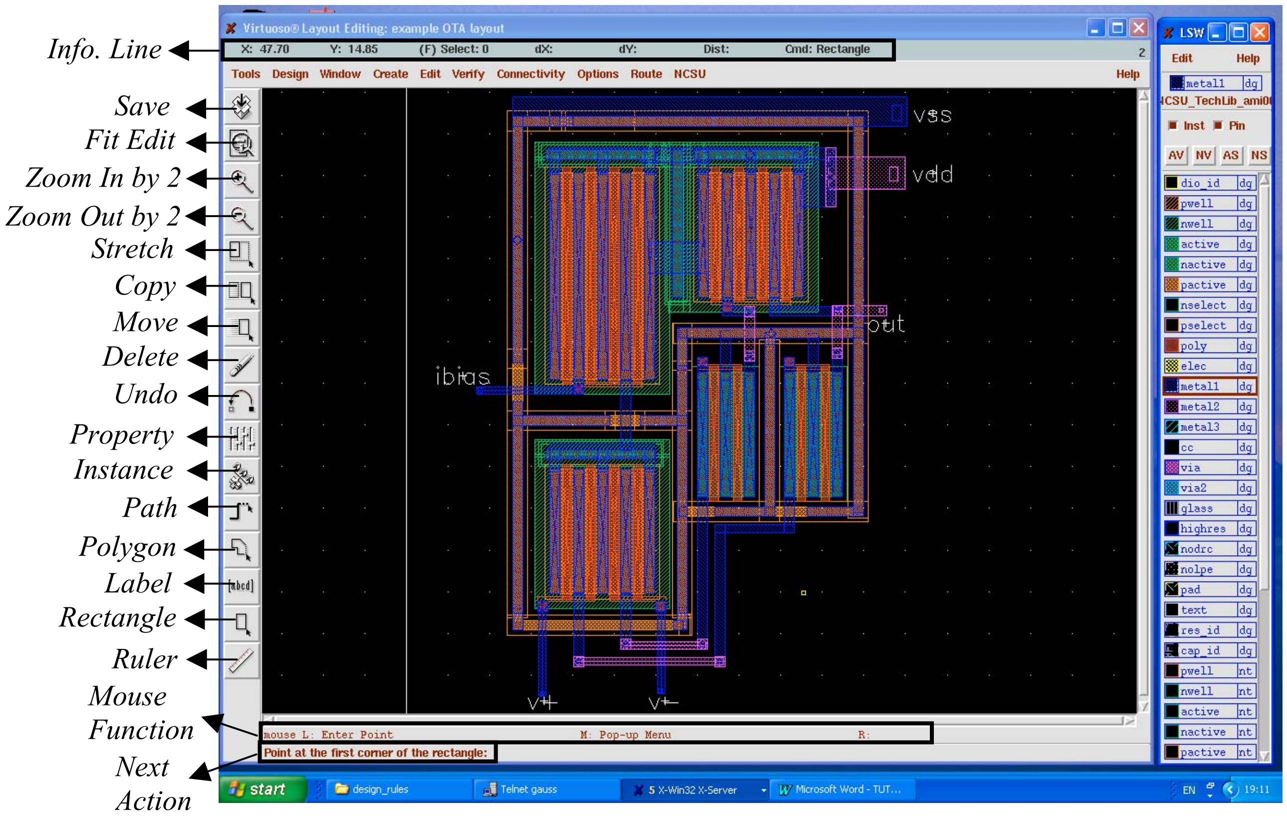
Technology Feature	CDK layer names	Description	Stream layer number	CIF layer abbreviation
electrode	elec	second polysilicon	56	CEL
	ce	metal1-electrode contact (obsolete; use cc instead)	55	CCE
poly capacitor	polycap	lower poly for thin-oxide capacitor	28	CPC
high-res implant	highres	implant for high-resistance elec (poly2)	34	CHR
npn	pbase	base for vertical NPN transistors	58	CBA
	cactive*	marker for collector of NPN transistors	43	CAA
third-layer metal	metal3	third metal	62	CMT
	via2	metal2-metal3 contact	61	CVS
fourth-layer metal	metal4	fourth metal	31	CMQ
	via3	metal3-metal4 contact	30	CVT
fifth-layer metal	metal5	fifth metal	33	CMP
	via4	metal4-metal5 contact	32	CVQ
sixth-layer metal	metal6	sixth metal	37	CM6
	via5	metal5-metal6 contact	36	CV5
metal-metal cap	metalcap	inter-metal layer for capacitor	35	CTM
silicide block	sblock	silicide implant block for resistor	29	CSB
thin-oxide cap	cwell	well implant for linear thin-oxide capacitor	59	CWC
high-voltage FETs	tactive	thick oxide	60	CTA

** cactive is simply a convenience layer for the user, not a mask layer, and is treated as ``active'' for purposes of streaming out, DRC, and extraction.*

Table III. Additional layers supporting the optional features

Virtuoso

Virtuoso is the main layout editor of Cadence design tools. There is a small (icon) button bar on the left side of the editor. Commonly used functions can be accessed pressing these buttons. There is an information line at the top of the window. This information line, (from left to right) contains the X and Y coordinates of the cursor, number of selected objects, the traveled distance in X and Y, the total distance and the command currently in use. This information can be very handy while editing. At the bottom of the window, another line shows what function the mouse buttons have at any given moment. Note that these functions will change according to the command you are currently executing.



```
X: 47.70    Y: 14.85    (F) Select: 0    dX:    dY:    Dist:    Cmd: Rectangle    2
mouse L: Enter Point                M: Pop-up Menu                R:
Point at the first corner of the rectangle:
```

Figure 11. Virtuoso and LSW windows

Similarly to Composer, most of the commands in Virtuoso will start a mode (the default mode is selection), and as long as you do not choose a new mode you will remain in that mode. To quit from any mode and return to the default selection mode, the *"Esc"* key can be used.

As mentioned before, we can do the layout of our OTA (such as that shown in Fig. 11) following different degrees of automation. The following section illustrates them.

5.4.1. "Basic" Full-Custom layout

The layout is made layer by layer, in any order. This is similar to the mask layout edition in *Magic* or *L-Edit*. In Cadence, this is not usually done in practice (unless the Layout Editor/Design Kit do not support more advanced methods). This is included here as a reference, you can skip this section and go to the more practical approaches of Sections 5.3.2 and 5.3.3.

Additional details are provided in the Cadence Online Help, and in the following tutorial:

http://turquoise.wpi.edu/cds/

In our example the technology selected is n-well, so that the black area represents the p substrate. The full custom layout process consist basically on repeatedly selecting a certain layer from the LSW window, choosing the Rectangle icon to create a rectangle and drawing it.

As mentioned above, the geometries (enclosure, extension, minimum dimensions, etc) of the objects created have to comply with a set of rules dependent on the technology, named Design Rules. Once your layout completed, a Design Rule Check (DRC) is performed to enforce this point. Object dimensions can be checked by observing the screen coordinates or using the ruler (Ruler icon or k shortkey).

- *For designing a PMOS transistor of aspect ratio W/L (the order of layer drawing is irrelevant):*

Since we are using an n-well process, the substrate will be p-substrate. To create a PMOS transistor we need an n-well in which the transistor will be formed.

- Draw the well
  a) Select the **n-well** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the n-well with the size required.

- Draw the p- and n-select regions for the p transistor
  a) Select the **pselect** layer from the **LSW** window; we will draw the pselect enclosing the transistor
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the pselect on the cellview. The pselect should be placed within the n-well (you can use the **Edit->Move** command to move the layer) with geometry according to the Design Rules.

- Draw Diffusions
  a) Select the **pactive** layer from the **LSW** window; we will draw the active region of the p-device
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the pactive on the cellview (it should be enclosed by the pselect and by the nwell, as determined by the Design Rules). Its width should be W, and its length as small as possible, for reducing device parasitics.

NOTE: In many SCMOS processes there are 3 active layers: active, nactive and pactive. They are interchangeable since they translate to the same physical layer in fabrication. As mentioned in Table I, nactive and pactive layers are provided for differentiating graphically n and p acive diffussions (assigning them different colors) but what makes an active layer of type n or p is the nselect or pselect layer, respectively, surrounding it).

- Draw Poly
  a) Select the **poly** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the poly on the cellview
  The poly should be placed at the center of the p-island, with length L, and should extend over the p-island as specified in the Design Rules.

- Place Contacts
  a) Select the contact layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the contact on the cellview. Most technologies require one (or two) fixed contact size/s.
  Contacts should be placed at both sides of the pactive. You can copy the generated contact throughout the drain and source areas. The number of contacts should be as large as possible. Their size is determined by the corresponding Design Rules.

- *For designing a NMOS transistor of dimensions W/L (the order of layer drawing is irrelevant):*

- Draw the nselect
  a) Select the nselect layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the nselect on the cellview

- Draw the N active region
  a) Select the nactive layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the nactive on the cellview ; it should be enclosed by the nselect (as determined by the Design Rules). Its width should be W, and its length as small as possible.

- Draw Poly
  a) Select the **poly** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the poly on the cellview
  The poly should be placed at the center of the n-island, with length L, and should extend over the n-island as specified in the Design Rules.

- Place Contacts
  a) Select the contact layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).

c) Using your mouse, draw the contact on the cellview. Most technologies require one (or two) fixed contact size/s.
Contacts should be placed at both sides of the nactive. You can copy the generated contact throughout the drain and source areas. The number of contacts should be as large as possible. Their size is provided by the corresponding Design Rules.

- *Designing resistors:*

Integrated resistors can be made by different layers (diffusion, well, poly). In analog applications linear resistors are usually required, so that they are typically implemented using poly, and if technology allows, using a special high resistive poly layer.

The procedure for designing a resistor is as follows:

a) Determine the number of squares required by the resistor geometry:

$$No.\ squares = \frac{Resistor\ Value(\Omega)}{Sheet\ Resistance(\Omega / square)}$$

Typical sheet resistances are 20-30 Ω/□ for (silicide) poly and about 1 kΩ/□ for the high-resistive poly.

b) Choose the width *W* of the resistor strip. It should be larger than the minimum given by the Design Rules *for fabricating resistors*.

c) Obtain the length *L* of the resistor strip: *L= W x No. Squares*.

d) If *L* is too long, several bends can be introduced (thus forming serpentine resistors). Recalculate (b) counting about ½ square for each bend corner.

- *Poly1 resistor:*

- Draw Poly
  a) Select the **poly** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the poly on the cellview of width *W* and length slightly larger than *L*. (for allowing contacts at the resistor ends)

- Place Contacts
  a) Select the contact layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the contact on the extremes of the poly strip.

- Identify the device as a resistor

  If the Design Kit does not extract resistances by default (as the AMI technologies in the NCSU Kit) a layer is provided to identify where a resistor is, and to obtain its resistance value during netlist extraction. It our example technology, it is **res_id** (this layer is not fabricated).

  a) Select the **res_id** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw a rectangle of length *L* limited by the contacts and surrounding the resistor (according to the Design Rules).

  Fig. 12 shows an example of a poly resistor, of 6 squares. Its approximate value is therefore 6 x (20-30 Ω/□) = 120 – 180 Ω

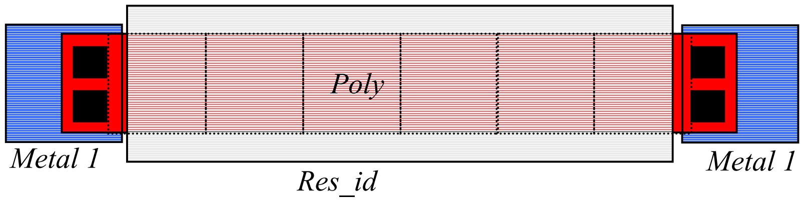


Figure 12. Poly resistor

- *High-Res resistor:*

Some technologies offer a highly resistive polysilicon layer, useful in analog design for obtaining linear resistors of large values using small silicon area. Usually they are made using the poly layer (more precisely, one of the poly layers if technology has two) and an additional layer that provides the high resistance properties.

The procedure for the AMI 0.5µ technology is as follows

- Draw Poly
  a) Select the **elec** layer from the **LSW** window (this is the second poly layer in this process)
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the poly on the cellview of width *W* and length slightly larger than *L*. (for allowing contacts at the resistor ends).

- Place Contacts
  a) Select the contact layer from the LSW window
  b) Select the Create->Rectangle (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw the contact on the extremes of the poly strip.

- Identify the device as a high-resistance resistor
  a) Select the **high_res** layer from the **LSW** window
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using your mouse, draw a rectangle of length *L*, separated from the contacts and surrounding the resistor (according to the Design Rules).

Fig. 13 shows an example of a poly resistor, of 6 squares. Its approximate value is therefore 6 x (1kΩ/□) = 6 kΩ

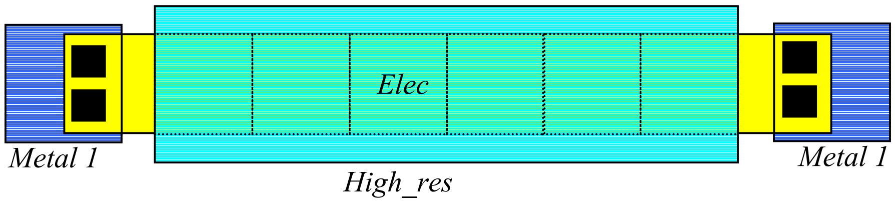


Figure 13. Hig-res resistor

NOTE: Fabricated resistance values can differ considerably from calculations. Therefore, the operation of a design should NEVER rely on exact resistance values, but on resistance RATIOS.

- *Other resistor:*

Resistors can also be made using diffusion or n-well layers, but the resulting resistances are strongly nonlinear and temperature-dependent, so that their application in analog design is marginal. The layout is very similar to that described for poly resistors, substituting the poly layer by the proper layer. *The resistor_id layer should be present for extracting these resistances in the layout netlist.*

- *Designing capacitors:*

Capacitors in analog circuits need typically be linear. Hence they are usually made (if technology provides a second poly layer) using two overlapping poly regions (poly1 and poly2) with a thin oxide layer in between. The overlapping region defines the required capacitance.

When two poly layers are available, the first poly layer (gate of MOS transistors and bottom plate of capacitors) is usually called *poly* or *poly1*. The second poly layer (top plate of capacitors and sometimes layer for resistors) is called *polycap* or *poly2* or *elec*.

The procedure for designing a poly-poly capacitor is as follows:

a) Determine the area required by the capacitor geometry:

$$Area\left(\mu m^2\right) = \frac{Capacitor \quad Value(F)}{Capacitance\ per\ unit\ area\ \left(F/\mu m^2\right)}$$

Typical poly-poly capacitances per unit area are 0.5 – 1 fF/µm^2. For the AMI 0.5µ process, it is approximately 0.9 fF/µm^2.

- Draw elec
  - a) Select the **elec** layer from the **LSW** window (this is the second poly layer in this process)
  - b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  - c) Using your mouse, draw a rectangle with the area calculated previously.

- Draw poly
  - a) Select the **poly** layer from the **LSW** window
  - b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  - c) Using your mouse, draw a rectangle overlapping the previous elec geometry according to the Design Rules.

- Place Contacts
  - a) Select the **contact** layer from the **LSW** window
  - b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  - c) Using your mouse, draw the contact on the top and bottom capacitor plates, according to the Design Rules.

Fig. 14 shows a poly capacitor in the AMI 0.5µ process.

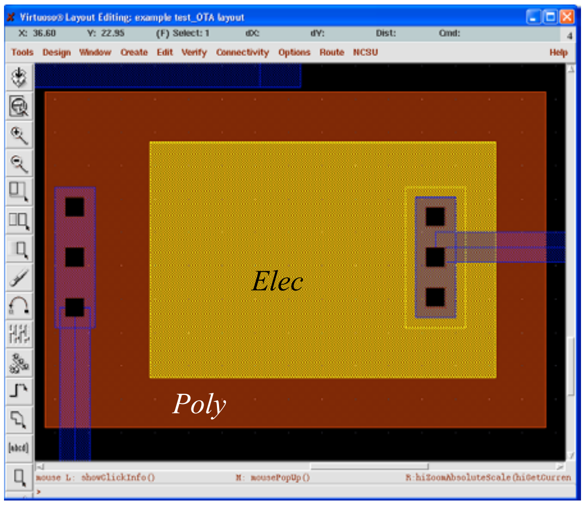

Virtuoso® Layout Editing: example test_OTA layout
X: 36.60 Y: 22.95 (F) Select: 1 dX: dY: Dist: Cmd:
Tools Design Window Create Edit Verify Connectivity Options Route NCSU Help
Elec
Poly
mouse L: showClickInfo() M: mousePopUp() R: hiZoomAbsoluteScale(hiGetCurren

Figure 14. Poly-Poly capacitor

NOTE 1: The geometry of the capacitor need not be a rectangle. It can fit the layout geometry in order to minimize area. In any case, if matched capacitors are required, they should not only have the same area but also the same geometry.

NOTE 2: Fabricated capacitor values may differ considerably from calculations. Therefore, the operation of a design should NEVER rely on exact capacitance values, but on capacitance RATIOS.

- *Layout: necessary connections:*

- Draw the well-contacts
  a) Select the **nselect** layer from the **LSW** window; we will draw the nselect enclosing the n-well contact for the PMOS transistors.
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using the mouse, draw the nselect on the cellview; If technology allows, the nselect can abut directly to the pselect of the PMOS transistor, but they should not overlap. Also, both selects should be placed within the n-well.
  d) Select the **nactive** layer from the **LSW** window; we will draw the body contact region of the P devices.
  e) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  f) Using your mouse, draw the nactive on the cellview
  g) Add contacts in the well-contact island.

- Draw the Substrate-contacts
  a) Select the **pselect** layer from the **LSW** window; we will draw the pselect enclosing the substrate contact for the NMOS transistors
  b) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  c) Using the mouse, draw the pselect on the cellview; If technology allows, pselect abuts directly to the nselect of the NMOS transistor, but they should not overlap.
  d) Select the **pactive** layer from the **LSW** window
  e) Select the **Create->Rectangle** (or choose the Rectangle icon from the side toolbar).
  f) Using your mouse, draw the p-island on the cellview.
  g) Add contacts in the substrate-contact island.

- Metal Connections
  a) Connect the transistor terminal contacts, terminal contacts of capacitors and resistors, and substrate/well contacts using the Metal1 layer (gates can be connected directly using the Poly1 layer).
  b) For interconnections, you can use any metal layer provided by the technology (Metal1, Metal2, Metal3, ...) so that metal paths can cross without being shorted. In two-metal technologies, usually one metal tends to be used for the horizontal paths and other for the vertical ones.

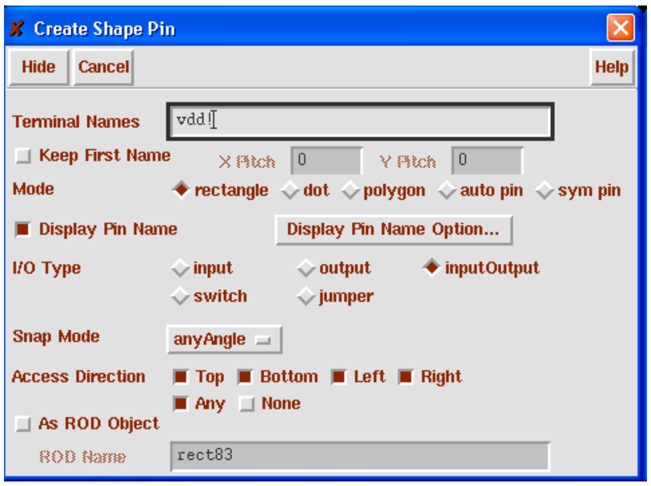
- *Additional steps for layout verification*

  - Add Pins:

    Once you have finished creating the layout, the next step is to add the I/O pins of your circuit. It is necessary to add the vdd! and vss! (or gnd!) pins to your circuit for the purpose of verification (if you have used these terminals in the schematic). Net labels ending in '!" mean global nodes (i.e., all wires in the entire design hierarchy labeled with this name are considered to be connected, even if they physically are not. They are usually power nets). The following is a procedure for adding I/O pins to your circuit:

    From your Layout window:

    1. Choose Create->Pin... from the menu. The Create Pin form will appear (Fig. 15).
    2. If the form is titled "Create Shape Pin", choose "sym pin" under the Mode option.
    3. Enter a TerminalName(the name of your pin).
    4. Make sure that the "Display Pin Name" option is selected.
    5. Specify the "I/O Type" as input, output, or inputoutput, according to the schematic.
    6. Specify the Pin Type as Metal1, Metal2,... depending on which is the top layer at the place that the pin is to be inserted (they should match).
    7. Specify the Pin Width to the desired pin width (the pin is square).
    8. Move the mouse to specify where the pin and the label should be placed.



Terminal Names: vdd!
Keep First Name [ ] X Pitch 0 Y Pitch 0
Mode: [✓ rectangle] dot polygon auto pin sym pin
Display Pin Name [✓] Display Pin Name Option...
I/O Type: input output [✓ inputOutput] switch jumper
Snap Mode: anyAngle
Access Direction: [✓ Top] [✓ Bottom] [✓ Left] [✓ Right] [✓ Any] None
As ROD Object [ ]
ROD Name: rect83

Figure 15. Create Pin window

In our example, you need to add pins for the input terminals (named in our OTA example v+, v-), outputs (out),ibias, vdd, and vss. This information will be employed for the extraction of the netlist from the layout and subsequent comparison with the schematic netlist. Therefore, pin names in layout and schematic must mach perfectly (comparison is case sensitive).

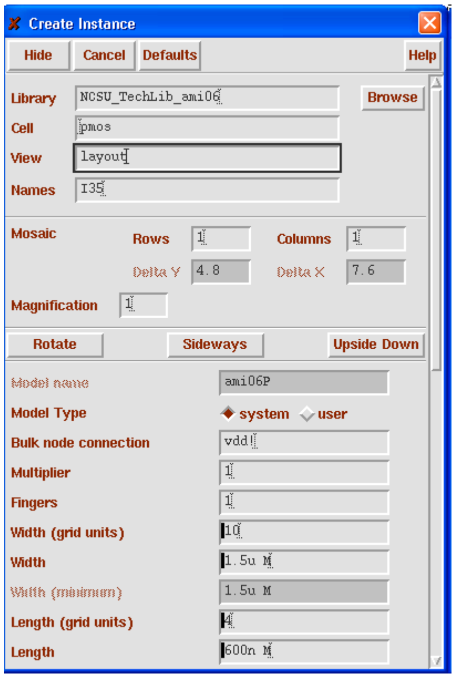
5.4.2. Full-Custom Layout using pcells

The above approach is useful for learning the design rules and layers available for a certain technology, but it is not usually followed (if one can avoid it). Instead, basic elements (transistors, resistances, capacitors, etc) can be directly created using parametrized cells (*pcells*), by defining their dimensions in a *Properties* window. Properties of a certain *pcell* can be modified after creation by selecting the *pcell* and clicking on the **Property** icon.

- For designing a PMOS transistor of aspect ratio W/L:

Select the Instance *NCSU_TechLib_ami06 > pmos*. In the Create Instance window (Fig. 16), you can select the desired *W* and *L*. Then, place the pcell.

For creating cascoded or interdigitized structures (e.g., for current mirrors or differential pairs) it is useful to use the ***Multiplier*** or ***Finger*** fields in the instance properties. They generate multiple instances of the transistor, sharing the adjacent instances a common (drain or source) terminal with metal contacts (Multiplier) or without them (Finger).



Library	NCSU_TechLib_ami06
Cell	pmos
View	layout
Names	I35
Mosaic	Rows 1, Columns 1, Delta Y 4.8, Delta X 7.6
Magnification	1
Model name	ami06P
Model Type	[✓ system] user
Bulk node connection	vdd!
Multiplier	1
Fingers	1
Width (grid units)	10
Width	1.5u M
Width (minimum)	1.5u M
Length (grid units)	4
Length	600n M

Figure 16. Creating a PMOS transistor

- For designing an NMOS transistor:

Similar to the PMOS, just select the Instance *NCSU_TechLib_ami06 > nmos*.

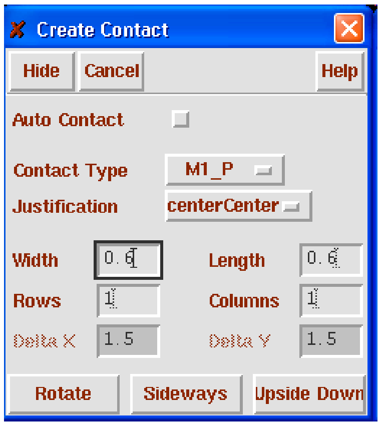
- For designing resistances and capacitances:

Similar to the procedure described in the former section for the AMI 0.5µ process (other processes may include resistor and capacitor pcells). However, pcells can be employed for the required poly contacts. There are also two rudimentary pcells for poly resistances, (*NCSU_TechLib_ami06 > res* (wide) or *res2* (thin)), but they are not very useful.

- *Creating contacts:*

Select **Create** > **Contact**, then select the type of contacts and the number of contacts required in the pop-up window (Fig. 17). The types of contacts available in the AMI 0.5µ process are:

- **M1-P**: Metal1-Pactive contact. Used mainly in PMOS drain and source contacts, substrate contacts.
- **M1-N**: Metal1-Nactive contact. Used mainly in NMOS drain and source contacts.
- **NTAP**: Metal1-Nwell contact. Used in N-well contacts.
- **M1-POLY**: Metal1-Poly contact. Used mainly in gate transistors and poly resistor terminals.
- **M1-ELEC**: M1 to poly2 contact. Used mainly in top plate contacts of poly-poly capacitors and in "high-res" resistor terminals.
- **M1-M2**: Metal1-Metal2 via.
- **M2-M3**: Metal2-Metal3 via.



Auto Contact

Contact Type	M1_P		
Justification	centerCenter		
Width	0.6	Length	0.6
Rows	1	Columns	1
Delta X	1.5	Delta Y	1.5

Figure 17. Creating a contact

- *Routing:*

Metal and poly paths can be created using the ***Create*** > ***Rectangle*** function. Alternatively, they can be made using the **Path** icon (or **Create Path**, or **p** shortkey). You select the path width and draw the path. Orthogonal or diagonal paths can usually be created. Moreover, you can create a via and change the metal layer without quitting the Path function, using **Change to layer ..** in the **Create Path** window. This tool is extremely necessary for wiring large paths in the highest hierarchies of the layout.

The Path tool is also very useful for creating serpentine resistors.

- *Pins:*

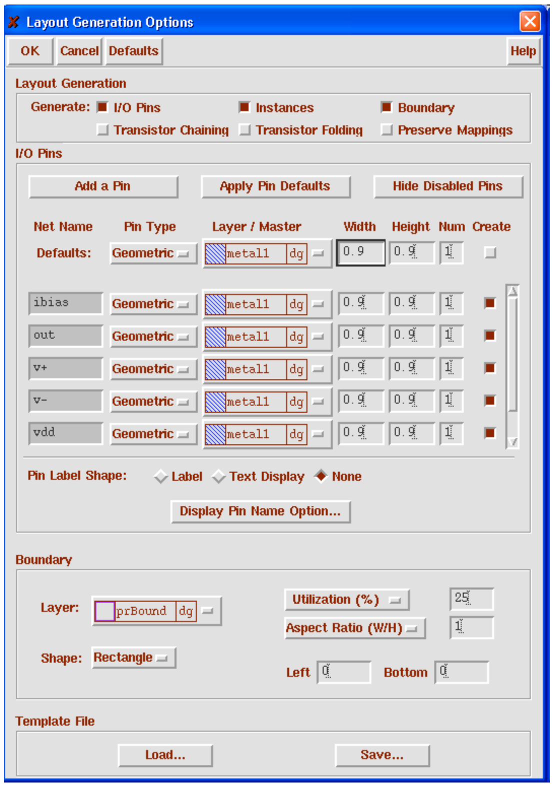
They are created as explained in the above section.

5.4.3 Full-Custom Layout using Virtuoso-XL

The **Virtuoso layout accelerator (Virtuoso XL)** is a connectivity-based editing tool that automates each stage of layout design, from component generation through automatic/interactive routing. When used as part of an automated custom physical design methodology, Virtuoso XL lets you generate custom layouts from schematics or netlists and edit layouts that have defined connectivity. Virtuoso XL continuously monitors connections of components in the layout and compares them with connections in the schematic. You can use Virtuoso XL to view incomplete nets, shorts, invalid connections, and overlaps to help you wire your design. It lets you both speed up and customize the layout process.

To start the Virtuoso XL environment, open the schematic view of cell OTA. Next, in the Composer window, click on ***Tools -> Design Synthesis -> Layout XL***. The Virtuoso XL Startup Option window will appear, asking whether a new layout cell view should be created or an existing layout cell should be used. Enable *Create New* and click on **OK**. A Create New File window will then appear. Use for *Library Name* **example**, for *Cell Name* **OTA**, for *View Name* **layout**, and for *Tool* **Virtuoso**, and click on **OK**. The Virtuoso layout window then appears.

To start the generation of the layout from the schematic, click on ***Design -> Gen from Source ...*** in the Virtuoso XL window. The Layout Generation Options window, as shown in Fig. 18, will appear. Under I/O Pins, change the pin layers to Metal1 and switch off the Create button for *Defaults* (to prevent the creation of a pin "BULK!" for the epi connections of the layout instance cells).



Layout Generation

Generate: [✓ I/O Pins] [✓ Instances] [✓ Boundary] Transistor Chaining, Transistor Folding, Preserve Mappings

I/O Pins

Net Name	Pin Type	Layer / Master	Width	Height	Num	Create
Defaults:	Geometric	metal1 dg	0.9	0.9	1	☐
ibias	Geometric	metal1 dg	0.9	0.9	1	■
out	Geometric	metal1 dg	0.9	0.9	1	■
v+	Geometric	metal1 dg	0.9	0.9	1	■
v-	Geometric	metal1 dg	0.9	0.9	1	■
vdd	Geometric	metal1 dg	0.9	0.9	1	■

Pin Label Shape: Label Text Display [✓ None]

Boundary

Layer: prBound dg

Utilization (%) 25

Aspect Ratio (W/H) 1

Shape: Rectangle

Left 0 Bottom 0

Template File

Figure 18. Layout Generation Options window

Next, click on **OK**. The Virtuoso XL window will show a large rectangle, which describes the boundary for placement and routing, and several small red rectangles describing the bounding boxes of the layout of the instance cells. Also the pins of the layout will be present in the layout shown by the Virtuoso XL window. To obtain an initial placement with the instances and pins inside the place and route boundary, click on ***Edit -> Place from Schematic*** in the Virtuoso XL window. Press the **f** key or click ***Window -> Fit All*** to fit the total design in the Virtuoso window. A picture similar to Fig. 19a will be obtained. The placement is not particularly good, but it is a good starting point.

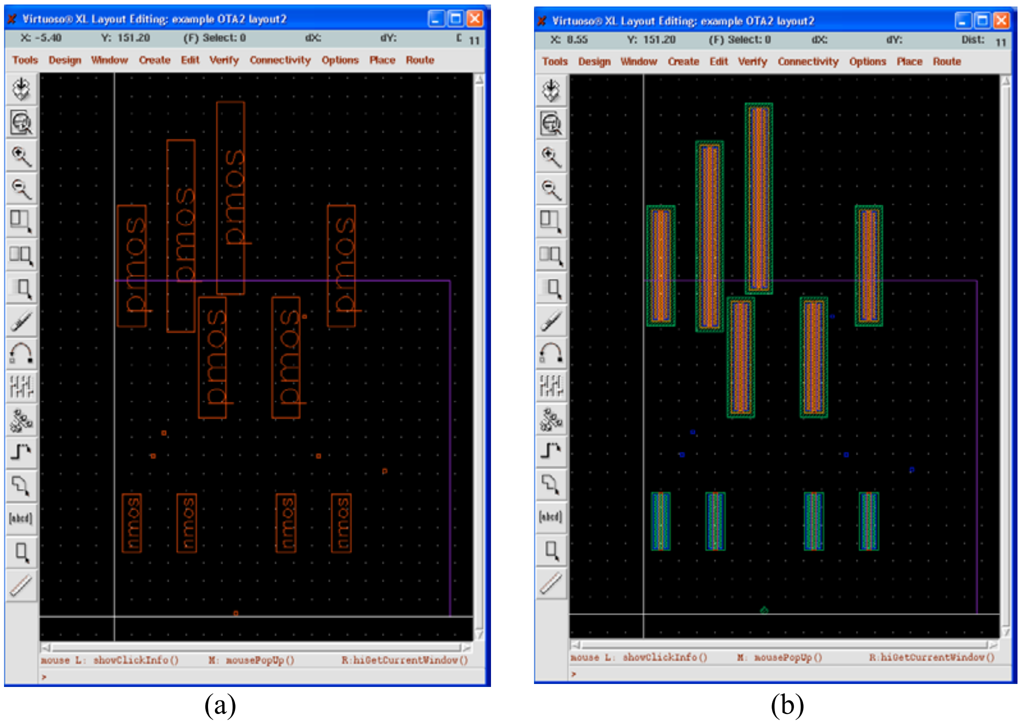

Virtuoso® XL Layout Editing: example OTA2 layout2
Tools Design Window Create Edit Verify Connectivity Options Place Route
pmos
nmos
mouse L: showClickInfo() M: mousePopUp() R:hiGetCurrentWindow()
(a)
(b)

Figure 19. Virtuoso-XL windows after placement (a) Bounding boxes displayed (b) Instances displayed

To see which layout instance corresponds to which schematic instance, click on an instance in one of the views and the corresponding instance in the other view will be highlighted.

To view the contents of the instances, press **Shift-f** (Fig. 19b results)
To view only the bounding boxes again, press **Ctrl-f**.

Virtuoso XL can show so-called flight lines for the nets in the layout which are not yet connected. The flight lines connect the pins that belong to the same net, and flight lines that belong to the same net have the same color. To show the flight lines for the unconnected nets, click on ***Connectivity -> Show Incomplete Nets***. The Show Incomplete Nets form will appear from which the nets that are shown are selected. Click on **OK** and a picture similar to that shown below will be obtained:

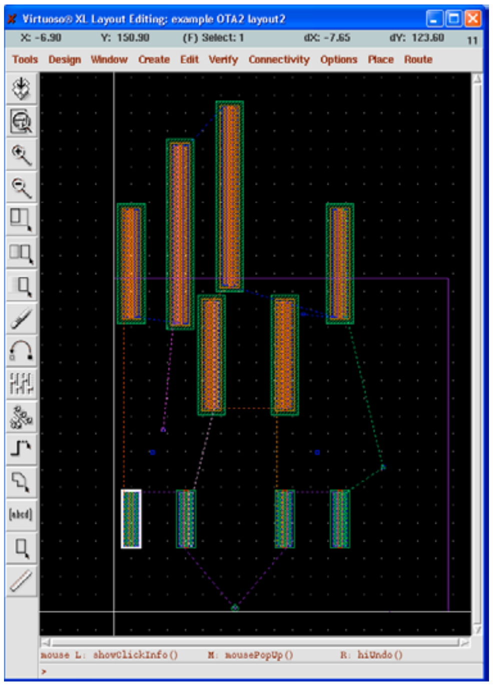


Figure 20. Virtuoso-Xl windows displaying conectivity

To hide the flight lines again, click on ***Connectivity -> Hide Incomplete Nets***.

<u>*5.4.4.Hierarchical layout*</u>

As for the schematic, the layout of a complex circuit usually is performed hierarchically. First, basic components are created (e.g., OpAmps, OTAs, current mirrors, etc) that are they instantiated in another layout at a higher hierarchical level. Such components are instantiated as pcells are, using e.g., the **Instance** icon. The only difference is that now they are selected from the user libraries.

The contents of the instances in the layout can be viewed pressing **Shift-f**, and can be converted to bounding boxes pressing **Ctrl-f**. Lower-level instances in a hierarchical layout cannot be modified by default. If we want to modify them, we can either go to the layout cellview of this component or directly edit it descending in the hierarchical layout, using the Edit in Place feature (Edit selecting ***Design>Hierarchy>Edit in Place*** or **^*x***, return using ***Design>Hierarchy>Return*** or ***b***). In both cases, changes will affect all the instances of this component.

<u>**5.5. Verification**</u>

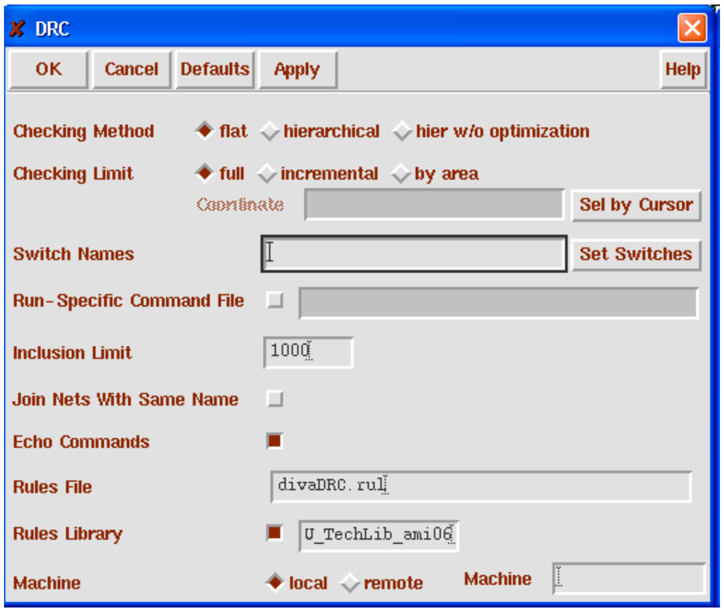
<u>*5.5.1. Design Rule Check (DRC)*</u>

The created mask layout must conform to a complex set of design rules, in order to ensure a lower probability of fabrication defects. A tool built into the Layout Editor, called **Design Rule Checker**, is used to detect any design rule violations during and after the mask layout design. The detected errors are displayed on the layout editor window as error markers, and the corresponding rule is also displayed in a separate window. The designer must perform DRC (in a large design, DRC is usually performed frequently - before the entire design is completed), and make sure that all layout errors are eventually removed from the mask layout, before the final design is saved.

The MOSIS SCMOS design rules (http://www.mosis.org/Technical/Designrules/scmos/tech-codes) are employed in the NCSU Design Kit. The Scalable CMOS (SCMOS) rules are a common set of rules widely supported by MOSIS that intend to simplify and unify the layout design and verification process. Circuit geometries (and layout rules) are specified in the Mead and Conway's lambda based methodology. The unit of measurement, lambda, can easily be scaled to different fabrication processes. Specific vendor rules are employed in other processes that lead usually to denser layouts.

The following is a procedure to perform design rule check (DRC) for a layout.

- From your Layout window:
  1. Choose Verify -> DRC from the menu. The Verify DRC form will appear.



Checking Method	[✓ flat] hierarchical hier w/o optimization
Checking Limit	[✓ full] incremental by area
Coordinate	
Switch Names	
Run-Specific Command File	☐
Inclusion Limit	1000
Join Nets With Same Name	☐
Echo Commands	■
Rules File	divaDRC.rul
Rules Library	■ U_TechLib_ami06
Machine	[✓ local] remote Machine

Figure 21. DRC Window

  2. Set the Switch Names field, if the process you're running requires it (in our example, leave as default). Switches allow customizing the rules check.
  3. Click OK to run DRC.
     - If your design has violated any design rules, DRC will report the errors in the CIW.
     - Errors are also indicated by the markers (white color) on the circuit.
     - You may then proceed to correct the errors according to the design rules.
  4. For huge layouts, the markers might not be easily located. To find markers, choose Verify -> Markers -> Find in layout window.
     - A pop-up menu will appear. Click on the Zoom to Markers box.
     - Click on the Apply button and Cadence will zoom in to the errors or warnings as desired.

5.5.2. Layout versus Schematic (LVS)

Once the layout fulfills all the design rules, the next verification step follows. The netlist behind the layout view is extracted and compared to that of the schematic view. This is the **Layout Versus Schematic (LVS) Check**.

Circuit extraction is performed after the mask layout design is completed, in order to create a detailed netlist (or circuit description) for the LVS and the simulation tool. The circuit extractor is capable of identifying the individual transistors and their interconnections (on various layers), as well as (usually) the parasitic resistances and capacitances that are inevitably present between these layers. Thus, the "extracted netlist" can provide a very accurate estimation of the actual device dimensions and device parasitics that ultimately determine the circuit performance. The extracted netlist file and parameters are subsequently used in Layout-versus-Schematic comparison and in detailed transistor-level simulations (post-layout simulation).

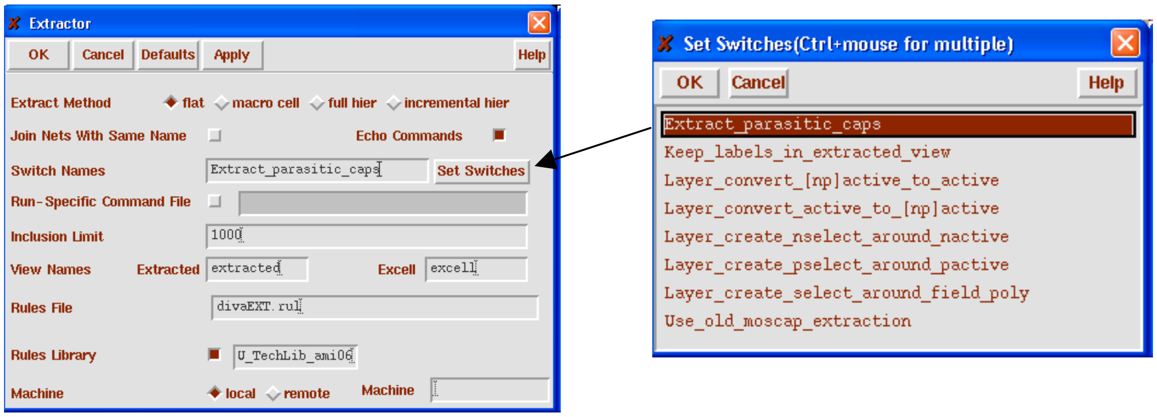
This is how a netlist is extracted:

1. Check and Save your design (schematic). Click on the first icon to the left of the schematic (it is the icon with a box and a check mark).
   - You will be warned if you have any floating wires or pins. You can also perform the same function by selecting Check and Save from the Design menu of the schematic window.
   - Make sure you select Check and Save, not just Save.
2. Open the layout view of your design.

   Make sure you have labeled the vdd! and vss! (or gnd!) pins for the power nets in layout, if they are employed in the schematic.
3. In the Virtuoso layout window, select Verify -> Extract.

   An extractor window appears (Fig. 22). Some switches can be set for customizing the extraction (e.g., for extracting parasitic capacitances). In our example, just click OK and watch any errors in the CIW.

   It extracts a netlist from the layout, and creates a new view name "extracted."



Extract Method [✓ flat] macro cell full hier incremental hier

Join Nets With Same Name ☐ Echo Commands [✓]

Switch Names: Extract_parasitic_caps

Run-Specific Command File ☐

Inclusion Limit: 1000

View Names Extracted: extracted Excell: excell

Rules File: divaEXT.rul

Rules Library [✓] U_TechLib_ami06

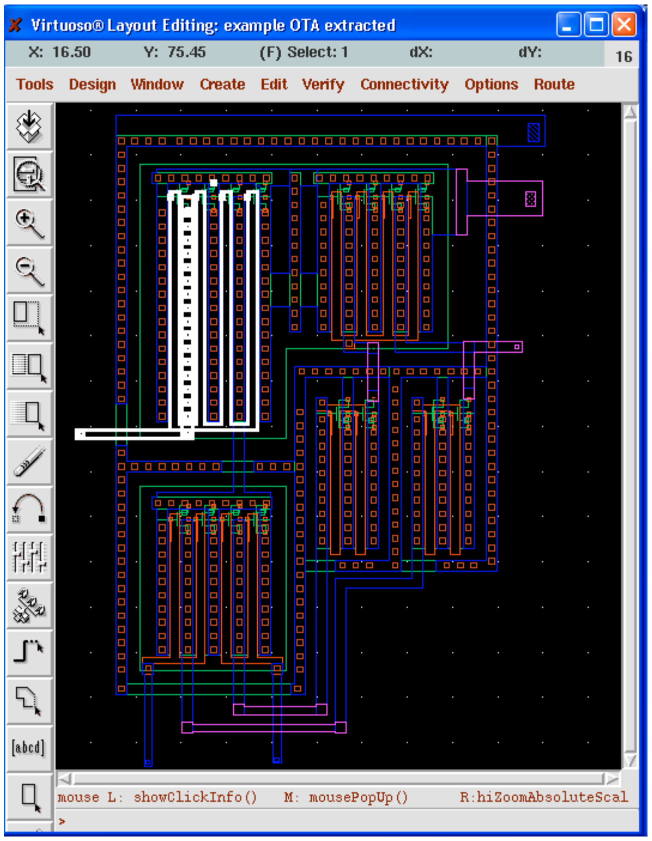
Machine [✓ local] remote Machine:

Extract_parasitic_caps [✓]
Keep_labels_in_extracted_view
Layer_convert_[np]active_to_active
Layer_convert_active_to_[np]active
Layer_create_nselect_around_nactive
Layer_create_pselect_around_pactive
Layer_create_select_around_field_poly
Use_old_moscap_extraction

Figure 22. Extraction window and switches applicable

4. Open the cell with view name "extracted". (Click Design -> Load ...)
   A new window appears. In our example, set the following items, and click OK.
   - Library Name: *example*
   - Cell Name: *OTA*
   - View Name: *extracted*

The following window appears. It is similar to the layout, but the devices are extracted (notice the device symbols close to the mask layout of the devices) and the connectivity is extracted. This last property is very useful in layout debugging. Selecting ***Verify > Probe>Add net*** and clicking a net, all the net connectivity is highlighted, and the net name is displayed (see the net corresponding to pin *iout* in Fig. 23). This allows to readily locate shorts or floating nets.



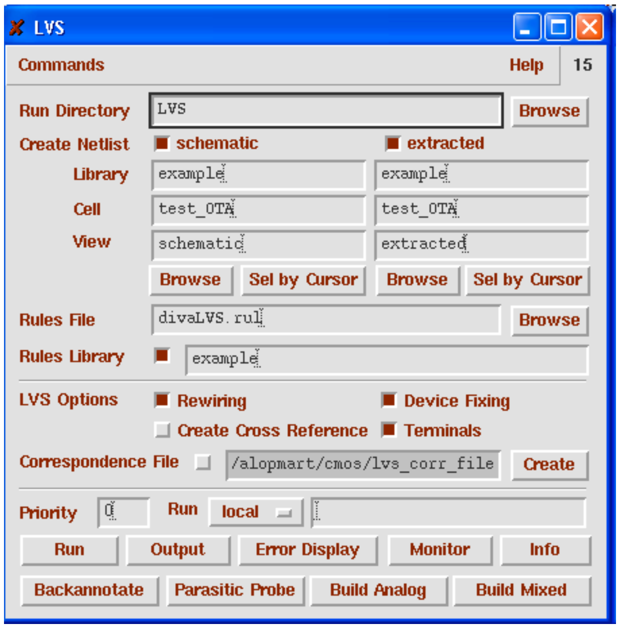
```
mouse L: showClickInfo()    M: mousePopUp()    R:hiZoomAbsoluteScal
>
```

Figure 23. OTA extracted view

After the mask layout design of the circuit is completed and the underlying design extracted, the design should be checked against the schematic circuit description created earlier. The tool called "Layout-versus-Schematic (LVS) Check" will compare the original network with the one extracted from the mask layout, and prove that the two networks are indeed equivalent. The LVS step provides an additional level of confidence for the integrity of the design, and ensures that the mask layout is a correct realization of the intended circuit topology. Note that the LVS check only guarantees topological match. A successful LVS will not guarantee that the extracted circuit will actually satisfy the performance requirements. Any errors that may show up during LVS (such as unintended connections between transistors, or missing connections/devices, etc.) should be corrected in the mask layout - before proceeding to post-layout simulation. Also note that the extraction step must be repeated every time you modify the mask layout.

This is the procedure for the LVS check:

1. In the Virtuoso window that shows the extracted view of the OTA, select Verify -> LVS ... A new window appears (Fig. 24). Select the schematic and the extracted views in the upper left and right areas of the window, respectively. Then click "RUN." The LVS check is performed in background mode. It takes a while, depending on the size of your design.



Run Directory	LVS	
Create Netlist	■ schematic	■ extracted
Library	example	example
Cell	test_OTA	test_OTA
View	schematic	extracted
Rules File	divaLVS.rul	
Rules Library	■ example	
LVS Options	■ Rewiring	■ Device Fixing
	□ Create Cross Reference	■ Terminals
Correspondence File	□ /alopmart/cmos/lvs_corr_file	
Priority	0	Run local

Figure 24. LVS Window

2. To see if the job is still running, you can click on the Job Monitor... button and a pop-up menu will appear.
3. After a while, a pop-up menu will appear notifying you of the successful completion or failure of the LVS job. Click OK.

   NOTE: most LVS failures are due to the fact that the schematic was not saved before LVS or that the layout was not extracted after some change.
4. In the LVS window, click "Output" to get the information regarding the LVS run. If the netlists doesn't match, click" Error Display" to find out the error.

This is a typical output for a successful LVS:

```
@(#)$CDS: LVS version 4.4.1 03/17/98 22:19 (cds10067) $

Like matching is enabled.
Net swapping is enabled.
Using terminal names as correspondence points.
Compiling Diva LVS rules...

    Net-list summary for /home/kgf/cds/LVS/layout/netlist
       count
   4                   nets
   4                   terminals
```

```
     1                  pmos
     1                  nmos

       Net-list summary for /home/kgf/cds/LVS/schematic/netlist
          count
     4                  nets
     4                  terminals
     1                  pmos
     1                  nmos


       Terminal correspondence points
               1        gnd!
               2        in
               3        out
               4        vdd!
```

The net-lists match.

```
                            layout  schematic
                                 instances
     un-matched                  0         0
     rewired                     0         0
     size errors                 0         0
     pruned                      0         0
     active                      2         2
     total                       2         2

                                   nets
     un-matched                  0         0
     merged                      0         0
     pruned                      0         0
     active                      4         4
     total                       4         4

                                 terminals
     un-matched                  0         0
     total                       4         4


Probe files from /home/kgf/cds/LVS/schematic
devbad.out:
netbad.out:
mergenet.out:
termbad.out:
prunenet.out:
prunedev.out:
audit.out:

Probe files from /home/kgf/cds/LVS/layout
devbad.out:
netbad.out:
mergenet.out:
termbad.out:
prunenet.out:
prunedev.out:
audit.out:
```

NOTE: The NCSU Kit by default does not check for transistor, resistor or capacitor size mismatches. Select the "NCSU->Modify LVS Rules..." menu entry and check the "Compare FET parameters", "Compare capacitor parameters" and "Compare resistor parameters" check box if you want LVS to warn you of size mismatches. By the way, take a look at all the LVS options you can set (for the most part they are self-explanatory):

- ***Allow FET Series Permutation****: disregard the connection order of series-connected FETs. Eg, in a NAND gate's pulldown stack this would allow the ``a'' and ``b'' inputs to be connected to either NMOS.*

- ***Combine Parallel FETs****: reduce parallel-connected FETs into one equivalent FET*
- ***Combine Parallel Resistors****: reduce parallel-connected resistors into one equivalent resistor*
- ***Combine Series Resistors****: reduce series-connected resistors into one equivalent resistor*
- ***Combine Parallel Capacitors****: reduce parallel-connected capacitors into one equivalent capacitor*
- ***Combine Series Capacitors****: reduce series-connected capacitors into one equivalent capacitor*
- ***Compare FET Parameters****: generate warning if FET widths and lengths don't match*
- ***Ignore FET Body Terminal****: ignore FET body terminal (useful for preliminary verification)*
- ***Compare Capacitor Parameters****: generate warning if capacitor values don't match. The amount by which the values can differ before they are considered mismatched is defined by the global variable NCSU_LVSCapSlack. A value of 0 indicates perfect agreement is required. The default is 0.1 (values must match within 10%).*
- ***Compare Resistor Parameters****: generate warning if resistor values don't match. The amount by which the values can differ before they are considered mismatched is defined by the global variable NCSU_LVSResSlack. A value of 0 indicates perfect agreement is required. The default is 0.1 (values must match within 10%).*

5.6. Post-Layout Simulation

The electrical performance of a full-custom design can be best analyzed by performing a post-layout simulation on the extracted circuit net-list. At this point, the designer should have a complete mask layout of the intended circuit/system, and should have passed the DRC and LVS steps with no violations. The detailed (transistor-level) simulation performed using the extracted net-list will provide a clear assessment of the circuit speed, the influence of circuit parasitics (such as parasitic capacitances and resistances), and any glitches that may occur due to signal delay mismatches. As can be guessed, this step is particulary important in circuits very sensitive to parasitics, such as wideband or RF circuits.

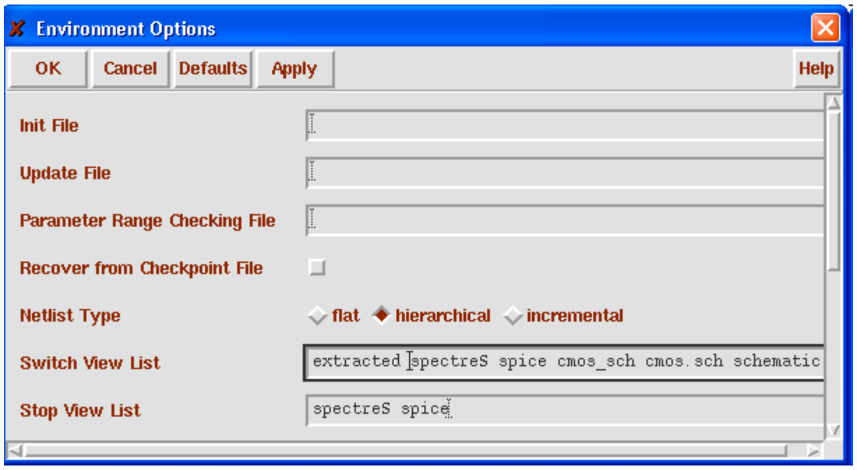
If the results of post-layout simulation are not satisfactory, the designer should modify some of the transistor dimensions and/or the circuit topology, in order to achieve the desired circuit performance under "realistic" conditions, i.e., taking into account all of the circuit parasitics. This may require multiple iterations on the design, until the post-layout simulation results satisfy the original design requirements.

Finally, note that a satisfactory result in post-layout simulation is still no guarantee for a completely successful product; the actual performance of the chip can only be verified by testing the fabricated prototype. Even though the parasitic extraction step is used to identify the realistic circuit conditions to a large degree from the actual mask layout, most of the extraction routines and the simulation models used in modern design tools have inevitable numerical limitations. This should always be one of the main design considerations, from the very beginning.

After a successful LVS you will have two main cell views for the same circuit. The first one is the schematic view, which is your initial (ideal) design. The second is the extracted view, that is based on the layout, and in addition to the basic circuit includes the layout associated parasitic effects (if the proper switches were activated during extraction, see Fig.22). Since both of these views refer to the same circuit they can be interchanged.

In this example we are going to re-run the simulation example of Section 5.2, but we will *make* the simulator to use the extracted OTA cell view (including parasitics) instead of the schematic cell view. Make sure that you are in the *test_OTA* schematic (Fig. 7), that you used to simulate your design earlier.

1. Start the simulation environment using ***Tools*** > ***Analog Environment***. The simulation window shown in Fig. 8 will pop up.
2. From the ***Setup*** menu choose the ***Environment*** option. A new dialog box controlling various parameters of Analog Artist will pop-up (Fig. 25). The line that we will have to alter is called the *Switch View List*. This entry is an ordered list of cell views that contain information that can be simulated. The simulator (in fact the netlister) will search until it finds one of these cellviews. The default entry does not contain an extracted cellview. We will simply add an entry for extracted cellview before the schematic cellview. As a result of this modification, the simulator will use the extracted cellview of the cell, if one is available.



Init File	
Update File	
Parameter Range Checking File	
Recover from Checkpoint File	☐
Netlist Type	flat [✓ hierarchical] incremental
Switch View List	extracted spectreS spice cmos_sch cmos.sch schematic
Stop View List	spectreS spice

Figure 25. Simulation environment options

From this point on the simulation will continue just as it has been described in the simulation example of Section 5.2, except for the fact that the results will now include parasitic effects from the actual layout.

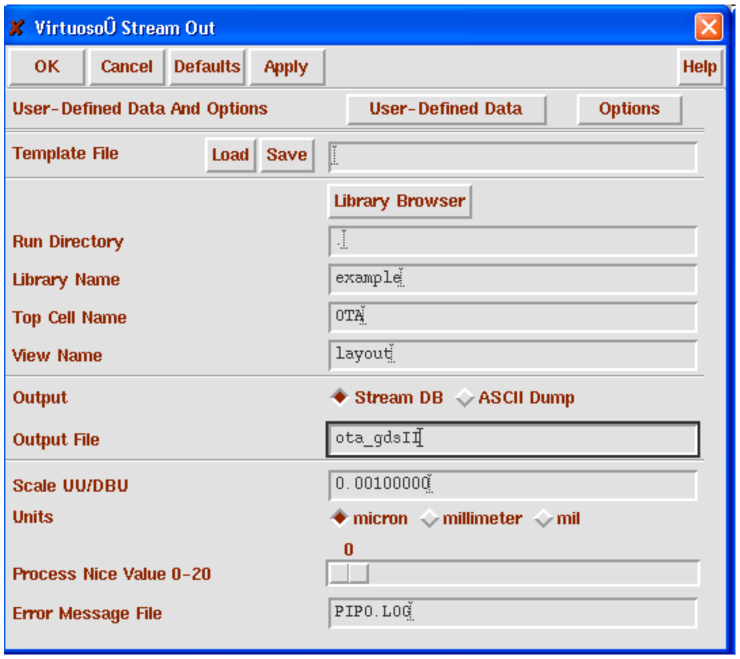
6. TRANSFER TO FOUNDRY

Once performed these verification steps at the highest hierarchical level (i.e., complete chip including pads) your design is ready to be submitted to the foundry. First, the design must be converted to a standard format (typically GDSII or CIF). This is the procedure:

1. Create GDSII file for layout. It is usually employed using design kits from Europractice, AMS, TSMC, etc.

- Open Stream Out form from CIW window. File->Export->Stream (Fig. 26)
- Fill in Library Name, Cell Name, and Output File boxes
- Include layer map in User Defined Data
  - Click User Defined Data button in Stream Out form
  - Enter the appropriate file name for the given technology in Layer Map box. It depends on the process employed.
  - Click ok.
- Click ok.

- You should get a message stating stream out was successful. The GDSII output file is the file you used in the Output File box. The PIPO.log file reports information about the conversion. This information can show if the required layers have been correctly translated.

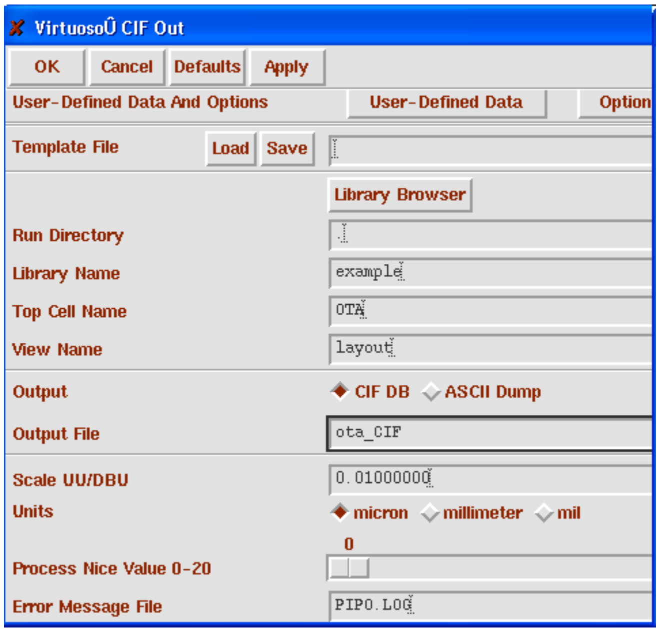


User-Defined Data And Options	
Template File	
Run Directory	.
Library Name	example
Top Cell Name	OTA
View Name	layout
Output	[✓ Stream DB] ASCII Dump
Output File	ota_gdsII
Scale UU/DBU	0.00100000
Units	[✓ micron] millimeter mil
Process Nice Value 0-20	0
Error Message File	PIPO.LOG

Figure 26: GDSII Stream Out window

2. Create CIF file for layout. This is the procedure employed for the NCSU Design Kit.

- Open CIF Out form from CIW window. File->Export->CIF.
- Follow all steps from 1. The layer map table is /Kits/NCSU/local/pipo/cifOutLayerMap.



User-Defined Data And Options	
Template File	
Run Directory	.
Library Name	example
Top Cell Name	OTA
View Name	layout
Output	[✓ CIF DB] ASCII Dump
Output File	ota_CIF
Scale UU/DBU	0.01000000
Units	[✓ micron] millimeter mil
Process Nice Value 0-20	0
Error Message File	PIPO.LOG

Figure 27: CIF Export window

3. Find checksum for output file (GDSII or CIF)

- Retrieve mosiscrc.c from MOSIS web-site at http://www.mosis.org/Customer/Email/mosiscrc.c
- Compile the C language file.
  >>gcc mosiscrc.c
- Use executable to find checksum of file. Record the two numbers that are given. For more information see the MOSIS web page http://www.mosis.org/Customer/Email/email-system.html
    - For GDSII
  >>a.out -b <GDSII file name>
    - For CIF
  >>a.out -t <CIF file name>

4. Submitting the Chip....

Use the one-step submission form to submit the chip to MOSIS for fabrication. Refer to http://www.mosis.org/Customer/Email/email-templates.html#one-step for details. The lambda and the technology code are specific to the process your are using. The technology codes and lambda values are found at http://www.mosis.org/Technical/Techcodes/menu-techcodes.html

If you are using the NCSU design kit, the technology code is specific to the kind of devices you are using. For instance, if you are using linear capacitors for the HP 0.5um process, the technology code is SCN3MLC_SUBM; if you did not use linear capacitors in the HP 0.5um process, the technology code is SCN3M_SUBM (LC indicates linear capacitors). All the SCMOS options are given at http://www.mosis.org/Technical/Designrules/scmos/scmos-main.html The options available are specific to the process (see http://www.mosis.org/Technical/Processes/menu-processes.html)The lambda values are given for each of the processes.

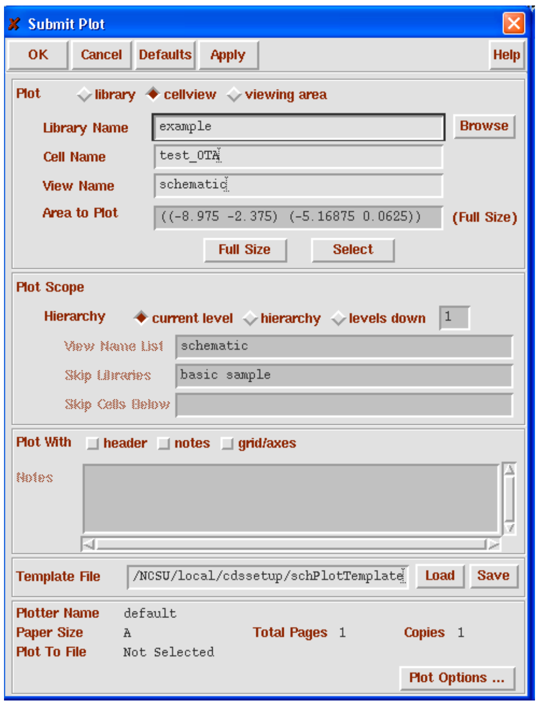
NOTE: It is highly recommended that before submitting the design the output format (GDSII or CIF) be imported again into Cadence (in a new library, so that the original design is not overwritten) and that the verification steps (DRC, LVS and Post-Layout simulation) be performed again. This ensures that all the layers were translated correctly to the final file.

8. PRINTING IN CADENCE

If you need to print your Cadence schematic, layout or simulation results, you can do the following.

a) **Printing schematics to a file**. In the Composer window containing the schematic you want to print, select **Design>Plot>Submit...** The form shown in Fig. 28a appears. Select **Plot Options**, and a new window, shown in Fig, 28b, appears. Choose the printer type (e.g., for a PS file, choose ps106-q). Then select **Send Plot only to File** and enter the filename. Click OK and the window disappears. Then click OK again in the Create Plot window. As a result, you have a (in this example, PS) file with the schematic.

The NCSU Kit allows modifying the schematic view to a better format for printing. If you want to use this alternative format, select from the Composer window NCSU > Create Publication schematic... and click OK. A new schematic_publication cellwiew appears. Open it and follow the guidelines of the former paragraph.



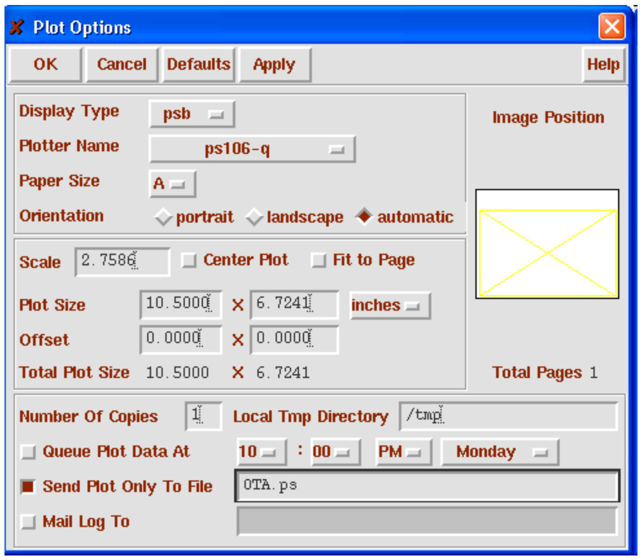
Plot: library [✓ cellview] viewing area
Library Name: example
Cell Name: test_OTA
View Name: schematic
Area to Plot: ((-8.975 -2.375) (-5.16875 0.0625)) (Full Size)

Plot Scope
Hierarchy: [✓ current level] hierarchy levels down 1
View Name List: schematic
Skip Libraries: basic sample
Skip Cells Below:

Plot With: header notes grid/axes
Notes:

Template File: /NCSU/local/cdssetup/schPlotTemplate

Plotter Name: default
Paper Size: A
Plot To File: Not Selected
Total Pages: 1
Copies: 1



Display Type: psb
Plotter Name: ps106-q
Paper Size: A
Orientation: portrait landscape [✓ automatic]
Scale: 2.7586 Center Plot Fit to Page
Plot Size: 10.5000 X 6.7241 inches
Offset: 0.0000 X 0.0000
Total Plot Size: 10.5000 X 6.7241
Total Pages 1
Number Of Copies: 1 Local Tmp Directory: /tmp
Queue Plot Data At: 10 : 00 PM Monday
[✓ Send Plot Only To File]: OTA.ps
Mail Log To:

(a) (b)

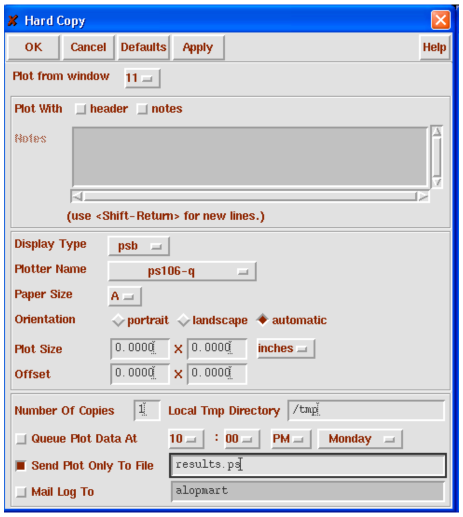
Figure 28. Printing to a file. (a) Submit Plot window (b) Plot Options Window

b) **Printing layouts to a file**. Perform the steps in the first paragraph of (a) from the Virtuoso layout editor containing the layout view you want to print.

c) **Printing a waveform**. If you want to save the results for a simulation, you have two options:

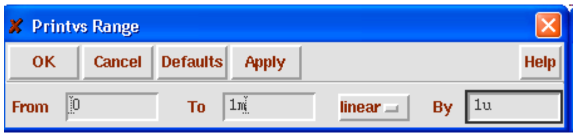
- *Save the waveforms displayed in the waveform window in a graphic file (e.g., PS)*. In the waveform window (Fig. 10), select **Window>Hardcopy**. The window in Fig. 29 pops up. Deselect **Plot with Header** and **Mail Log to ...** options. Select the type of graphic format choosing the **Plotter Name** (ps106-q for PS), select **Send Plot only to File** and write the filename. Finally, click OK.

- *Save the data of a certain waveform in a text file*. You may prefer to save the waveform data in a spreadsheet text file (X axis in first column, Y values in second column) for importing it in Matlab, Excel, etc. In this case, from the Calculator window (Fig. 9), select **printvs**. A window pops up (Fig. 30) where you can introduce the data range you want to save. A results display window appears with the data (Fig. 31a). Usually you prefer to save the data in scientific notation. You can change the data representation by choosing **Expressions> Display Options**. The window of Fig. 31b appears. Then, save the data to a file by Selecting from this window **Window>Print**. The window of Fig. 32 appears. Select **Print to File**, and write the filename.



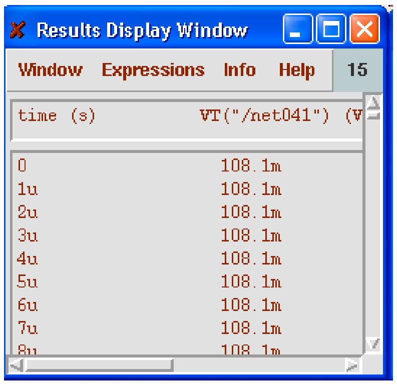
Plot from window 11
Plot With ☐ header ☐ notes
Notes
(use <Shift-Return> for new lines.)
Display Type psb
Plotter Name ps106-q
Paper Size A
Orientation ◇ portrait ◇ landscape ◆ automatic
Plot Size 0.0000 x 0.0000 inches
Offset 0.0000 x 0.0000
Number Of Copies 1 Local Tmp Directory /tmp
☐ Queue Plot Data At 10 : 00 PM Monday
■ Send Plot Only To File results.ps
☐ Mail Log To alopmart

Figure 29. Printing the waveform window to a file.

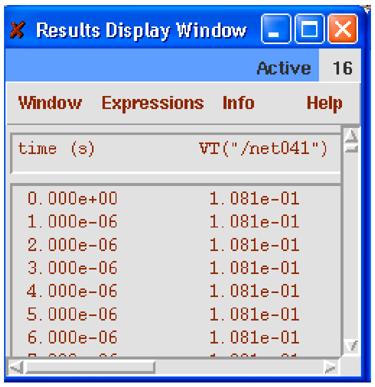


From 0 To 1m linear By 1u

Figure 30. Data range selection



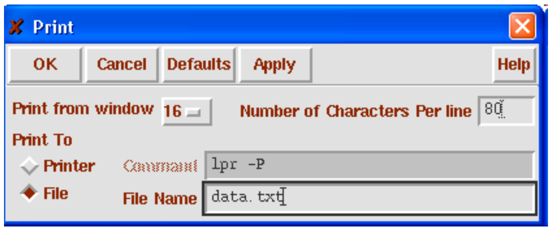
```
time (s)               VT("/net041")  (V
0                      108.1m
1u                     108.1m
2u                     108.1m
3u                     108.1m
4u                     108.1m
5u                     108.1m
6u                     108.1m
7u                     108.1m
```



```
time (s)               VT("/net041")
 0.000e+00              1.081e-01
 1.000e-06              1.081e-01
 2.000e-06              1.081e-01
 3.000e-06              1.081e-01
 4.000e-06              1.081e-01
 5.000e-06              1.081e-01
 6.000e-06              1.081e-01
```

(a) (b)

Figure 31. Data displayed (a) Engineering notation (b) Scientific notation



Print from window 16 Number of Characters Per line 80
Print To
◇ Printer Command lpr -P
◆ File File Name data.txt

Figure 32. Printing data to a file

7. REFERENCES

The basic and most complete reference is the Cadence OnLine Help. It can be accessed from the **Help** menu of any Cadence Window, or writing ***cdsdoc&*** in the UNIX shell. The window shown in Fig. 33 appears. You can browse among the libraries to find the document required.

Online information concerning the NCSU Kit can be found in **tesla**, in the following directory:

/Kits/NCSU/local/doc.

Try opening the index page, writing from this directory: **$ netscape index.html**

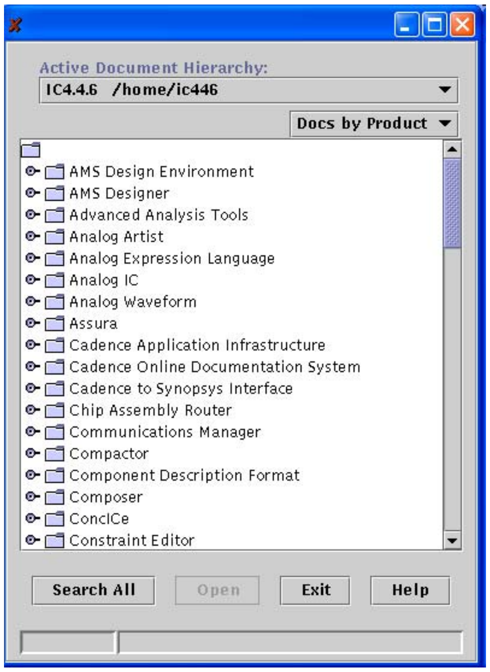


Figure 33. Cadence online help

There are also several of Cadence tutorials spread over the Internet (better than this manual in most cases), with varied contents and quality of presentation. Some examples are:

1. Worcester Polytechnic Institute.

http://turquoise.wpi.edu/cds/flow.html

> Complete on-line tutorial for a typical bottom-up analog design flow using CADENCE Custom IC Design Tools (version 97A). The examples were generated using the HP 0.6 um CMOS14TB process technology files, prepared at North Carolina State University (NCSU) and made available through MOSIS.

2. Virginia Polytechnic Institute and State University

http://www.ee.vt.edu/~ha/cadtools/cadence/cadence.html

More up-to-date tutorial covering more topics, but with less graphical support. It is a tutorial customized to the machines and tools of the Virginia Tech.

3. Delft University of Technology

http://warga.et.tudelft.nl/cadmgr/dimes03Tutorial

It describes how a integrated circuit in the (owned by Delft University) DIMES03 technology can be designed, from schematic to GDS file, using the Cadence design system. As an example a simple differential amplifier is created that consists of 4 bipolar transistors and 5 resistors. Important aspects of integrated circuit design are discussed, such as schematic entry, simulation, layout synthesis, DRC, extraction, LVS, re-simulation and GDS file generation. The part concerning Virtuoso XL is particularly interesting.

Relevant additional information can be found in:

1. Cadence homepage: **http://www.cadence.com/**

2. MOSIS homepage: **http://www.mosis.org/**
A basic reference for design kits, transistor models, submission, etc, etc.

3. North Carolina State University Cadence Homepage: **http://www.cadence.ncsu.edu/**
Detailed info concerning the NCSU kit can be found here.

4. Cadence University Program: **http://www.cadence.com/company/university/na.html**

APPENDIX. ADVANCED TOPICS

A.1. TRANSITION GUIDE FROM TANNER TOOLS TO CADENCE

Designs already completed using Tanner Tools (L-Edit, Spice Netlists, etc.) can be employed in the Cadence environment, so that modification of a former design performed using these tools does not have to start from scratch. However particular attention should be paid to the verification of an imported design.

Layouts performed in L-Edit can be imported in Cadence if they are saved in a common standard format such as CIF. Concerning Spice netlists, they can be imported as text files and simulated in the Cadence environment with minor modifications. For simulating a Spice netlist using Spectre in the Cadence framework:

1) Log in tesla. You should set up the correct environment before, as described in Section 3.

2) Copy to your home directory in tesla the Spice netlist and model files required. For instance, for simulating an inverter you can use file inv.scs containing the netlist. You have to include the path to the model files and a line where you instruct spectre what netlist syntax is employed (in this case, spectre):

```
CMOS inverter Transfer Characteristics

include "./d25.m"
simulator lang=spice

** The above line tells spectre that we use the standard SPICE syntax instead
** of Spectre's own syntax
VDD 1 0 2.5

** MOS D G S B type Width Length
M1 3 2 0 0 d25P W=1.8U L=0.24U
M2 3 2 1 1 d25N W=0.84U L=0.24U

***** For DC analysis  only ***********
VIN_DC 2 0 0
.DC VIN_DC 0 2.5 0.1
.PLOT DC VIN_DC(2 0) VOUT(3 0)

***** For Transient analysis only ****
**VIN_TR 2 0 PWL(0NS 2.5 0.1NS 2.5 0.15NS 0 2NS 0 2.05NS 2.5)
**.TRAN 0.05NS 3NS

*.PROBE
.END
```

Study the netlist. The source file performs a DC analysis. If you wish to make a transient analysis, comment out the lines under "For DC analysis only," and remove "*" for the lines under "For Transient analysis only."

The model file is in this case d25.m. It also includes the **simulator lang** line:

```
*SPICE 3f5 Level 8, Star-HSPICE Level 49, UTMOST Level 8

simulator lang=spice

* DATE: Feb 23/01
* LOT: T0BM              WAF: 07
* Temperature_parameters=Default
.MODEL d25N  NMOS (                          LEVEL   = 11
+VERSION = 3.1          TNOM    = 27          TOX     = 5.8E-9
+XJ      = 1E-7         NCH     = 2.3549E17    VTH0    = 0.3907535

..........

+PK2     = 2.589681E-3   WKETA   = -1.866069E-3   LKETA   = -0.0166961     )
*
.MODEL d25P  PMOS (                          LEVEL   = 11
+VERSION = 3.1          TNOM    = 27          TOX     = 5.8E-9
+XJ      = 1E-7         NCH     = 4.1589E17    VTH0    = -0.583228

..........

+PK2     = 2.100359E-3   WKETA   = 5.428923E-3    LKETA   = -0.0111599     )
*
```

- Spectre is case sensitive unlike standard SPICE. This model description, however, uses SPICE syntax, not Spectre's (notice the "simulator lang" line). This command allows to use a different syntax with Spectre.

- To peform Spice simulation of this file, type the command. `spectre inv.scs`

- To plot the results, first set the DISPLAY environment variable in tesla (if this was not done before):

```
Bash_2.05a$ DISPLAY=your_ip_address:0
Bash_2.05a$export DISPLAY
```

- Type the command. `awd -dataDir inv.raw`
  Four windows appear. Activate "Result Browser" window.

- Click left buton on inv.raw. Yellow node numbers show up on the right end of the hierarchy.

- Clcik right button on nodes 2 and 3. "Waveform Window" displays the waveform.

- To make a hard copy of the plot:
  - Choose hardcopy menu from Windows menu on "Waveform Display" window. A windows appears.
  - Verify Laser Writer is chosen as "Plotter Name".
  - Click "Send Plot Only to File.," and type in the file name. Click apply to generate a postscript file.

You can also save a text file with X and Y values in columns.

ADDITIONAL INFO

For details of Spice and Spectre, refer to the Cadence online manuals. Choose the following menus in the sequence.

- For SPICE choose HSPICE > "HSPICE/SPICE Interface ..."
- For Spectre choose Spectre > "Affirma Spectre Circuit Simualtor User Guide."

A.2. INTRODUCTION TO SKILL

The SKILL language has been developed by Cadence to be used with their tool suites. It allows the user to write a "script" to perform any command in Cadence. SKILL is an interpretive language like *LISP* and *Perl*. SKILL was designed to work on repetitive tasks and several of its functions are based on lists.

For more information about the structure of SKILL code, you can go to the following sources:

- Finder: Allows the user to search its database by entering a keyword. Finder will display all functions containing the keyword. For each function, it gives the usage and a brief description about the function.
  To invoke: From the CIW, type startFinder()
- On-line help: Detailed manual about Cadence that includes function reference and user guides to SKILL. It contains the functions for the other cadence tool suites as well.
  To invoke: From the terminal, type cdsdoc& --> click on the SKILL Language menu

A simple SKILL procedure is given as an example below. The objective of this procedure is to modify all nmos and pmos objects' bulk node value to be vss! and vdd!. This function is library specific and is given here as an example only.

To execute this procedure:

1. Copy the following procedure and save it as "ChangeProperty.il"
2. From the CIW, type: load "ChangeProperty.il"
3. To execute the procedure, type: cb(libraryname cellview)

Example: cb("vlsi_proj" "tspc_rs")

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;;This procedure will change the bulk node values of nmos and pmos
    ;;to be vss! and vdd!, respectively.
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    procedure(cb(lib cell)
    ;let will allow you to access the cv and newlabel variables outside
    ;of this procedure
    let((cv newLabel)
    ;open the cellview as a database object and assign it to cv
    cv = dbOpenCellViewByType(lib cell "schematic" "" "a")
    ;for loop to go through all instances in this cellview
    foreach(instance cv~>instances
    ;open the property window -- this will display the window!
    geSelectObjectNoFilter(instance)
    schHiObjectProperty()
    ;check the cellname and change the bulk node (bn) property
    if(schObjPropForm->cellName->value == "nmos"
    then
    schObjPropForm->bn->value = "vss!"
    )
    if(schObjPropForm->cellName->value == "pmos"
    then
    schObjPropForm->bn->value = "vdd!"
    )
    ;close the form
    hiFormDone(schObjPropForm)
    geDeselectAll()
    );end foreach
    );end of let
    );end procedure
```

- **Building Nets, Terminals, and Pins**

The following steps show how to define pins in a layout using SKILL.

1. Create the shape that will serve as the pin. The shape is usually a rectangle. Note: The shape cannot be a polygon.
`fig = dbCreateRect( pcCellView layer list( x1:y1 x2:y2))`

2. Create the net to which the pin attaches. In this example, the pin name n1 matches the name of the corresponding pin in
the schematic symbol for this cell:
`net = dbCreateNet( pcCellView "n1")`

3. Create a terminal on the same net. The terminal must have the same name as the net and match the terminal type. In this example, the terminal type is inputOutput, the same terminal type as the corresponding pin in the schematic symbol:
`trm = dbCreateTerm( net "n1" "inputOutput")`

4. Create a pin:
`pin = dbCreatePin( net fig "n1")`

The pin database object connects the pin figure with the net. The pin name can match the terminal name, but does not have to. In the example, the pin name n1 matches the terminal name. Within the pcell, you can have multiple shapes that all belong to the same electrical terminal. Each shape would have a pin to associate it to the same net. In such cases, each pin is created on the same net and must have a unique pin name.

5. If your tool requires pins to have an access direction, define the access direction:
`pin~>accessDir = '( "top" "left")`

The access direction is a list identifying the correct sides of the pin shape for connection.

A.3. LOGIC SIMULATION WITH VERILOG

This section shows how to perform logic simulation using Verilog. The procedure is for a quick and simple solution, and it does not explore full features of Verilog. The example to be used is a 2x1 multiplexer. At this point, you should have set up the environment. Otherwise, refer to Section 3.

- Before you start, you have to have a schematic diagram (view name: schematic) of a multiplexer. Each gate symbol used in the schematic diagram should have a switch-level schematic diagram (view name: schematic) in MOSIS library.

- Verilog simulator imposes two severe constraints on the logic style for switch-level schematic diagrams. The constraints for a switch-level schematic diagram are;
  - Current should flow only in one direction for a transistor. It means that the drain and the source of each transistor are to be fixed in the schematic diagram.
  - It seems that high impedance Z is not defined. Hence, transmission gates cannot be simulated properly.
- Move to your working directory. Edit the library path file, cds.lib, if necessary. Invoke Cadence software by typing icfb&
- In Command Interface Window (CIW), select the following menus.
  file -> open ...
  A new window appears. Set the following items, and click OK.
  - Library Name: MOSIS
  - Cell Name: My_Mux
  - View Name: schematic
- In Composer Window, select the following menus.
  Tools -> Simulation -> Verilog-XL
  A new window appears. Verify Run Directory is "My_Mux.run1", and click OK.
- In Verilog window, select the following menus.
  Stimulus -> STL -> Edit Stimulus...
- A window informing non-existent of the netlist appears, and click OK.
  Edit STL windows appear. Check "Compile STL after Editing?", and click OK.
- A text editor window appears. You should enter stimulus patterns in the file. Look for the line xv(0 0 0). This line means the first stimulus vector is (0 0 0) for the inputs defined in the top part of the file. Add other stimulus vectors. For the 2x1 multiplexer with three inputs, enter the following vectors.

  ```
  xv(0 0 0)
  xv(0 0 1)
  xv(0 1 0)
     ...
  xv(1 1 1)
  ```

  Save the file.
- A window asking "Do you want to make..." appears. Click YES.
- In Verilog window, click "Start Interactive" menu button which is on the top left in the menu. It starts to compile the netlist. Watch for any errors in the window.
- In Composer window, select all inputs and outputs to be monitored. To select multiple nodes, hold down SHIFT key, and encircle input/out pins as a group or individually.
- In Verilog window, click "View Waveforms" menu button which is on the bottom right in the menu. A waveform window with list of select I/Os appears.
- In Composer window, Click "Continue" menu button which is the right button on the second row.
- Examine the waveforms in "View Waveforms" window.