Module 1: Introduction to ADE 5.0

Topics in this module

- Course objectives
- Course outline
- Class schedule
- Getting help, technical support, and documentation
- The Design Framework II Design Environment
- Accessing design tools
- Creating a library
- Creating cells and cell views
- Schematic capture
- Analog simulation
- Analyses
- Summary

Terms and Definitions

CDSDoc	Cadence® online help tool that uses a Netscape browser interface.
CIW	Command interpreter window. Interface used to access DFII applications.
command line	A line buffer in the CIW that accepts SKILL-based commands.
text field	An area on a tool window where the user provides data.
cyclic field	Set of selectable options in a tool window, denoted by a small rectangle.
library	A set of design directories that includes ‘cells’ and ‘cellviews’.
Library Manager	A Cadence tool that allows user to browse and edit a design library.
Composer	Schematic editor and symbol generation tool in DFII.
cell	A basic unit of a design hierarchy described by cell views.
cell view	A specific view of a cell that includes schematic, symbol, or layout.
instance	A uniquely named placement of a symbol onto a schematic.
pin	A connection point on a schematic and symbol used for accessing signals.
bindkey	A predefined key on the keyboard that invokes a preselected command.

Course Objectives

- Learn how to create schematics, symbols, and a design hierarchy
- Set up and run analog simulations
- Analyze simulation results
- Evaluate sensitivities and mismatches to improve circuit performance.
- Run Corners, Monte Carlo, and Optimization tools to improve yield
- Create and use OCEAN scripts and SKILL to set up and run simulations
- Understand the Component Description Format (CDF)
- Create configurations with the Hierarchy Editor (HED)
- Use subcircuits and macromodels
- Run the parasitic simulation flow
- Use advanced tools to solve special problems

Course Objectives

The objective of this class is to provide both instruction and materials on using the Cadence Analog Design Environment. The objective for this course of instruction is that the student can utilize the entire front-to-back design flow of the Analog Design Environment software.

Course Outline

1 Introduction to ADE 5.0

2 Schematic Entry

3 Analog Simulation

4 Simulation Results Display Tools

5 Analyzing Simulation Results

6 SKILL and OCEAN

7 Parametric Analysis

8 Corners Analysis

9 Monte Carlo Analysis

10 Optimization Analysis

11 Circuit Surfer

11 Component Description Format (CDF)

12 Macromodels, Subcircuits, and Inline Subcircuits

13 Inherited Connections

14 The Hierarchy Editor

15 Overview of Parasitic Simulation

16 Assura Parasitic Simulation Flow

Appendixes:
A Diva Parasitic Simulation Flow
B WaveScan Display Tools
C Spectre MDL
D Match Analysis, dcmatch
E Advanced Topics in ADE

Course Outline

The Analog Design Environment classroom series is an intensive, fast-paced, 4-day class on using Cadence design software to run analog circuit simulations.

This Analog Design Environment course comprises 17 modules, including both lectures and corresponding lab activities.

Module 15 provides an overview of the Parasitic Simulation theory and flow, followed by Module 16, Assura Parasitic Simulation Flow. Appendix A, Diva Parasitic Simulation Flow, will be taught when conditions require this flow to be used. The selection of Diva or Assura will be based on the specific classroom and work site circumstances.

The Appendixes B, C, and D discuss other tools and features of the Analog Design Environment. Finally, Appendix E, Advanced Topics in ADE, provides a brief overview of Verilog-A and Mixed Signal Design Environment.

Class Schedule

Day 1

1	Introduction to ADE 5.0
2	Schematic Entry
3	Analog Simulation
4	Simulation Results Display Tools

Day 2

5	Analyzing Simulation Results
6	SKILL and OCEAN
7	Parametric Analysis
8	Corners Analysis
9	Monte Carlo Analysis

Day 3

10	Optimization Analysis
11	Circuit Surfer
11	Component Description Format (CDF)
12	Macromodels, Subcircuits, and Inline Subcircuits

Day 4

13	Inherited Connections
14	The Hierarchy Editor
15	Overview of Parasitic Simulation
16	Assura Parasitic Simulation Flow

Class Evaluations

Class Schedule

The class schedule listed above is the recommended schedule for the standard 4-day classroom series of ADE 5.0. Private classes taught at customer facilities may require adjustments or modifications to this schedule.

The class schedule is intended to adequately cover all topics of the ADE course. As such, each day covers 4 to 5 modules. Each module requires 1 to 2 hours to complete both the lecture presentation and the lab activity.

To ensure adequate coverage of the lecture material, the instructor may elect to defer answers to detailed questions that are asked during the lecture. The instructor may also deviate from course schedule due to conditions that cause unexpected delays or other circumstances. In any case, the instructor will attempt to answer relative questions in a timely manner or obtain the answer for the student as soon as possible.

Getting Help

You can get help with Cadence software from the following sources:

- Help button on forms and windows
- Cadence online documentation (CDSDoc)
- Education Services training manuals
- SourceLink® online customer support
- Customer Response Center (CRC)

Getting Help

Online Help

Cadence reference manuals and online help files for each product are installed automatically when installing the product. Hard copies of the reference manuals are available from Cadence. All these online documents are part of the online help system, which can be accessed as follows:

- View relevant product information by clicking the help button on windows and forms. Use this information to complete a form or what can be done in the window.
- Start the CDSDoc documentation from a UNIX shell by typing `cdsdoc&` at the command line and search through all Cadence reference manuals and online help systems installed with each product. Also, use CDSDoc to print the reference manuals entirely or just the relevant material.

Other Means of Getting Information

- With a software maintenance agreement, subscribe to the SourceLink online support system and view known problems and solutions or communicate with other users. The SourceLink system is accessible via the internet. To open an account, send email to crc_customers@cadence.com.
- Training manuals, like this one, can supplement reference manuals.
- When the above information is insufficient, call the Customer Response Center.

Overview of Analog Design Environment

Analog Design Environment is a software tool set within Design Framework II that is used to set up and run analog simulations. The Analog Design Environment also accesses and views the simulation results.

The Analog Design Environment allows you to:

- Choose the simulator host
- Choose the type of analysis: ac, dc, transient, parametric, sensitivity, etc.
- Set design variables: Vdd, frequency, Cout, etc.
- Append model files and include files
- Netlist and run simulations
- Quickly alter the simulation setup and rerun the simulation
- Plot simulation results in the Waveform display tool
- Evaluate simulation results using waveform expressions
- Run multiple simulation tools: Corners, Monte Carlo, Optimizer, etc.
- Automatically set up, save, and run OCEAN scripts

Overview of Analog Design Environment

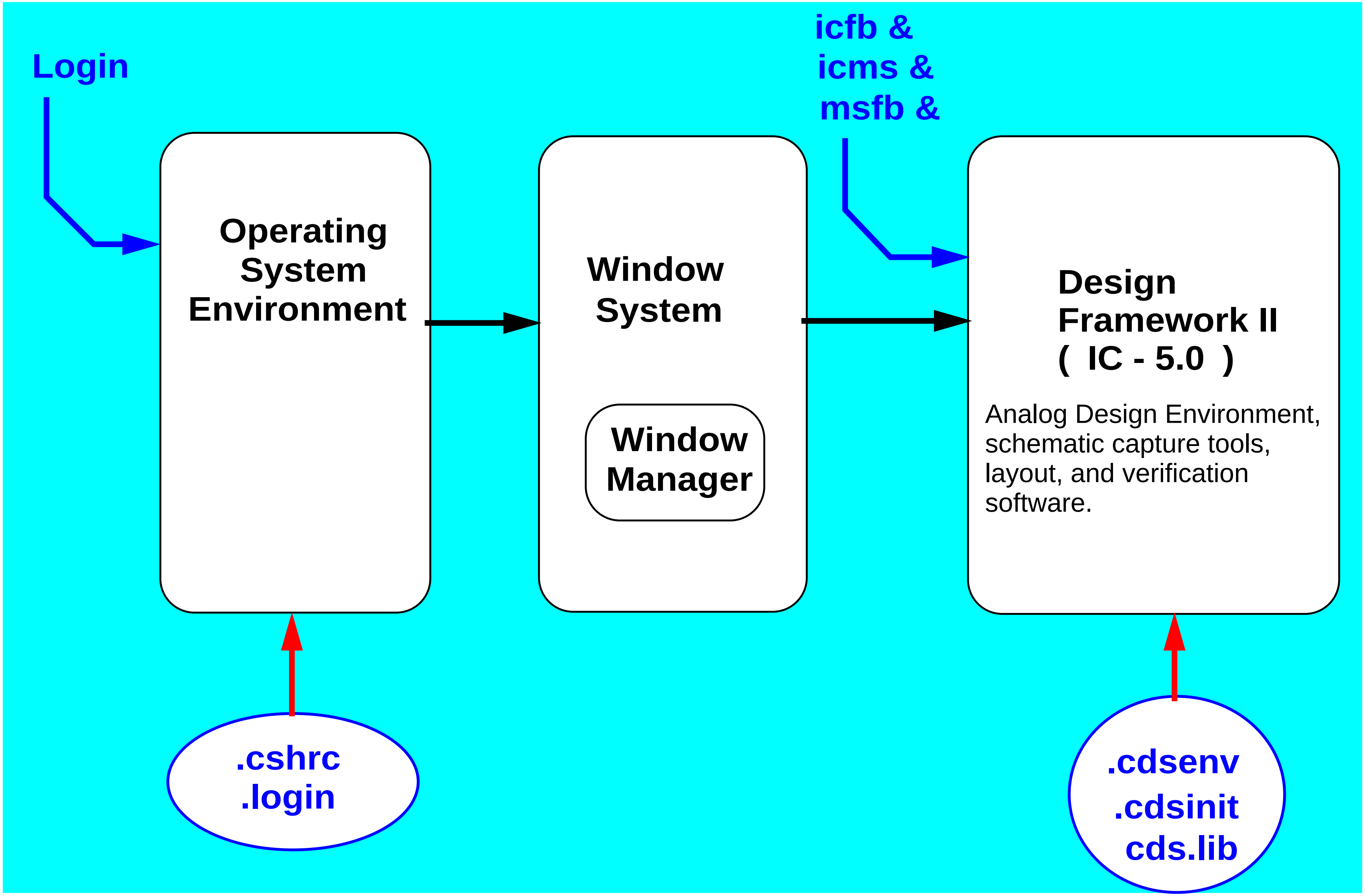
The Analog Design Environment is a set of software design tools used to set up, control, and run circuit simulations. ADE allows you to choose the simulator host, set design variables, select model files, and to select analyses to add, modify, or delete from next simulation run.

The Analog Design Environment provides a user-friendly graphical interface that includes pull-down menus and icons for making fast and easy changes.

The Analog Design Environment also provides control for accessing the simulation results and displaying the results to the Waveform Display tool. The results can be entered into other tools for waveform processing or to obtain specific data using expressions.

The Analog Design Environment provides access to multiple simulation tools. ADE can be used to start Corners, Monte Carlo, and the Circuit Optimizer.ADE also allows you to automatically set up, save, and run OCEAN scripts.

Design System Initialization Files



Analog Design Environment, schematic capture tools, layout, and verification software.

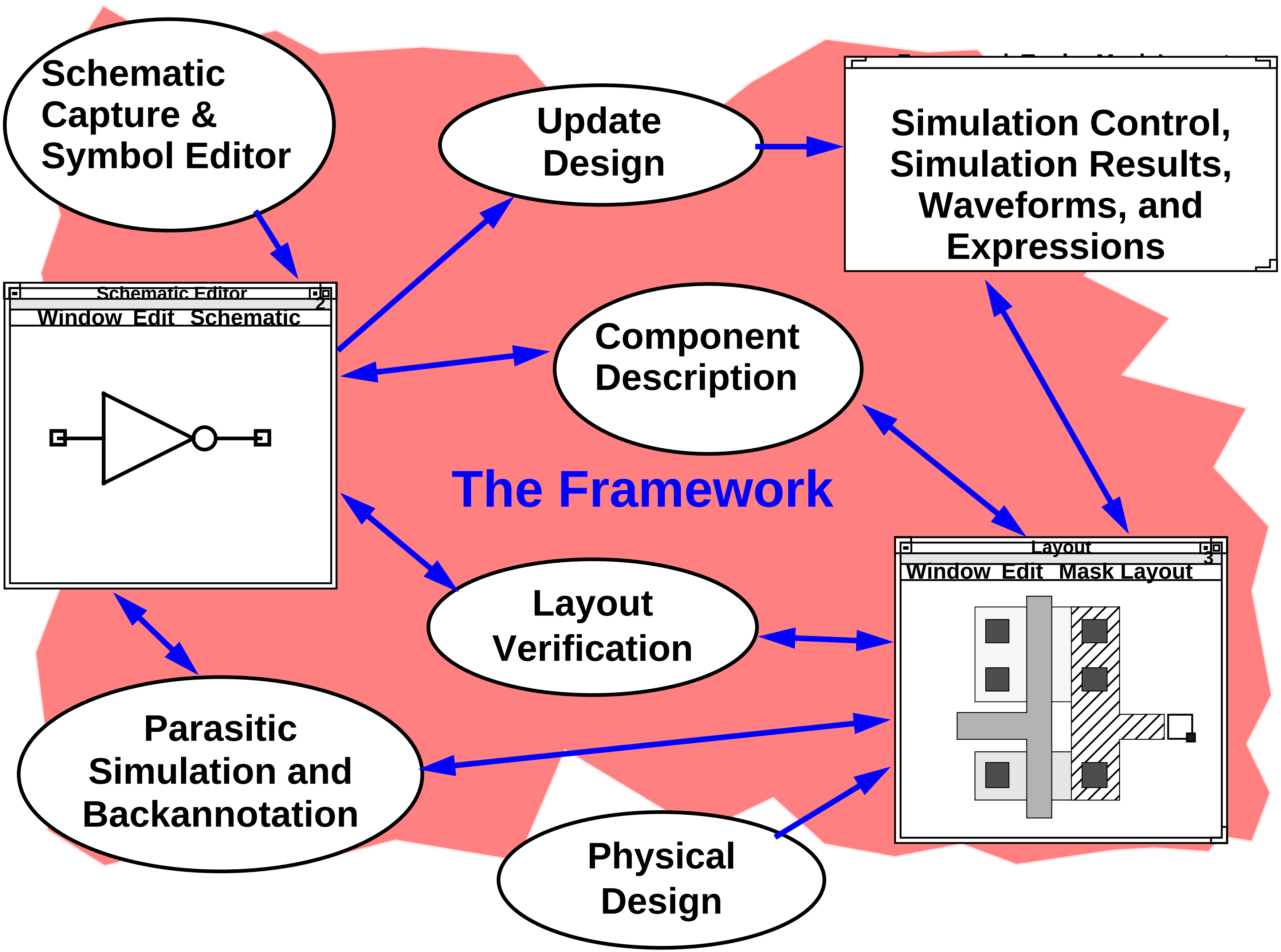
Design System Initialization Files

There are some design system initialization files that configure the operating system environment. For example, the *.cshrc* and *.login* files configure the UNIX environment when you log in and start a UNIX application.

The initialization file, *.cdsinit*, customizes the Analog Design Environment. The *cds.lib* file sets the paths to the libraries. These files, along with the *.cdsenv* file, are discussed later.

For more information on configuring your operating system environment for the Analog Design Environment, consult the *Cadence Design Framework II Configuration Information* guide.

Overview of the Design Framework II Environment



Overview of the Design Framework II Environment

The Cadence Analog Design Environment is a set of design tools that operate within Design Framework II. Design Framework II is the underlying structure for Cadence design tools for schematic capture, analog simulation, and layout. It provides a single integrated environment for accessing all tools and design data, including the ability to:

- Access to the Command Interrupter Window (CIW) using ***icfb***, ***icms***, or ***msfb***.
- Use the Library Manager Tool to browse design libraries and open cell views.
- Create new libraries, cells, and cell views.
- Start or edit a schematic view or symbol view.
- Start or edit a layout design.
- Run layout verification.
- Start the Analog Design Environment and run simulations.
- Access simulation results directly using the Results Browser.
- Run OCEAN scripts.

Advantages of Using Design Framework II

- Common software environment for using schematic capture, simulation, layout, and design verification
- Easy-to-learn, consistent user interface
- Similar appearance between most forms and windows
- Communication between software tools within the DFII environment
- Tool windows remain open while running other applications
- Data can be “back annotated”
  - From layout to schematic
  - From simulation to schematic
  - From simulation to layout
  - etc.
- Applications may be customized or automated using SKILL or the OCEAN command language

Advantages of Using Design Framework II

The Cadence Design Framework II environment is an integrated design environment. An integrated environment means that numerous tools and applications operate together. For Design Framework II the environment provides schematic capture, simulation control, netlist generator, circuit simulator, waveform display, layout and verification tools.

For the design software tools, DFII provides:

- Consistent user interface

  Analog applications in the design framework have the same “look and feel.” Menu items are often in the same place in every application.

- Consistent database

  A consistent database stores all design information. Tools share data in real time so long formalized translations between tools are not needed. The DFII environment also saves time during schematic to layout verification, because it updates layout geometries as the schematic component parameters change.

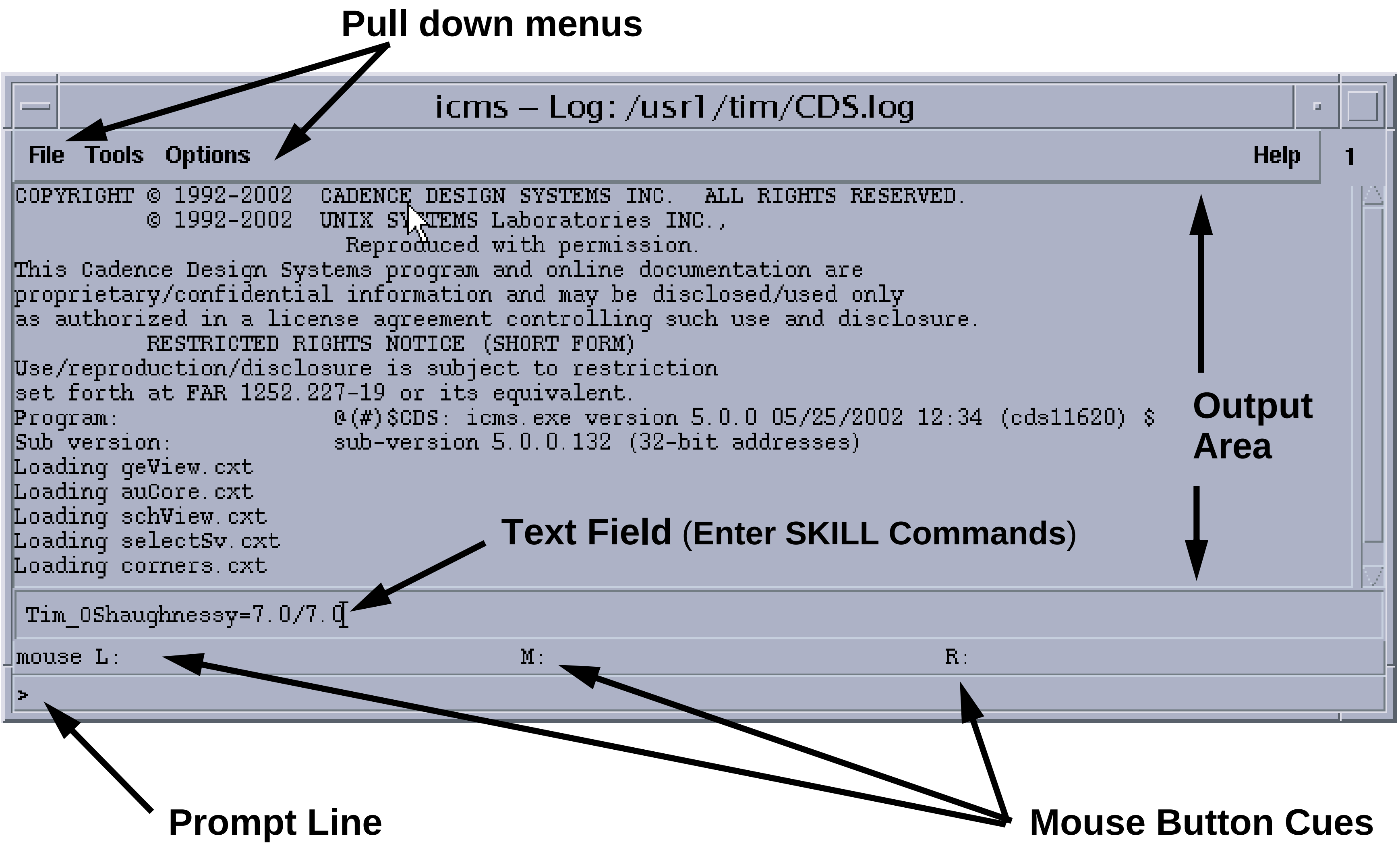
- Cooperating tools

  Applications run concurrently, with results available to all other tools, eliminating the need to open and close applications when changing tool contexts. For example, update and simulate a schematic without restarting the simulation environment. Updates are known to the simulation window as soon as they are made in the schematic entry window.

The Command Interpreter Window (CIW)

Enter: ***icfb &***, ***icms &***, *or* ***msfb &***

This Is Your Control Panel for DFII Applications!



```
COPYRIGHT © 1992-2002  CADENCE DESIGN SYSTEMS INC.   ALL RIGHTS RESERVED.
          © 1992-2002  UNIX SYSTEMS Laboratories INC.,
                         Reproduced with permission.
This Cadence Design Systems program and online documentation are
proprietary/confidential information and may be disclosed/used only
as authorized in a license agreement controlling such use and disclosure.
          RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Program:                @(#)$CDS: icms.exe version 5.0.0 05/25/2002 12:34 (cds11620) $
Sub version:            sub-version 5.0.0.132 (32-bit addresses)
Loading geView.cxt
Loading auCore.cxt
Loading schView.cxt
Loading selectSv.cxt
Loading corners.cxt
```

```
Tim_OShaughnessy=7.0/7.0
```

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

```
>
```

Pull down menus

Output Area

Text Field **(Enter SKILL Commands)**

Prompt Line

Mouse Button Cues

The Command Interpreter Window (CIW)

The Command Interpreter Window (CIW) is at the heart of the framework system. Use this window to access framework-based applications. System and error messages from applications are reported in this window.

- **Output Area**

  The output area displays a running history of the commands used with their results. For example, it issues a status message when a cell library is opened. This data is saved in the *Log File* whose path appears as the application title of the CIW. Use scroll bars to view previous output pane data without having to resize the CIW.

- **Text Entry Field**

  Enter Cadence SKILL commands in this area. Every pull-down menu command in the Design Framework II environment has an equivalent SKILL command. Advanced users can define and execute their own SKILL commands by entering them here.

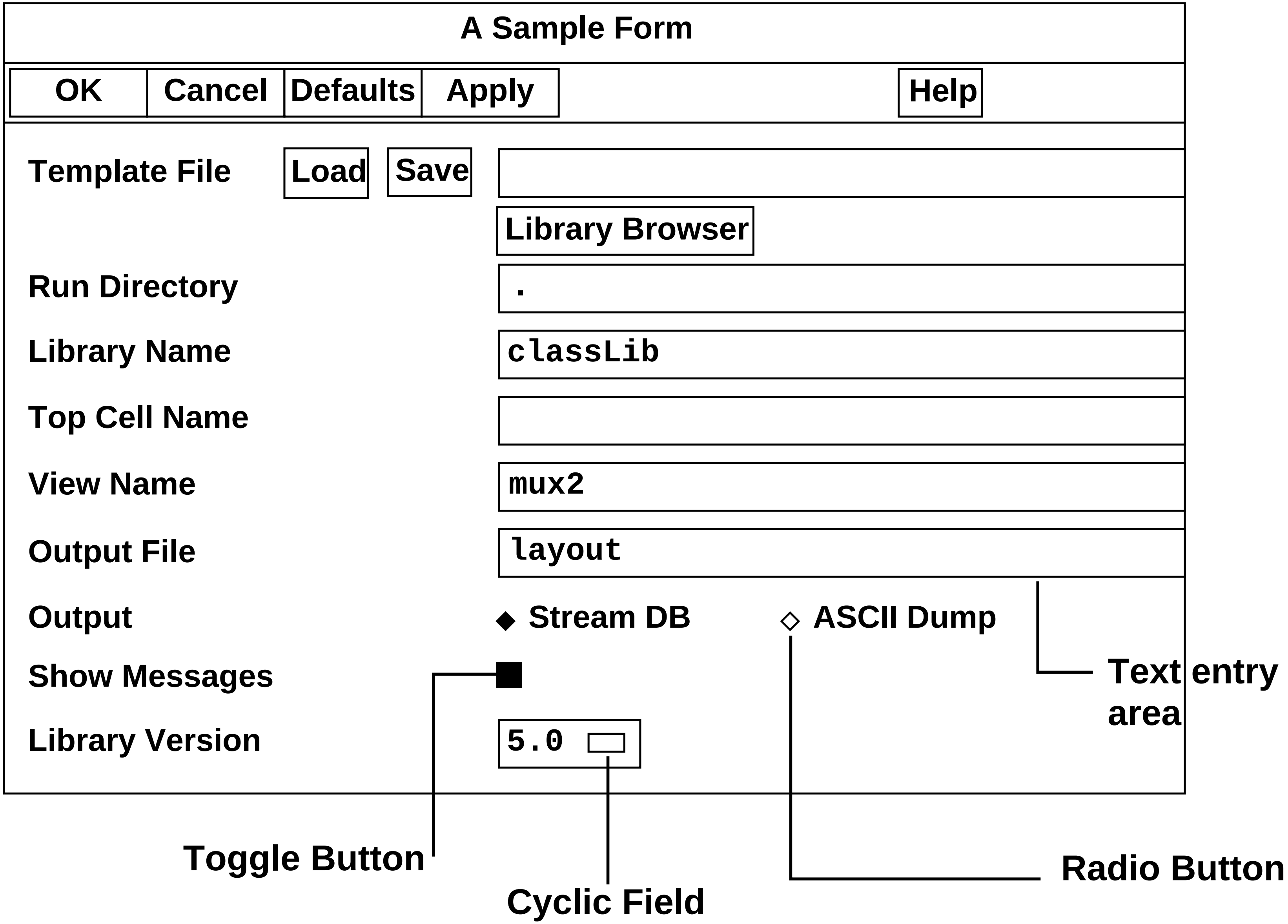
- **Prompt Line**

  The prompt line at the bottom of the CIW indicates the next step when carrying out a command executed in any Design Framework II application window.

- **Mouse Button Cues**

  Tells which mouse button to push to execute a command in a Design Framework II window.

Using a Form



A Sample Form

Template File	
Run Directory	.
Library Name	classLib
Top Cell Name	
View Name	mux2
Output File	layout
Output	◆ Stream DB ◇ ASCII Dump
Show Messages	■
Library Version	5.0

Using a Form

Forms provide a place to enter the information required by a command.

The top of the form has a *title bar* and a set of *banner buttons*. The body of the form contains *prompts* that indicate which option is being set. Next to the prompt is one of the following:

- *radio button*, for choosing one of several options
- *text entry area*, for typing information
- *toggle button*, for turning options on or off
- *cyclic field*, for choosing one of many options. Initially only one option is shown. Move the pointer to the field and hold down the left mouse button, the other options appear.

The form might also have buttons such as *Browse*, which shows a browser window, or *More Options*, which displays another form.

Change of an entry on a form is disabled when the name appears in gray instead of black, and the text entry area is shaded.

- Press the **Tab** key or mouse to move to the next text entry field.
- Use the left and right arrows on the keyboard to move the cursor in a text entry field.

  Press **Control-a** to go to the beginning of a line; **Control-e** to go to the end of a line; **Control-u** to erase to the beginning of a line.

Initializing the Design Framework II Environment

The Design Framework II software reads your *.cdsinit* file at startup to set up your environment. The *.cdsinit* file:

- Sets user-defined bindkeys when the Design Framework II environment is started.
- Redefines system-wide defaults.
- Contains SKILL commands.

The search order for the *.cdsinit* file is:

- <install_dir>/tools/dfII/*local*
- the current directory
- the home directory

Here is the path to a sample *.cdsinit* file:

```
<install_dir>/tools/dfII/samples/artist/cdsinit
```

Initializing the Design Framework II Environment

Start the Design Framework II environment, it reads the *.cdsinit* file to set up your configuration. The search order for the *.cdsinit* file is *<install_dir>/tools/dfII*/*local*, the current directory, and finally the home directory. When a *.cdsinit* file is found, the search stops unless a command in a *.cdsinit* file reads other user files.

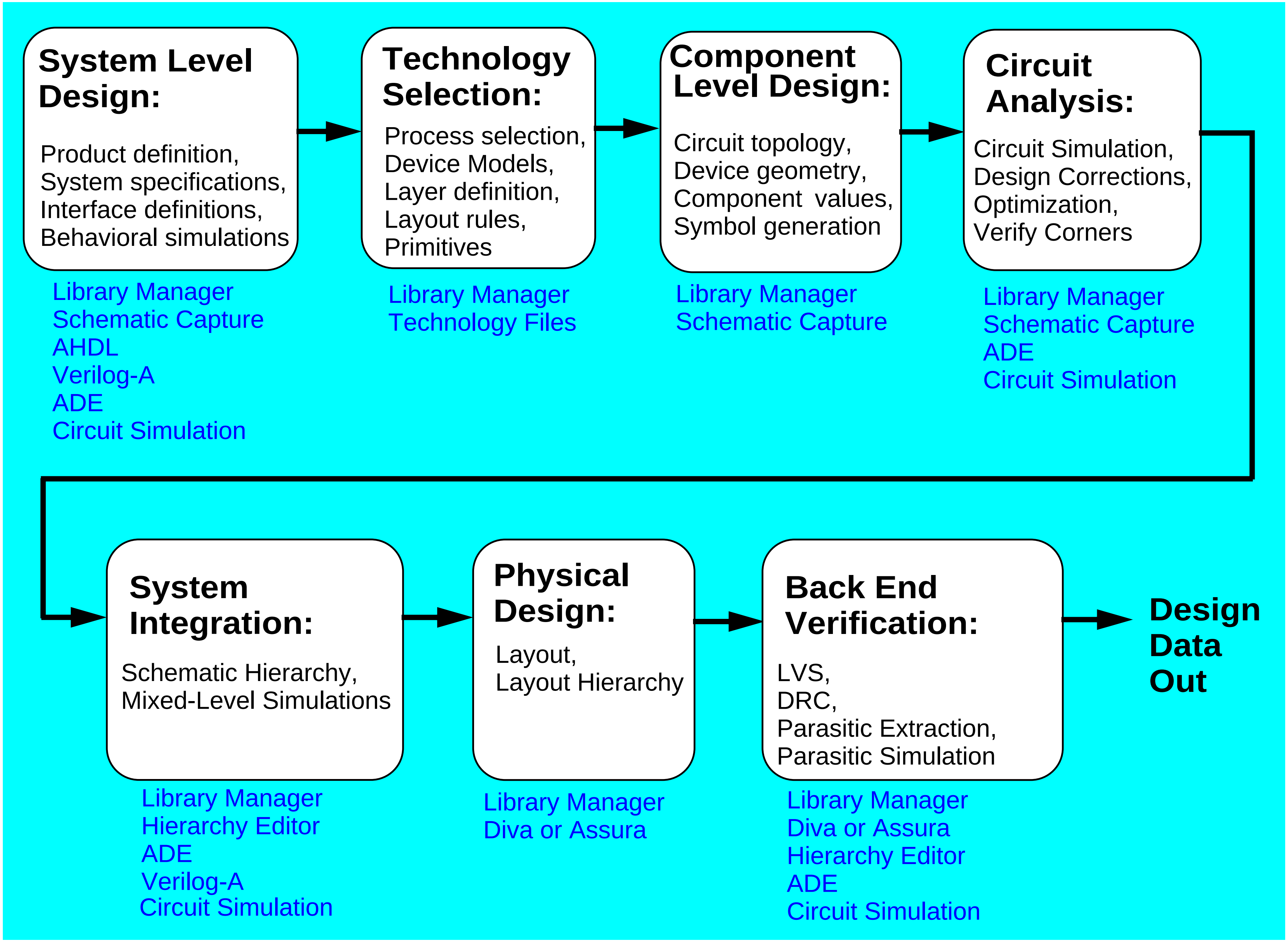
The *.cdsinit* file is a text file written in SKILL. A statement in a *.cdsinit* file can load user-defined bindkeys. Another statement might set Waveform Window defaults.

A sample *.cdsinit* file included with the software contains examples of statements to copy into your own *.cdsinit* files. It has very detailed comments about command usage. This sample is located at *<install_dir>/tools/dfII/cdsuser/.cdsinit*. An additional sample *.cdsinit* file exists for analog designers at *<install_dir>/tools/dfII/samples/artist/cdsinit*.

The Installation Path

The Design Framework II software product hierarchy is discussed in detail in the *Cadence Design Framework II Configuration Information* guide.

IC Design Flow, Front to Back



System Level Design:

Product definition,
System specifications,
Interface definitions,
Behavioral simulations

Library Manager
Schematic Capture
AHDL
Verilog-A
ADE
Circuit Simulation

Technology Selection:

Process selection,
Device Models,
Layer definition,
Layout rules,
Primitives

Library Manager
Technology Files

Component Level Design:

Circuit topology,
Device geometry,
Component values,
Symbol generation

Library Manager
Schematic Capture

Circuit Analysis:

Circuit Simulation,
Design Corrections,
Optimization,
Verify Corners

Library Manager
Schematic Capture
ADE
Circuit Simulation

System Integration:

Schematic Hierarchy,
Mixed-Level Simulations

Library Manager
Hierarchy Editor
ADE
Verilog-A
Circuit Simulation

Physical Design:

Layout,
Layout Hierarchy

Library Manager
Diva or Assura

Back End Verification:

LVS,
DRC,
Parasitic Extraction,
Parasitic Simulation

Library Manager
Diva or Assura
Hierarchy Editor
ADE
Circuit Simulation

Design Data Out

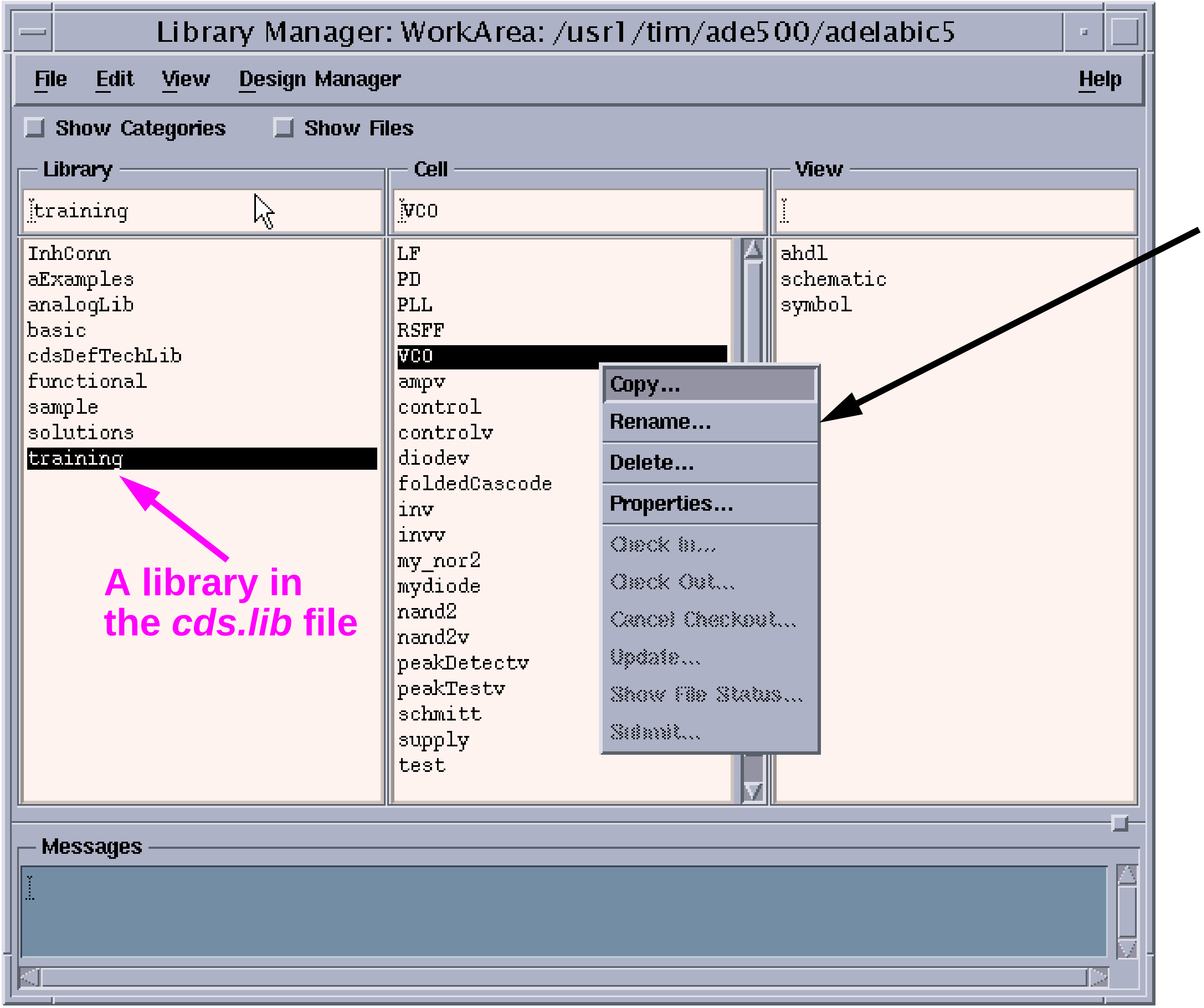
IC Design Flow, Front to Back

The graphical flow above shows a Front to Back design flow for integrated circuits or related system design. The blocks show the major steps or design categories. The text below each block shows the software tools used in the corresponding design steps.

- At the front end, the product, device or system is defined. The system-level specifications are used for behavioral simulation of the system.
- A fabrication process or technology is selected.
- A schematic of a specific block is captured.
- The design of the circuit is simulated. If needed, the circuit is redesigned to achieve specified goals.
- The circuit is integrated into a hierarchy. The hierarchy is then simulated.
- Physical design or layout capture of the circuit is completed. The layout of the hierarchy is then completed.
- Back end verification of the layout includes design rule checks, layout versus schematic checks, and parasitic extraction. The extracted parasitics are “backannotated” to the schematic for parasitic simulation using the circuit simulation software.

The Library Manager

The Library Manager is a graphical data management tool.



□ Show Categories □ Show Files

Library: training

Cell: VCO

View:

Library:
InhConn
aExamples
analogLib
basic
cdsDefTechLib
functional
sample
solutions
training

Cell:
LF
PD
PLL
RSFF
VCO
ampv
control
controlv
diodev
foldedCascode
inv
invv
my_nor2
mydiode
nand2
nand2v
peakDetectv
peakTestv
schmitt
supply
test

View:
ahdl
schematic
symbol

Object Sensitive Menus

A library in the *cds.lib* file

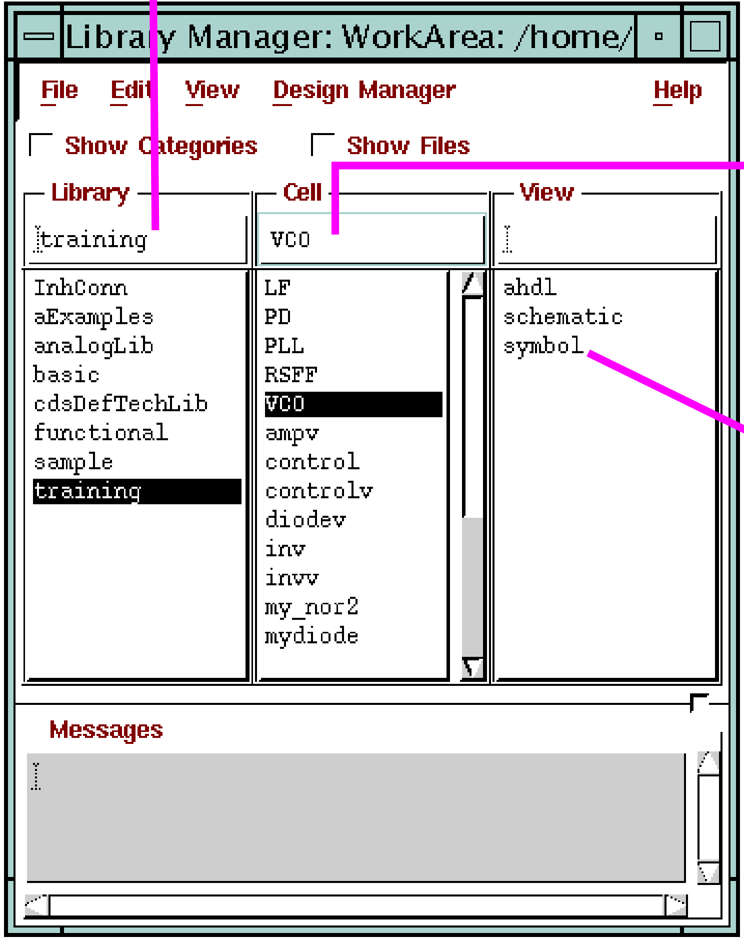
The Library Manager

The Library Manager provides a convenient way to browse libraries containing cells and cell views. The most common use is to display the contents of libraries graphically. Other functionality includes renaming, copying, specifying permissions for, creating categories for, deleting, and viewing properties of design data. Use the Library Manager to create cells and views, to edit or read a design, and to access the design manager.

The illustration above shows a fully expanded library. Initially, the Library Manager lists only the library names that are set in the *cds.lib* file. This file contains the paths to the libraries used in the design session, including example libraries provided by Cadence, such as *analogLib* and *basic*.

Expand design data with Object Sensitive Menus (OSM) or with the mouse. To expand data, point at the word that represents the data in the Library Manager and choose the appropriate mouse button or menu command.

The Library Structure

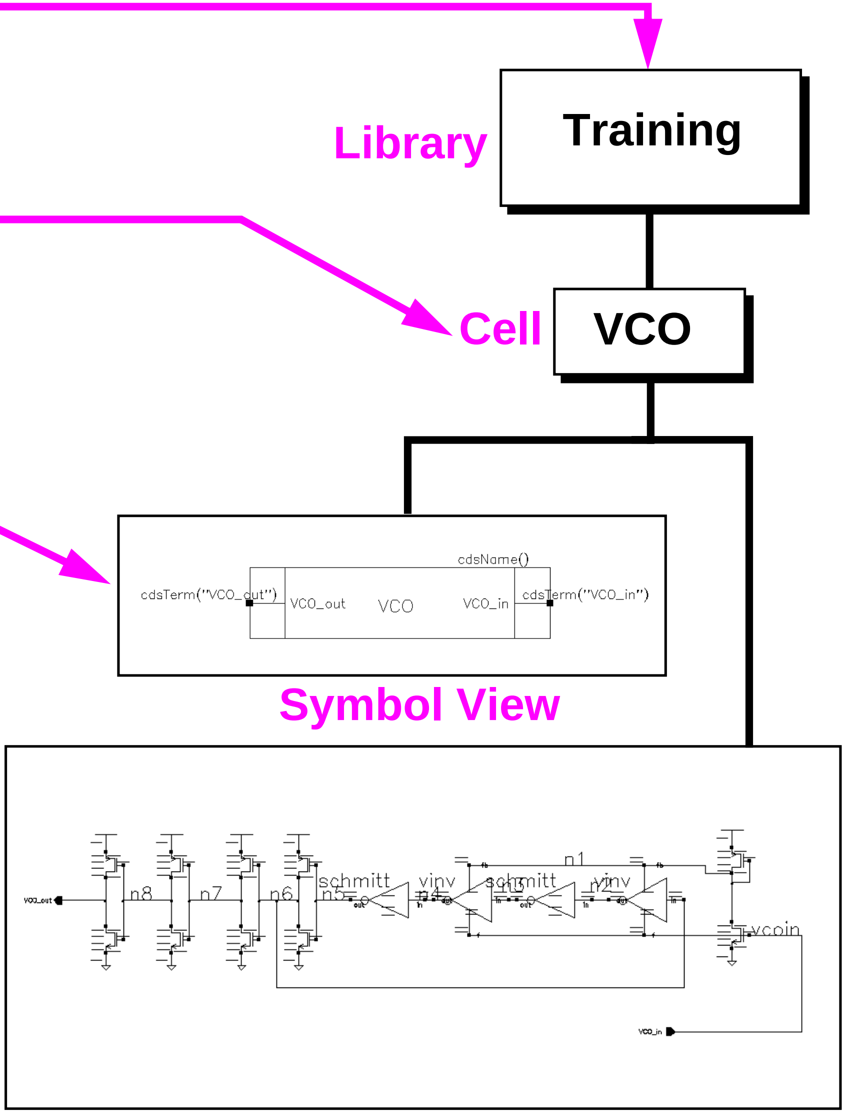


Library: training

Cell: VCO

View:

Library	Cell	View
InhConn	LF	ahdl
aExamples	PD	schematic
analogLib	PLL	symbol
basic	RSFF	
cdsDefTechLib	VCO	
functional	ampv	
sample	control	
training	controlv	
	diodev	
	inv	
	invv	
	my_nor2	
	mydiode	



Library

Training

Cell

VCO

Symbol View

Schematic View

Tools—Library Manager...

The Library Structure

- Library

  A *library* is a collection of cells. The library also contains all the different views associated with each of the cells. **Reference** libraries typically contain well-characterized cells that can be instantiated in many different designs. Examples are the *analogLib* and *basic* libraries. **Design** libraries contain cells currently under development by a particular user, group, or for a particular design project.

- Cells

  A *cell* is a logical component in your library. It can be a building block such as a *VCO* or *amplifier*. It can also be the top level chip name.

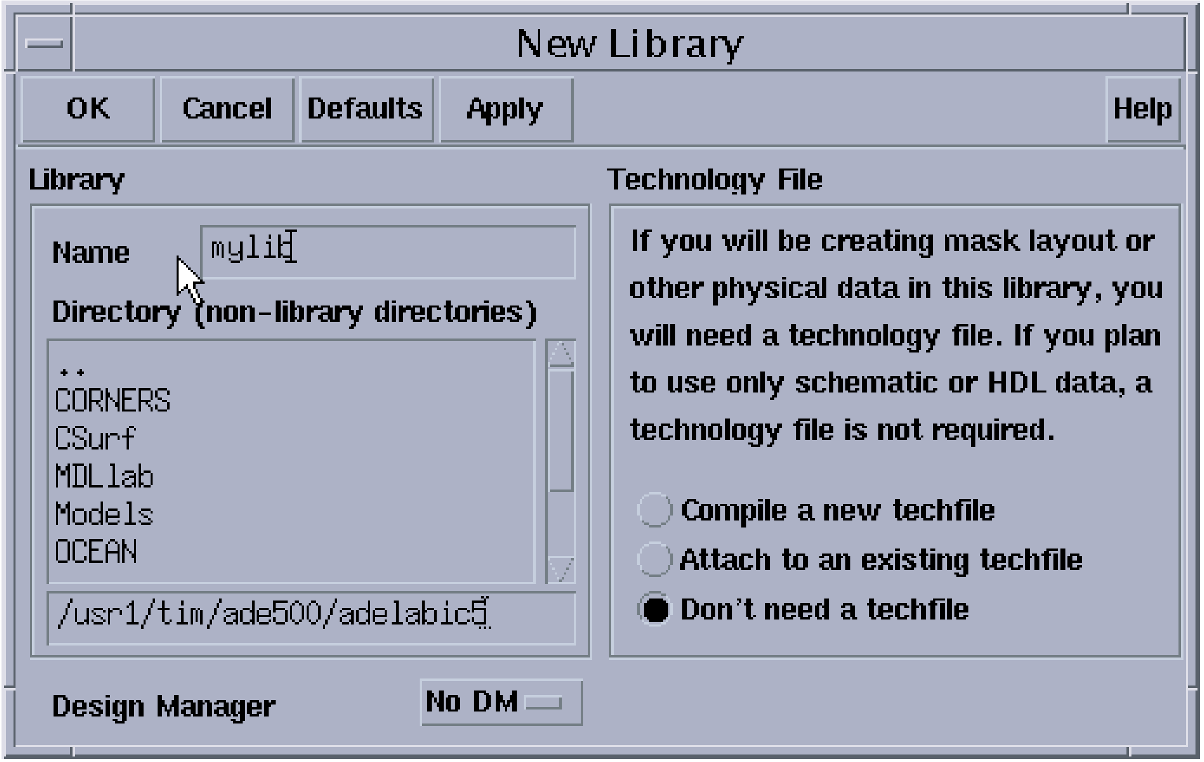
- Views

  A *view* is a particular representation of a cell such as a *layout*, *symbol*, or *schematic*. An application tool, such as *Composer-Schematic*, creates a view. Although a chip can include many levels of cell hierarchy, none of the hierarchical complexity is reflected in the libraries.

  A library is a flat collection of cells. Details of the design hierarchy exist inside the views that contain instances of other cells. The library treats all cells the same.

Creating a New Library

In the CIW or the Library Manager, select **File—New—Library**.



Library

Name: mylib

Directory (non-library directories)

```
..
CORNERS
CSurf
MDLlab
Models
OCEAN
```

/usr1/tim/ade500/adelabic5

Technology File

If you will be creating mask layout or other physical data in this library, you will need a technology file. If you plan to use only schematic or HDL data, a technology file is not required.

○ Compile a new techfile
○ Attach to an existing techfile
● Don't need a techfile

Design Manager: No DM

- Specify the library name and path.
- Specify the design manager to use.
- For Physical Design and Verification, specify the ASCII technology file or technology file library to be attached to the new library.

  The new library is entered into the *cds.lib* file.

Creating a New Library

When creating library, use a form to specify the library name and path, the design manager to use, and the technology file to attach to the library.

- Technology File Contents

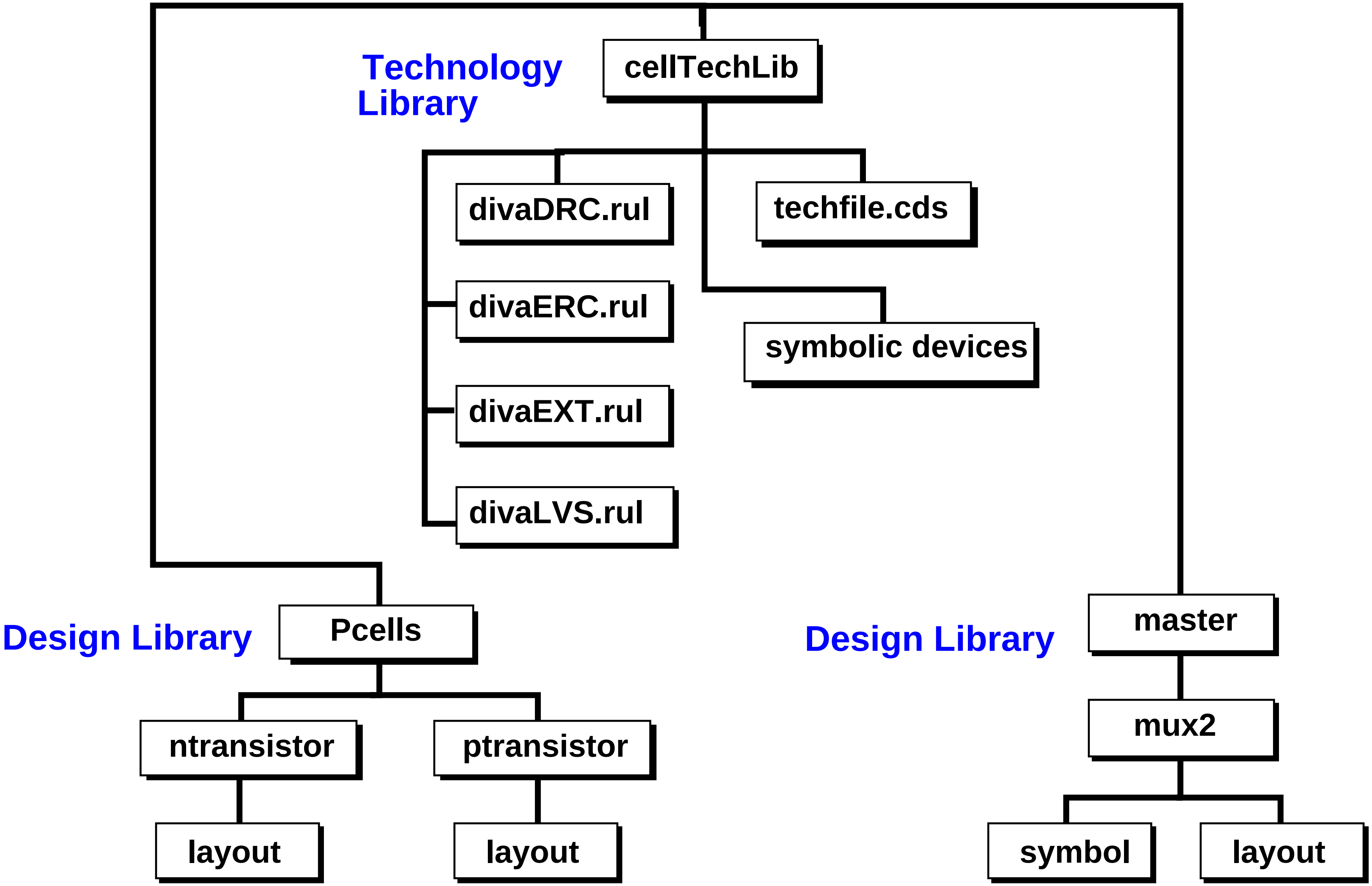
  The *technology file* is a large data file that specifies all of the technology-dependent parameters associated with that particular library. Design rules, symbolic device definitions, and parasitic values are some of the technology-specific parameters common to all cells in a library.

- *cds.lib* File

  The software automatically updates the *cds.lib* file when creating a library through the CIW's **File—New—Library** command, when copying one library to another name, or renaming a library. This file contains the paths to all of the libraries used in the design session, and can be accessed through CIW's **Tools—Library Path Editor** command.

Shared Technology Library

This example shows several libraries sharing the same technology file library.



Shared Technology Library

Share technology file information between different libraries. Create a technology file library and attach your design libraries to the technology file library. Use the **Technology File—Attach To** command in the CIW to attach the design library to the technology file library. Sharing a technology file library with other libraries share the same Diva® rules, layer information, and symbolic devices amongst a group of libraries. Sharing a technology file can help reduce the size of the design libraries, because the technology information is stored at only one location.

- *techfile.cds* file

  The *techfile.cds* file contains the binary technology file. This file name must be called *techfile.cds*.

- Diva Rules

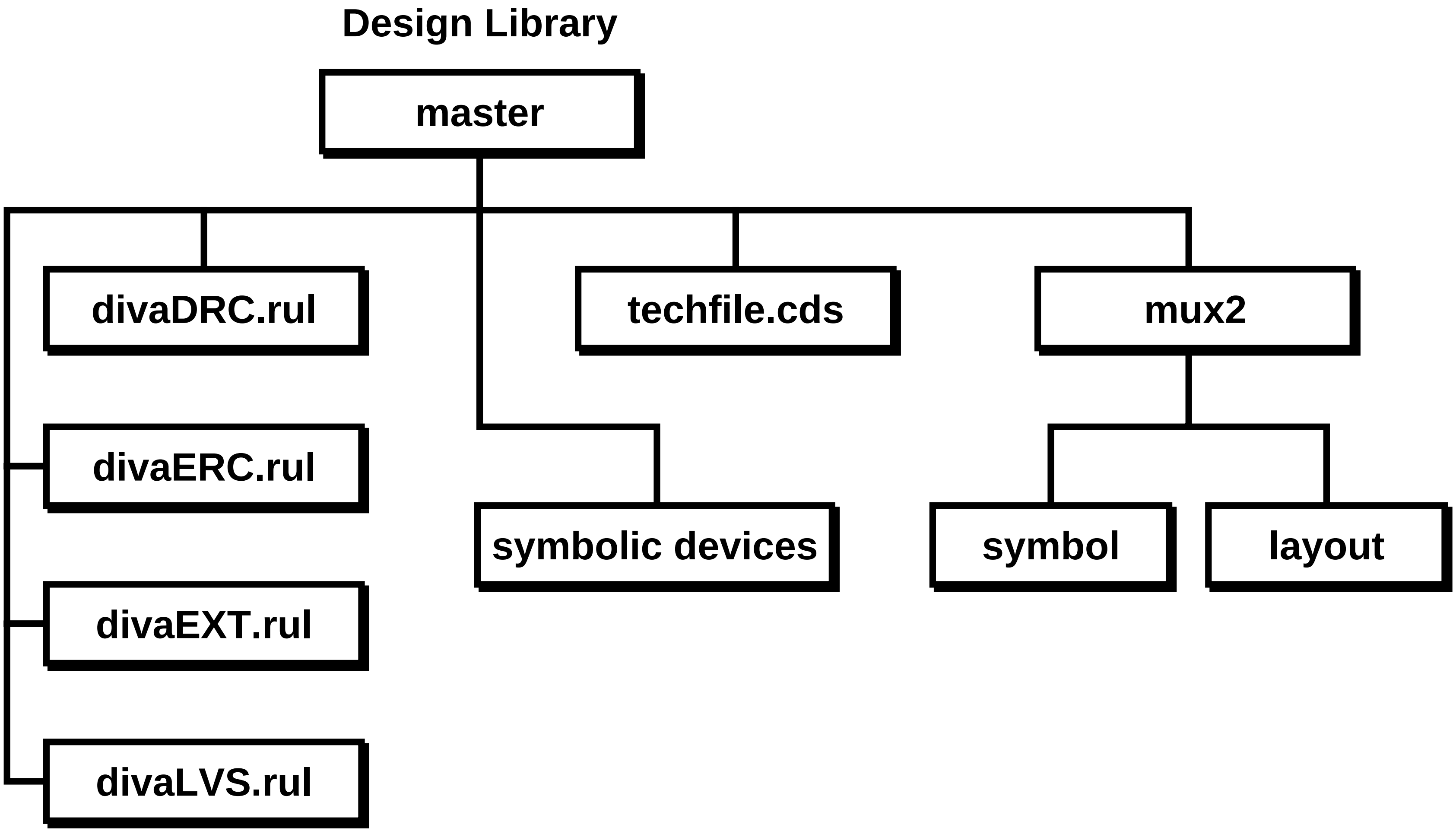
  The Diva rules are stored as separate ASCII files. For each type of rule (DRC, Extraction, ERC, and LVS), there is a Diva rules file.

- Symbolic Devices

  The symbolic devices such as contacts, pins, transistors, and wire information can be shared between libraries.

Technology File Stored in the Design Library

This example shows a technology file being stored inside a design library and not being shared with other libraries.



Technology File Stored in the Design Library

A library can have its own technology file information that is stored inside of the library.

Design Data Management

- Version control
- Configuration management
- Access control
- Release process

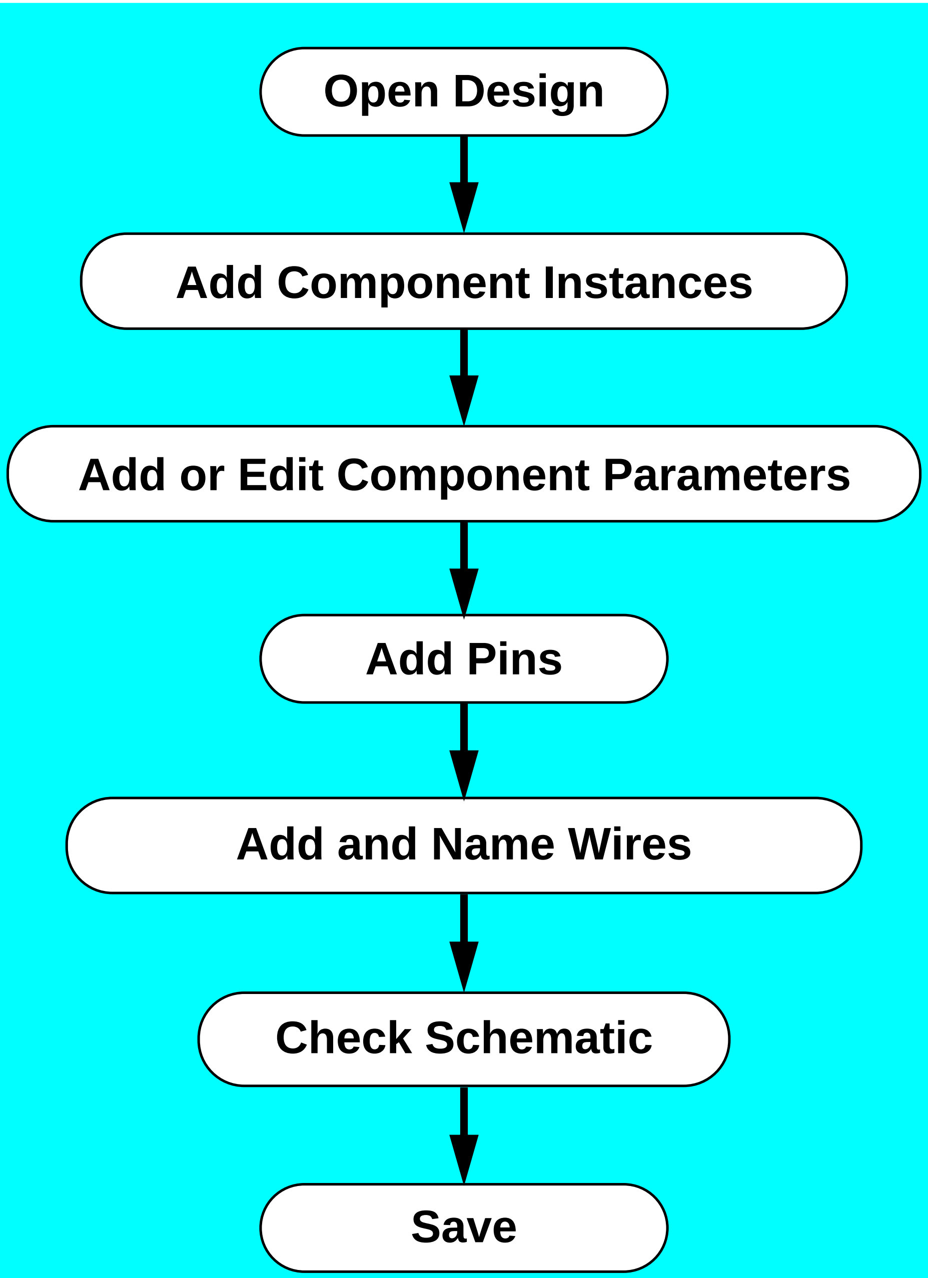
Design Data Management

Manage your design data using a suitable data management system. The following are some of the benefits of design management:

- Maintaining multiple versions of design data
- Releasing correct versions of designs
- Preventing accidental deletion of someone else’s design in a team environment
- Tracking or monitoring design status
- Isolating “what-if” design scenarios
- Effectively sharing stable design data
- Easily rolling back to a previous state of a design

The Cadence Team Design Manager (TDM) is our design data management system. Your system administrator or CAD support group must decide what data management system (if any) your projects will use.

Overview of Schematic Entry Flow



Open Design

Add Component Instances

Add or Edit Component Parameters

Add Pins

Add and Name Wires

Check Schematic

Save

Overview of Schematic Entry Flow

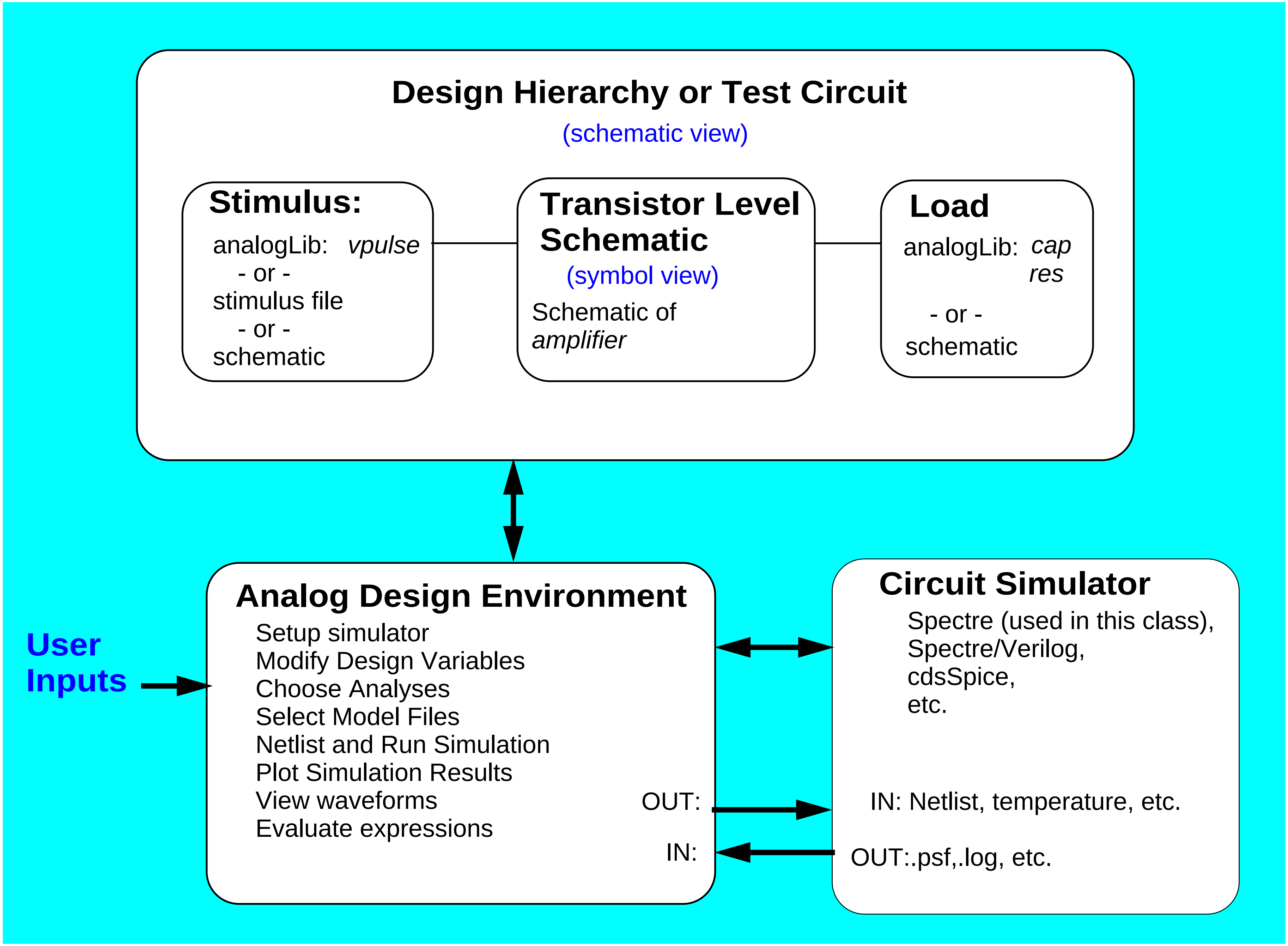
Perform the following steps when creating a schematic:

1. Open the design.
2. Add component instances by placing cellviews from libraries.
3. Add or modify component parameters.
4. Add pins to indicate connections outside of this schematic.
5. Connect the components and pins.

   Use wires to do this. This step also includes giving meaningful names to signals in the design.
6. Check the design to ensure that it is correct.
7. Save the design.

Note: These steps shall be discussed in detail in the next module.

Overview of Circuit Simulation



Design Hierarchy or Test Circuit

(schematic view)

Stimulus:

analogLib: *vpulse*
- or -
stimulus file
- or -
schematic

Transistor Level Schematic

(symbol view)

Schematic of *amplifier*

Load

analogLib: *cap*
res
- or -
schematic

Analog Design Environment

Setup simulator
Modify Design Variables
Choose Analyses
Select Model Files
Netlist and Run Simulation
Plot Simulation Results
View waveforms
Evaluate expressions

OUT:

IN:

User Inputs

Circuit Simulator

Spectre (used in this class),
Spectre/Verilog,
cdsSpice,
etc.

IN: Netlist, temperature, etc.

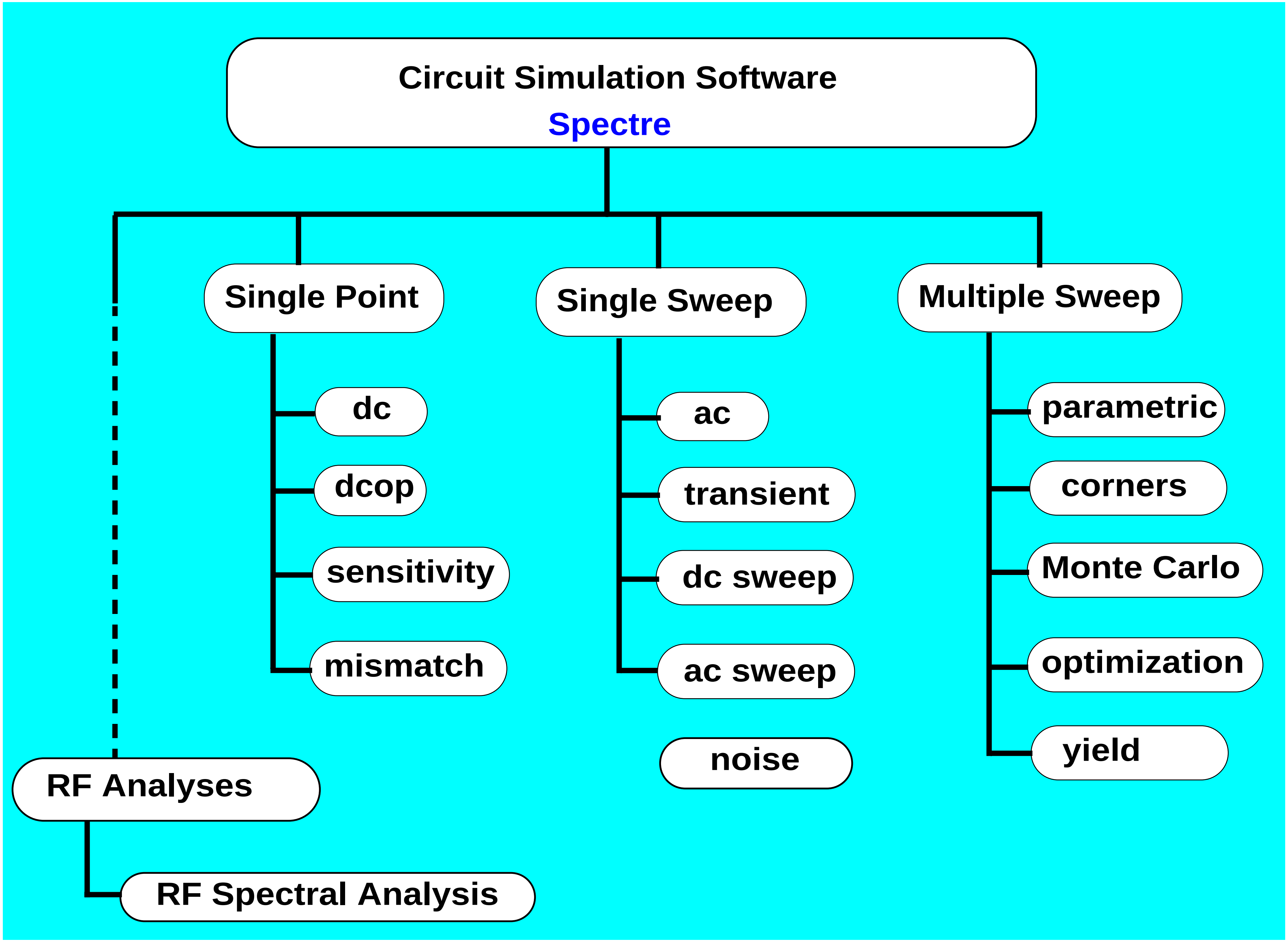
OUT:.psf,.log, etc.

Overview of Circuit Simulation

The block diagram above shows an overview of the circuit simulation process.

- The circuit schematic is captured or edited.
- A symbol of the schematic is placed in a hierarchy or test circuit schematic.
- The Analog Design Environment is started.
- The user provides input to the Analog Design Environment to set up and control what information is netlisted and then sent to the circuit simulator.
- The user used the Analog Design Environment to run the circuit simulator.
- The user selects the information to be printed, plotted, or to be analyzed.
- The user modifies the setup or edits the schematic for the next simulation.

Types of Circuit Simulation Analyses



Types of Circuit Simulation Analyses

The diagram shows the variety of analyses available with analog circuit simulation.

Single-point analyses often include the steady state **dc** solution of the circuit. The operating point **dcop** solves the operating point device parameters and low frequency gain of the circuit.

Single-sweep analyses often include **ac** and **transient** analysis. The **ac** analysis is a frequency sweep of the circuit. The **transient** analysis is a time sweep of the circuit operation to a time domain stimulus. A **dc sweep** analysis is a multiple point dc analysis performed while stepping a parameter such as temperature, design variable, or a model parameter. Solving the dc gain of an amplifier as a function of temperature is often called a **temperature sweep**. Solving the gain of an amplifier as a function of a model parameter is called a parametric sweep. It is also possible to sweep the **ac** gain of an amplifier at a specified frequency of the amplifier over temperature. This is called an **ac temperature sweep**.

Multiple sweep analyses refer to sweeping one variable and then stepping another variable between successive sweeps. In **parametric** analysis, the two or more variables are altered at specified intervals. In the **Corners** analysis, variables are specified at named corners. In the **Monte Carlo** analysis, the parameters are altered using random number generators and a specified distribution. In **optimization** analysis, the parameter are altered using the results of the previous simulation and a search algorithm.

The Analog Design Environment and the Spectre circuit simulator are capable of performing the analyses shown above. In addition, the Spectre circuit simulator can perform special steady state **ac spectral analysis** on RF waveforms. These analyses are discussed in the Spectre RF classes.

Summary

In this module we discussed:

- Course objectives
- Course outline
- Class schedule
- Getting Help, including CDSDoc
- Design Framework II environment
- Using forms
- Creating a library
- Creating cells and cell views
- Overview of schematic capture
- Overview of circuit simulation in the Analog Design Environment
- Types of simulation analyses

Summary

In this module we provided an introduction to the class, including:

- Class objectives
- Class schedule
- Online documentation

This module also provided discussion on the Design Framework II environment, including:

- Starting DFII with *icfb*, *icms*, or *msfb*.
- The Command Interpreter Window (CIW)
- Use of forms
- Front to Back design flow using Design Framework II
- Overview of schematic capture
- Overview of circuit simulation in the Analog Design Environment
- Types of simulation analyses

Labs

Lab 1-1 Getting Started

Lab 1-2 Top-Down System Modeling

Labs

Module 2: Schematic Entry

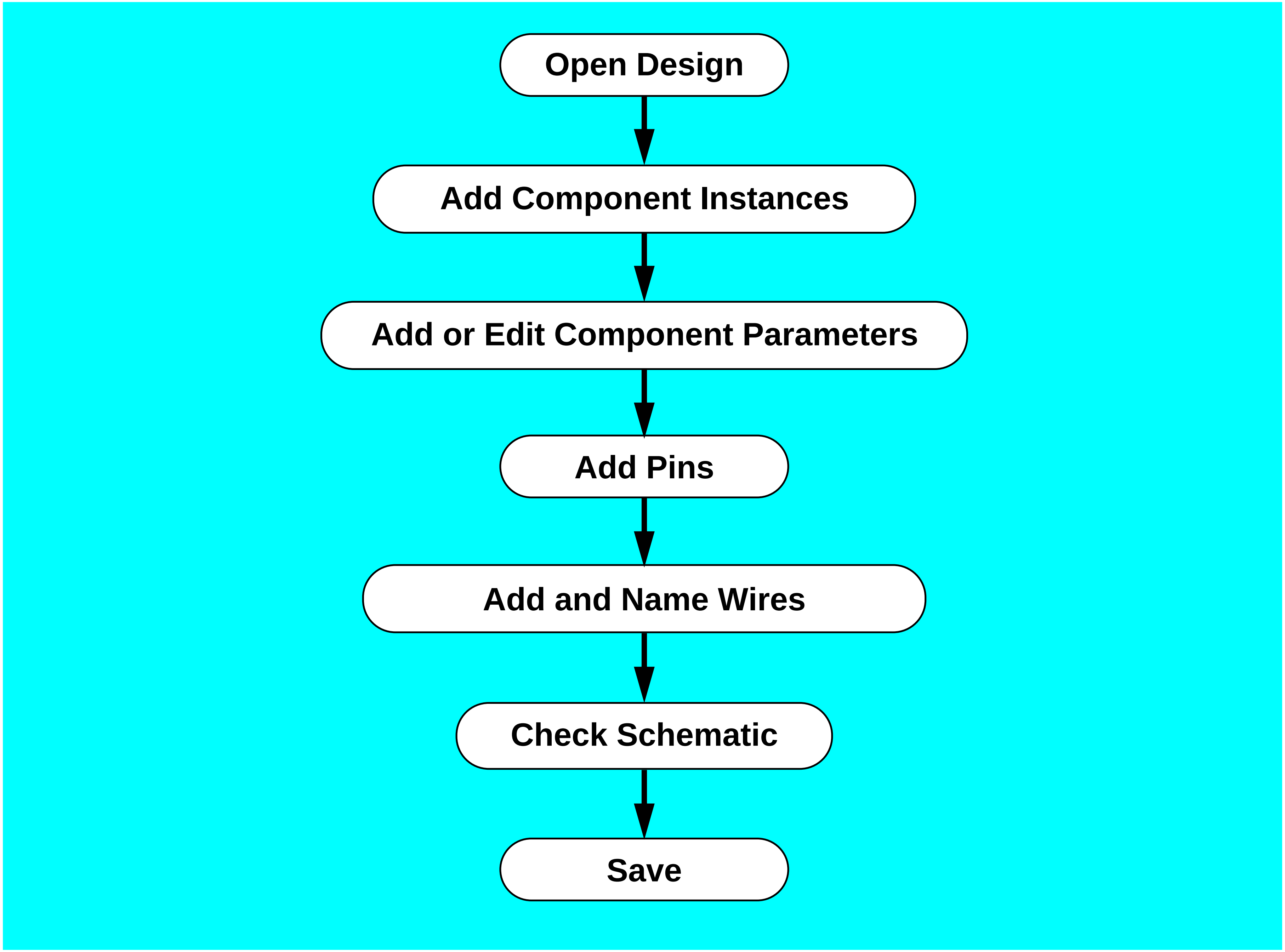
Topics in this module

- The schematic capture flow
- Creating a schematic view
- Contents of a schematic
- Adding component instances
- Adding pins
- Adding wires
- Editing object properties
- Using Accelerator keys (also known as bindkeys) and schematic window icons
- Checking the schematic for errors
- Symbol generation and editing
- Using a design hierarchy

Terms and Definitions

library	A set of design directories that includes ‘cells’ and ‘cellviews’.
Library Manager	A Cadence tool that allows user to browse and edit a design library.
Composer	Schematic editor and symbol generation tool in DFII.
cell	A basic unit of a design hierarchy described by cell views.
cell view	A specific view of a cell that includes schematic, symbol or layout.
instance	A uniquely named placement of a symbol onto a schematic.
pin	A connection point on a schematic and symbol used for accessing signals.
bindkey	A predefined key on the keyboard that invokes a preselected command.

Schematic Entry Flow



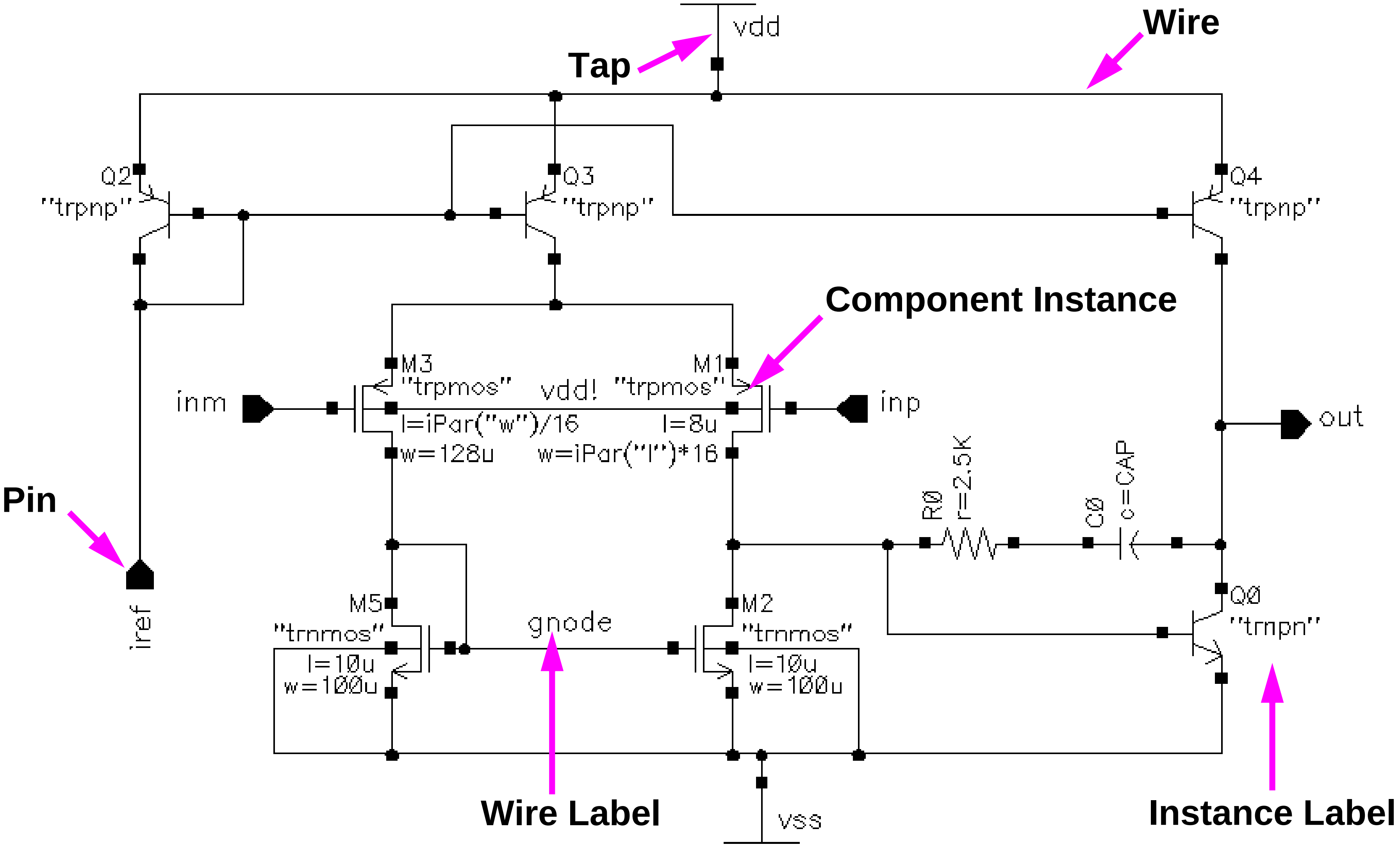
Schematic Entry Flow

Perform the following steps when creating a schematic:

1. Open the design. Use the CIW or Library Manager tool.
2. Add component instances by placing cellviews from libraries.
3. Add or modify component parameters.
4. Add pins to indicate connections outside of this schematic.
5. Connect the components and pins.

   Use wires to do this. This step also includes giving meaningful names to signals in the design.
6. Check the design to ensure that it is correct.
7. Save the design.

Contents of a Schematic



Tap

Wire

Component Instance

Pin

Wire Label

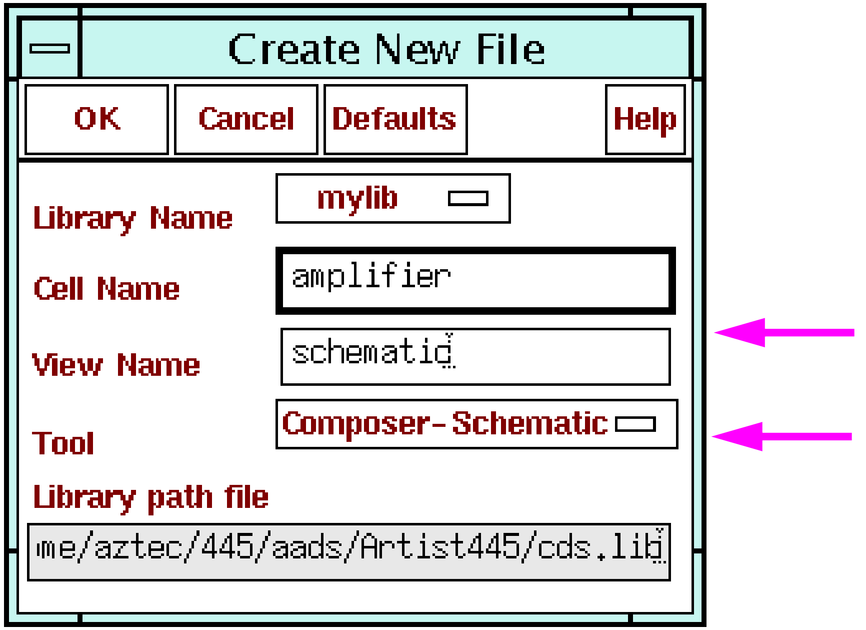
Instance Label

Contents of a Schematic

- Component instances represent instantiations of other cellviews in this cellview.
- Instance labels display component information in the design entry window.
- Pins can be inputs and outputs of a schematic or connection point when an instance is placed in another cellview.
- Wires can be drawn between pins to connect them.
- Wire labels provide meaningful signal identifiers for simulation results analysis.
- Analog taps and sources can be included directly in the design.

Creating a New Cellview

In the CIW or Library Manager, select **File—New—Cellview**.



Create New File

Library Name: mylib

Cell Name: amplifier

View Name: schematic

Tool: Composer-Schematic

Library path file: me/aztec/445/aads/Artist445/cds.lib

Default View Name subject to override.

Select: Composer-Schematic

- Specify the Library Name, Cell Name, View Name, and Tool to use. The path to the *cds.lib* file will appear in the form and is not editable.
- Modify the Tool field to create a layout, verilog, symbol, schematic, vhdl, or ahdl view.

For an ADE schematic, select **Composer-Schematic** from the Tool cyclic field

Creating a New Cellview

Create a new cell views from the Library Manager or CIW.

Specify the Library Name, Cell Name, View Name, and Tool to use. The path to the *cds.lib* file will appear in the form and is not editable.

Modify the Tool field to create a *layout*, *verilog*, *symbol*, *schematic*, *vhdl*, or *ahdl* view. For a schematic, use *Composer-Schematic*. This will automatically enter *schematic* into the View Name text field.

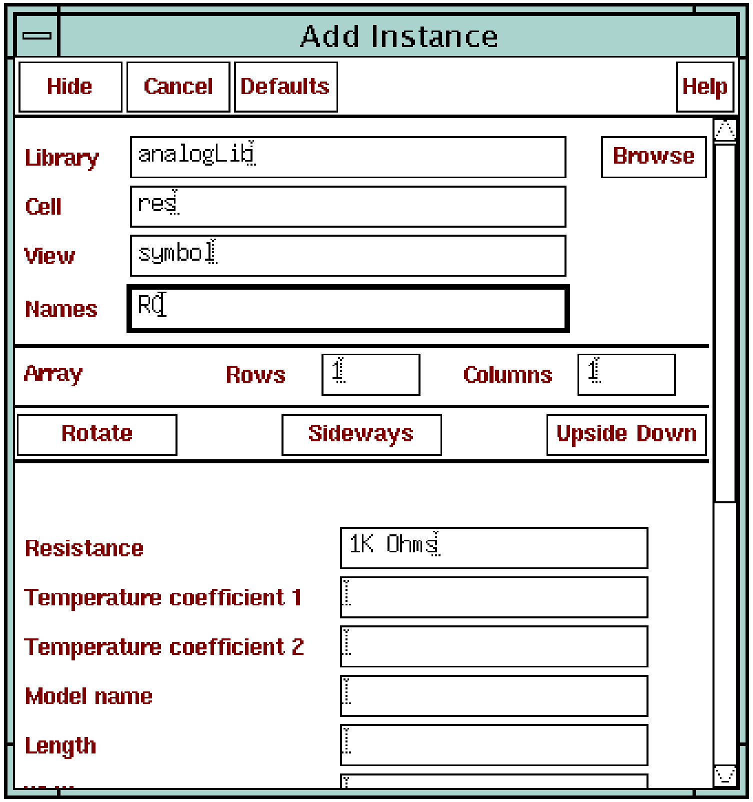
Important

Although *schematic* is automatically entered into the View Name text field by default, you have the option to name the view anything you want. For example, the View Name may be altered to *schem1*, and the process repeated for *schem2*. As such, there are two or more schematic views for the same cell name. This allows parallel circuit designs within the same cell name. A common symbol for both schematics is then used within the hierarchy, until the final schematic is selected. This feature is useful in exploratory designs where the final circuit topology has not yet been finalized.

Adding Component Instances

Select **Add—Instance** or press the **i** key to display the Add Instance form.

- Attach multipliers to values. Enter **1k** (not 1 k) so that *k* is not mistaken as a variable.
- Parameter units, such as *ohms*, are implicit.



Library	analogLib
Cell	res
View	symbol
Names	R0
Array Rows	1
Columns	1
Resistance	1K Ohms
Temperature coefficient 1	
Temperature coefficient 2	
Model name	
Length	

Use these buttons while placing components to control orientation.

Adding Component Instances

Design components are generally instances of a *symbol* cellview and might be design primitives. Here are some properties associated with design component instances:

Parameter	Example Value
Library Name	analogLib
Cell Name	res
View Name	symbol
Instance Name	R2

The Instance Name is assigned automatically, unless explicitly specified.

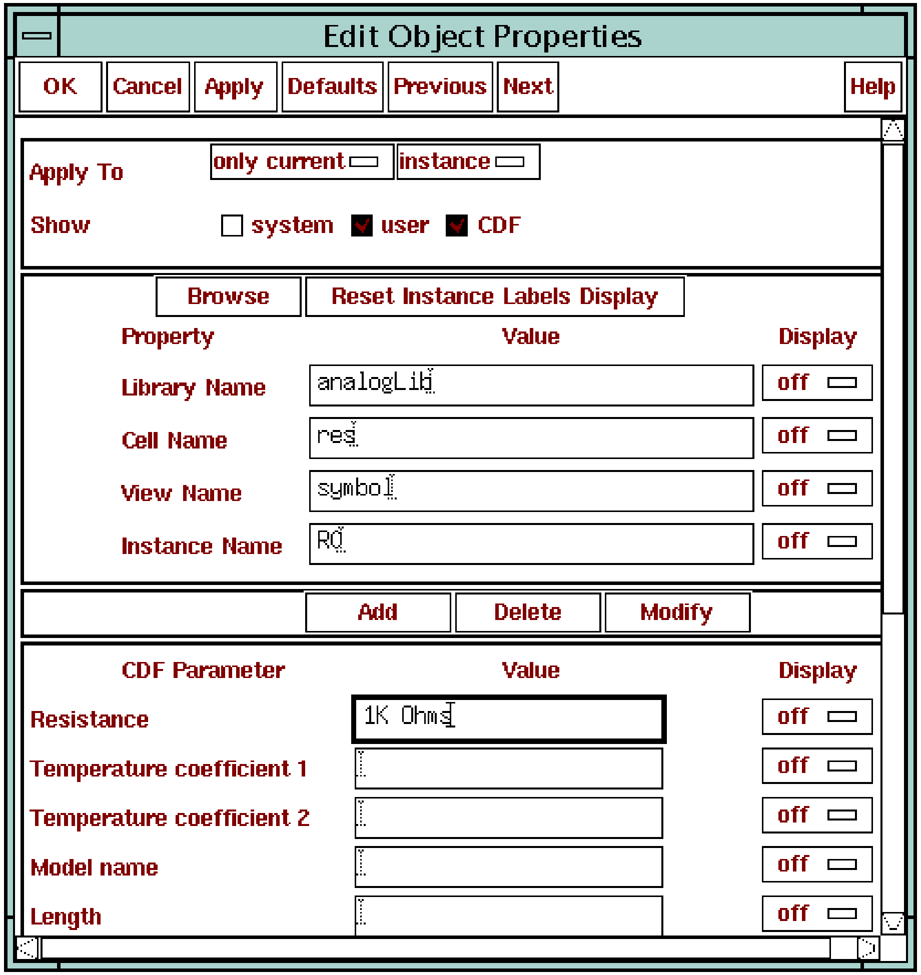
Find analog design primitives in the *analogLib* library. This library is included wherever the Analog Design Environment software is installed in the path *<install_dir>/tools/dfII/etc/cdslib/artist*. Include this path in your library search path to use *analogLib* components.

The system prompts for component parameters when instantiating the components. Attach multiplier suffixes, such as **k** for 1000, to numerical quantities.

Use the **Rotate**, **Upsidedown**, and **Sideways** buttons to change the orientation of your components as they are placed in the schematic.

Updating Design Objects

- Select **Edit—Properties—Objects** or bindkey **q** to start the form. The **Next** and **Previous** buttons highlight single objects in a selected set.
- Use **Design—Renumber Instances** to renumber instances in a design.

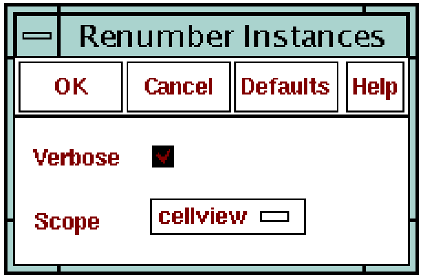


Apply To: only current, instance

Show: □ system ☑ user ☑ CDF

Property	Value	Display
Library Name	analogLib	off
Cell Name	res	off
View Name	symbol	off
Instance Name	R0	off

CDF Parameter	Value	Display
Resistance	1K Ohms	off
Temperature coefficient 1		off
Temperature coefficient 2		off
Model name		off
Length		off



Verbose ☑

Scope: cellview

Design—Renumber Instances

Updating Design Objects

You can either update single or multiple objects in a selected set. Use the *Next* and *Previous* buttons on the Edit Object Properties form to scroll through a set of selected objects and update them. Only one object at a time will highlight in the schematic window. It is possible to modify most quantities that appear on the form.

The most common changes concern components parameters, pin name, and pin direction.

In addition, use the Stretch, Copy, Move, Delete, and Rotate commands to update your design. These commands are located under the **Edit** menu in the schematic window. To display an options form that is associated with any of these commands, use the **Cmd Options** icon or press the **F3** key while these commands are active.

Use the Renumber Instances form to renumber instances. This form renumbers component names in sequential order to make it easier to track component totals. In addition, adding and deleting components in a schematic during the design process can leave components labeled improperly.

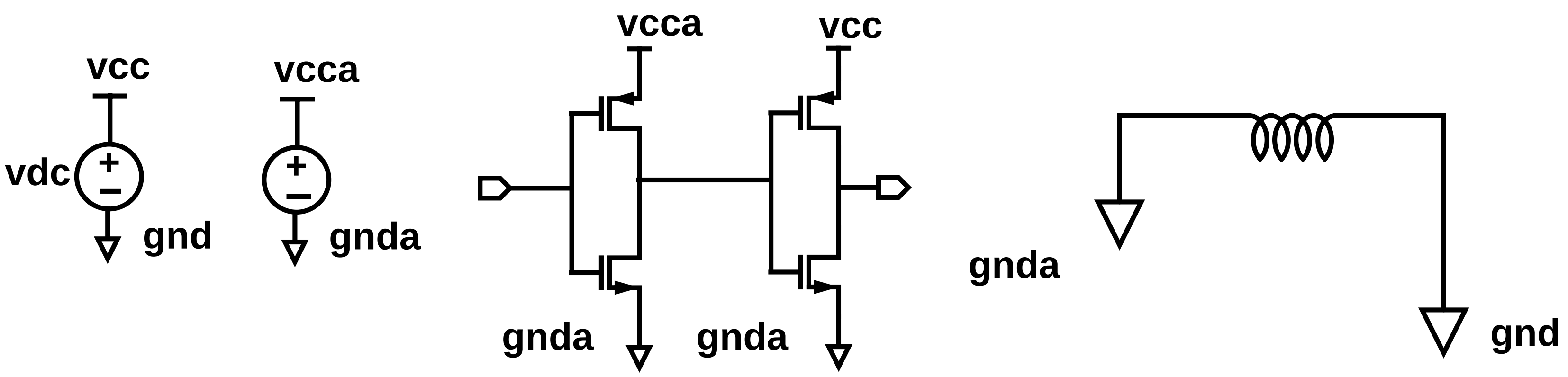
Note: The renumber sequence depends on the order that the symbols were added to the schematic.

Adding Sources and Ground

Sources, taps, and grounds are instances of cells.

Sample source cells are in the *analogLib* library.

- Choose from independent, dependent, and piece-wise linear (PWL) sources.
- Choose tap and ground cells, which are used to establish global nets.
- An instance of the cell *gnd* is required in the design for DC convergence.



Adding Sources and Ground

- Ground

  Always include the symbol *gnd* (found in the *analogLib* library). Analog simulators require that all nodes in the circuit must have a DC path ground. This would be represented as *node 0* in the Cadence SPICE circuit simulator, for example. Use other ground symbols, such as *gnda*, for a ground that is connected to the reference ground through an analog circuit.

- Voltage Sources

  Include all of your DC and transient voltage and current sources in the schematic. There are many types of voltage sources in *analogLib*. For example, some of the independent voltage sources are *vdc*, *vsin*, *vpulse*, *vexp*, *vpwl*, and *vpwlf*. Each source has a current equivalent that begins with the letter ***i***. There are also equivalent dependent sources.

  All sources generate input waveforms except for *pwlf* sources, which stimulate a circuit using a text file of data tables. It is not necessary to include sources in the schematic, although this is often convenient. Attaching a stimulus file to the final netlist is discussed in the analog simulation section of this course.

- Voltage Taps

  Use tap symbols to transfer voltages and currents throughout the design without using wires. Voltage tap symbols, such as *vcc*, *vdd*, *vcca*, and *vccd*, are in the *analogLib* library.

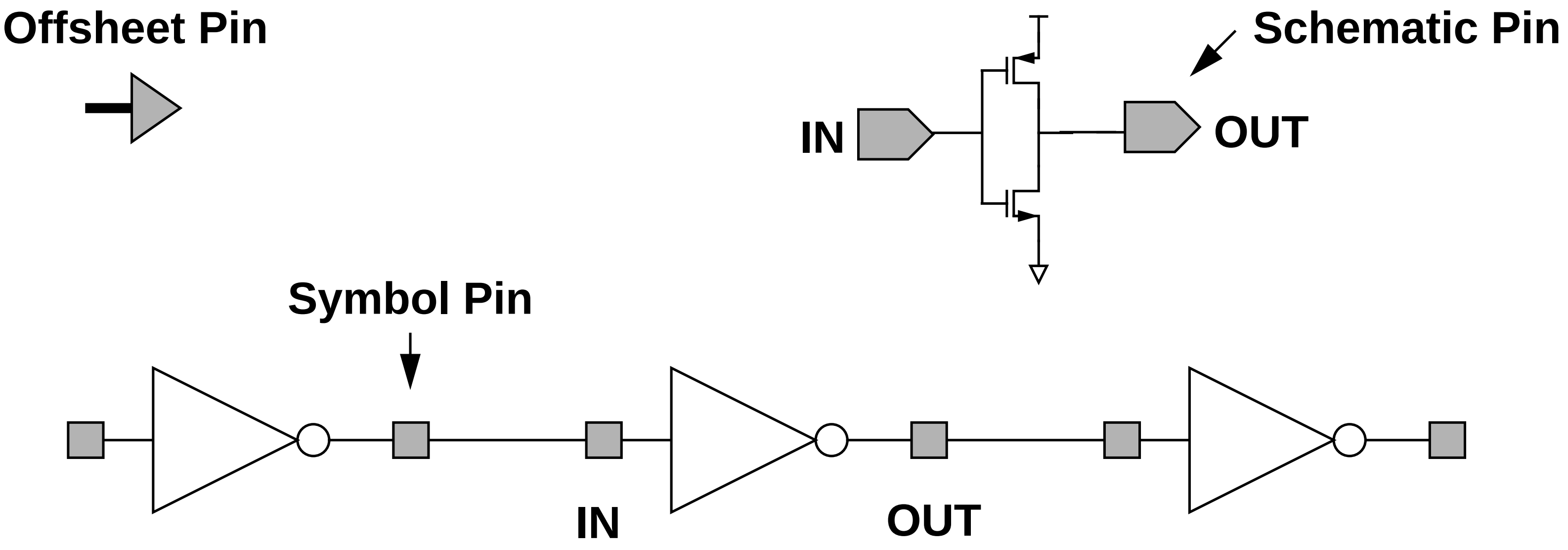
Pins

Pins have a user-defined *Name* and a *Direction* (input, output, or input/Output).

Pins are one of three *types*:

- *Schematic* pins provide ports to a schematic.
- *Symbol* pins provide ports to a symbol representing a schematic, and are connection points to the symbol in a hierarchical design.
- *Offsheet* pins are used in large designs without hierarchy.

Pin names and directions must match in all cellviews of a cell.



Offsheet Pin

Schematic Pin

Symbol Pin

Pins

For analog designers, pins have two primary functions:

- Pins represent connection points between different cell views such as schematic, symbol, and layout representations. Using named pins identifies equivalent input, output, and I/O ports throughout the design environment.
- Pins provide connection points for objects that are hierarchically instantiated.

Pin Properties

Pins have a pin name, pin type, and pin direction. These should be consistent throughout your design.

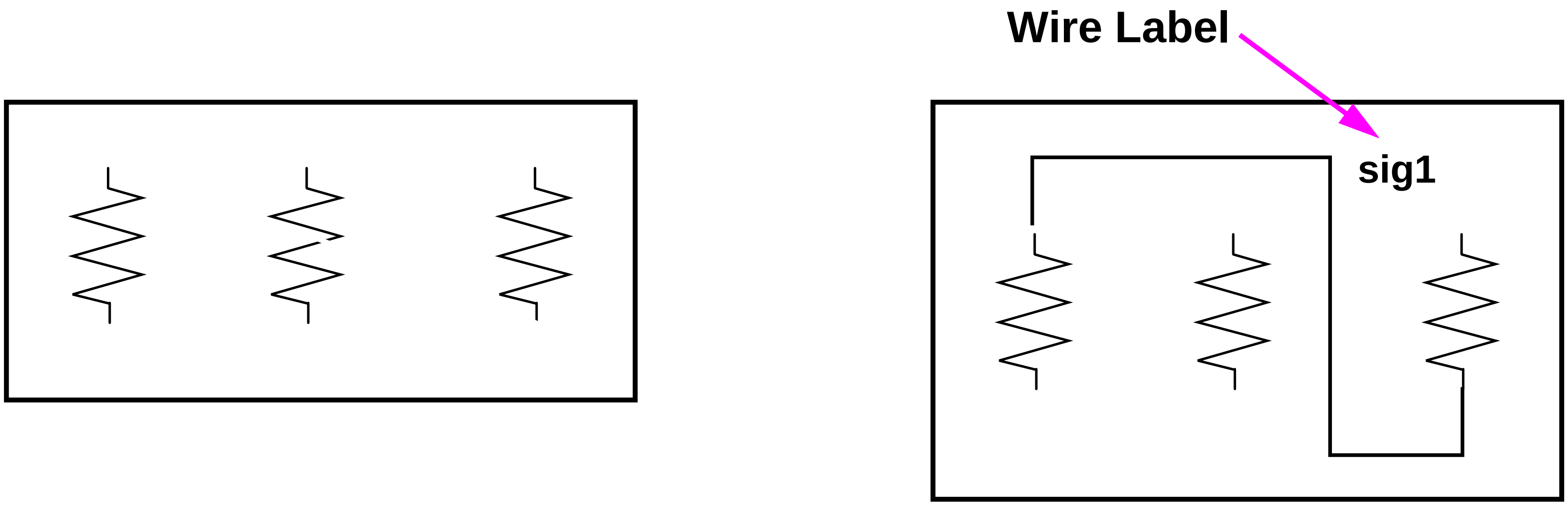
Multiple Sheet Design with Offsheet Pins

The *Virtuoso Schematic Composer User Guide* manual includes a section on multiple sheet design methodology and information on the *offsheet* pin type. You can get other help for the Composer software in the Cadence online help.

Pins (*ipin*, *opin*, *iopin*, *sympin*) now come from “basic” library.

Wires and Wire Labels

Automatic routing is the default mode.



Route Entered The System Routes

When not labeling a wire, the system names the net formed by the wires.

If the router cannot find a path between two points,

- A dotted “flight line” is placed to establish connectivity only.
- Click on intermediate points to guide the router to yield a solid line of connectivity.
- Use the **Cmd Options** icon or **F3** key to modify the wiring options.

Wires and Wire Labels

Draw wires between the instance pins and schematic pins to connect them. Use wide wires to indicate multiple signals on a wire, the system does not force or check this. Draw wires at any angle, but most designers frequently restrain wires to orthogonal lines.

- Using Route Methodology

  The route draw mode chooses two points in your design and then it automatically routes a wire around components. If a routed net remains dotted, it is because there was no clear routing channel. This can happen if the instances are too close or overlap the selection boxes. To solve this, move the components further apart to give a routing channel.

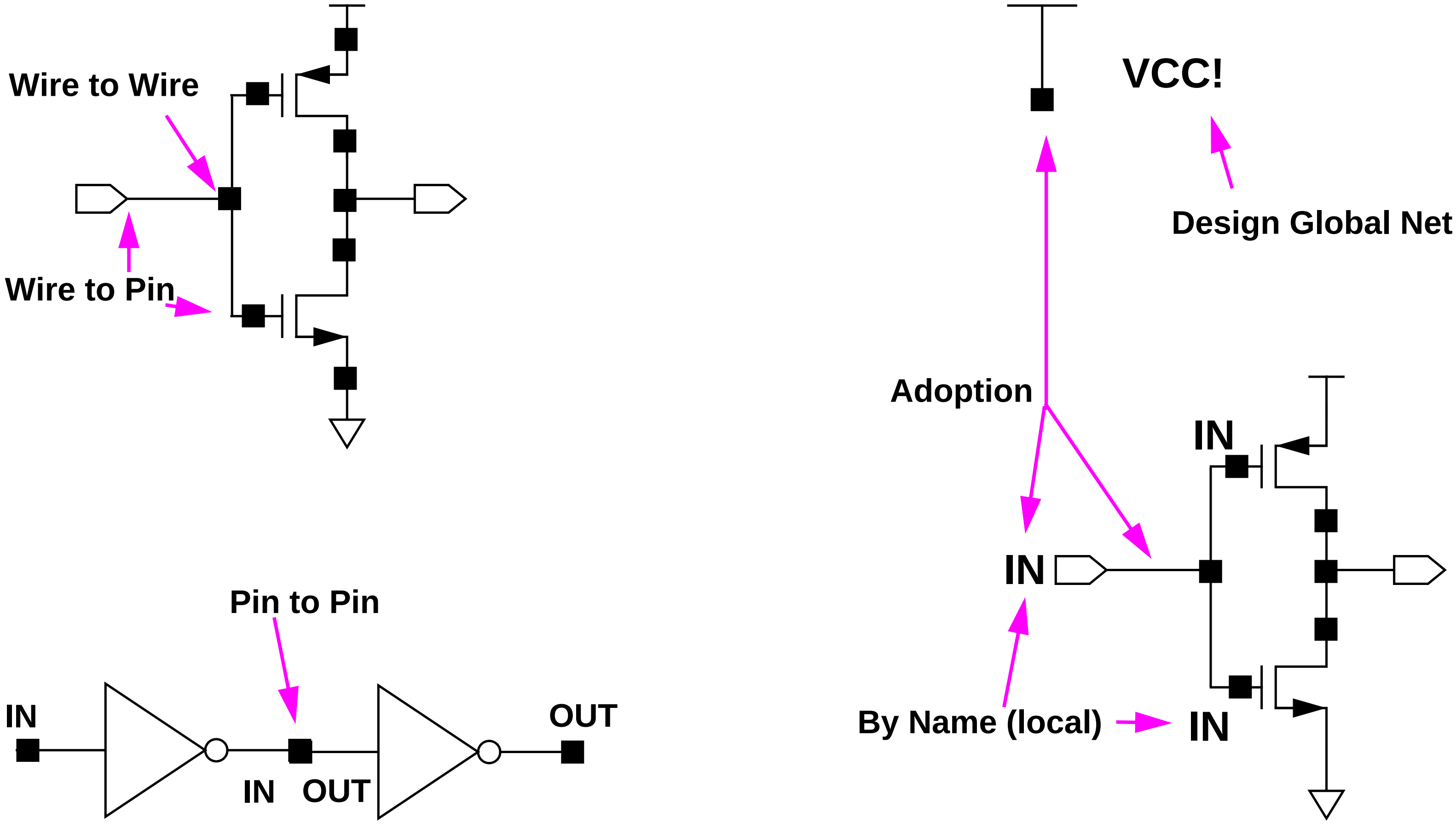
  Route method options exist to wire together two points immediately (the default) or indicate many points to route together later in a single step. More information on route methods is included in the design entry reference documentation.

- Wire Labels

  Labeling wires gives the corresponding net a meaningful name in the simulation results data. Otherwise nets are system named. There is some control over the automatically generated names, but these may not be as meaningful as custom names.

  Click the **Cmd Options** icon in the schematic window or press the **F3** key to change the default wiring setup.

Interconnecting Components



Avoid this when possible.

Schematic pins and global symbol pins name wires by adoption.

Note: Inherited connections, not shown, are discussed in the *Advanced Topics in ADE* module.

Interconnecting Components

- Physical Connectivity

  All physical connections are made by wire-to-pin, wire-to-wire, or pin-to-pin connections.

- Connectivity by Name

  If two wires have been labeled with the same name, they become part of the same net when connectivity is established.

- System-Assigned Names

  If a net is unnamed, the system generates a name such as *net100* or *net7*. Optionally change the base name from *net* to something else. If a wire is connected to a schematic pin, then the pin name is used to name the net by adoption when connectivity is established.

- Global Nets

  Any net or pin name that ends in an exclamation point will be part of a global net when connectivity is established. Global nets are automatically connected through the hierarchy without the use of wires. For example, voltage taps have symbol pin names that end in an exclamation point. If a wire is connected to a pin that has a global name, the pin name is used to name the net by adoption. This is how voltage and ground signals are propagated throughout a design.

Note: A net named *net!* is not connected to a net named *net*.

Schematic Checking

During schematic checking, all of the following are performed by default:

- Update Connectivity

  This process associates wires and pins with logical connections called nets.

- Schematic Rules Check
  - Logical checks
  - Physical checks
  - Name checks

- Cross-View Checker

  This option checks for pin name and direction consistency between cellviews.

Select **Check — Rules Setup** from a schematic window to edit the rules. Disable any or all of these schematic checking features, if not needed.

Schematic Checking

Schematic checking is a critical step in the design process.

Either check a single cellview or descend through the hierarchy to check all cellviews in your design.

Checking a schematic accomplishes the following:

- Update Connectivity—When connectivity is established, wires and pins in the design entry window become associated with logical connections called nets. It is necessary to correct connectivity problems prior to going on to the next design phase.
- Schematic Rules Check—This process checks the schematic with a set of rules. Access them with the **Check—Rules Setup** command from the schematic window.

  The checks include:

  — Logical checks, such as *Floating Input Pins* and *Shorted Output Pins*.

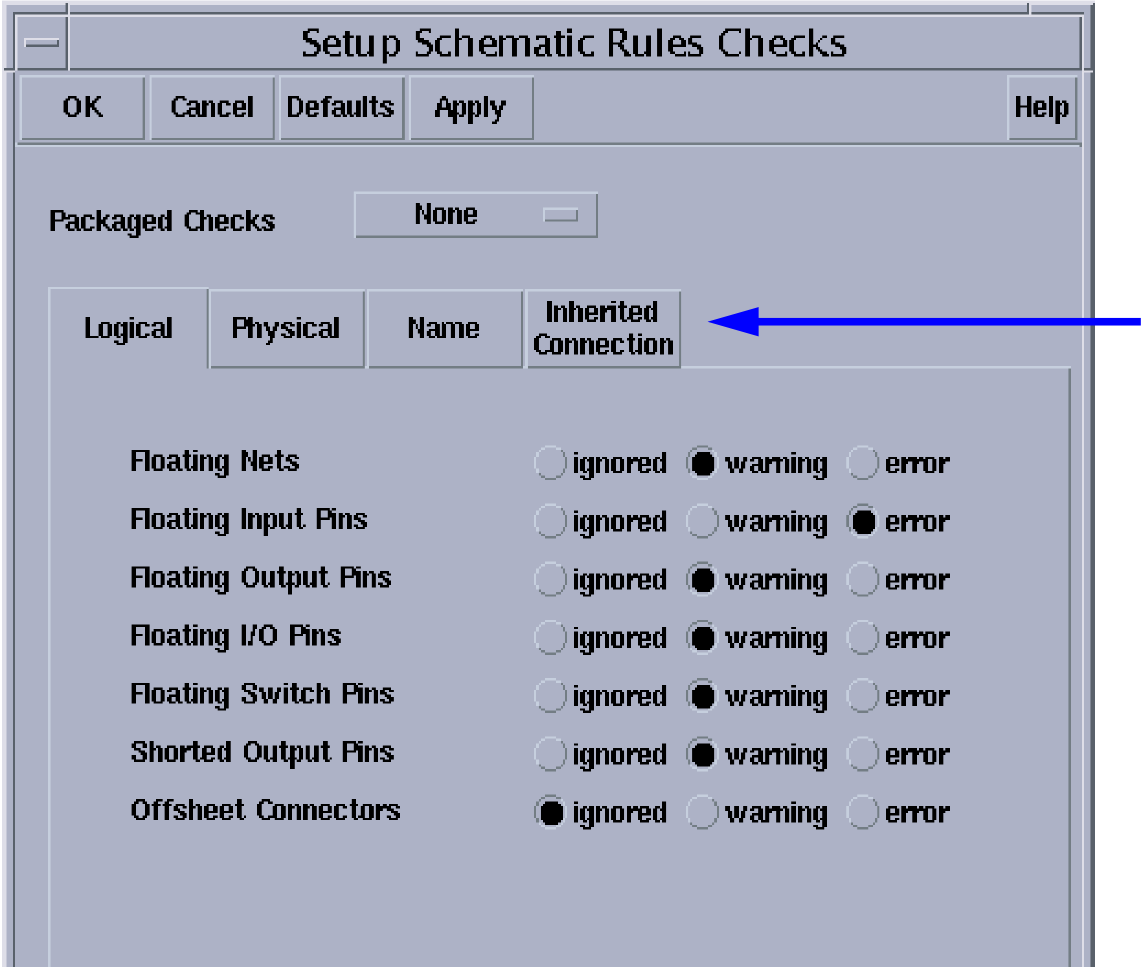
  — Physical checks, such as *Unconnected Wires* and *Overlapping Instances.*

  — Name checks, such as *Instance Name Syntax.*
- Cross-View Checker—This option checks the pin consistency between different views of the cell. Pin names and directions must match between cellviews.

Schematic Checking Rules

The system sets the default schematic checking rules.

The following set and selections are logical types of rules:



Packaged Checks: None

Floating Nets	○ ignored	● warning	○ error
Floating Input Pins	○ ignored	○ warning	● error
Floating Output Pins	○ ignored	● warning	○ error
Floating I/O Pins	○ ignored	● warning	○ error
Floating Switch Pins	○ ignored	● warning	○ error
Shorted Output Pins	○ ignored	● warning	○ error
Offsheet Connectors	● ignored	○ warning	○ error

Note:
There are four sets of Rules Checks as indicated by the tabs.
- Logical
- Physical
- Name
- Inherited Connections

Note: *Ignored* means do not check for a condition. It is permitted to generate a netlist and run a simulation with *warnings*, but not with *errors*.

Schematic Checking Rules

The Schematic Rule Checker (SRC) performs schematic syntax checks. Select and override the default values of schematic rule checks. The defaults are acceptable for most applications. Select and set the severity level for SRCs. There are three levels for each check:

ignored	SRC does not perform the check.
warning	Warnings do not need to be corrected before continuing. SRC treats schematic connectivity as valid. The netlist program can still read the schematic.
error	Problems must be corrected before continuing. SRC treats schematic connectivity as invalid. The netlist program refuses to read the schematic.

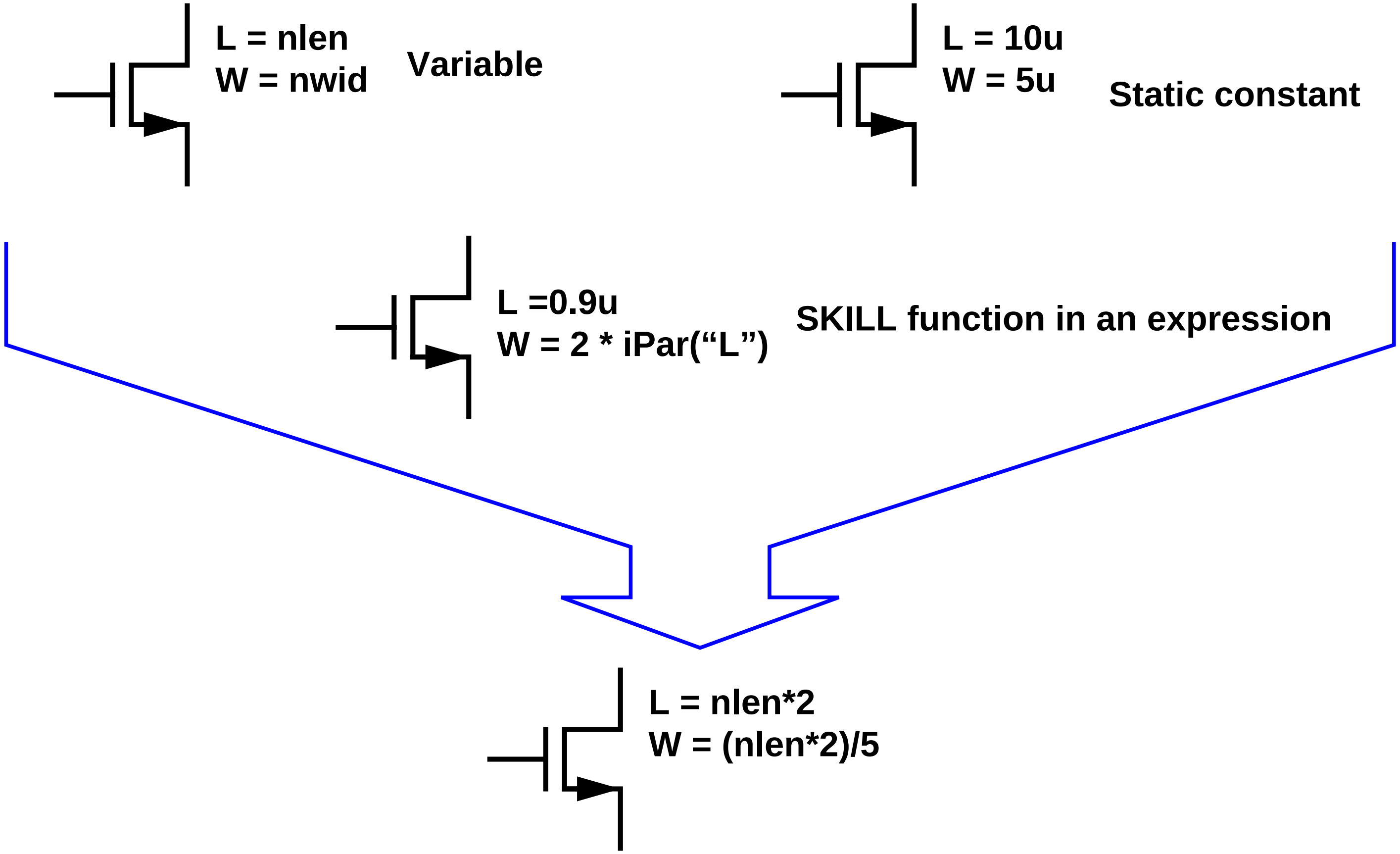
The checks are done for three different areas:

Logical checks	These checks consist of component connections that could affect the functionality of the circuit.
Physical checks	These checks consist of problems dealing with overlapping components, unconnected wires, and solder dots.
Name checks	These checks consist of problems associated with name syntax, behavioral model syntax, etc.

A description of the rules is in the *Virtuoso Schematic Composer User Guide*.

Component Parameter Types

Use user-defined functions to describe parameters.



L = nlen
W = nwid

Variable

L = 10u
W = 5u

Static constant

L =0.9u
W = 2 * iPar(“L”)

SKILL function in an expression

L = nlen*2
W = (nlen*2)/5

Any mathematical expression using the above

Note: iPar is only used on the same sheet of the schematic, and not for hierarchy.

Component Parameter Types

Some basic types are static constants, global variables, and dependent variables. Use these in combination with a mathematical expression to create parameters.

To change a value during simulation, assign a variable name. Before running simulation, set design variables. All quantities assigned the same variable name will get the same value.

Hierarchical variables, user-defined functions, and user-defined constants are discussed later.

SKILL Functions

Use built-in SKILL functions that return design parameter values and use them in expressions to set component parameters. For example, to make the width of a MOS device a function of its length. In this example, if the length of a component is defined as *L*, then the width can be set as

```
w= 2 * iPar(“L”)
```

to make the width of the device twice the length.

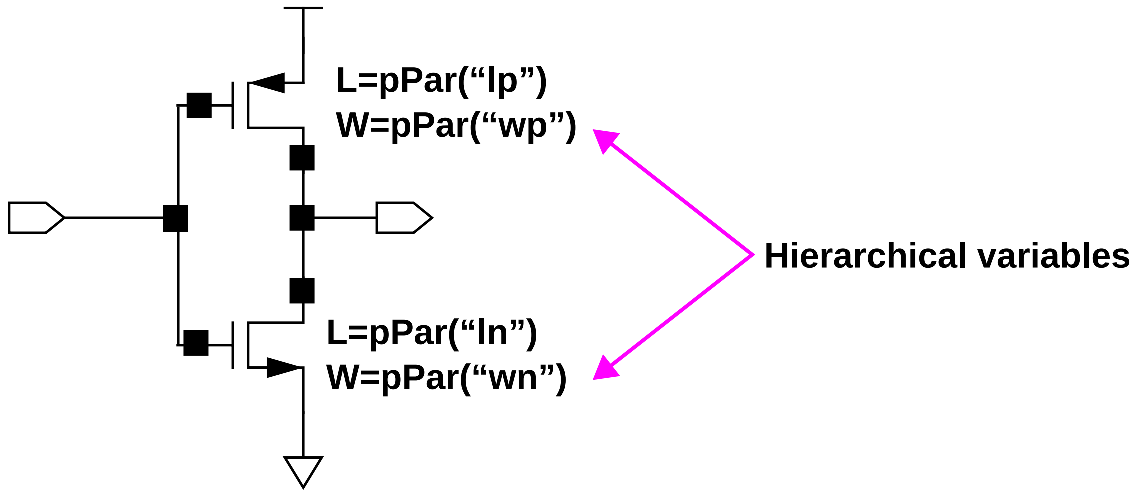
In general, the SKILL function

```
iPar(“parameter name”)
```

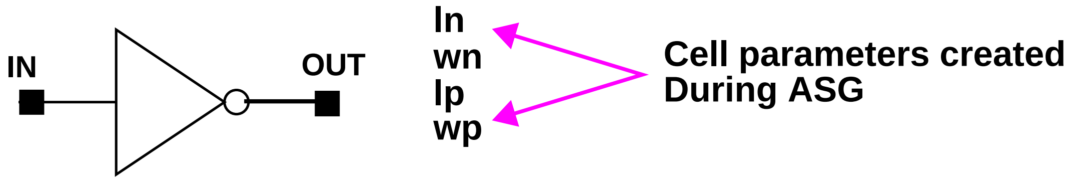
returns the value of a component parameter of the local cell.

More information on *pPar* and *iPar* is found in CDSDoc.

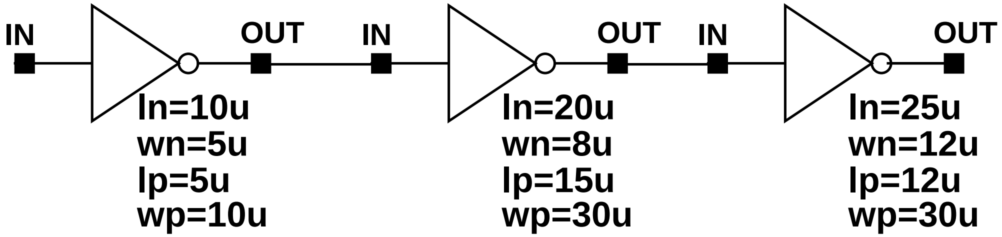
Passing Parameters Through the Hierarchy



Hierarchical variables



Cell parameters created
During ASG



During Automatic Symbol Generation (ASG), hierarchical variables are scanned. The system creates component parameters for the symbol from these variables.

Parameters become editable when the instances are selected.

Passing Parameters Through the Hierarchy

You can place the same symbol many times and alter its schematic component parameter values at the instance level.

To accomplish this, assign expressions to schematic components using the following syntax:

```
pPar(“variable”)
```

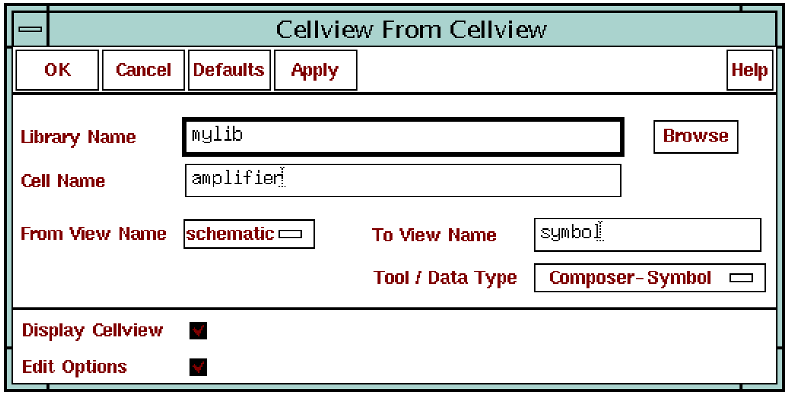
The value of the *variable* will be passed from the symbol level in the hierarchy to a component parameter in the schematic. Set the value of the *variable* when placing the symbol that was automatically generated from a schematic with hierarchical variables. During Automatic Symbol Generation (ASG), the system analyzes the hierarchical variables to determine what component parameters to prompt for when placing a symbol in another cellview.

Automatic Symbol Generation (ASG)

Automatic symbol generation assists in the creation of symbols. The quickest way to automatically create a symbol is from another cellview. Creating a symbol from an existing cellview also ensures that the pin properties will match between the cellviews. Other advantages to ASG are the automatic creation of symbol parameters from hierarchical variables in the schematic and the creation of CDF for the cell.

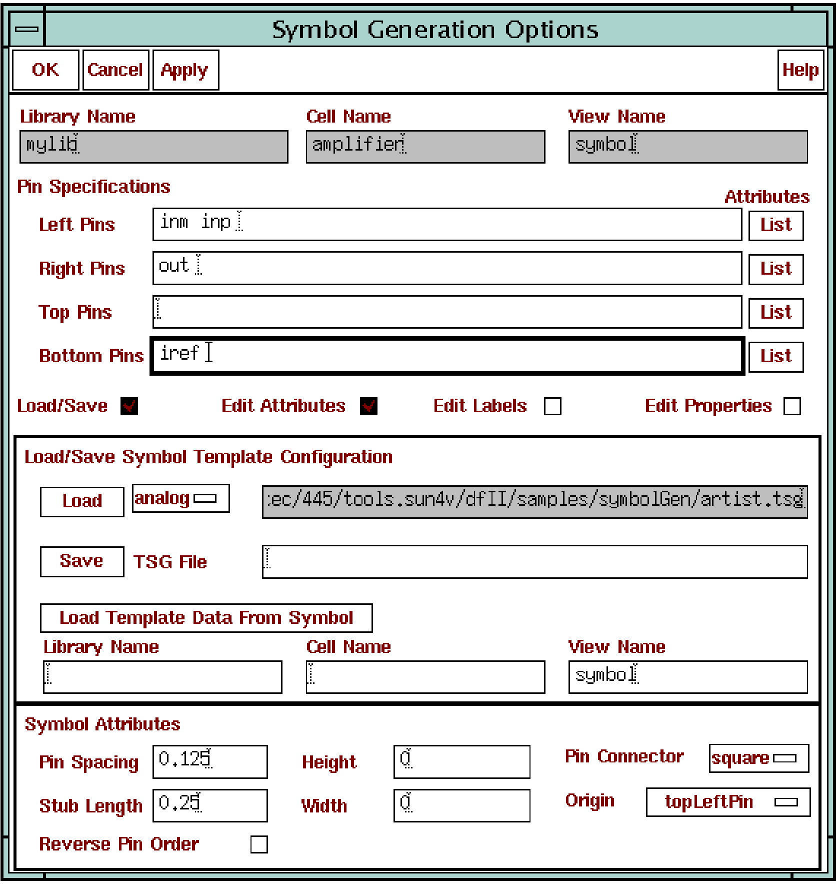
Symbol Generation

Design—Create Cellview—From Cellview



Library Name	mylib
Cell Name	amplifier
From View Name	schematic
To View Name	symbol
Tool / Data Type	Composer-Symbol
Display Cellview	☒
Edit Options	☒

Select **Apply** or **OK**



Library Name	Cell Name	View Name
mylib	amplifier	symbol

Pin Specifications

		Attributes
Left Pins	inm inp	List
Right Pins	out	List
Top Pins		List
Bottom Pins	iref	List

Load/Save ☒ Edit Attributes ☒ Edit Labels ☐ Edit Properties ☐

Load/Save Symbol Template Configuration

Load analog :ec/445/tools.sun4v/dfII/samples/symbolGen/artist.tsg

Save TSG File

Load Template Data From Symbol

Library Name	Cell Name	View Name
		symbol

Symbol Attributes

Pin Spacing	0.125	Height	0	Pin Connector	square
Stub Length	0.25	Width	0	Origin	topLeftPin
Reverse Pin Order	☐				

This form opens with only the top portion, press these buttons to extend the symbol generation capabilities of the form.

Select a symbol generation template, other than the.cdsinit entry.

Select symbol generation attributes to control the symbol drawing.

Symbol Generation

Bring up ASG from the schematic window. A template file will be used for the symbol creation. There are different symbol template files for different tools in the Design Framework II environment.

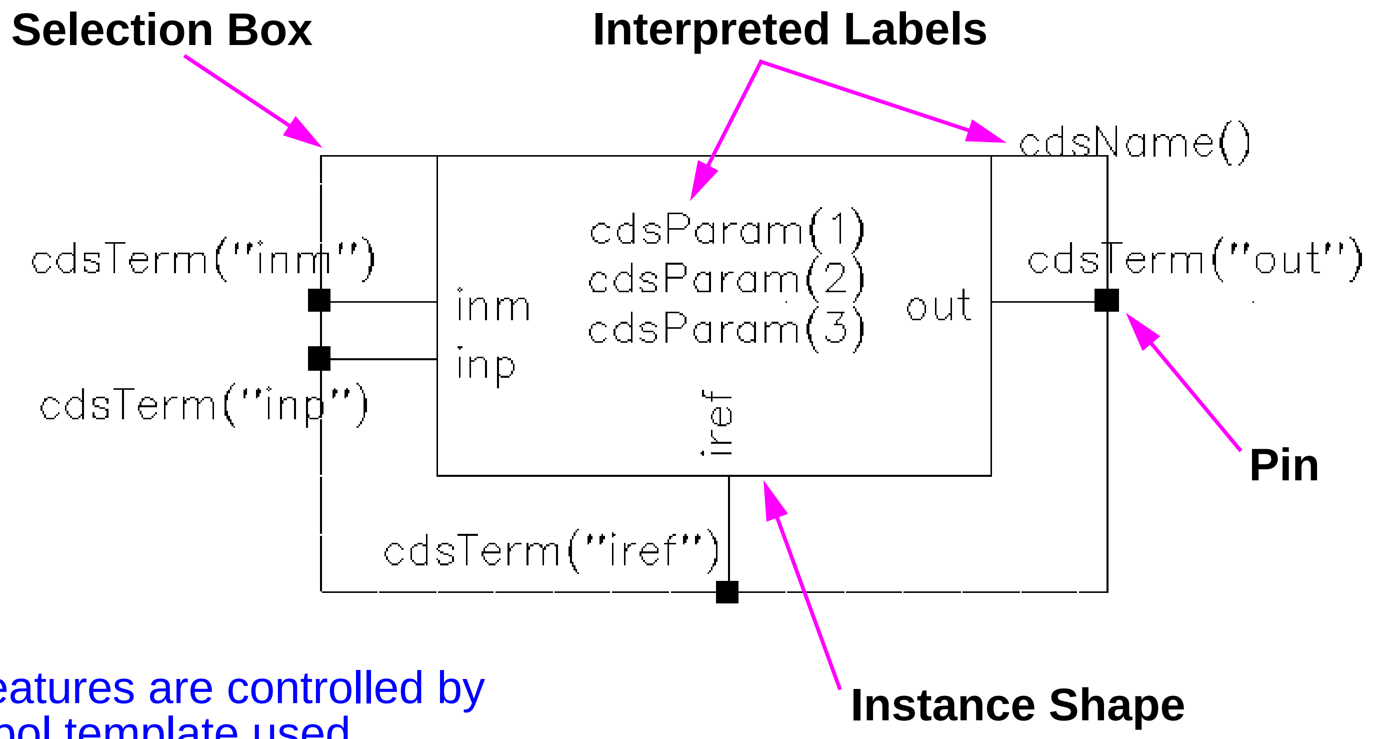
To make sure the *analog* symbol generation template is used in your design, put the following command in your *.cdsinit* file:

```
schSetEnv( "tsgTemplateType" "analog" )
```

Note: Failure to set this will result in a digital symbol generation.

Notice that the From View Name and To View Name fields can be modified in the Cellview from Cellview form. This provides a way to create other views, such as *behavioral* or *ahdl* from your schematics.

Characteristics of an Automatically Generated Symbol



These features are controlled by the symbol template used.

Interpreted labels on the symbol act as “placeholders” for different types of information to be displayed in the schematic.

- *cdsTerm()* labels display pin names or the net names.
- *cdsParam()* labels display parameters of an instance.
- *cdsName()* labels display the instance or cell name.

Characteristics of an Automatically Generated Symbol

An automatically generated symbol cellview includes pins, a rectangular graphic and labels. It can be modified using the symbol editor.

Note: The generation depends on the symbol template selected either in .cdsinit or by the symbol generation form.

There is some control over how automatically generated symbols are drawn. By default, pins are placed at the left side of the symbol if their direction is input, at the right side of the symbol if their direction is output, and on top of the symbol if the direction is InputOutput. Options exist to change the pin appearance and order of the pins.

Interpreted Labels allow information to appear near the symbol after it is placed in another cellview. For the label generation template, the three label types are:

- *cdsTerm()* labels display pin names or the net names pins connect to.
- Each *cdsParam()* label can display a parameter of the instance. There can be multiples of this label.
- *cdsName()* labels display the instance or cell name.

All placeholder labels can be rearranged so that labels on instantiated instances are moved accordingly. These three labels have meaning in the Analog Design Environment. They display certain attributes, which are discussed later.

A selection box is drawn around the symbol and can be edited. It defines the symbol’s selectable region after it is placed in another cellview.

Schematic Window Icons and Accelerator Keys

The Composer Schematic software provides both icons and “Accelerator” keys to simplify schematic capture. The icons and Accelerator keys also reduce the time needed to capture and edit schematics.

- The icons appear on the left-hand side of the schematic editing window.
- An icon is activated by using a left click over the icon.
- Accelerator keys are activated by pressing specified keys on the keyboard.
- Accelerator keys are sometimes referred to as **bindkeys**.

Schematic Window Icons and Accelerator Keys

Composer Command Summary

COMMAND:	bindkey	Mouse Sequence:	icon?
Add Component	**i**	Add — Instance	yes
Select Component(s)	LMB	left click, or drag LMB	
Copy	**c**	Edit — Copy	yes
Delete	**Del**	Edit — Delete	yes
Move	**m**	Edit — Move	
Stretch	**M**	Edit — Stretch	yes
Rotate	**r**	Edit — Rotate	
Repeat		RMB	yes
Modify Properties	**q**	Edit — Properties — Objects...	yes
Add Wire	**w**	Add — Wire	yes
Add Wire Name	**l**	Add — Wire — Name...	yes
Add Pin	**p**	Add — Pin	yes
Undo	**u**	Edit — Undo	yes
Redo	**U**	Edit — Redo	

Composer Command Summary

COMMAND:	bindkey	Mouse sequence:	icon?
Zoom in by 2	**]**	Window — Zoom — zoom in by 2	**yes**
Zoom out by 2	**[**	Window — Zoom — zoom out by 2	**yes**
Zoom in	**z**	Window — Zoom — Zoom in	
Zoom out	**Z**	shift — RMB	
Fit	**f**	Window — Fit	
Redraw	**f 6**	Window — Redraw	
Check and Save	**X**	Design — Check and Save	**yes**
Save As	**^s**	Design — Save As	
Delete Marker	**^g**	Check — Delete Marker	
Descend Edit	**E**	Design — Hierarchy — Descend Edit	
Return	**^e**	Design — Hierarchy — Return	
Delete All Edits		Design — Discard Edits	
Close		Window — Close	

Bindkeys

Many of the schematic capture commands have alternative ways to be invoked.

1. A command sequence such as: **Edit—Properties—Objects**
2. An icon such as:
3. A bindkey such as **q**

Bindkeys include the following features:

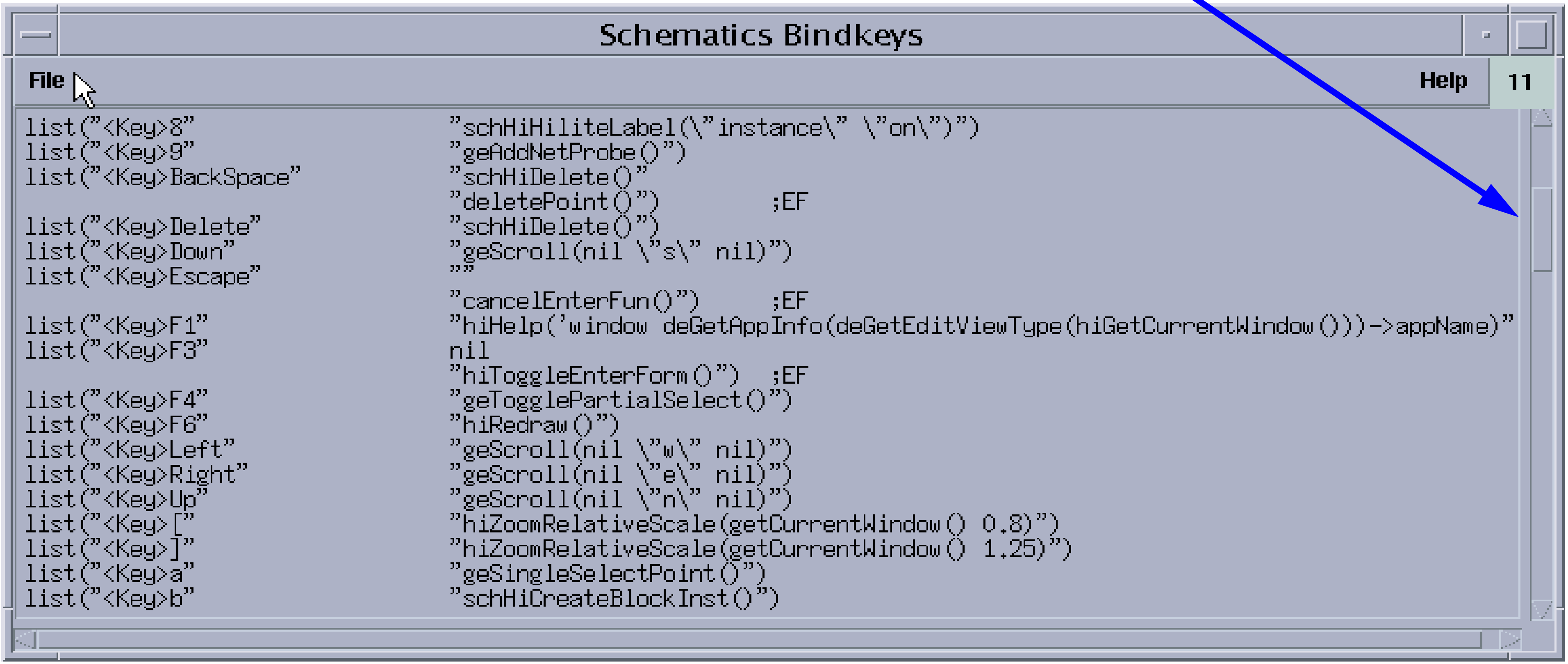
- Speed up schematic capture flow
- Default set of functions with installation
- Functions may be customized
- Full set of may be viewed or changed using **Options—Bindkeys** in the CIW

Bindkeys

Bindkeys simplify the schematic capture flow. A default set of Accelerator keys is provided; however, the keys are programmable. To view the key settings and the corresponding SKILL syntax, select in the CIW: Options—Bindkeys. A **“Key or Mouse Binding Window”** appears. This window shows bindkeys for the schematic editor and other tools. In the Application Type Prefix cyclic field select *schematic*.

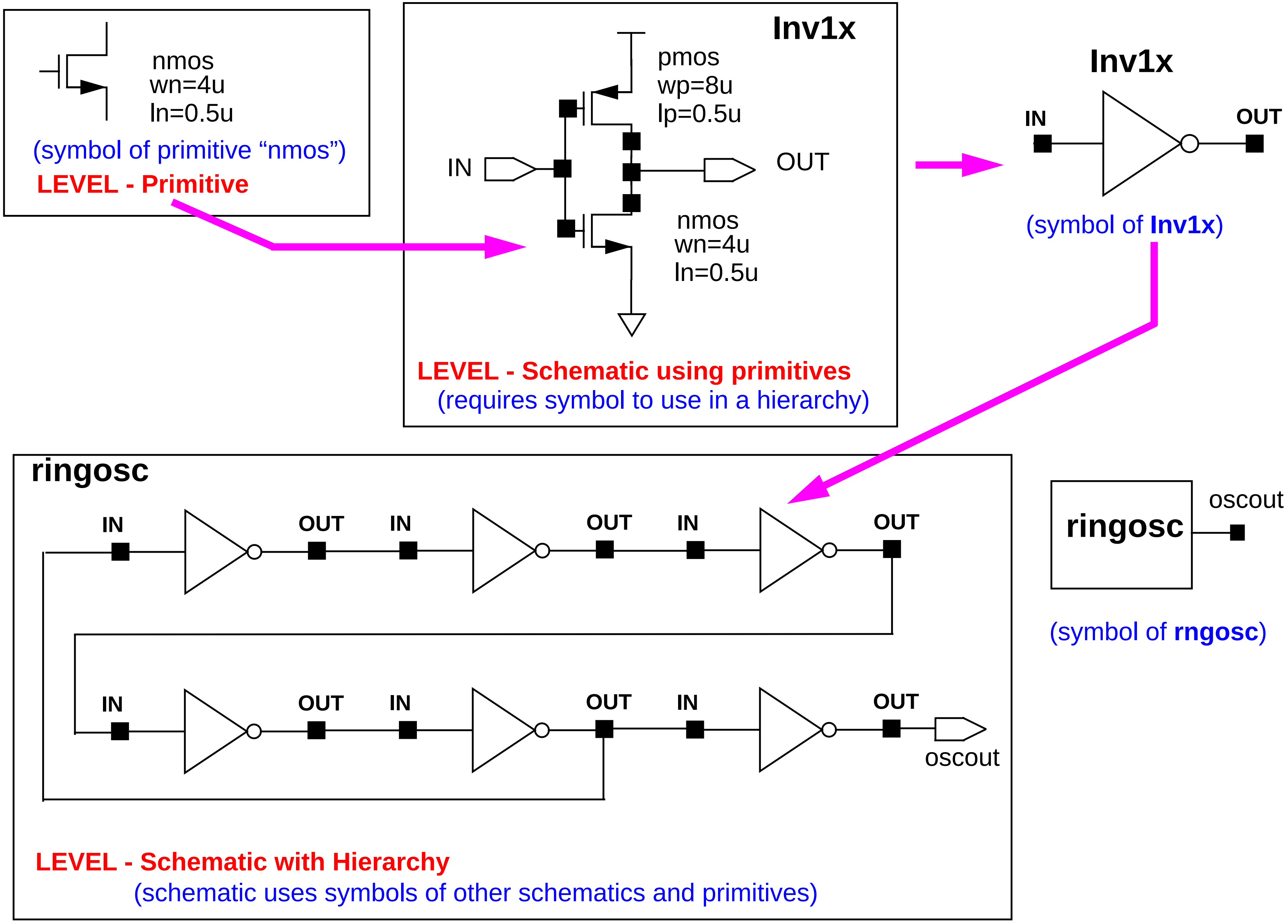
Then select the *Show Bind Keys* button. The Schematic Bindkeys window will appear.

Note: This is a partial list. To view all bindkeys use scroll bar.



```
list("<Key>8"                "schHiHiliteLabel(\"instance\" \"on\")")
list("<Key>9"                "geAddNetProbe()")
list("<Key>BackSpace"        "schHiDelete()"
                             "deletePoint()")          ;EF
list("<Key>Delete"           "schHiDelete()")
list("<Key>Down"             "geScroll(nil \"s\" nil)")
list("<Key>Escape"           ""
                             "cancelEnterFun()")       ;EF
list("<Key>F1"               "hiHelp('window deGetAppInfo(deGetEditViewType(hiGetCurrentWindow())->appName)"
list("<Key>F3"               nil
                             "hiToggleEnterForm()")    ;EF
list("<Key>F4"               "geTogglePartialSelect()")
list("<Key>F6"               "hiRedraw()")
list("<Key>Left"             "geScroll(nil \"w\" nil)")
list("<Key>Right"            "geScroll(nil \"e\" nil)")
list("<Key>Up"               "geScroll(nil \"n\" nil)")
list("<Key>["                "hiZoomRelativeScale(getCurrentWindow() 0.8)")
list("<Key>]"                "hiZoomRelativeScale(getCurrentWindow() 1.25)")
list("<Key>a"                "geSingleSelectPoint()")
list("<Key>b"                "schHiCreateBlockInst()")
```

Using a Hierarchy



(symbol of primitive “nmos”)

LEVEL - Primitive

LEVEL - Schematic using primitives
(requires symbol to use in a hierarchy)

(symbol of **Inv1x**)

LEVEL - Schematic with Hierarchy
(schematic uses symbols of other schematics and primitives)

(symbol of **rngosc**)

Using a Hierarchy

A hierarchy is the design data of a complex system organized into simple and manageable data at different levels. A hierarchy simplifies the complex structure of a system. It often reduces the storage requirements for the data. It also simplifies and reduces the time to design the system.

Primitives are the basic design elements and exist at the bottom of the hierarchy. The design does not descend below this level. A schematic may consist entirely of primitives. Such a schematic is also referred to as a flat schematic or a primitive-level schematic. For large systems, for example a 16-bit analog-to-digital converter, it is difficult capture the design with a flat schematic.

A design of a complex system can consist entirely of a single schematic. The flat schematic can be simulated by including sources for power and stimulus. A design using only a flat schematic (without hierarchy) is inefficient when repeated structures are used. Such a schematic becomes difficult to manage as the circuit complexity increases.

A schematic can use symbols of primitives and symbols for other schematics. Such a schematic is more efficient to design repeated structures and complex systems. This use of symbols to represent schematics continues to higher and higher levels of hierarchy until the TOP level of the design is reached.

A symbol for a schematic view is only required when that schematic is used within a hierarchy.

Labs

Lab 2-1 Schematic Entry

Lab 2-2 Symbol Creation

Lab 2-3 Building the Supply Circuit

Lab 2-4 Building the ampTest Design

Labs

Lab Reference Material: Mouse Buttons

Left Mouse Button—Select and Deselect

Click	Select point
Double click	Extend select
Shift-click	Select point (add)
Control-click	Deselect point
Draw through	Select box or Direct Edit*
Shift draw through	Select box (add) or Direct Edit*
Control draw through	Deselect box or Direct Edit*
(EF)	Add point

Middle Mouse Button
Pop-Up Menus

Click	Pop-up menus
(EF)	Pop-up menus

Right Mouse Button
Repeat, Zoom, Options

Click	Repeat last command
Draw through	Zoom in
Shift draw through	Zoom out
(EF)	Command options (command-specific bindings)

Note: EF (Enter Function) bindkeys used within an active command.
*Direct Edit applies only when over object.

Lab Reference Material: Mouse Buttons

Module 3: Analog Simulation

Topics in this module

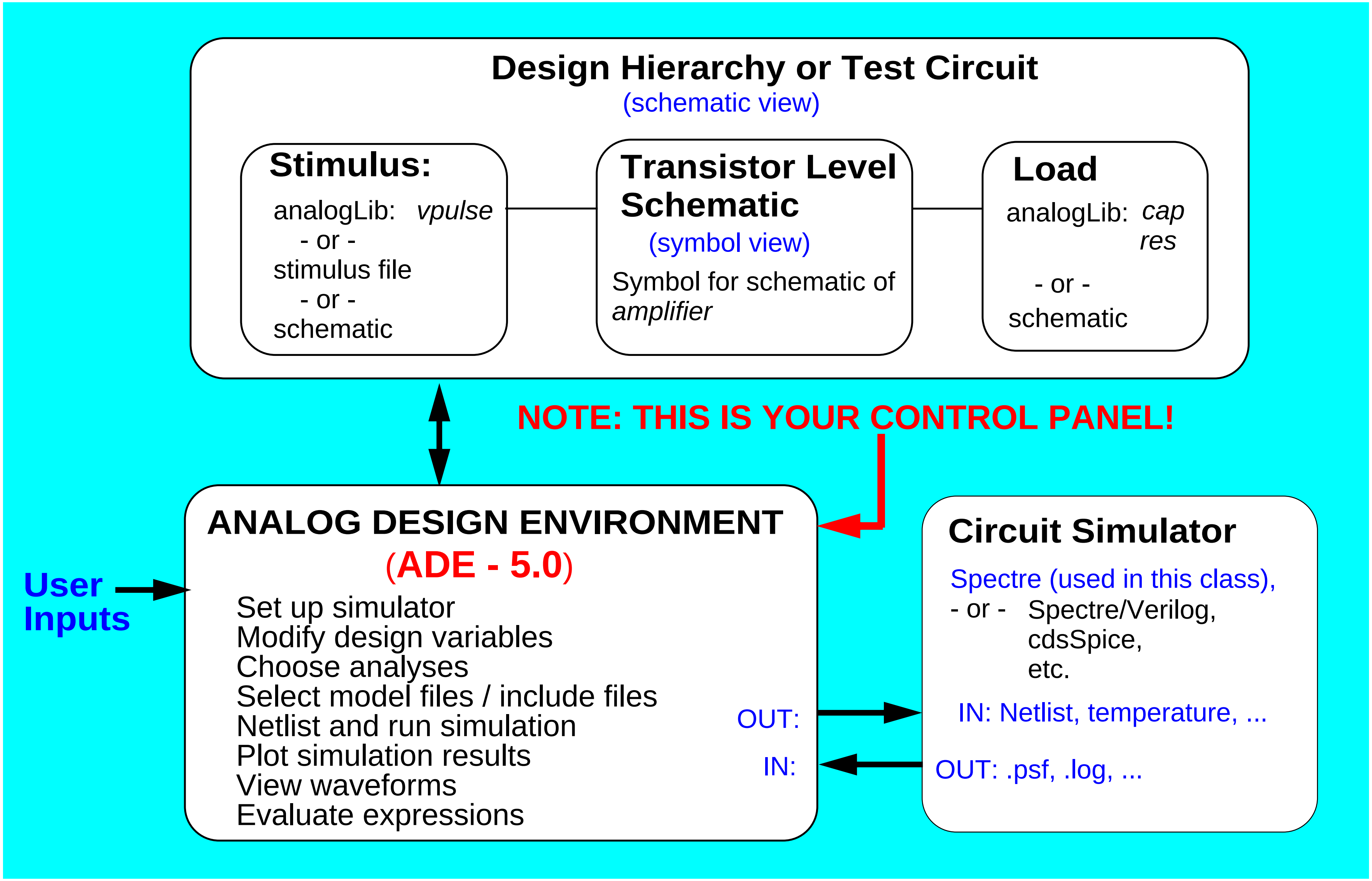
- Overview of the simulation environment
- Setting up the simulation environment
- Model files
- Design variables
- Choosing analyses
- Netlisting
- Running simulation
- Viewing simulation results with the Waveform display tool
- Saving simulator sessions

Terms and Definitions

simulation window	The ADE user interface to control and view simulations.
analyses field	A text field in the simulation widow indicating selected analyses.
output field	A text field in the simulation widow indicating selected output.
Spectre	A Cadence simulation tool for simulating analog circuits.
simulator host	Software tool such as Spectre, cdsSpice, etc. to be used for simulation.
model library	A text file having model description used by the simulator host.
stimulus template	A user interface used to establish signals used in simulation.
netlist	A textual description of a schematic used by the simulator host.
Waveform Window	A graphical interface used to plot simulation data.
direct plot	User command used for ‘special’ plots to the Waveform Window.
annotating	Process of displaying data back to another window or schematic.
label display	A label attached to a component for displaying information.
snapshot	A command to the Waveform Window to update intermediate results.

Overview of the Analog Design Environment

The **Analog Design Environment** is a user-friendly graphical interface tool to set up, run, and evaluate analog circuit simulations:



Design Hierarchy or Test Circuit
(schematic view)

Stimulus:
analogLib: *vpulse*
- or -
stimulus file
- or -
schematic

Transistor Level Schematic
(symbol view)
Symbol for schematic of *amplifier*

Load
analogLib: *cap*
res
- or -
schematic

NOTE: THIS IS YOUR CONTROL PANEL!

User Inputs

ANALOG DESIGN ENVIRONMENT
(ADE - 5.0)
Set up simulator
Modify design variables
Choose analyses
Select model files / include files
Netlist and run simulation
Plot simulation results
View waveforms
Evaluate expressions

OUT:
IN:

Circuit Simulator
Spectre (used in this class),
- or - Spectre/Verilog,
cdsSpice,
etc.

IN: Netlist, temperature, ...

OUT: .psf, .log, ...

Overview of the Analog Design Environment

The Analog Design Environment is a software design tool the provides an easy way to set up, run, and evaluate analog circuit simulations. It provides a simple way to change the input file to run the next simulation. It provides access to the simulation output files to view waveforms, make waveform measurements, and evaluate expressions. Think of the Analog Design Environment as your “control panel”.

Use the Analog Design Environment to:

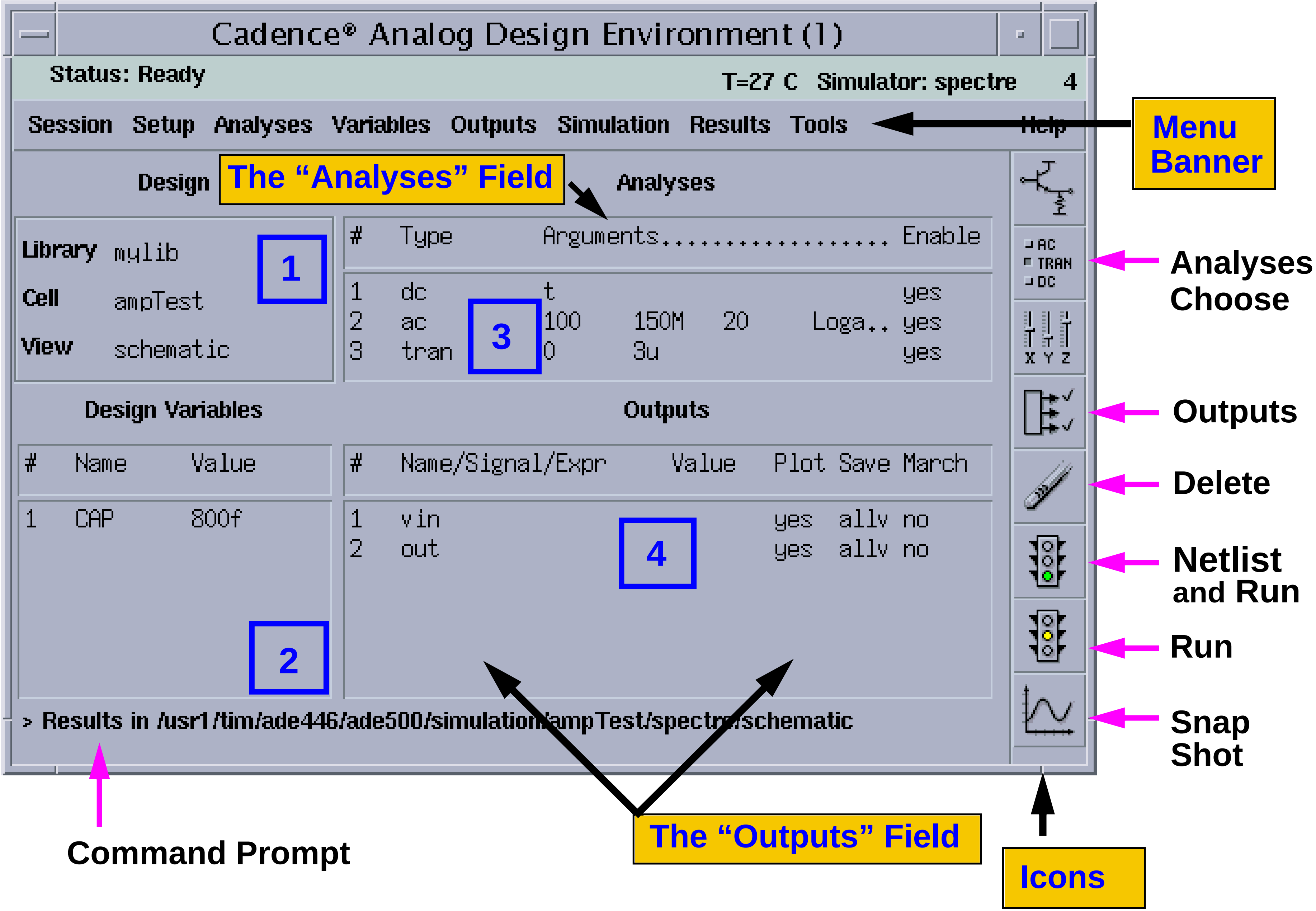
- Setup simulations
- Netlist the circuit
- Run the simulation
- Evaluate the simulation results

Important

The Analog Design Environment is often referred to as the **“simulation environment”** or the **“simulation window”**.

Note: It was also known in prior releases as “Analog Artist” and sometimes as “Artist”.

Important Features of the Simulation Window



Cadence® Analog Design Environment (1)

Status: Ready T=27 C Simulator: spectre 4

Design

Library mylib

Cell ampTest

View schematic

Analyses

#	Type	Arguments...................				Enable
1	dc	t				yes
2	ac	100	150M	20	Loga..	yes
3	tran	0	3u			yes

Design Variables

#	Name	Value
1	CAP	800f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	vin		yes	allv	no
2	out		yes	allv	no

> Results in /usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic

Important Features of the Simulation Window

The diagram shows the Analog Design Environment window. The simulation window is annotated with notes used to describe the important features.

The are four regions identified by numbered boxes 1, 2, 3, and 4.

- Region 1 is the “Design Field”. It indicates the library, cell, and cell view to be simulated.
- Region 2 is the “Design Variables Field”. It shows the design variables and set values.
- Region 3 is the “Analyses Field”. It shows the status, ranges, and types of analyses.
- Region 4 is the “Outputs Field”. It indicates the selected outputs and expressions.

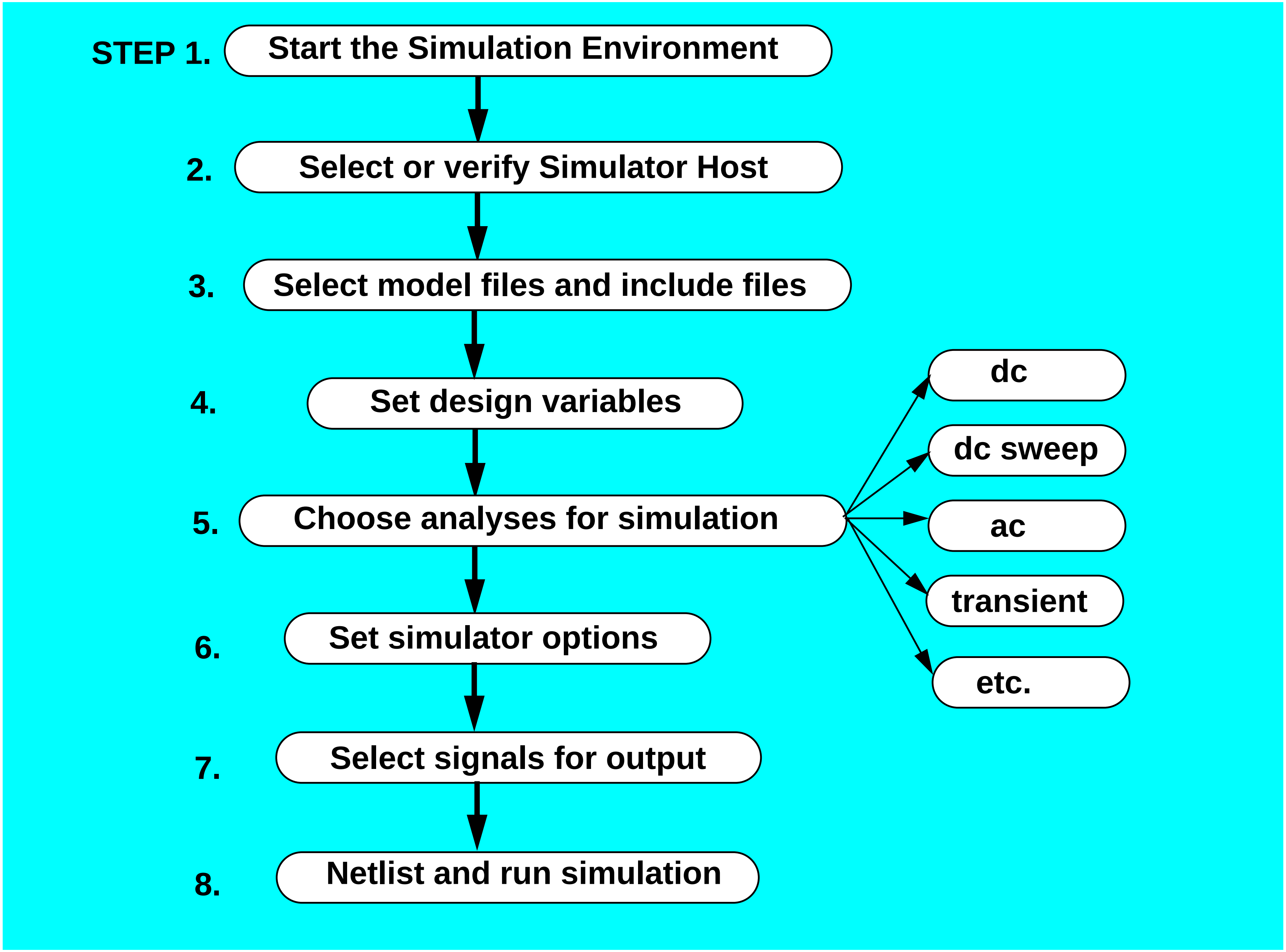
Important

The entries within Regions 2, 3, and 4 are accessible by executing a left mouse click. It is sometimes easier and faster to make changes within these fields than using the menu banner.

The simulation window also has a **Menu Banner** used to select analyses, models, outputs, and other simulation controls. The Menu Banner also has a Tools menu used to set up **advanced** simulation tools such as Corners, Monte Carlo, and Optimization.

The simulation window also has a set of icons for along the right-hand side for quick selection of specific commands.

Analog Simulation Flow



STEP 1. Start the Simulation Environment

2. Select or verify Simulator Host

3. Select model files and include files

4. Set design variables

5. Choose analyses for simulation

- dc
- dc sweep
- ac
- transient
- etc.

6. Set simulator options

7. Select signals for output

8. Netlist and run simulation

Analog Simulation Flow

This diagram shows the major steps required to set up and run an analog circuit simulation in the simulation environment. To set up the simulation the first time, most of these steps are required. However, on all subsequent simulations, perhaps only one or two steps are needed.

Each of the steps numbered above shall be discussed within this module.

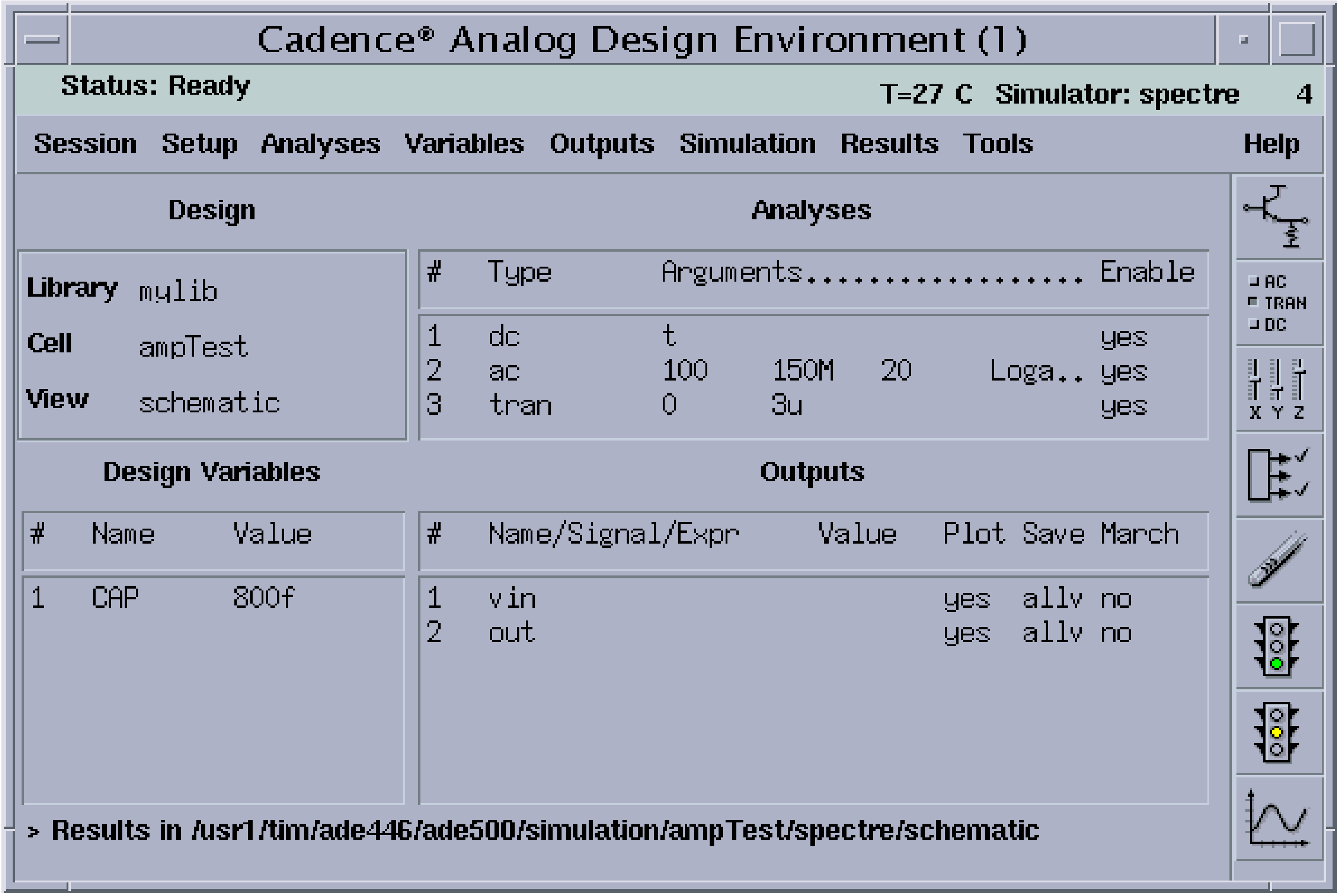
Important

If you are unfamiliar with the flow, this may appear to be a large amount of work to set up the simulation. However, after the initial setup is completed, the flow is greatly simplified. Once you are familiar with the flow, all of these steps are performed within a few minutes, or less.

Starting the Simulation Environment

STEP 1. Select **Tools—Analog Environment** from the schematic menu banner, or select **Tools—Analog Environment—Simulation** from the CIW.

Then the Analog Design Environment "**Simulation Window"** appears.



Status: Ready T=27 C Simulator: spectre 4

Design

Library	mylib
Cell	ampTest
View	schematic

Analyses

#	Type	Arguments...................	Enable
1	dc	t	yes
2	ac	100 150M 20 Loga..	yes
3	tran	0 3u	yes

Design Variables

#	Name	Value
1	CAP	800f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	vin		yes	allv	no
2	out		yes	allv	no

> Results in /usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic

REMEMBER: This is your Control Panel.

Starting the Simulation Environment

STEP 1.

In this class, use the Spectre simulator that is integrated directly into the Analog Design Environment.

Start the simulation environment from the schematic window or the CIW. If the simulation environment is started from the design window, the design in the window is understood to be the target of the analysis. Start the simulation environment from the CIW to simulate any design without viewing it.

Commands on pull-down menus in the Simulation window establish a simulation flow if used in order, starting at the top and left and ending with the last command on the right most pull-down menu.

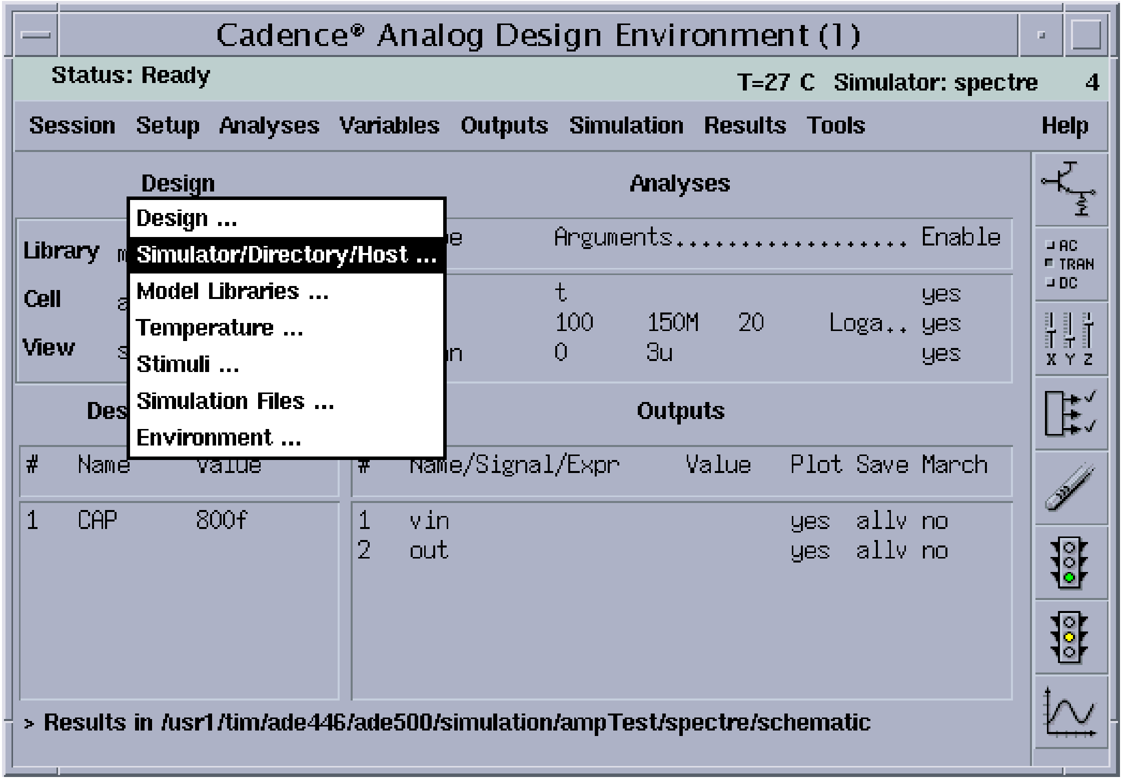
For analog analysis, include this statement in your *.cshrc* file before starting the Cadence Design Framework II environment:

```
setenv CDS_Netlisting_Mode “Analog”
```

Setting the Simulator

STEP 2.
Select the simulator to be used.

Select: Setup—Simulator/Directory/Host



Cadence® Analog Design Environment (1)

Status: Ready T=27 C Simulator: spectre 4

	Arguments.................				Enable
	t				yes
	100	150M	20	Loga..	yes
n	0	3u			yes

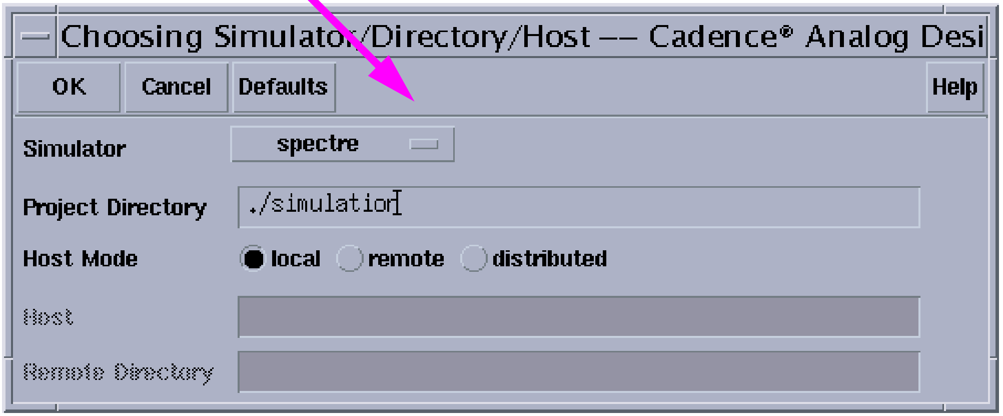
#	Name	Value
1	CAP	800f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	vin		yes	allv	no
2	out		yes	allv	no

> Results in /usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic

For this class select **spectre** in the cyclic field.



Choosing Simulator/Directory/Host –– Cadence® Analog Desi

Simulator spectre

Project Directory ./simulation

Host Mode ● local ○ remote ○ distributed

Host

Remote Directory

Setting the Simulator

STEP 2.

The second step of the Analog Simulation Flow is to set the target simulator and Project Directory. The Project Directory is the location where the simulator puts all of the data, including netlists, waveforms, include files, and mapping information.

Set the simulator to **spectre** to start direct simulation with the Spectre tool. Note that a cyclic field appears that allows other host simulator choices, such as *spectreVerilog*.

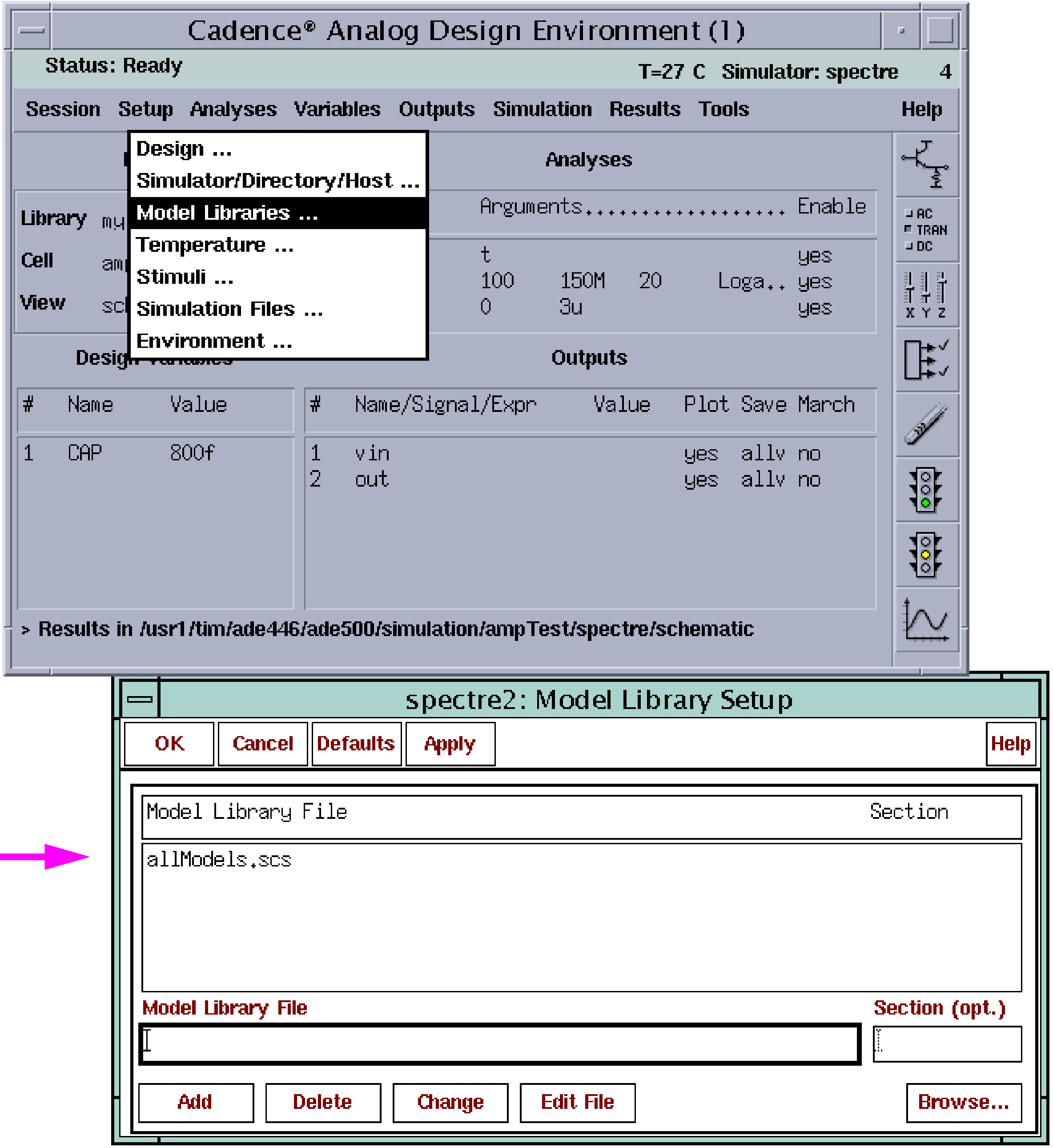
The Choosing Simulator/Directory/Host form has a button that provides a way to choose the **distributed** Host Mode. When this button is pressed, there are two additional fields added to your form: **Auto Job Submit**, and **E-Mail Notify**. This function uses multiple machines to handle large or computationally intensive jobs. Jobs are sent to the default queue named in your *.cdsenv* file or to the last setting in the Job Submit form. When **Auto Job Submit** is not selected, the Job Submit form appears whenever a simulation is run. More information can be found in CDSDoc.

Note: In the 4.4.2 and older releases, the Cadence analog design environment depended on socket interfaces to analog simulators. These socket interfaces used Cadence SPICE to compensate for simulator limitations. The most important of these limitations was the lack of an expression evaluation capability. Today, most analog simulators no longer have these limitations. Direct simulation (with Spectre) takes advantage of evaluation capability.

Setting the Model Libraries

STEP 3.
Select the model files. In simulation window, select:

Setup—Model Libraries...



Status: Ready T=27 C Simulator: spectre 4

Library my
Cell am
View sc

Analyses

```
Arguments.................  Enable
t                           yes
100    150M   20     Loga.. yes
0      3u                   yes
```

Design Variables

```
#   Name    Value
1   CAP     800f
```

Outputs

```
#   Name/Signal/Expr    Value   Plot Save March
1   vin                         yes  allv no
2   out                         yes  allv no
```

> Results in /usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic

```
Model Library File                                  Section
allModels.scs
```

Model Library File:

Section (opt.):

This example uses a relative path. The path is set by the Include Path in the Simulation Files Setup form.

Setting the Model Libraries

STEP 3.

This is a critical step that must be performed to ensure that model files for active devices, such as transistors, are included in the netlist.

With Spectre Direct:

- There are no **.m* files (**.m* is *cdsSpice* socket syntax).

  A conversion utility exists to convert **.m* files to **.scs* files.

  .*scs* = Spectre circuit simulator. All files ending in .*scs* are implied to be Spectre syntax in the netlist unless **simulator lang=spice** is included in the file. (SPICE in this case means general Berkeley SPICE syntax and NOT cdsSpice). Spectre example:

  ```
  model npn bjt type=npn is=3.26E-16 va=60 bf=100 \
  br=6 nc=2 ikr=100m rc=1 vje=0.7 \
  cjc=1e-12 fc=0.5 cje=0.7e-12 \
  tr=200e-12 tf=25e-12 itf=0.03 vtf=7 xtf=2
  ```

- There are several modeling techniques to use. The main use model is one file with all of the models contained within.

- The contents of the model file do not appear in the netlist. The library model file is referenced as an include file.

There are samples of all of the models available at:

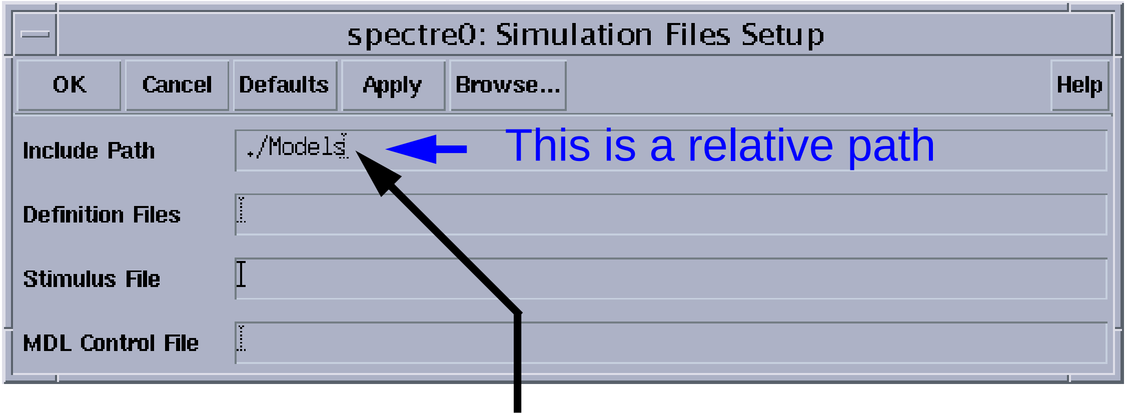
```
<install_dir>/tools/dfII/samples/artist/models/spectre
```

Simulation Files

STEP 3.

Setup—Simulation Files

For setting the path to other simulation files



Include Path ./Models

Definition Files

Stimulus File

MDL Control File

This is a relative path

Enter the absolute or relative path into the text field

Include Path	Path to location of any Model Library Files, Definition Files, and Stimulus Files (the paths in these fields can be relative paths)
Definitions Files	File or files that contain function definitions and parameter declarations not in the Design Variables section of the Simulation Window
Stimulus Files	Location of text-based stimulus file
Spectre MDL Files	File for measurements using Spectre MDL commands

- Each of the above fields can have one or more paths or file declarations separated by a space.
- Any type of Spectre include file can be included in the netlist by either using the Definitions Files field above or the Model Libraries Setup form.

Simulation Files

STEP 3.

Include Path—Example of absolute path declaration: *“/usr/home/models/project/models”*

The simulator resolves a relative file name by first looking in the directory where the file is located. Subsequently, it searches for the file in each of the directories specified by the include path, from left to right.

Definitions File—Defines functions and global variables that are not design variables. It is included in the netlist before model files. Example:

```
simulator lang=spectre
real PiRho() {
    return 2500;
} Functions returning constant values
real Rpb(real l, real w) {
    return PiRho()*l/w;
} Simple passing parameters
real rpoly(real value, real tdc) {
value*(1+.01*(tdc-25)+.002*(tdc-25)**2);
} poly resistor function of temperature
```

A sample Definitions File exists at:
`<install_dir>/tools/dfII/samples/artist/models/spectre/defaults.scs`

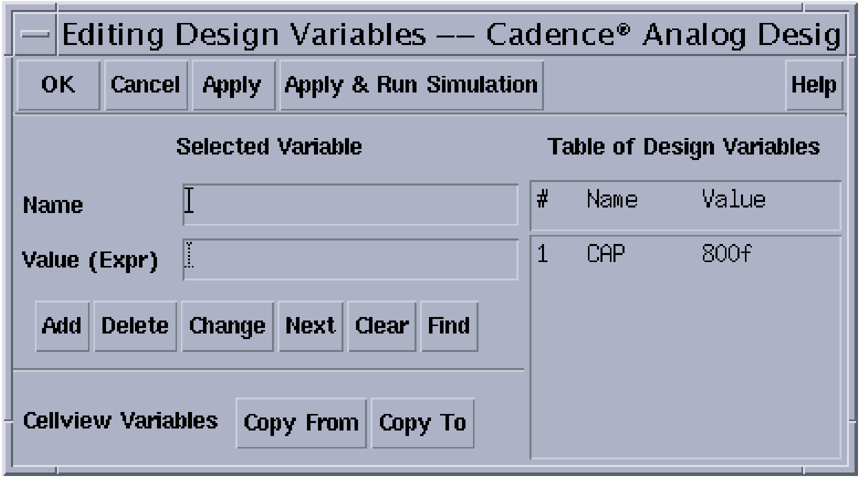
Stimulus File—Enter the full path to the directory where the stimulus file resides. A file name ending in **.scs* defaults to Spectre. All other files default to SPICE syntax.

Spectre MDL File—For more information on using Spectre MDL see Appendix C of this manual.

Setting Design Variables

STEP 4.

- Extract variables in a design with the **Copy From** button.
- Copy variable settings back to the design with the **Copy To** button.
- Add variables used in parameterized model files that are not extracted.
- Update a variable value and run simulation. Netlisting does not occur again.
- Extract new variables added after a simulation run.
- Use the **Find** Button to locate the Selected Variable in your design.



#	Name	Value
1	CAP	800f

Select **Variables—Edit** or click the **Edit Variables** Icon.

Setting Design Variables

STEP 4.

Use the Editing Design Variables form to:

- Enter design variable values. These values can be numbers or expressions.
- Copy variables from a design, or copy set variables back to the design to be saved.
- Add new variables. Add variable names to this form that control the simulation engine or appear in model files and then set their values.
- Find a variable in the design. This useful feature highlights the component to which the selected variable is attached. This can be very helpful in a large design when trying to locate a particular design variable.

When starting the simulator from this form, change a variable value, apply the change, and repeat simulations quickly. The system will not create a new netlist when a variable value is updated.

Spectre Direct has no character limit for the size of a variable. The following examples are legal statements:

```
parameters thisIsAReallyLongDesignVariableName=10000 desVar2=10p \
desVar3=2u desVar4=1.15
```

Design variables are not evaluated. They appear in the netlist as Spectre parameter statements.

Choosing Analyses

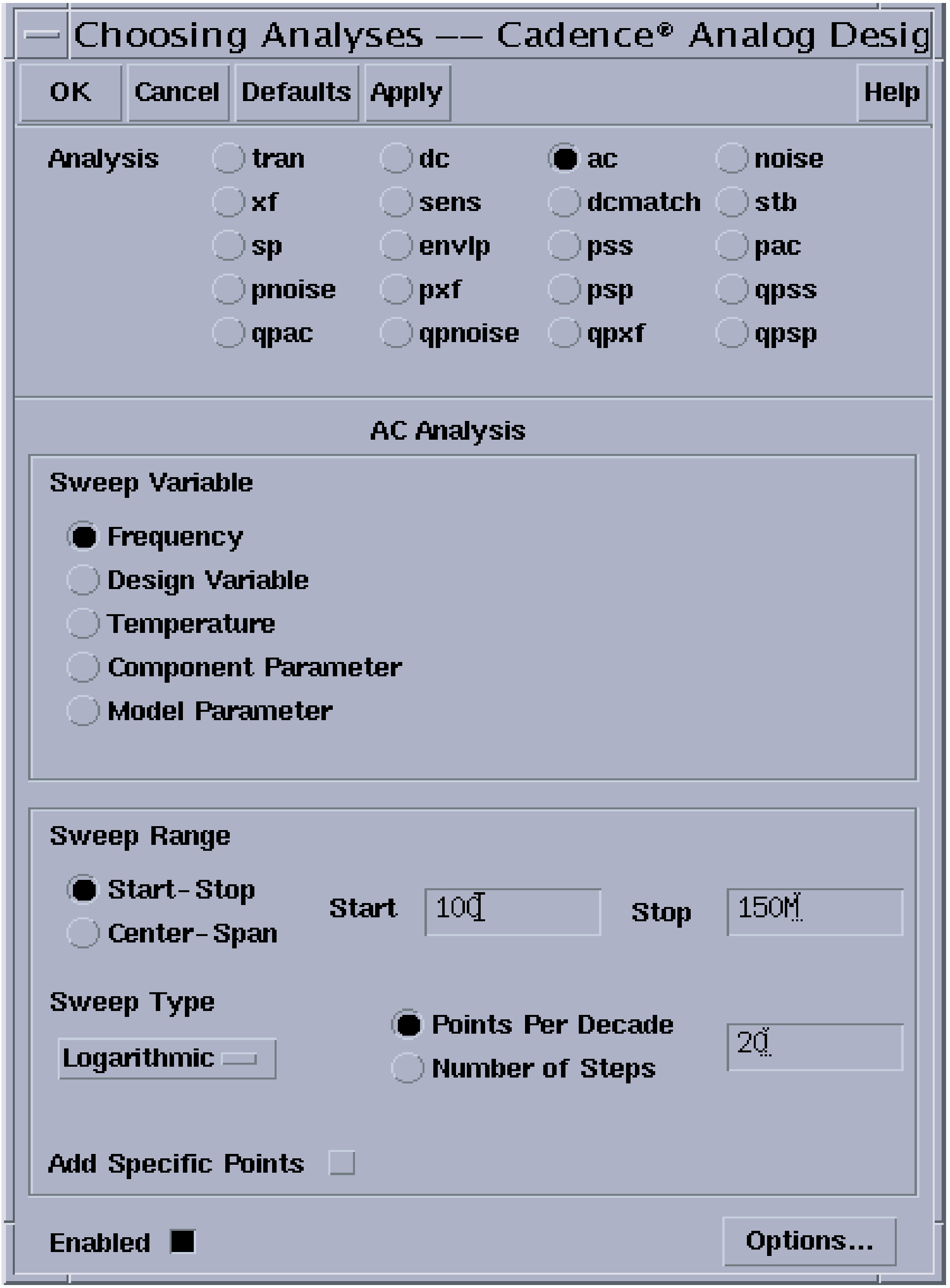
STEP 5.

Select

Analyses—Choose

or click the

Choose Analyses icon.



Analysis ○ tran ○ dc ● ac ○ noise
○ xf ○ sens ○ dcmatch ○ stb
○ sp ○ envlp ○ pss ○ pac
○ pnoise ○ pxf ○ psp ○ qpss
○ qpac ○ qpnoise ○ qpxf ○ qpsp

AC Analysis

Sweep Variable

● Frequency
○ Design Variable
○ Temperature
○ Component Parameter
○ Model Parameter

Sweep Range

● Start-Stop Start 100 Stop 150M
○ Center-Span

Sweep Type

Logarithmic

● Points Per Decade 20
○ Number of Steps

Add Specific Points □

Enabled ■

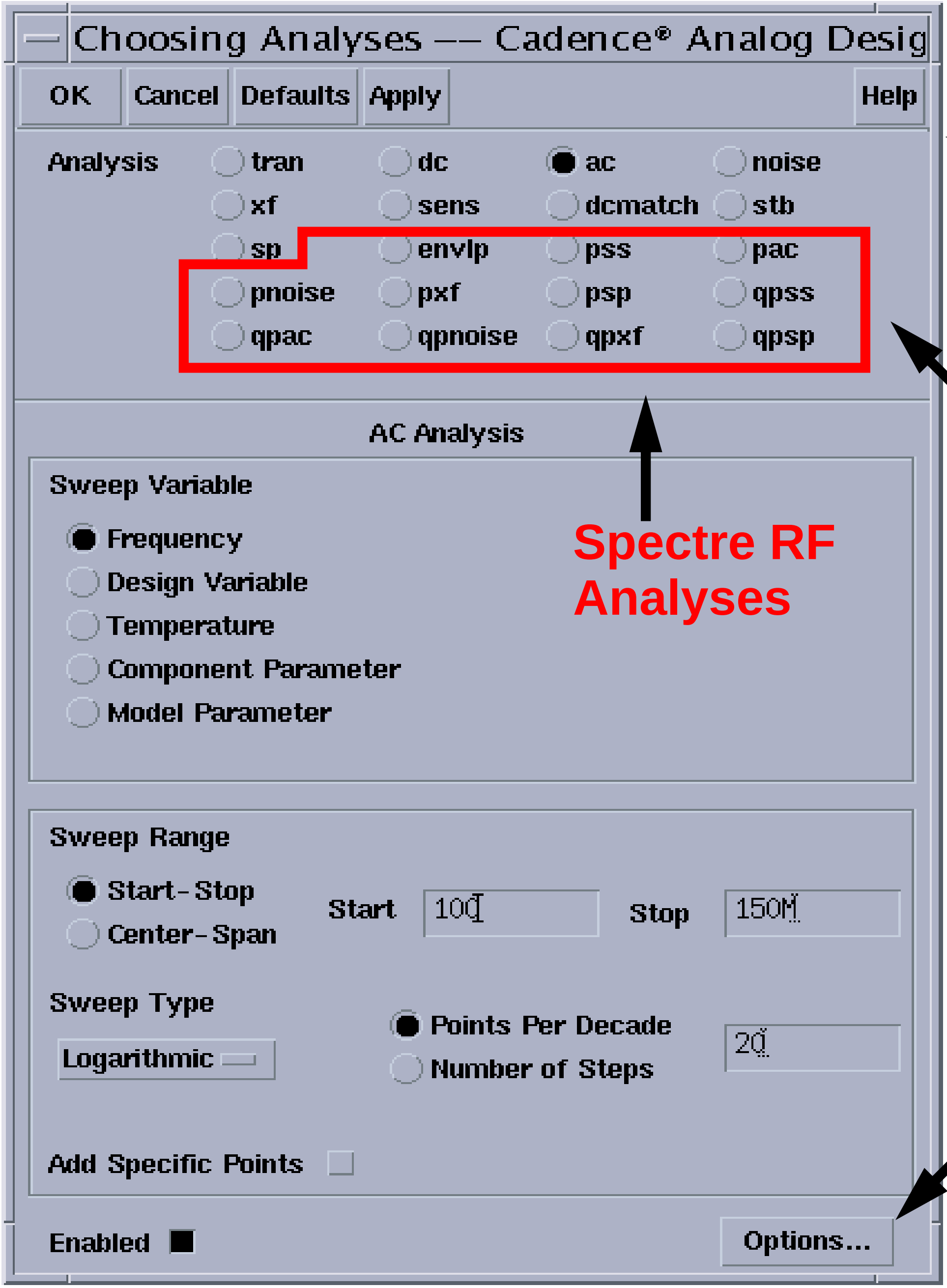
Choosing Analyses

STEP 5.

Choosing the analyses to perform on the design is a straightforward process. Run analyses together, separately, and in any combination. Analysis units are determined by the simulation engine.

Output data is generated for each specified analysis during a simulation.

Choosing Analyses Details



Choosing Analyses —— Cadence® Analog Desig

Analysis				
	○ tran	○ dc	● ac	○ noise
	○ xf	○ sens	○ dcmatch	○ stb
	○ sp	○ envlp	○ pss	○ pac
	○ pnoise	○ pxf	○ psp	○ qpss
	○ qpac	○ qpnoise	○ qpxf	○ qpsp

AC Analysis

Sweep Variable

● Frequency
○ Design Variable
○ Temperature
○ Component Parameter
○ Model Parameter

Sweep Range

● Start-Stop
○ Center-Span

Start 100 Stop 150M

Sweep Type

Logarithmic

● Points Per Decade 20
○ Number of Steps

Add Specific Points □

Enabled ■

Spectre RF Analyses

The Choosing Analysis form dynamically changes based on the host simulator and the selections made on the form.

The types of analyses that are available depend on the host simulator.

Sweep Variables depend on the selected analysis.

Sweep Range

Analysis Options are entered by selecting the Options... button.

Choosing Analyses Details

The Choosing Analysis form depends on the host simulator. The cdsSpice form has only four selectable types of analyses. The form above is used for the Spectre circuit simulator, which supports 15 types of analyses. These include **ac, dc, tran, stb, dcmatch**. In addition:

sp	S-Parameter Analysis

Also, the following types of RF analyses are supported by Spectre RF tool:

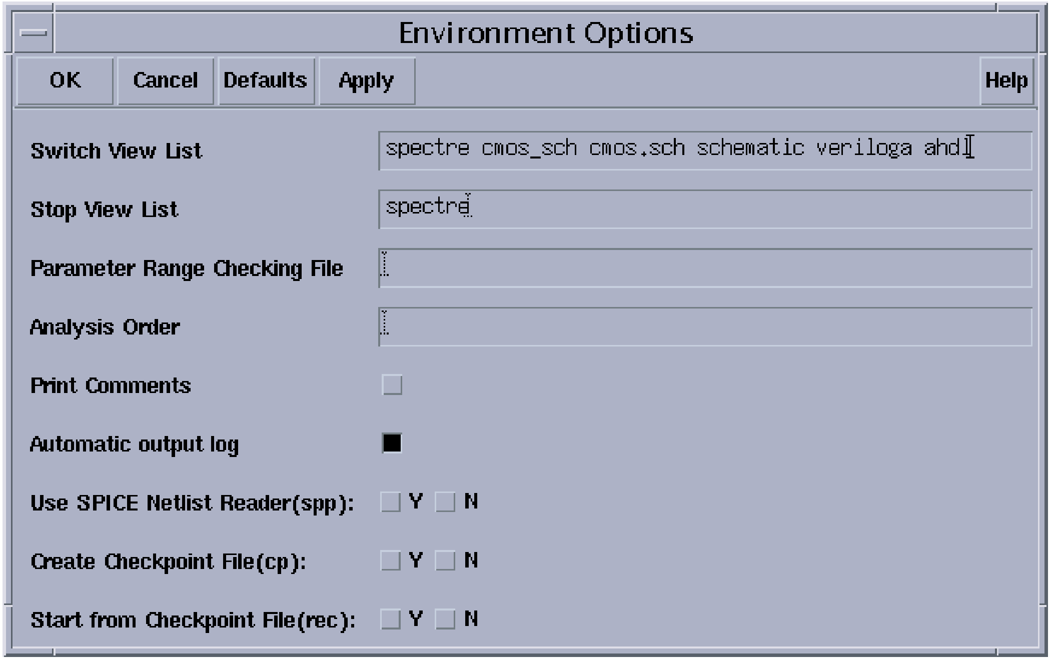
envlp	Envelope Following Analysis
pss	Periodic Steady State
pac	Periodic AC
pnoise	Phase Noise Analysis
pxf	Periodic XF Analysis
psp	Periodic S-Parameter
qpss	Quasi-Periodic Steady State Analysis
qpac	Quasi-Periodic AC
qpnoise	Quasi-Periodic Noise
qpxf	Quasi-Periodic XF
qpqsp	Quasi-Periodic S-Parameter

For more information on Spectre RF analysis, see CDSDoc. In addition, Cadence Education Services provides training in Spectre RF usage.

Simulation Environment Options

STEP 6.

Select **Setup—Environment**



Switch View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop View List: spectre

Parameter Range Checking File:

Analysis Order:

Print Comments □

Automatic output log ■

Use SPICE Netlist Reader(spp): □ Y □ N

Create Checkpoint File(cp): □ Y □ N

Start from Checkpoint File(rec): □ Y □ N

- Switch View List and Stop View List establish netlisting rules.
- Parameter Range Checking File
- Use the SPICE Netlist Reader (spp)—Read in HSPICE/SPICE netlists and run with the Spectre simulator.
- Checkpoint and Restart

Simulation Environment Options

STEP 6.

The Switch View List specifies the order that the cellviews are netlisted.

The Stop View List specifies the view at which the netlist statements are generated.

For the Spectre simulator, the Parameter Range Checking File contains the parameter range limits. Use either the full path to the file, or a period (.) to specify a relative path to the directory where the Cadence tools were started.

(Optional) Check and set the options for view switching to control how the system netlists hierarchical designs. Normally, there is no need to modify the Switch View List and Stop View list.

The SPICE reader (*spp*) option is provided for include files or subcircuits that are in SPICE syntax. The *spp* automatically converts them to Spectre syntax, so a Spectre simulation can be run.

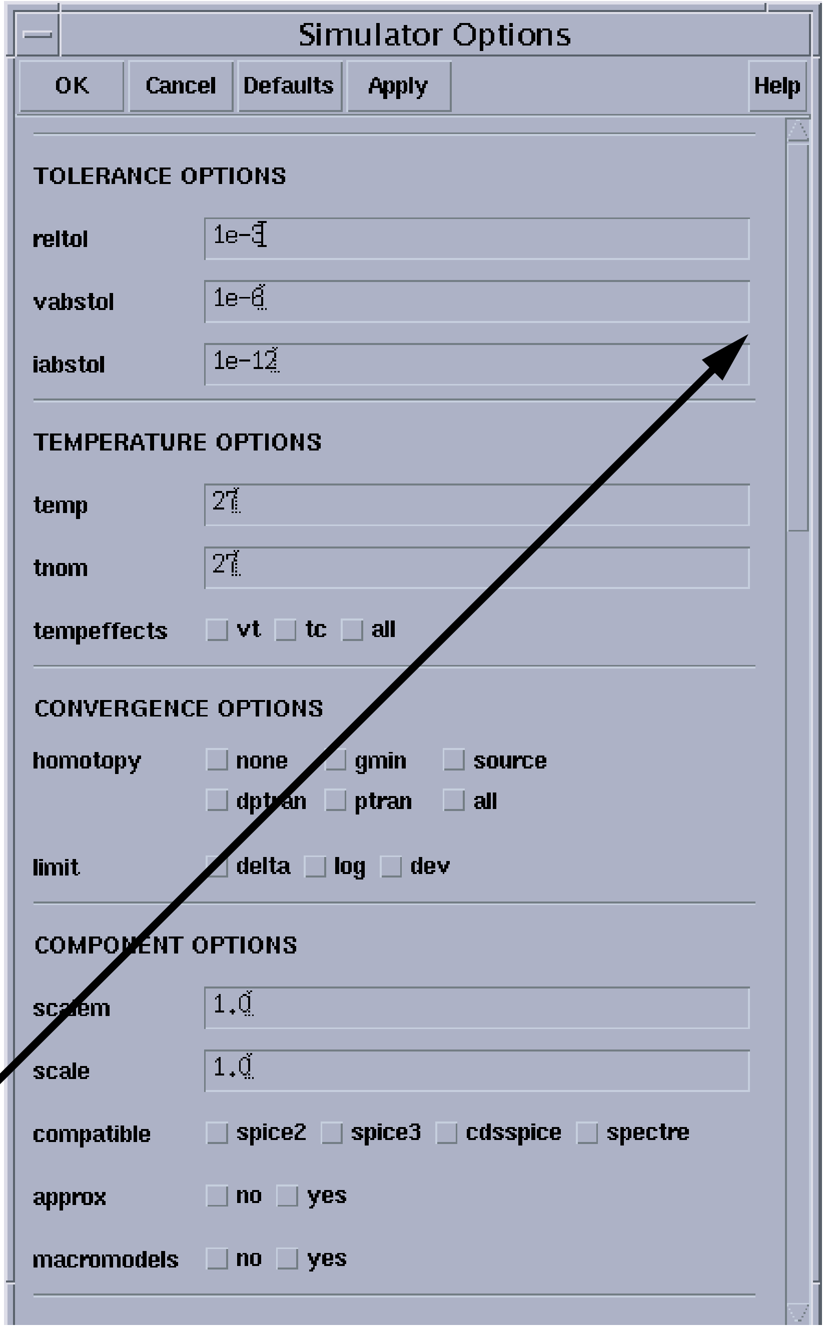
The **Create Checkpoint File (cp)** and **Start from Checkpoint File (rec)** options allow Spectre to save checkpoint files while a simulation is running, and then restart the analysis from this file. Currently, only transient analysis supports checkpoint and restart.

Simulator Options

STEP 6.

Select **Simulation—Options—Analog**

Use this form to set the simulator tolerance values, convergence options, and other settings.



TOLERANCE OPTIONS

reltol	1e-3
vabstol	1e-6
iabstol	1e-12

TEMPERATURE OPTIONS

temp	27
tnom	27
tempeffects	□ vt □ tc □ all

CONVERGENCE OPTIONS

homotopy	□ none □ gmin □ source □ dptran □ ptran □ all
limit	□ delta □ log □ dev

COMPONENT OPTIONS

scalem	1.0
scale	1.0
compatible	□ spice2 □ spice3 □ cdsspice □ spectre
approx	□ no □ yes
macromodels	□ no □ yes

NOTE:
This is a very long form. The scroll bar indicates the amount of the form that is visible.

Simulator Options

STEP 6.

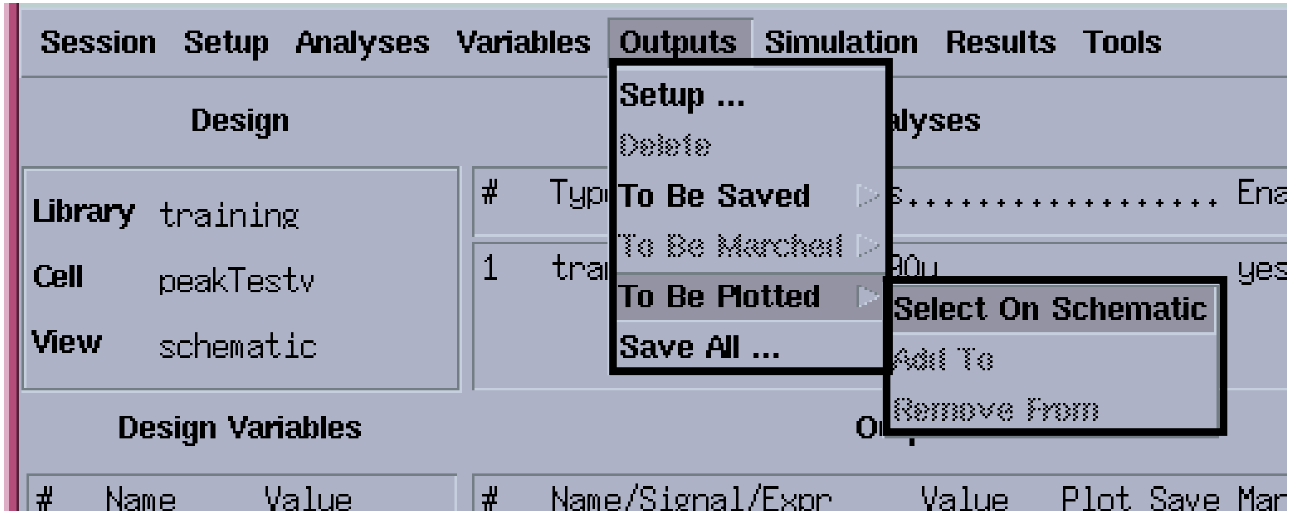
Select **Simulation—Options—Analog** to activate the Simulator Options form. Use this form to set convergence, tolerances, and other simulator settings.

In addition to this form, the Choosing Analyses form has an **Options** button for setting specific options for transient, ac, and dc analyses. For example, the **Infotimes** and **Captab** options are found by selecting the **tran** button in the Choosing Analysis form and then selecting the **Options** button.

Probing the Schematic to Save Output Data

STEP 7.

Select signals for output

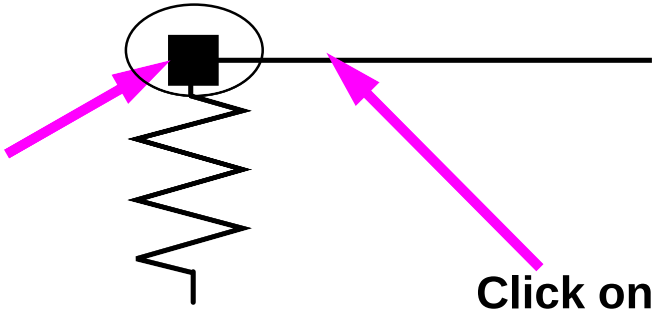


Library	training
Cell	peakTestv
View	schematic

#	Typ	s	Ena
1	tra	90u	yes

#	Name	Value

#	Name/Signal/Expr	Value	Plot	Save	Mar



Click on pins to save currents.

Click on wires to save voltages and frequency data.

Select: Outputs—To Be Plotted—Select On Schematic

- You **must** terminate this command by pressing the **Esc** key.
- Save and load data sets.
  An optional command saves quantities associated with all wires, all pins, or both.

Probing the Schematic to Save Output Data

STEP 7.

Specifying Outputs

Use **Outputs—To Be Plotted—Select On Schematic**

When using the command to select a set of outputs, click on device pins to save terminal currents or wires to save node voltage or frequency data. Data will be available for plotting and analysis on nodes or terminal pins that are saved before simulation.

Select the **Edit—Select—Filter** command in the schematic window to display the *Schematic Selection Filter* to choose which objects, such as wires and pins, are selectable.

Saving All Outputs

Save all quantities associated with wires or pins in a design.

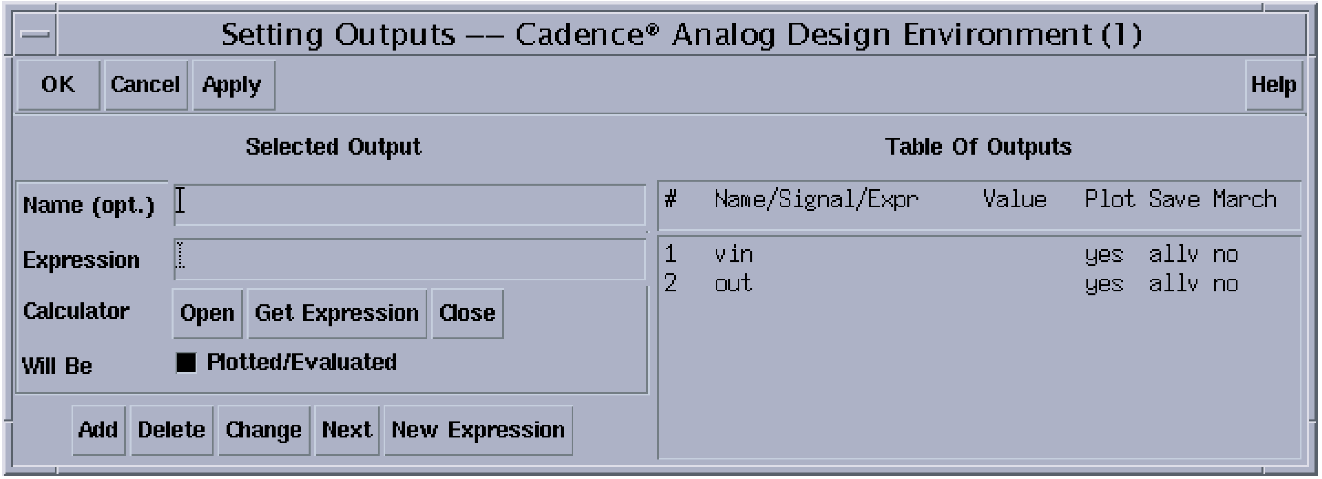
Outputs Section of Simulation Window

STEP 7.

#	Name/Signal/Expr	Value	Plot	Save	March
1	out	wave	yes	allv	yes
2	input	wave	yes	allv	no
3	VDC("/out")	-1.004m	yes		
4	phaseMargin	73.64	yes		
5	gainMargin	-16.00	yes		

Automatically Evaluated

Automatically Plotted



■ Plotted/Evaluated

```
#   Name/Signal/Expr      Value    Plot Save March
1   vin                            yes  allv no
2   out                            yes  allv no
```

Outputs Section of Simulation Window

STEP 7.

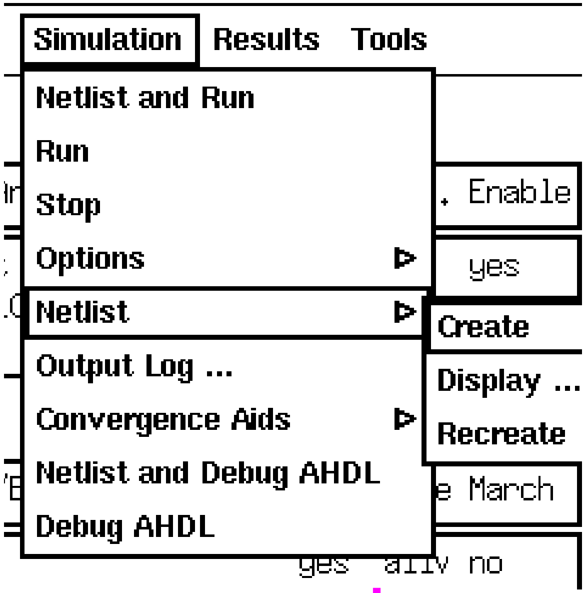
In the Outputs section of your Simulation window display the following:

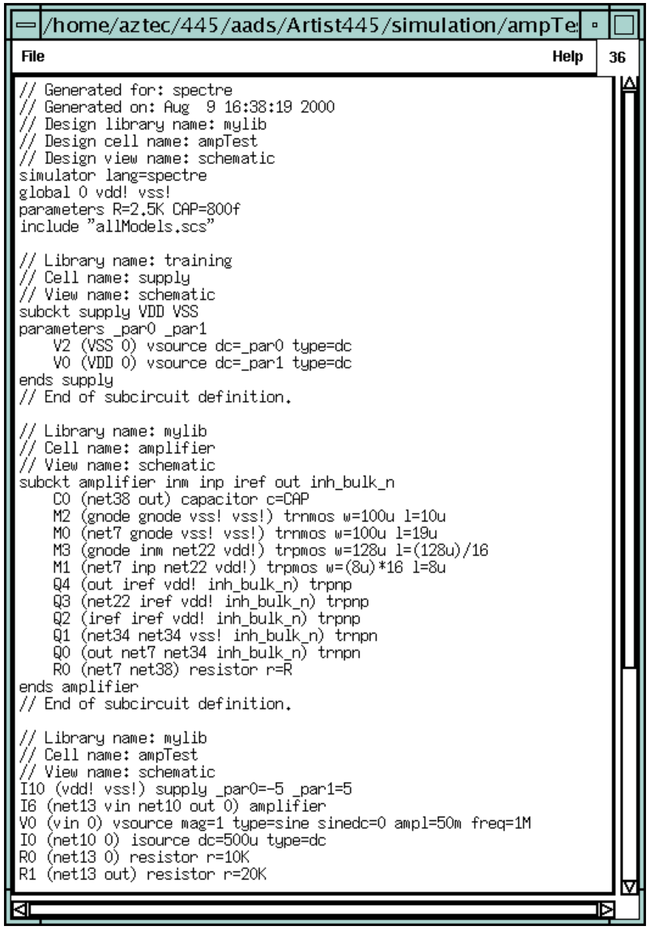
- Signal names and evaluated expression values (created by the calculator)
- The signals to plot
- The signals to save
- The signals to march (this is an option in **spectreS** only.)

Use calculator expressions to return either a waveform or numeric value.

If signals and expressions are saved prior to running the simulation, the Analog Design Environment will automatically plot the waveforms and evaluate the numeric expressions and display their values when the simulation ends.

Netlisting





```
// Generated for: spectre
// Generated on: Aug  9 16:38:19 2000
// Design library name: mylib
// Design cell name: ampTest
// Design view name: schematic
simulator lang=spectre
global 0 vdd! vss!
parameters R=2.5K CAP=800f
include "allModels.scs"

// Library name: training
// Cell name: supply
// View name: schematic
subckt supply VDD VSS
parameters _par0 _par1
    V2 (VSS 0) vsource dc=_par0 type=dc
    V0 (VDD 0) vsource dc=_par1 type=dc
ends supply
// End of subcircuit definition.

// Library name: mylib
// Cell name: amplifier
// View name: schematic
subckt amplifier inm inp iref out inh_bulk_n
    C0 (net38 out) capacitor c=CAP
    M2 (gnode gnode vss! vss!) trnmos w=100u l=10u
    M0 (net7 gnode vss! vss!) trnmos w=100u l=19u
    M3 (gnode inm net22 vdd!) trpmos w=128u l=(128u)/16
    M1 (net7 inp net22 vdd!) trpmos w=(8u)*16 l=8u
    Q4 (out iref vdd! inh_bulk_n) trpnp
    Q3 (net22 iref vdd! inh_bulk_n) trpnp
    Q2 (iref iref vdd! inh_bulk_n) trpnp
    Q1 (net34 net34 vss! inh_bulk_n) trnpn
    Q0 (out net7 net34 inh_bulk_n) trnpn
    R0 (net7 net38) resistor r=R
ends amplifier
// End of subcircuit definition.

// Library name: mylib
// Cell name: ampTest
// View name: schematic
I10 (vdd! vss!) supply _par0=-5 _par1=5
I6 (net13 vin net10 out 0) amplifier
V0 (vin 0) vsource mag=1 type=sine sinedc=0 ampl=50m freq=1M
I0 (net10 0) isource dc=500u type=dc
R0 (net13 0) resistor r=10K
R1 (net13 out) resistor r=20K
```

STEP 8.

- Netlists are hierarchical and created incrementally. Re-netlist only the modified schematics.
- Force all schematics to re-netlist with the **Simulation—Netlist—Recreate** command.

Netlisting

STEP 8.

The system automatically creates netlists when running a simulation, but a netlist can be created and viewed before simulation.

Netlisting

Use the **Simulation—Netlist—Create** command to:

- Use the Analog Design Environment to create a netlist that is simulated in standalone mode.
- Modify the netlist, perhaps to take advantage of features that the interface to your simulator does not support.
- Read the netlist before starting the simulation.

Incremental Netlisting

Incremental netlisting is faster than full hierarchical netlisting because only the schematics that have changed since the previous netlist was generated are re-netlisted. This substantially speeds up netlisting of hierarchical designs containing many small schematics. The system keeps track of the status of each schematic during and between design sessions.

Running the Simulation

STEP 8.

After the netlist has been created or recreated, the simulation is ready to run.

To run the simulation:

- Select **Simulation—Run** or
- Select the **Run** icon on the right side of the simulation window.

If preselected outputs appear in the output field of the simulation window, then the Waveform display will automatically appear when the simulation is completed.

Running the Simulation

Running Additional Simulations

The purpose of running a simulation is to verify the operation and performance of the circuit. This often requires running additional simulations. Most steps of the simulation flow have now been completed. So running additional simulations is greatly simplified.

- To make changes to the simulation, simply modify the entries to the Design Variables, Analyses, or Output fields, and then press the **Run** icon.
- To change the temperature, select **Setup—Temperature** in the menu banner, enter the new value, and the press the **Run** icon.
- If you did not edit the schematic, you do not need to netlist the circuit.
- If you did edit the schematic, you must do a Check and Save in the schematic window.
- After the Check and Save, you must select **Netlist—Recreate** within the simulation window.
- The simulation setup can also be saved for running additional simulations at a later time, or even for running simulations on similar circuits.

Running Additional Simulations

Upon completion of the first simulation, the setup has been complete. It is now very simple to change the parameters of the next simulation, or set of simulations.

Just select by:

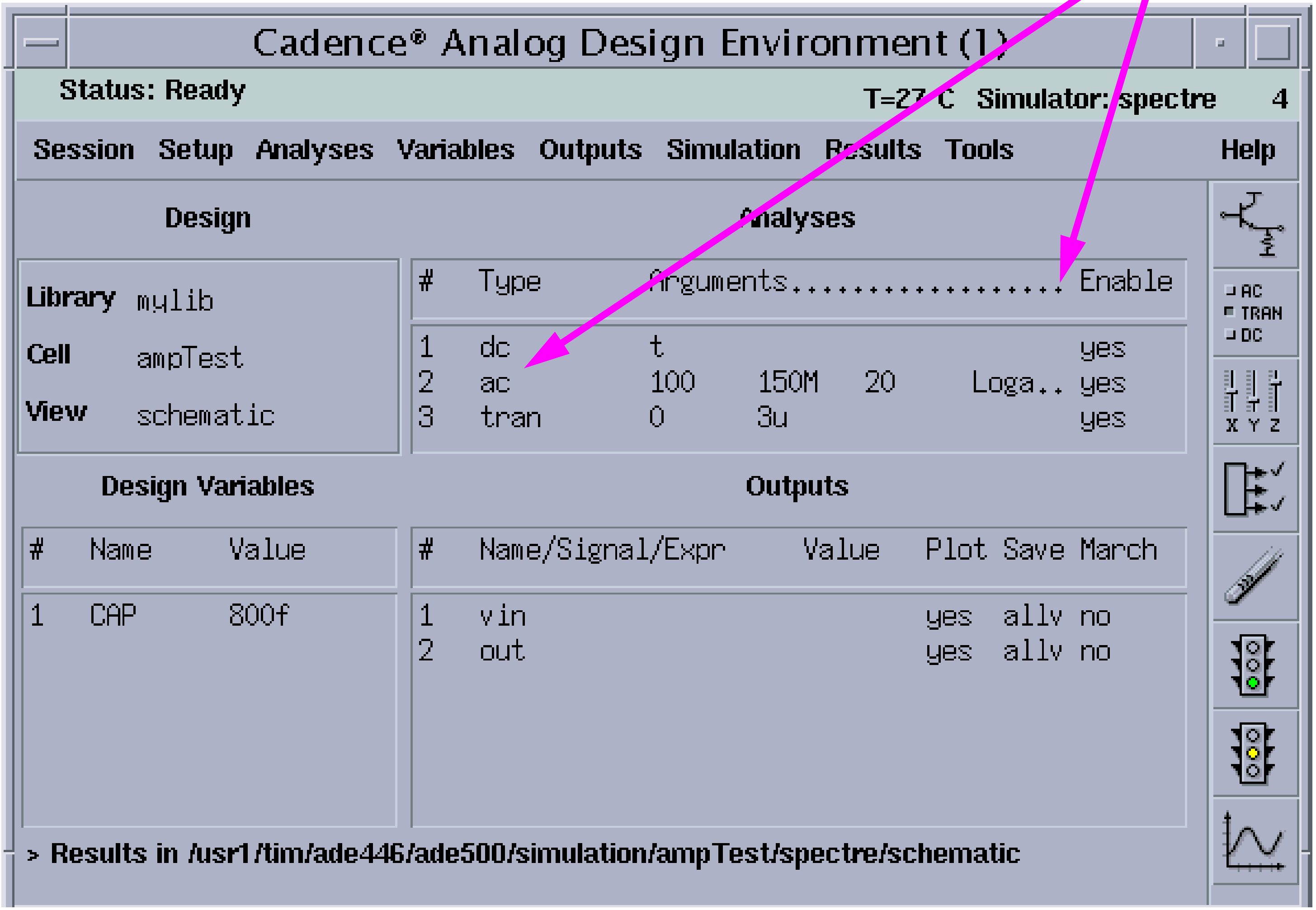
- a mouse sequence using the menu banner,
- selecting an icon on the right-hand side of the simulation window, or
- selecting the appropriate line within an editable field of the simulation window, such as the “Analyses” field.

If you make changes only to the simulation window, then you are able to run the new simulation without netlisting.

If you make changes to the schematic, then the schematic must be checked and saved, and a new netlist must be generated. However, the simulation window accesses the design database using the Design Framework. There is no requirement to open or close additional windows. A few simple clicks and the updated schematic is simulated.

Control of Analyses for Simulation

Analyses Control Field



Status: Ready T=27 C Simulator: spectre 4

Design

Library	mylib
Cell	ampTest
View	schematic

Analyses

#	Type	Arguments.................				Enable
1	dc	t				yes
2	ac	100	150M	20	Loga..	yes
3	tran	0	3u			yes

Design Variables

#	Name	Value
1	CAP	800f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	vin		yes	allv	no
2	out		yes	allv	no

> Results in /usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic

Control of Analyses for Simulation

The Analog Design Environment provides additional control for running analyses. The Analyses Control Field lets you select which analyses to complete during the next simulation run. This field displays all analyses that have been activated by using the Choosing Analyses window.

To select a specific analysis, move the mouse into the Analyses Control Field and click **left** to select the specific analysis line. The selected line is highlighted.

The menu banner of the Simulation Window provides these options:

- Select **Analyses—Delete** to remove the analysis completely!
- Select **Analyses—Enable** to include the analysis in the next simulation.
- Select **Analyses—Disable** to deactivate analysis, but not delete from Analyses field.

The disable mode means the selected analysis does not run for net simulation. However, the **setup** from **Analyses—Choose** remains. Simply select, and then enable the analysis to run.

Additional Options Using ADE

The Analog Design Environment provides additional features that simplify running additional simulations, or modify the performance of the simulation.

These options are:

- Analog Default Options
- Save State
- Load State
- Stimulus Template
- Simulation Environment Options
- Infotimes
- Captab

Additional Options Using ADE

Analog Default Options modifies the default appearance of the simulation window.

Save State is used to save the setup of the simulation window for reuse.

Load State is used the restore the setup of the simulation window.

Stimulus Template is an alternative method to provide stimulus to the circuit.

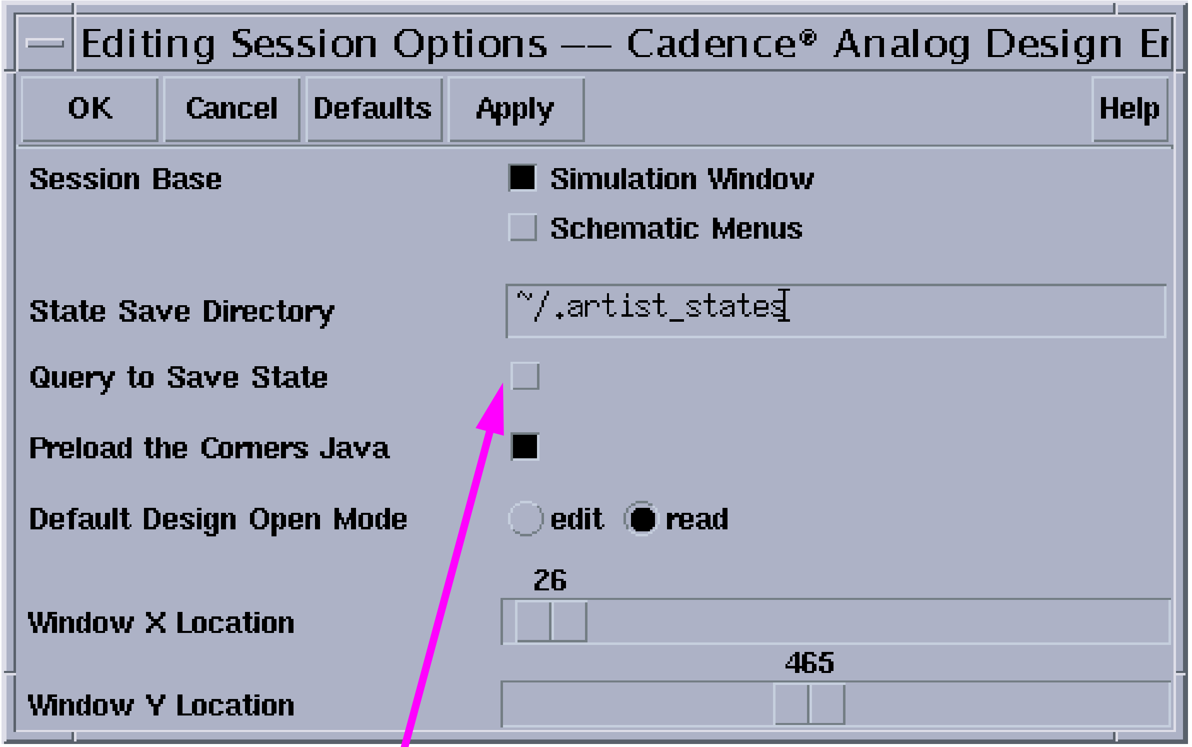
Simulation Environment Options alters the switch view list and the stop view list.

Save Options sets the default levels of signals to be saved.

Analog Default Options

In the Simulation window, select **Session — Options**.

Then an Editing Session Options window appears.



Session Base ■ Simulation Window □ Schematic Menus

State Save Directory ~/.artist_states

Query to Save State □

Preload the Corners Java ■

Default Design Open Mode ○ edit ● read

Window X Location 26

Window Y Location 465

Select this button to be queried to save the present working state.

Analog Default Options

With this command, you set up certain options to affect the way the Simulation Environment looks. Choose which method you prefer: Simulation Window-based (default) or Composer-based (using schematic window menus) or both.

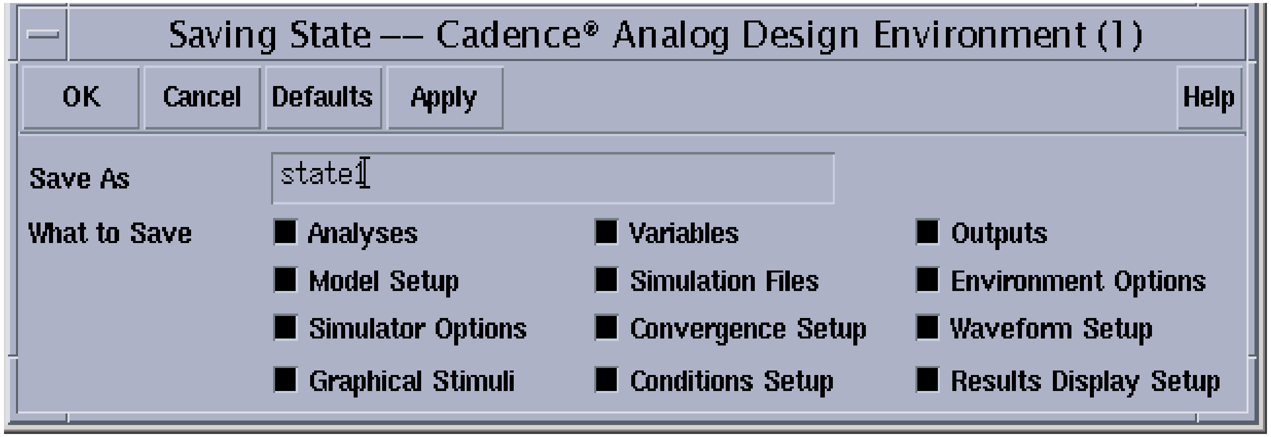
In addition, specify the directory to save state files in. State files characterize the Simulation Environment setup, including the Model Library File, convergence parameters, outputs, and design variables. Keep the default *./.artist_states* location (this will place the states in the project which launched the Cadence session), or choose one in your design database.

The option called **Preload the Corners Java** is a new feature that works with the Corners Analysis tool. It is turned on by default and can slow down your Simulation Environment start-up time. When not using the Corners tool, to turn it off or add the following statement to your *.cdsenv* file:

```
asimenv loadCorners     boolean nil
```

Also set the Default Design Open Mode and the location of the Simulation window.

Simulation States

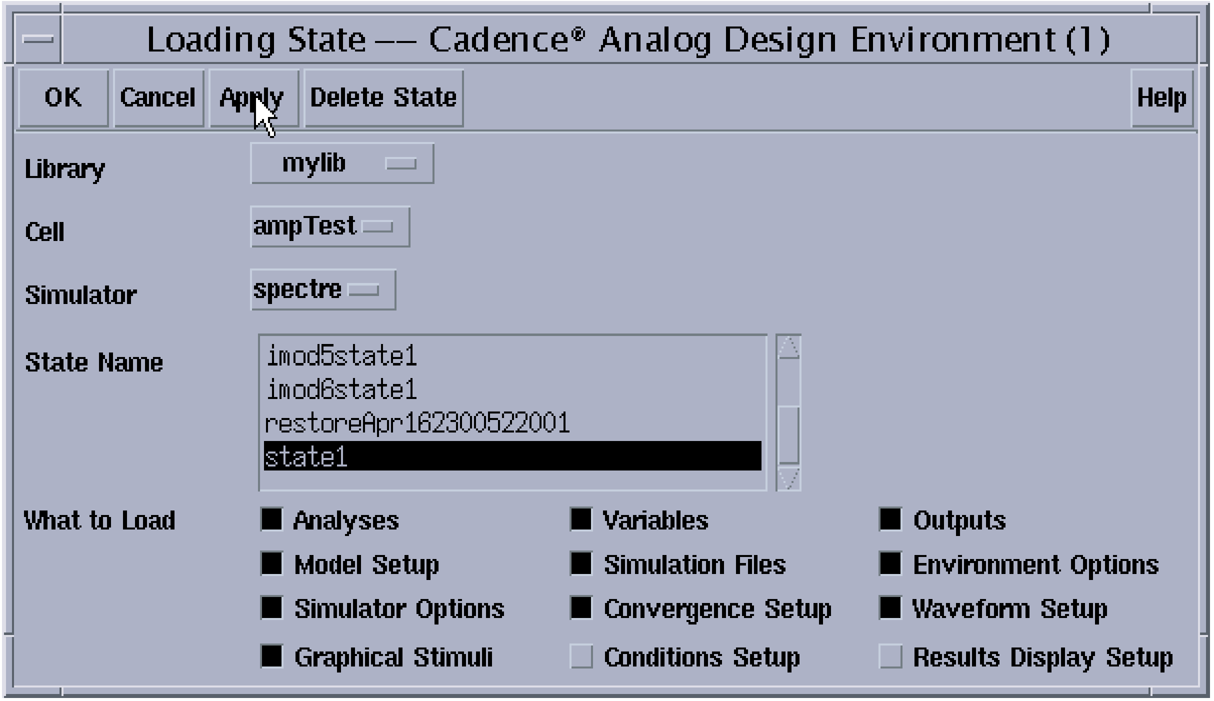


Save As: state1

What to Save:

■ Analyses ■ Variables ■ Outputs
■ Model Setup ■ Simulation Files ■ Environment Options
■ Simulator Options ■ Convergence Setup ■ Waveform Setup
■ Graphical Stimuli ■ Conditions Setup ■ Results Display Setup

Session—Save State



Library: mylib

Cell: ampTest

Simulator: spectre

State Name:

- imod5state1
- imod6state1
- restoreApr162300522001
- state1

What to Load:

■ Analyses ■ Variables ■ Outputs
■ Model Setup ■ Simulation Files ■ Environment Options
■ Simulator Options ■ Convergence Setup ■ Waveform Setup
■ Graphical Stimuli □ Conditions Setup □ Results Display Setup

Session—Load State

Simulation States

Save State

This command saves simulation states during a session. Items that can be saved include Analyses, Variables and Outputs, and the Waveform Window. Save each of these items individually or in groups. Name the file anything by typing the name in the Save As field. The default name is **state1**.

By default, the files are saved under a directory called *./.artist_states*. Change the location of this directory by specifying a new directory in the State Save Directory field in the Editing Session Options form. Access this form with the **Session—Options** command in the Simulation window.

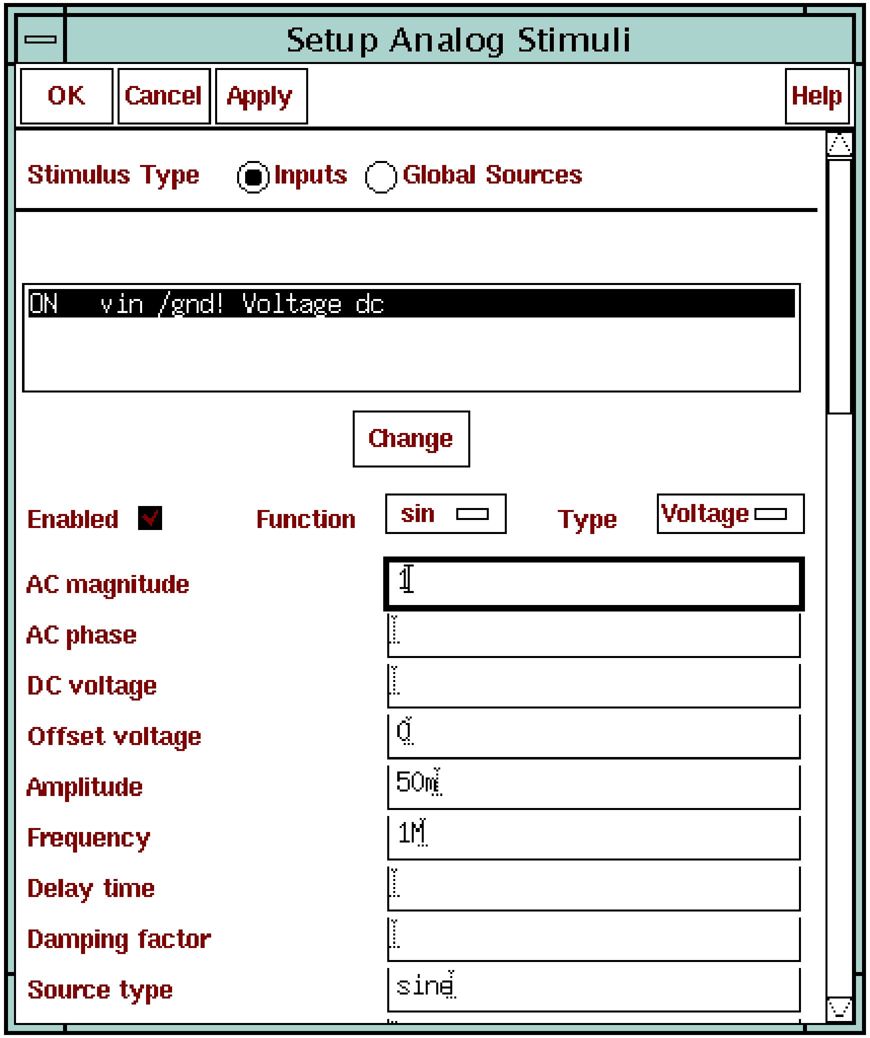
Load State

Use the **Load State** command to load saved states for a design. The cyclic fields next to Library and Cell are used to pick a particular design. The state files are simulator dependent, if Analyses are saved as part of the file. Use Outputs and Variables with any simulator. The State Name listbox will show all of the saved files for the design. Save individual objects.

This feature is particularly useful for adding items to the Outputs section. For example, the user might have specific equations that are always used for testing the AC stability of a circuit. Once these equations are defined, they can be loaded them into other designs very quickly.

Stimulus Template

Setup—Stimuli

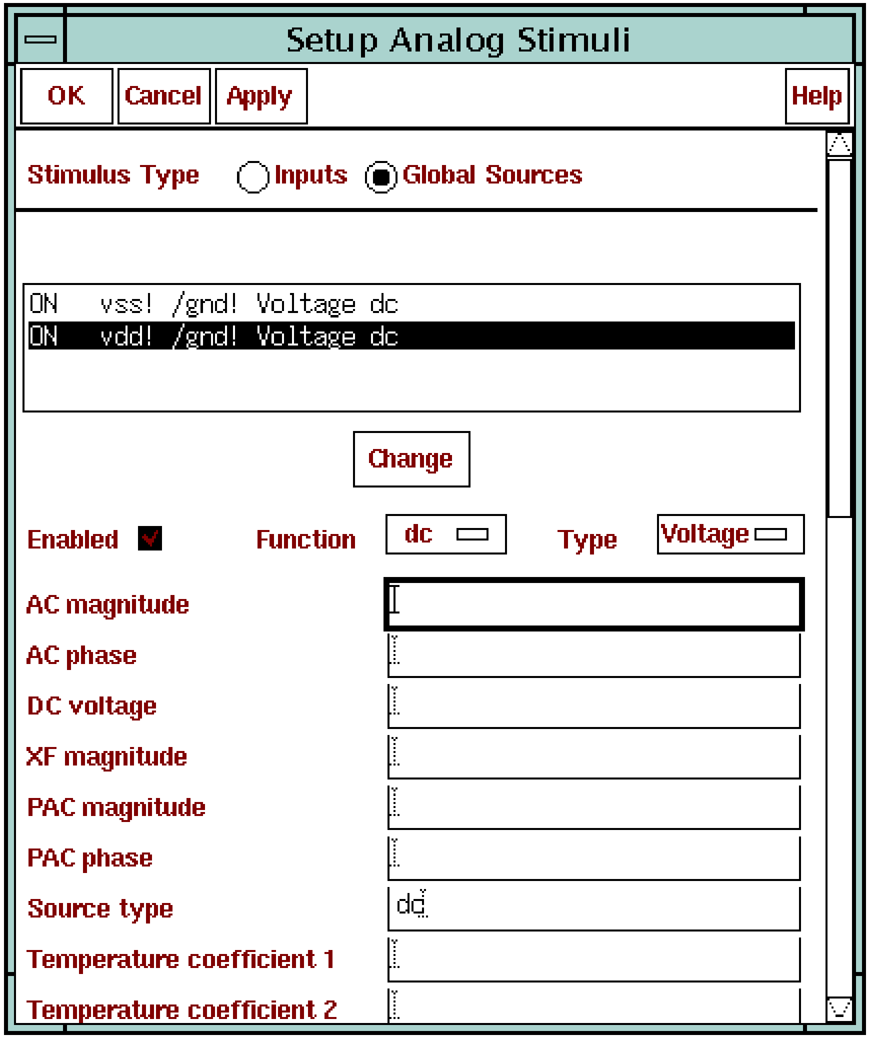


Stimulus Type ● Inputs ○ Global Sources

```
ON    vin /gnd! Voltage dc
```

Enabled ☑ Function sin Type Voltage

AC magnitude	
AC phase	
DC voltage	
Offset voltage	0
Amplitude	50m
Frequency	1M
Delay time	
Damping factor	
Source type	sine



Stimulus Type ○ Inputs ● Global Sources

```
ON    vss!  /gnd!  Voltage dc
ON    vdd!  /gnd!  Voltage dc
```

Enabled ☑ Function dc Type Voltage

AC magnitude	
AC phase	
DC voltage	
XF magnitude	
PAC magnitude	
PAC phase	
Source type	dc
Temperature coefficient 1	
Temperature coefficient 2	

Stimulus Template

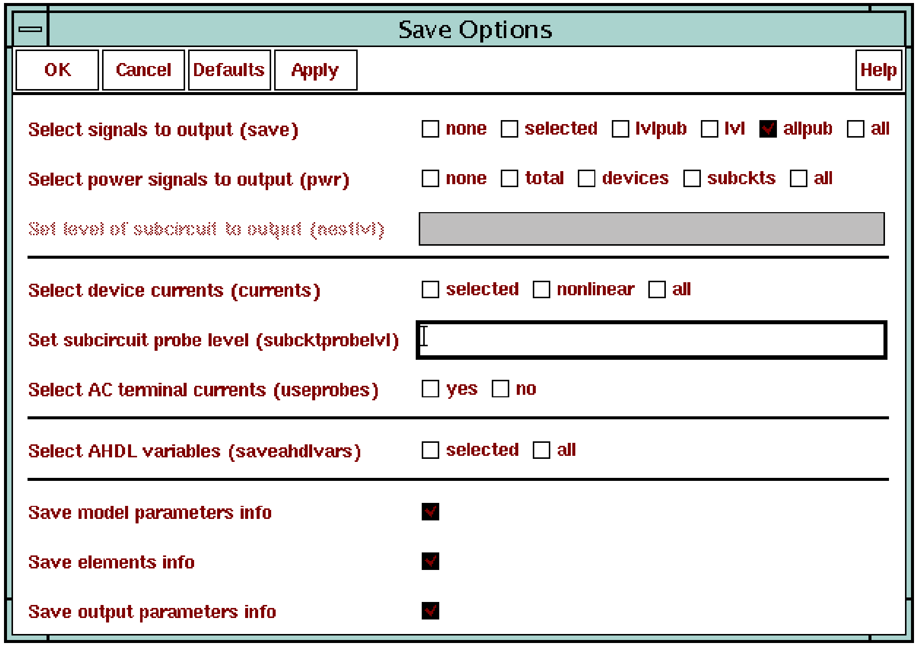
Use this graphical interface to create a stimulus file for specifying input stimuli and power supply stimuli to your design. Attach any type of source to the input pins or global pins that are in your design.

To use input stimuli, instantiate input pins into your top-level schematic for those stimuli signals. The power stimuli requires a defined global name on a signal (such as *vdd!*). Use this option to create designs that can run in multiple simulation scenarios, that do not require power sources and input stimuli that can clutter up the schematic window.

Note: To serve as optional stimuli, use both standard sources mixed with stimuli sources in your schematic. The stimulus template provides other options to creating stimuli for your circuit, depending on the analysis selected.

Save Options

Select: **Outputs—Save All**



Select signals to output (save) □ none □ selected □ lvlpub □ lvl ☑ allpub □ all

Select power signals to output (pwr) □ none □ total □ devices □ subckts □ all

Set level of subcircuit to output (nestlvl)

Select device currents (currents) □ selected □ nonlinear □ all

Set subcircuit probe level (subcktprobelvl)

Select AC terminal currents (useprobes) □ yes □ no

Select AHDL variables (saveahdlvars) □ selected □ all

Save model parameters info ☑

Save elements info ☑

Save output parameters info ☑

Setting	Description
none	Does not save any data. (Currently saves one node chosen at random.)
selected	Saves only signals selected in schematic.
lvlpub	Saves all signals that are normally useful up to *nestlvl* deep in the subcircuit hierarchy. This option is equivalent to *allpub* for subcircuits.
lvl	Saves all signals up to *nestlvl* deep in subcircuit hierarchy. Relevant to subcircuits.
allpub	Saves only signals that are normally useful.
all	Saves all signals.

Save Options

Specify which signals to save with the **save** parameter. Use the *nestlvl* parameter when saving signals in subcircuits. (Set **save** to *lvl* or *lvlpub*.)

To save power dissipated on a circuit, subcircuit, or device, use the pwr parameter. Power is calculated only during DC and transient analyses. The results are saved as a waveform, representing the instantaneous power dissipated in the circuit, subcircuit, or device.

The *nestlvl* parameter specifies how many levels deep into the subcircuit hierarchy to save signals. The default setting for *nestlvl* is infinity, which saves all levels.

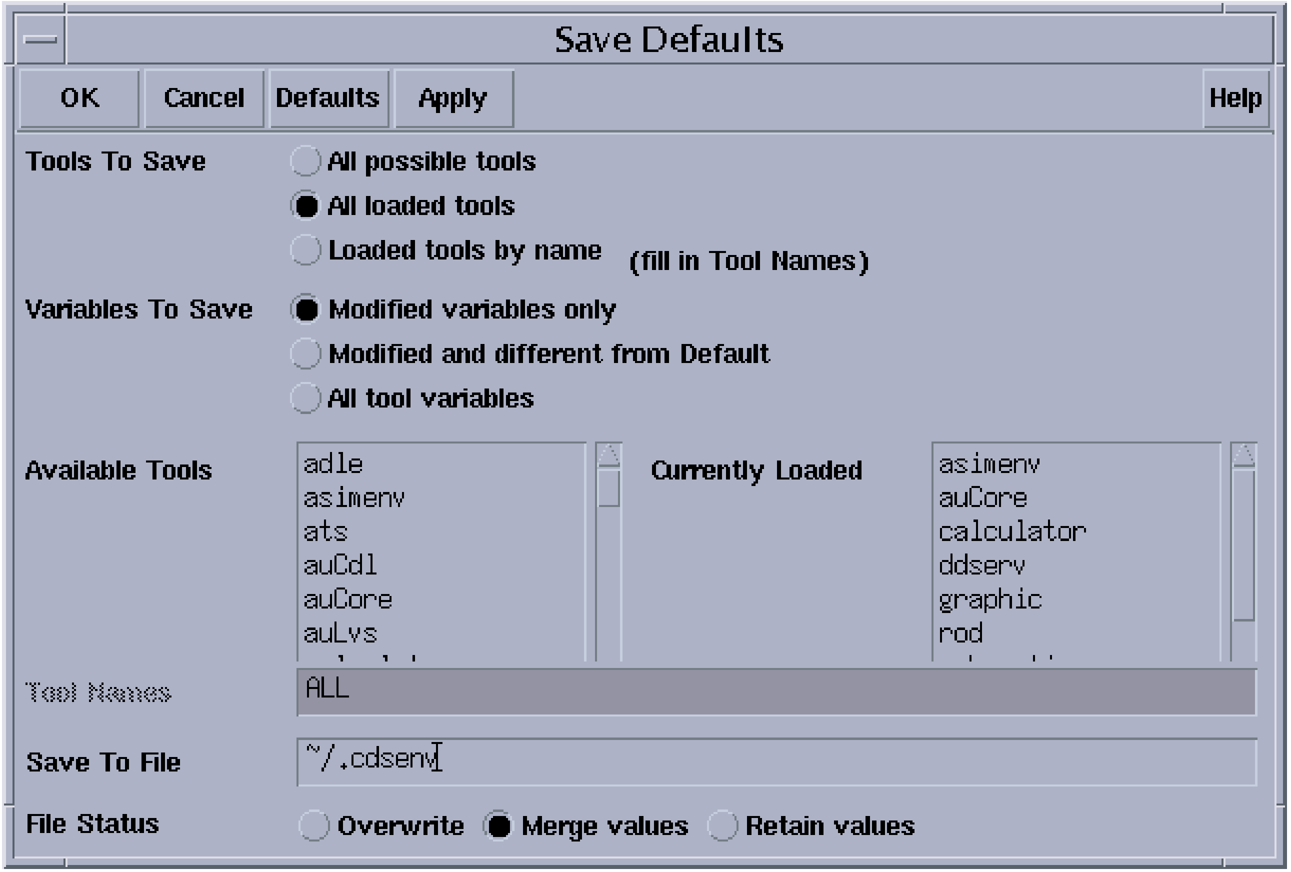
The **currents** parameter of the options statement computes and saves terminal currents. The **selected** parameter saves only currents select. The **nonlinear** parameter saves all terminal currents for nonlinear devices, naturally computed branch currents, and currents specified with save statements. The **all** parameter saves all currents. The **nonlinear** and **all** parameters can significantly increase simulation time.

Use the *subcktprobelvl* parameter to control the calculation of terminal currents for subcircuits. Current probes are added to the terminals of each subcircuit, up to *subcktprobelvl* deep. Specify all currents to be calculated with current probes by setting *userprobe* to **y**es. To save ahdl variables in ahdl instances, set *saveahdlvars* to **selected** or **all**.

Sometimes there is a need to set a large number of current probes. This could happen, for example, if one needs to save a number of ACs. (Current probes can find such small signal currents when they are not normally computed.) Specify that all currents be calculated with current probes by placing *useprobes=yes* in an options statement.

To save all the ahdl variables belonging to all the ahdl instances in the design, set the *saveahdlvars* option to all using a Spectre options command. For example: *Saveahdl* options *saveahdlvars=all.*

Save Defaults and Save Session



Tools To Save ○ All possible tools ● All loaded tools ○ Loaded tools by name (fill in Tool Names)

Variables To Save ● Modified variables only ○ Modified and different from Default ○ All tool variables

Available Tools: adle, asimenv, ats, auCdl, auCore, auLvs

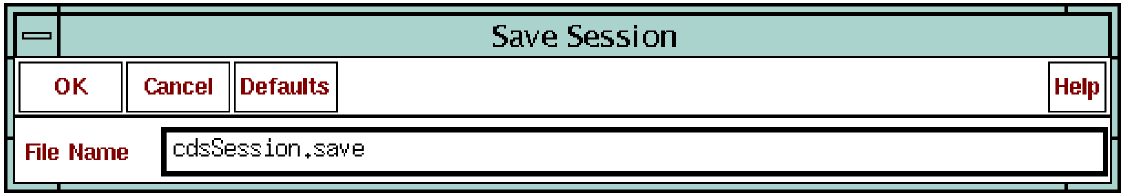
Currently Loaded: asimenv, auCore, calculator, ddserv, graphic, rod

Tool Names: ALL

Save To File: ~/.cdsenv

File Status ○ Overwrite ● Merge values ○ Retain values

In the CIW, select **Options—Save Defaults**.



File Name: cdsSession.save

In the CIW, select **Options—Save Session**.

Save Defaults and Save Session

The *.cdsenv* file contains default information for all of the tools in the executable. In the Analog Design Environment, the information that you set in the Editing Session Options form and the Setting Plotting Options form is saved in the *.cdsenv* file. Additionally, the Simulator and Project Directory are also saved.

File Status

- Overwrite: Stores the values entered in the Save Defaults form by overwriting the previous *.cdsenv* file in pointed to in the path.
- Merge values: Stores the modified values entered in the Save Defaults form in the previous *.cdsenv* file but does not delete pre-existing, unmodified values.
- Retain values: Stores the values entered in the Save Defaults form by creating another file. It is necessary to enter the name of the file in the Save to File field.

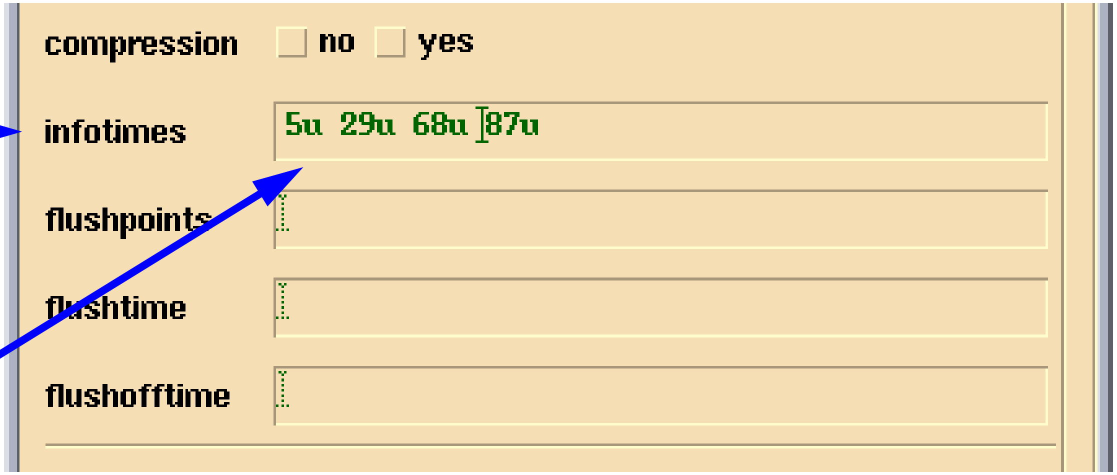
Save Session

When starting the Design Framework II environment with a *-restore <sessionFileName>* option, the specified file is loaded. Windows that were active at the time of the session save will be restored, as well as each open form. Also restore sessions by typing *load* (“*<sessionFileName>”)* in the CIW. Use caution when loading a session file in the CIW, because SKILL commands in the file might interfere with the current design session. The restored session will be added to the existing session, but no existing windows will close.

Infotimes

Infotimes is a transient analysis option to display transient operating point information.

1. From the Simulation Window, select **Analyses—Choose.**
2. Select the **tran** button in the Choosing Analyses form.
3. Select **Options** at the bottom of the Choosing Analyses form.
4. A very long Transient Options form appears, scroll down to **infotimes**.
5. Enter infotimes, as shown.



compression □ no □ yes

infotimes 5u 29u 68u 87u

flushpoints

flushtime

flushofftime

The line on the Transient Options form for entering infotimes.

Enter your time points in this text field **in any order.**

Infotimes

Accessing the *infotimes* Text Field

Multiple values entered in this field should be separated by blank spaces. If invalid separators or non-numeric values are specified here, Spectre reports the error/warning in the simulation output window. For example, if you use “comma” between values, it works but gives you following warning:

```
Warning from spectre during circuit read-in.
"input.scs" 75: Use of the comma character in lists will not be
supported in future releases.
```

The infotimes netlist statements

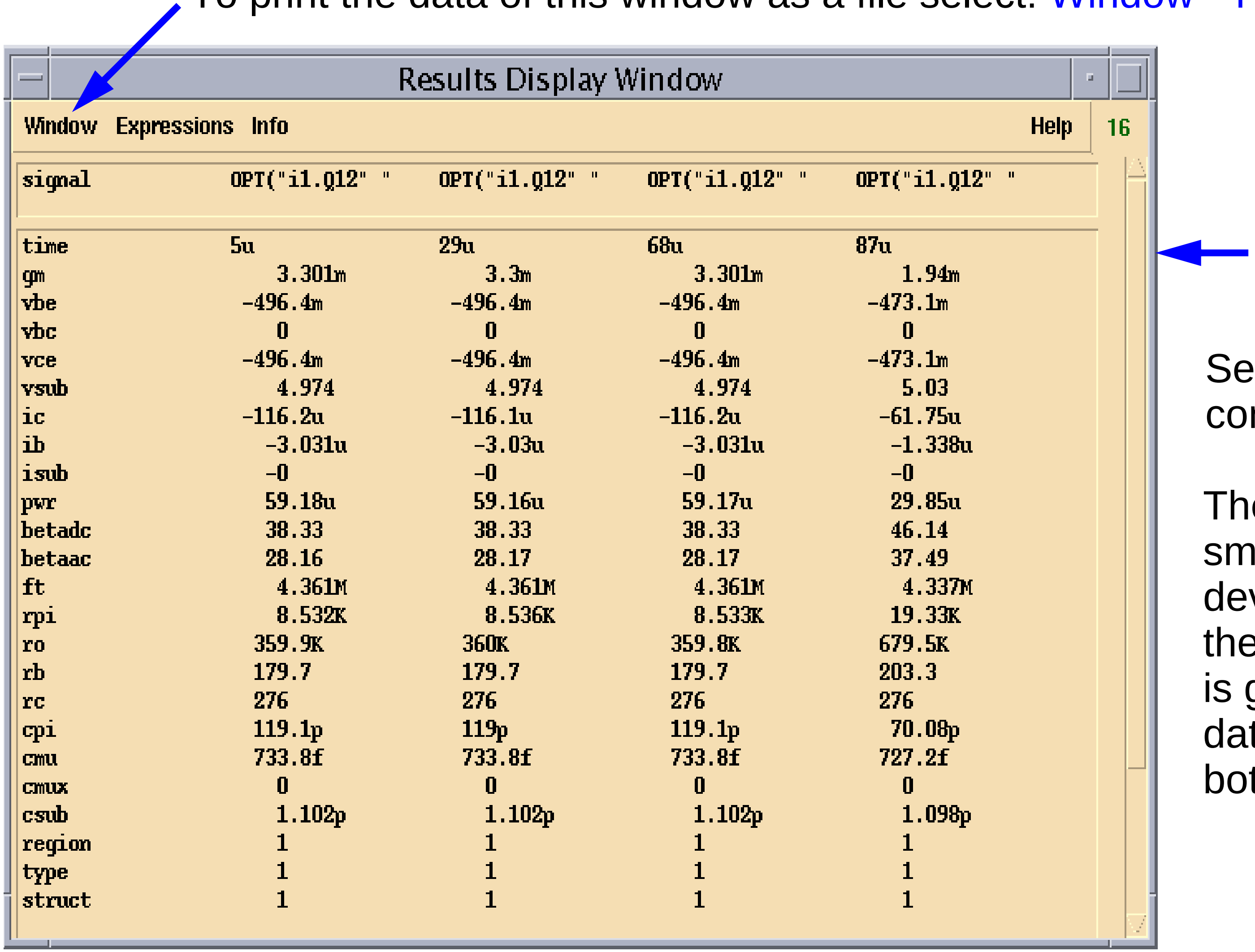
Netlist from ADE:

```
tran tran stop=100u write="spectre.ic" writefinal="spectre.fc" \
    annotate=status infotimes=[5u 29u 68u 87u] maxiters=5 \
    infoname=tran_Info
tran_Info info what=oppoint where=rawfile
```

Infotimes Results

Select **Results—Print—Transient Operating Points**, then click components on the schematic.

To print the data of this window as a file select: Window—Print



```
signal          OPT("i1.Q12" "   OPT("i1.Q12" "   OPT("i1.Q12" "   OPT("i1.Q12" "

time            5u                29u               68u               87u
gm                  3.301m            3.3m              3.301m            1.94m
vbe              -496.4m           -496.4m           -496.4m           -473.1m
vbc                 0                 0                 0                 0
vce              -496.4m           -496.4m           -496.4m           -473.1m
vsub                4.974             4.974             4.974             5.03
ic               -116.2u           -116.1u           -116.2u            -61.75u
ib                 -3.031u           -3.03u            -3.031u           -1.338u
isub               -0                -0                -0                -0
pwr                59.18u            59.16u            59.17u            29.85u
betadc             38.33             38.33             38.33             46.14
betaac             28.16             28.17             28.17             37.49
ft                  4.361M            4.361M            4.361M            4.337M
rpi                 8.532K            8.536K            8.533K            19.33K
ro               359.9K            360K              359.8K            679.5K
rb               179.7             179.7             179.7             203.3
rc               276               276               276               276
cpi              119.1p            119p              119.1p             70.08p
cmu              733.8f            733.8f            733.8f            727.2f
cmux                0                 0                 0                 0
csub                1.102p            1.102p            1.102p            1.098p
region              1                 1                 1                 1
type                1                 1                 1                 1
struct              1                 1                 1                 1
```

Scroll bar

Select additional components!

The scroll bar becomes smaller as additional devices are selected on the schematic. The file is getting larger, but the data is added to the bottom of the file.

Infotimes Results

- Once infotimes values are specified, netlist and run the simulation.
- When the simulation has been completed, select:

  Results—Print—Transient Operating Point
- A Results Display window appears. If you have not selected a component, the window will be empty. Select a component in the schematic.
- The operating point information appears in the window.
- Select another component, the data is added to the bottom. You may not see it; however, the scroll bar on the window gets smaller.
- To print the file, select **Window—Print** in the Results Display Window.

Captab

- Transient analysis option or dc analysis option
- Provides a table of node capacitances at specified times
- Has three node detail options: **node**, **nodetoground**, and **nodetonode**
- Has a threshold feature; default is 0.0F
- Similar to the CAPTAB option in HSPICE
- Used with infotimes in the transient analysis options form
- Simple to use

Captab

Captab provides a tabulation of node and device capacitance, either at the dc operating point or at the specified infotimes. The tabulation appears in the simulation output log file.

The CAPTAB parameters

detail = node

Provides details of the capacitance. Possible values are node, nodetoground, or nodetonode.

sort=name

How to sort the capacitance table. Possible values are name or value. If sort-by-value is selected, then the table will be sorted in a descending order of the *total node capacitance*. (The rows with the same “From_Node” will remain together.)

If sort-by-name is selected, then the table will be sorted in alphabetical order according to the “From_Node:To_Node” column.

threshold=0 F

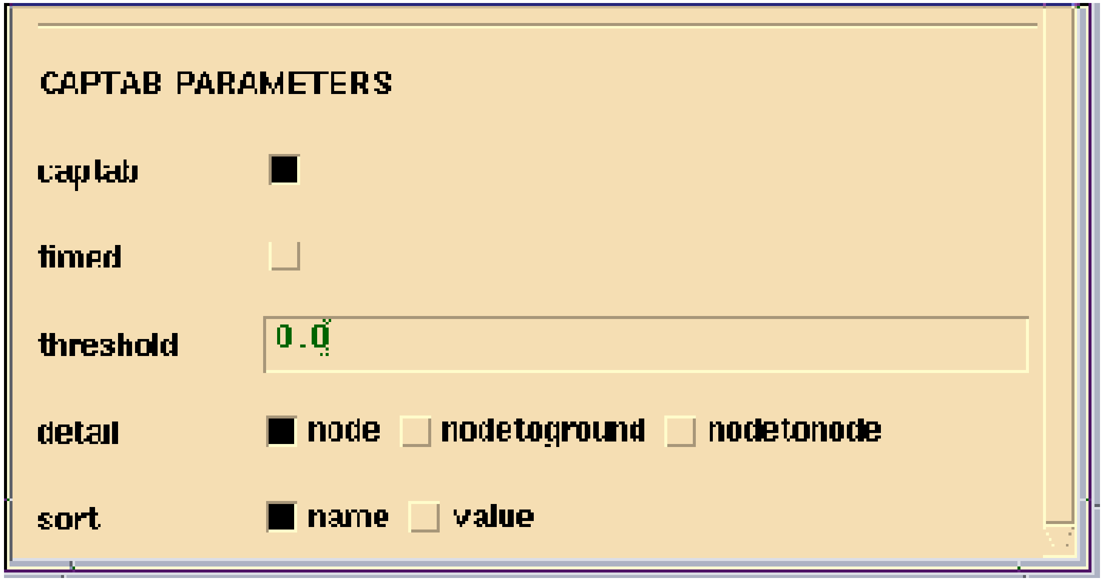
Threshold capacitance value for printing. This feature allows you to specify the threshold capacitance value. The nodes for which the *total node capacitance* is below the threshold value will not be included in the output.

Selecting the captab Option from ADE

DC and Transient Analysis has CAPTAB PARAMETERS option.

For transient analysis:

1. Select **Analyses—Choose**.
2. Select the **tran** button in the Choosing Analyses form.
3. Select **Options** at the bottom of the Choosing Analyses form.
4. On the Transient Options form scroll down to infotimes.
5. Enter infotimes (see page 3-51).
6. Scroll down to CAPTAB PARAMETERS at the bottom of Transient Options form.
7. Enter CAPTAB selections.



CAPTAB PARAMETERS

captab ■

timed □

threshold 0.0

detail ■ node □ nodetoground □ nodetonode

sort ■ name □ value

Selecting the captab Option from ADE

Reminder to Terminate Select "Outputs..."

When doing the lab activities, it is extremely important to remember to terminate the command **Outputs—To be Plotted—Select On Schematic.**

This command allows you to select wires and terminals for plotting. The command continues to select wires and terminals until it is terminated by pressing the **Esc** key.

A common error when using the simulation environment is to continue work without terminating this command. The user will move the mouse to select Netlist and Run, or other commands in the menu banner. Then the user attempts to change a component parameter on a symbol in the schematic. The simulation environment responds by selecting all terminals of the component to be plotted and these appear in the "Outputs" field of the simulation window.

Now the user must unselect the terminals and delete the entries in "Outputs" field.

Reminder to Terminate Select “Outputs ...”

When using the Outputs—To Be Plotted—Select On Schematic command, it is important to terminate this command. It is terminated using the Esc key (also known as the escape key).

Important

If you forget to terminate (a common mistake) and later attempt to select a component on the schematic for edit, all terminals of the component will be selected for plotting and appear in the “Outputs” field the simulation window. At this point it is easy to recover. Select the component again. Fortunately, the Outputs—To Be Plotted—Select On Schematic command is a toggle command. The second mouse click is an unselect. The terminals of the component are unselected and also removed from the “Outputs” field. Now press the Esc key.

Labs

Lab 3-1 Running Simulation

Lab 3-2 Using the Stimulus Template

Lab 3-3 Transient Operating Point Analysis, “infotimes”

Lab 3-4 Captab

Labs

Module 4: Simulation Results Display Tools

Topics in this module

- Overview for using Waveform Display tools
- Viewing simulation results with the Waveform Window
- Accessing and appending data on the Waveform Window
- Accessing the Waveform Calculator
- Using subwindows
- WaveScan
- Label displays

Terms and Definitions

simulation window	The ADE user interface to control and view simulations.
analyses field	A text field in the simulation widow indicating selected analyses.
output field	A text field in the simulation widow indicating selected output.
spectre	A Cadence simulation tool for simulating analog circuits.
simulator host	Software tool such as Spectre, cdsSpice, etc. to be used for simulation.
model library	A text file having model description used by the simulator host.
stimulus template	A user interface used to establish signals used in simulation.
netlist	A textual description of a schematic used by the simulator host.
Waveform Window	A graphical interface used to plot simulation data.
direct plot	User command used for ‘special’ plots to the Waveform Window.
annotating	Process of displaying data back to another window or schematic.
label display	A label attached to a component for displaying information.
snap shot	A command to the Waveform Window to update intermediate results.

Overview of Simulation Display Tools

A useful method to evaluate a simulation is to examine and make measurements on the simulation results. Waveform display tools are used to display simulation data. Some waveform display tools and related software tools include:

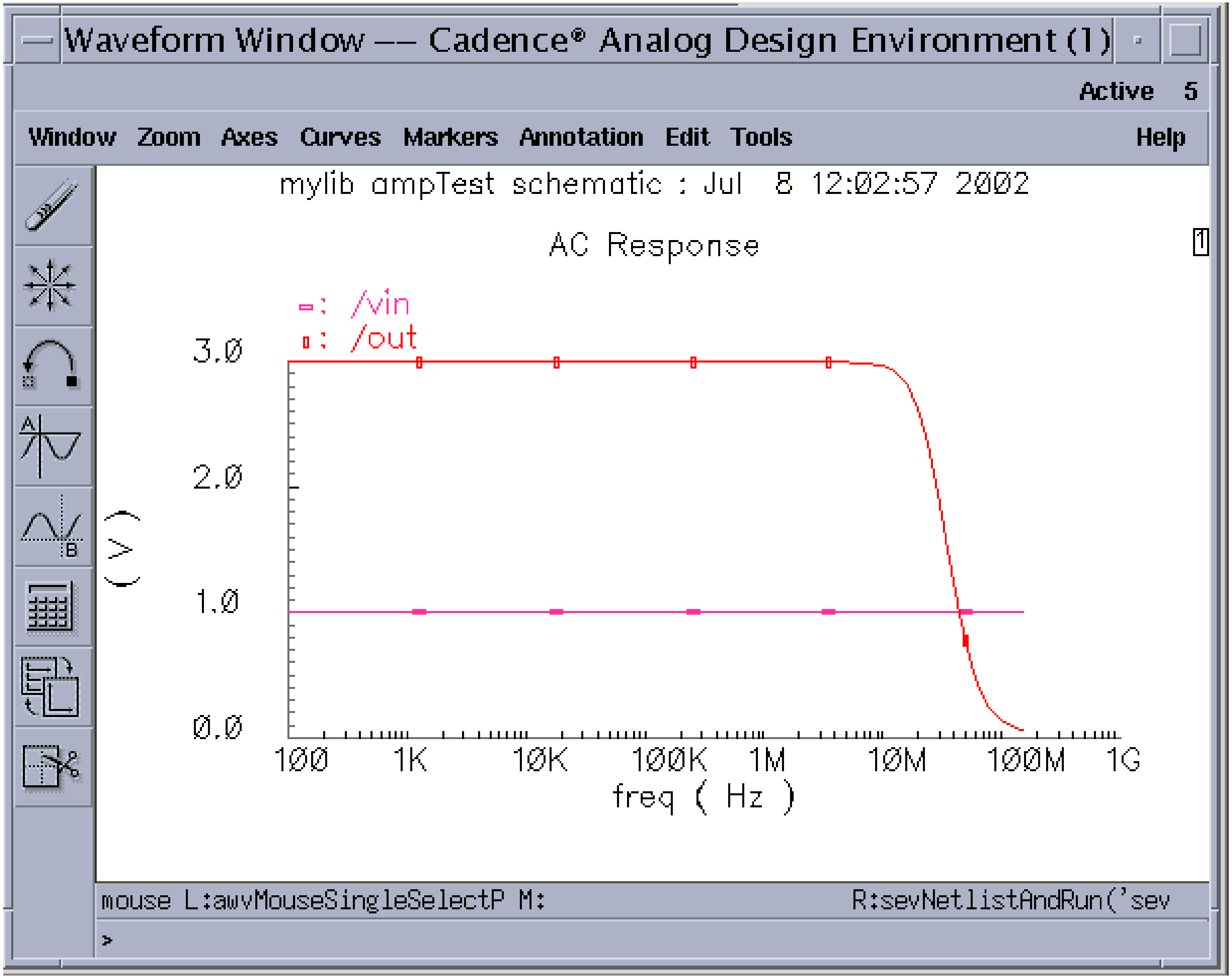
- Waveform Window
- Waveform Calculator
- Results Browser
- Snapshot Tool
- WaveScan
- Annotating Component Display

Overview of Simulation Display Tools

This module discusses the numerous methods and software tools that are used to display simulation results. Waveforms, printed tables, histograms are some methods that are used. In this module, the focus is on viewing the simulation results. As such, the Waveform Window and the WaveScan tool are discussed. The results of a dc simulation will also be backannotated to the schematic.

The Waveform Calculator and Result Browser will be discussed in the next module on Analyzing Simulation Results.

The Waveform Window


Waveform Window –– Cadence® Analog Design Environment (1)
Active 5
Window Zoom Axes Curves Markers Annotation Edit Tools Help
mylib ampTest schematic : Jul 8 12:02:57 2002
AC Response
1
▭: /vin
▯: /out
3.0
2.0
1.0
0.0
(V)
100
1K
10K
100K
1M
10M
100M
1G
freq (Hz)
A
B

mouse L:awvMouseSingleSelectP M: R:sevNetlistAndRun('sev

>

Double-click on any item in the window to display an options form that controls it.

The Waveform Window

Use the left mouse button to double-click on any item in the Waveform Window to display a form that controls it. Additionally, many of the schematic editor features, such as bindkeys, selection, deletion, and drag to move an item, are supported within the Waveform Window.

You can easily save and recall waveform setups. There is no limit on the number of curves that can be plotted, or on the number of subwindows into which a single Waveform Window can be split.

Use the annotation features from the Analog Design Environment window to display such items as the *temp = 25*, *beta = 100*, and *CAP = 500f* labels in the Waveform Window selected. In addition, display the design name, temperature, scalar outputs, design variables, and simulation date. These labels are turned on in the **Setting Plotting Options** form, which are accessed by the **Results—Printing/Plotting Options** command in the simulation window. This is discussed more in the next module, *Analyzing Simulation Results*.

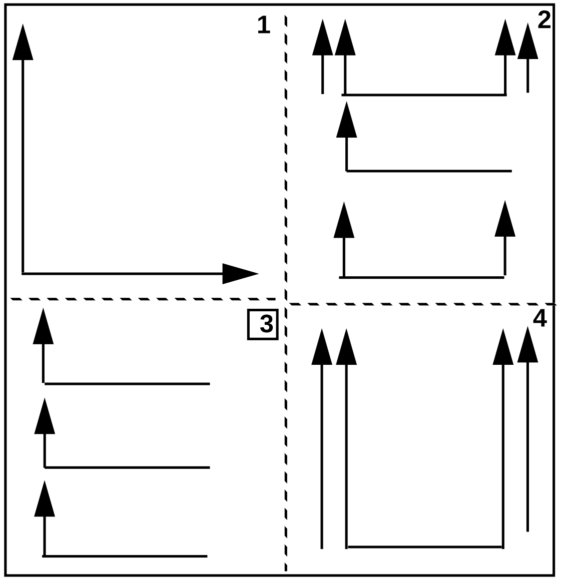
Note: The gain margin is calculated as the magnitude of the gain in dB at f0. The frequency f0 is the lowest frequency in which the phase of the gain provided is –180 degrees. For stability, the gain margin must be less than 0 dB.
gainMargin(gain) = 20 * log10(value(gain f0))
Example: Gain Margin = gainMargin(VF(“/out”))

The phase margin is calculated as the difference between the phase of the gain in degrees at f0 and at –180 degrees. The frequency f0 is the lowest frequency where the gain is 1. For stability, the phase margin must be positive.
phaseMargin(gain) = 180 + phase(value(gain f0))
Example: Phase Margin = phaseMargin(VF(“/out”))

The waveform window shows results for a closed loop simulation, or:
***VCL=Vout/Vin+Vout* (due to Vin=1V)**

The Open Loop Gain is expressed as:
***VOL=Vout/(Vinp-Vinn)*, for both Magnitude and Phase.**

Waveform Window Features



Layout of Window

Independent Subwindows

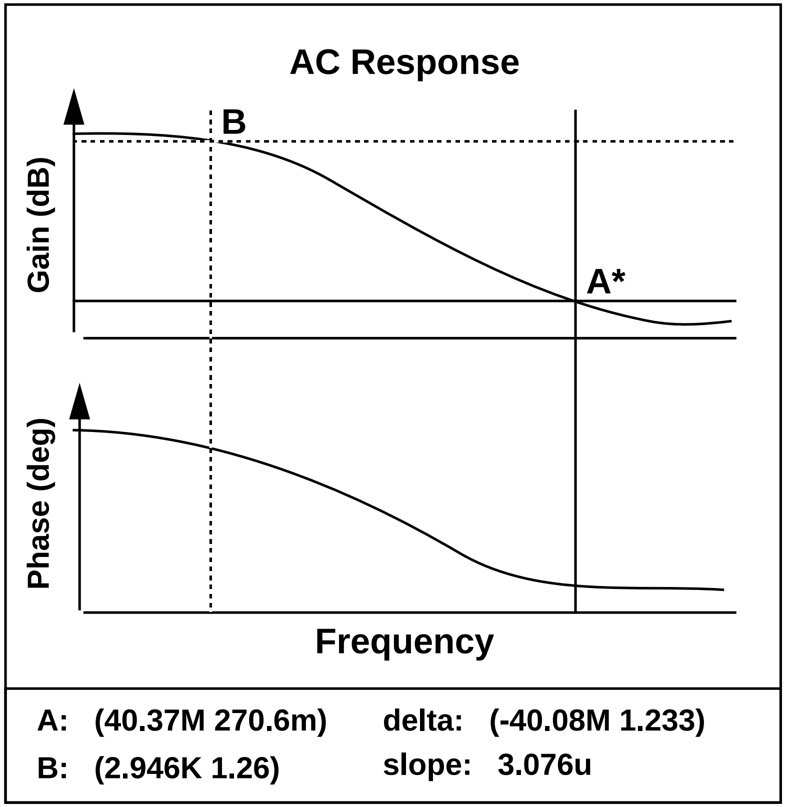
Strip Mode

Labels

Vertical and Horizontal Markers

Crosshair Markers w/ On-Screen Display

Select Letters A or B to Delete Markers



```
A:   (40.37M 270.6m)        delta:   (-40.08M 1.233)
B:   (2.946K 1.26)          slope:   3.076u
```

Waveform Window Features

Place as many as many curves on a single strip as desired, and each subwindow can use a different window setup. Additionally, a single strip or composite window can have from 1 to 4 Y-axes, with user-assigned curves on each axis.

Waveform curves can be selected and their colors and design styles can be changed with a simple options form, which modifies your Waveform Window to display X/Y Plots.

Both vertical and horizontal markers are available. In addition, crosshair markers, which display information directly on the Waveform Window, can be used for quick measurements. To delete the **A** and **B** markers, select the letter (A or B) that labels the marker and note the asterisk that appears next to it. Next, click the **Delete** icon.

Axes labels can be modified, and there is a main title along with separate subtitles that can be used on each subwindow. Add labels that use calculator expressions, which update with each successive simulation.

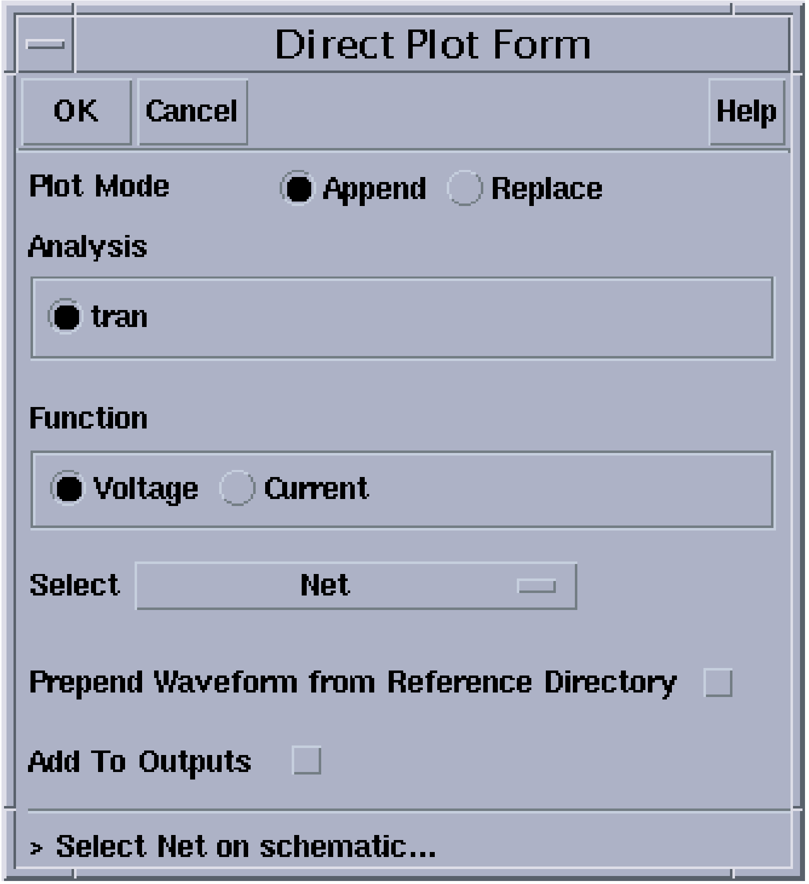
Display multiple Waveform Windows at the same time. Use one window to display results from a previous simulation run, while a new window displays the current data.

Use the Waveform Window interactively with the Calculator (discussed later).

Pan and zoom in the window, too.

Direct Plot

Main Form...
Transient Signal
Transient Minus DC
Transient Sum
Transient Difference
AC Magnitude
AC dB10
AC dB20
AC Phase
AC Magnitude & Phase
AC Difference
Equivalent Output Noise
Equivalent Input Noise
Squared Output Noise
Squared Input Noise
Noise Figure
DC



Plot Mode ● Append ○ Replace

Analysis

● tran

Function

● Voltage ○ Current

Select Net

Prepend Waveform from Reference Directory □

Add To Outputs □

> Select Net on schematic...

Main Form...

Results—Direct Plot

Direct Plot

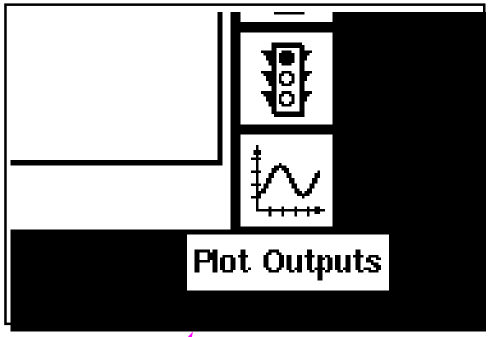
Use the Direct Plot commands to plot common waveforms quickly without the use of a Plot Set or the Calculator. There are quantities available that cover all of the analyses. The choices include Spectre RF analyses, magnitude and phase simultaneously, and transient waveforms without DC offset. The menu will only display commands that are specific to the simulator used.

Groups of signals created using these commands are not saved and cannot be loaded during a different session.

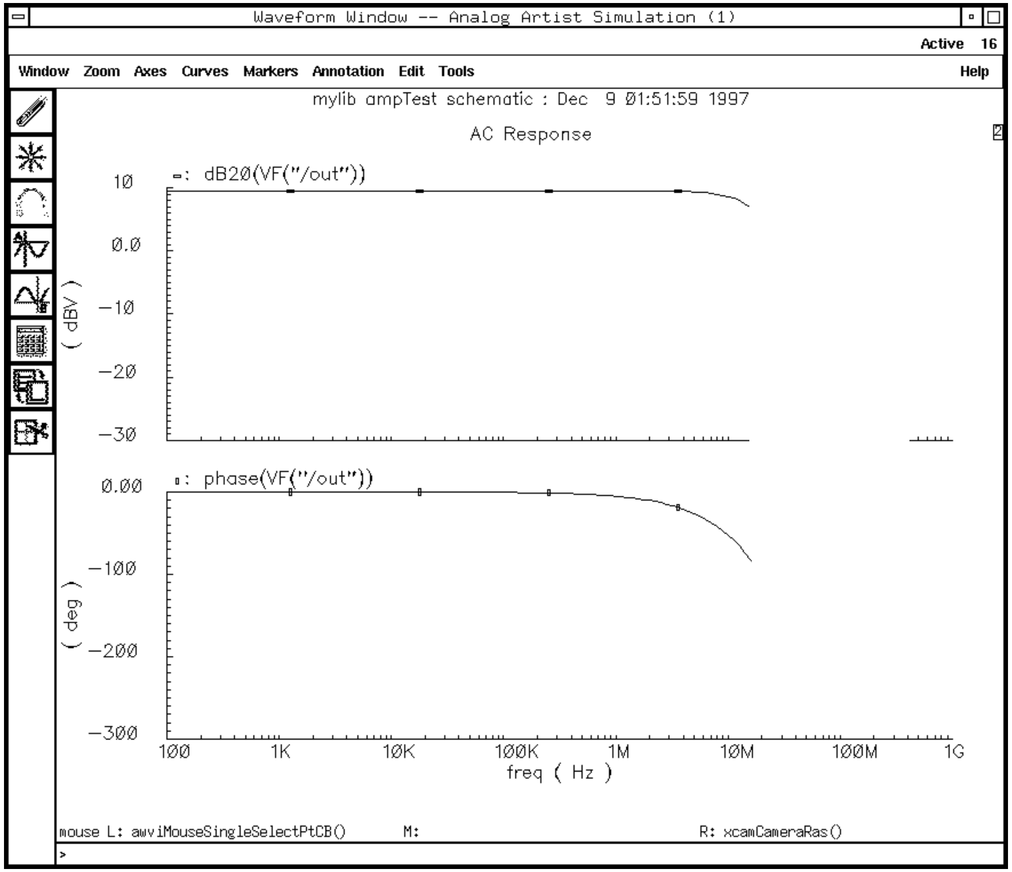
Use the **Results—Plotting Options** form to specify the signals selected as a group before plotting, or to plot after each selection. After selecting all of the signals of interest on the schematic, press the **Esc** key to end the selection and plot your results.

Also set up bindkeys to use the Direct Plot options.

Snapshot



Bottom of Simulation Window


Waveform Window -- Analog Artist Simulation (1)
Active 16
Window Zoom Axes Curves Markers Annotation Edit Tools Help
mylib ampTest schematic : Dec 9 01:51:59 1997
AC Response
dB20(VF("/out"))
10
0.0
−10
−20
−30
(dBV)
phase(VF("/out"))
0.00
−100
−200
−300
(deg)
100
1K
10K
100K
1M
10M
100M
1G
freq (Hz)
mouse L: awviMouseSingleSelectPtCB() M: R: xcamCameraRas()
>

- Set up outputs to plot in the Simulation Window before simulating.
- During simulation, click the **Plot Outputs** icon to plot the data available. This requires the simulation to be running slowly enough for this feature to work.
  Each time the icon is clicked, the Waveform Window displays the results that have accrued since the previous click of the icon.

Snapshot

Snapshot is a capability in the Analog Design Environment that looks at waveforms during simulation. Plot selected outputs while the simulation is running by clicking the Plot Outputs icon. Use this tool to discover problems with the current simulation, stop it, fix the problem, and resimulate.

In addition, monitor the progress of properly working simulations, without the need to use marching waveforms (these waveforms are not available in Spectre direct).

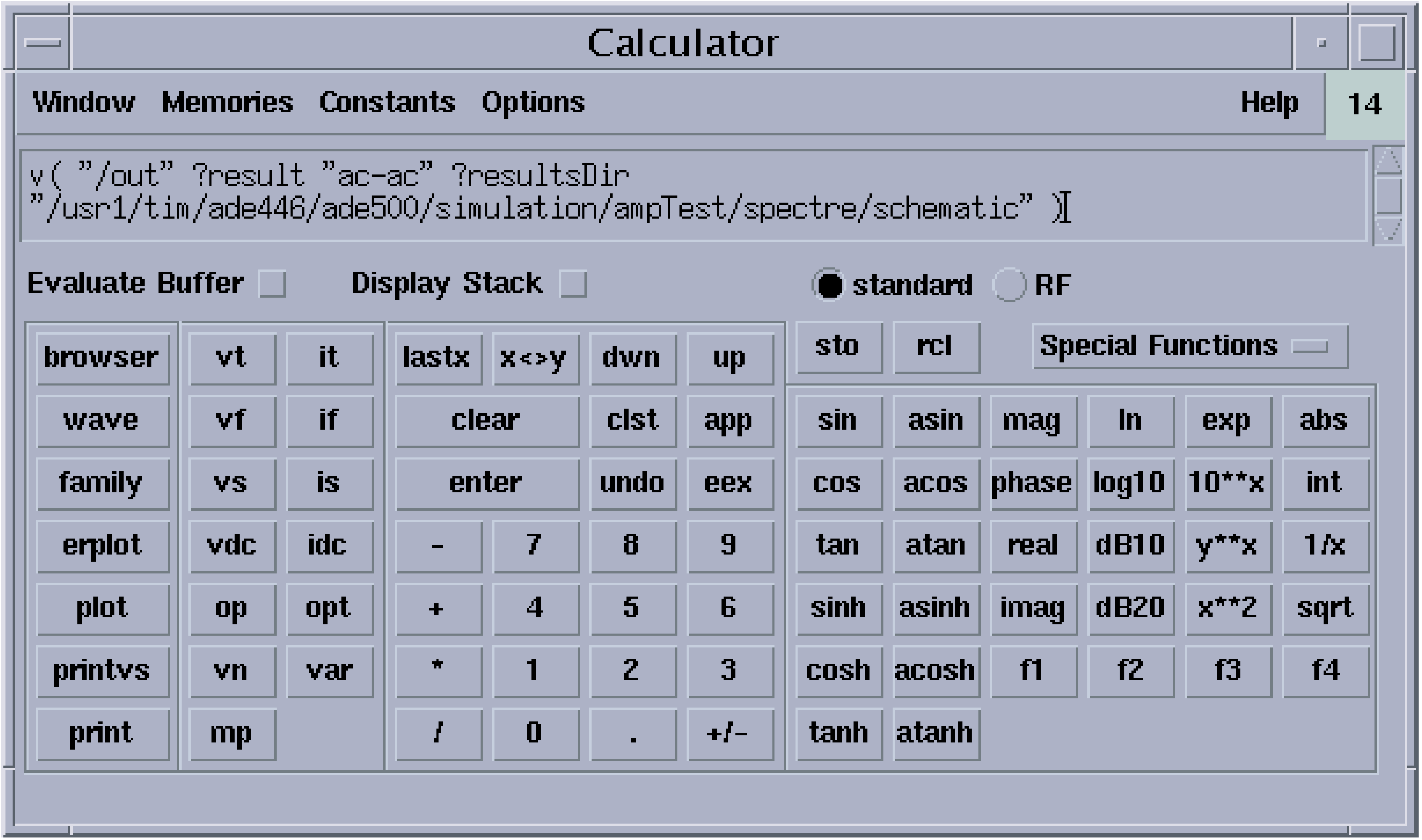
For mixed-signal simulations, analog and digital simulation data will be available at every synchronization timepoint, which will allow the Waveform tool to display analog and digital waveforms in sync.

The windows might not display some waveform data up to the current timepoint, because it is displaying intermediate simulation results while the simulation is running. The waveform reader might not be able to read the new data while the simulator is writing it to disk. As a result, the window might not display both the analog and digital signals up to the current timepoint.

Use Snapshot to plot waveforms during long transient runs. The waveforms **must** be set up in the Outputs section of the simulation environment window to use this feature. Snapshot can also serve as a workaround instead of using the “checkpoint and restart” functionality, because the simulation continues to run after taking a “snapshot” of your current results.

Waveform Calculator

The Waveform Calculator is integrated into the Analog Design Environment and is used to analyze simulation results. It is discussed in Chapter 5.



```
v( ”/out” ?result ”ac-ac” ?resultsDir
”/usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic” )
```

Evaluate Buffer □ Display Stack □ ● standard ○ RF

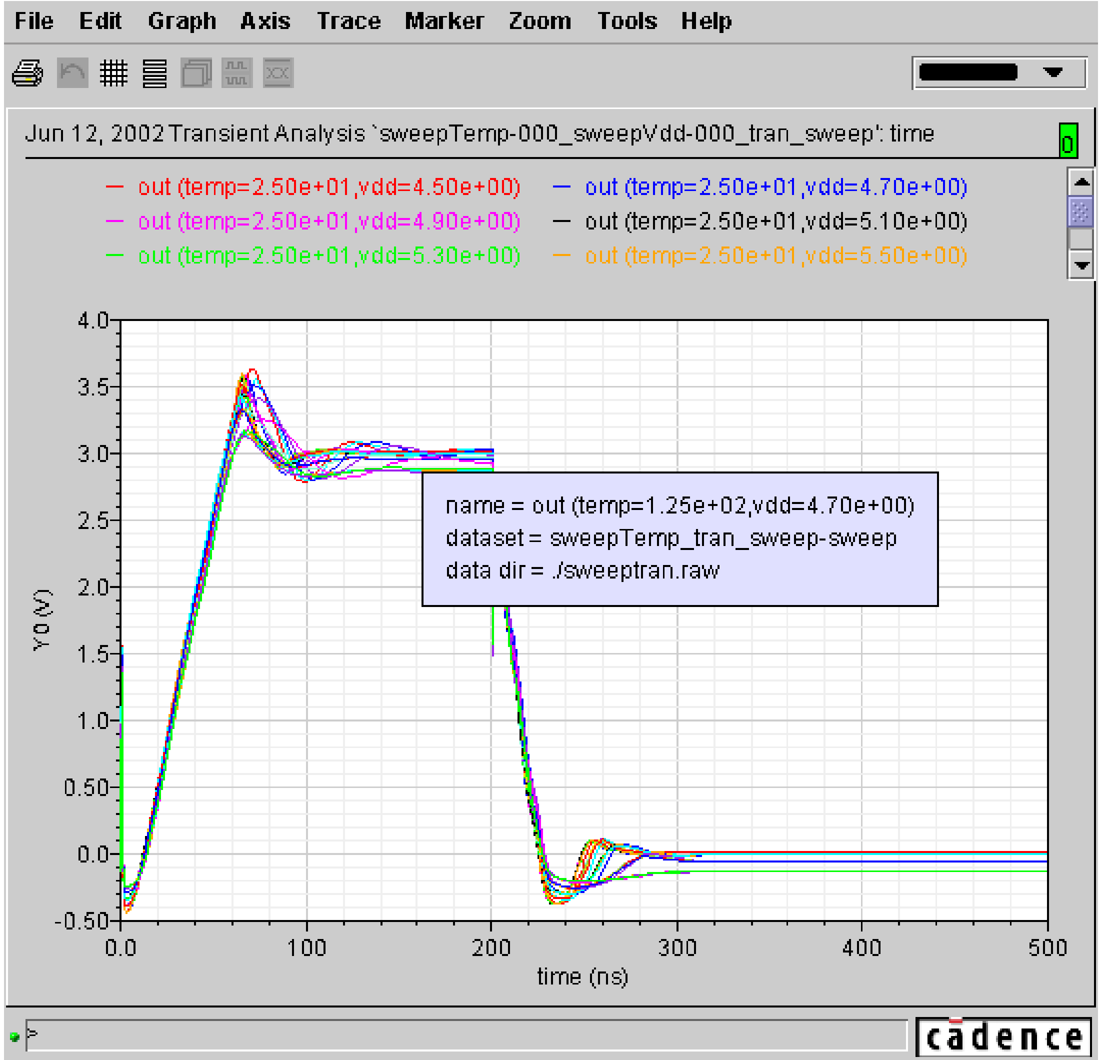
Waveform Calculator

The Waveform Calculator Too is integrated into the Analog Design Environment. It works directly with the Waveform Window to display results. The Calculator displays numerical results of the simulation data. It also processes waveform data obtained from the schematic, the Waveform Window, or the Results Browser.

The Calculator is discussed in detail in the chapter **Analyzing Simulation Results**.

WaveScan

A alternate waveform display tool is available. **WaveScan** is a new waveform display tool for viewing simulation results. It features composite plots, strip mode, subwindows, and a card view. A sample WaveScan plot is shown below.


File Edit Graph Axis Trace Marker Zoom Tools Help
Jun 12, 2002 Transient Analysis `sweepTemp-000_sweepVdd-000_tran_sweep': time
0
out (temp=2.50e+01,vdd=4.50e+00)
out (temp=2.50e+01,vdd=4.70e+00)
out (temp=2.50e+01,vdd=4.90e+00)
out (temp=2.50e+01,vdd=5.10e+00)
out (temp=2.50e+01,vdd=5.30e+00)
out (temp=2.50e+01,vdd=5.50e+00)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.50
0.0
-0.50
Y0 (V)
0.0
100
200
300
400
500
time (ns)
cadence

name = out (temp=1.25e+02,vdd=4.70e+00)
dataset = sweepTemp_tran_sweep-sweep
data dir = ./sweeptran.raw

1. New waveform display tool
2. Standalone!
3. Separate Results Browser
4. Separate Calculator

For more information on WaveScan see Appendix B.

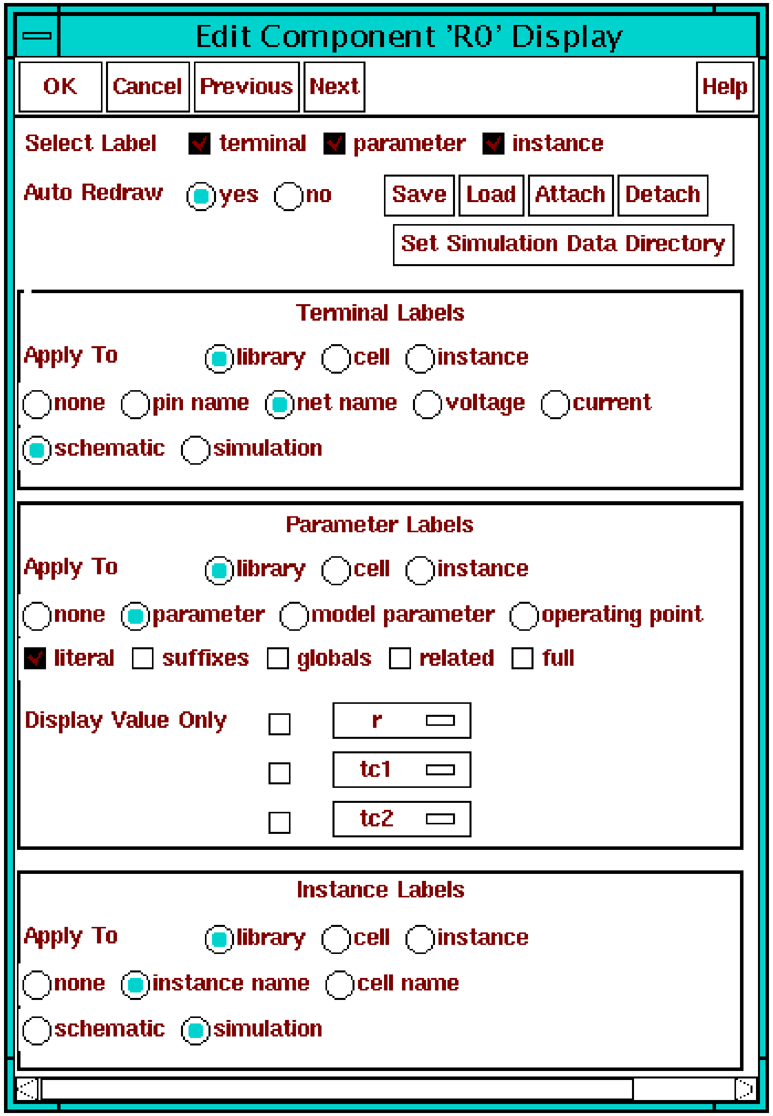
WaveScan

WaveScan is a new waveform display tool. In this release, it is a standalone tool. WaveScan uses a separate Results Browser to access simulation data and a separate Calculator to analyze the simulation data.

Appendix B of this manual provides more detailed information on using the WaveScan tools.

Controlling Schematic Label Displays

In the schematic window, select **Edit—Component Display**.



Select Label ☒ terminal ☒ parameter ☒ instance

Auto Redraw ● yes ○ no

Terminal Labels

Apply To ● library ○ cell ○ instance

○ none ○ pin name ● net name ○ voltage ○ current

● schematic ○ simulation

Parameter Labels

Apply To ● library ○ cell ○ instance

○ none ● parameter ○ model parameter ○ operating point

☒ literal □ suffixes □ globals □ related □ full

Display Value Only

□ r

□ tc1

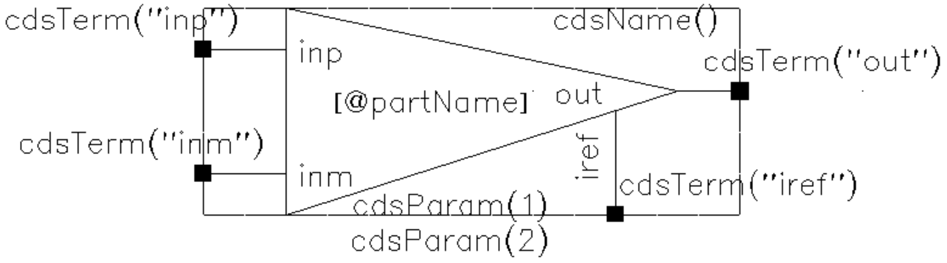
□ tc2

Instance Labels

Apply To ● library ○ cell ○ instance

○ none ● instance name ○ cell name

○ schematic ● simulation



Modifies cdsTerm ( ) labels

Modifies cdsParam ( ) labels

Modifies Instance labels

Controlling Schematic Label Displays

This form controls the label displays at different levels:

- *Instance Level*: Affects a selected instance
- *Cell Level*: Affects all instances in the cellview with the same cell name as the selected instance.
- *Library Level*: Affects all instances in the cellview from the same library as the selected instance.

Save label display changes to a file.

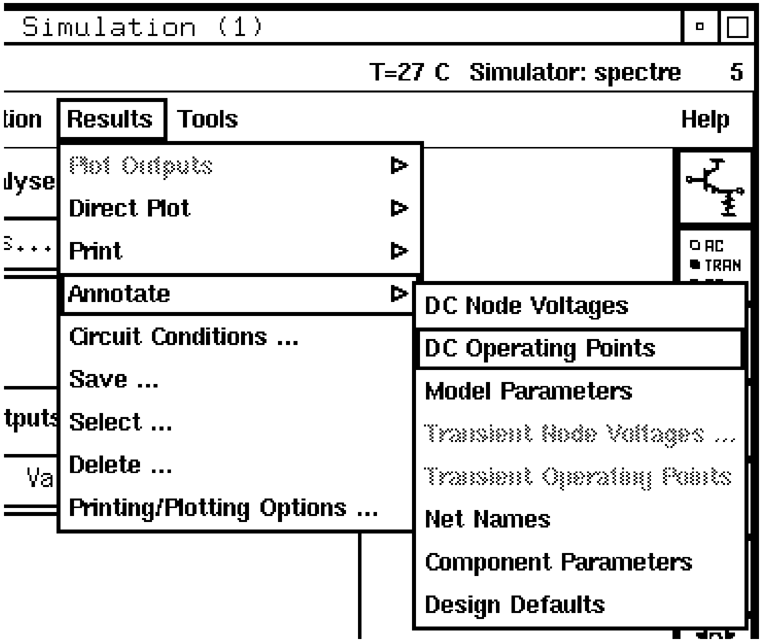
- Attach label displays to a library, so the latest settings appear when a design opens.
- Label display changes at the instance level are always saved with the design. Modify these settings later on at the instance level only.

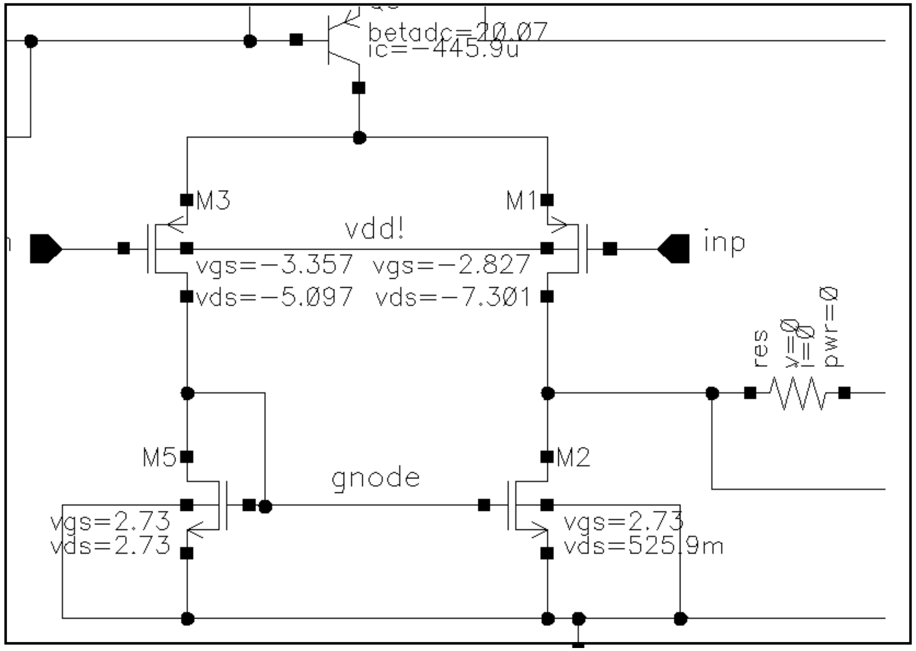
In addition, terminal and instance labels can have different values in the simulation and schematic environments. This is due to design hierarchy. For example, if an instance of the *amplifier* cell is placed in a hierarchical design, an *nmos* transistor in the *amplifier* schematic might be named *M2*. In the final netlist, the transistor might appear with a name of *xi0/m2*.

Similarly, a terminal can appear to be named *net52* in the *amplifier* schematic, but might have a name of *xi0/net52* in the final netlist.

The schematic and simulation values of terminal and instance labels can be displayed, too.

Annotating Simulation Information to the Schematic





DC Operating Points

Annotation labels appear near all components in the design window.

- Display configurations can be saved and loaded again.
- Unless explicitly saved, the database is overwritten or lost when exiting the Design Framework II session.
- The **Design Defaults** command removes annotated simulation data and restore design information on the schematic.
- Use **Results—Print** command and menus to print results to a file.

Annotating Simulation Information to the Schematic

Annotate simulation information to the schematic window with the **Results—Annotate** command and menu in the simulation window. Print the same data to a file using the **Results—Print** command and menu.

The annotation is achieved through the interpreted labels on component symbols that serve as “placeholders” for the data to be displayed. Set the visibility and type of data by using the Label Display menu. Bring up the Label Display menu with the **Edit—Label Display** command in the schematic window.

Apply changes to all cells in the design window, to all cells in the library of the selected instance, to the same cells as the selected instance, or to only the selected instance. The different types of interpreted labels are:

- *terminal* or *cdsTerm(pin_name)* labels: Display net names, pin names, node voltages, port currents and node numbers (referenced to either the design environment or simulation netlist).
- *parameter* or *cdsParam()* labels: Display component parameters, model parameters, transient data, and operating points. Hierarchical variables can be displayed literally or evaluated.
- *instance* or *cdsName(*) labels: Display instance or cell names (referenced to either the design environment or simulation netlist).

Labs

Lab 4-1 Displaying Results with the Waveform Window

Lab 4-2 Saving the Simulation Session

Lab 4-3 Displaying Interpreted Labels Near Schematic Components

Lab 4-4 Annotating Simulation Results to the Schematic Window

Labs

Lab Reference Material

When using a command such as **Outputs—To Be Plotted—Select On Schematic** or **Outputs—To Be Saved—Select On Schematic** in the simulation environment, follow the prompts in the schematic window or CIW to graphically probe the design. When finished probing all desired nodes and terminals, press the **Esc** key with the cursor in the design entry window to cancel the select function. Failure to cancel the select function explicitly before starting a different one, might temporarily disable the system.

If this condition occurs, the *Nest Limit* of the environment has been violated, and a warning appears in the CIW. Change the *Nest Limit* with the **Options—User Preferences** form through the CIW to get around this feature.

Make sure to cancel each selection or probing function using the Esc key when done.

Lab Reference Material

Module 5: Analyzing Simulation Results

Topics in this module

- The Waveform Calculator
- The Print Engine
- Storing and managing simulation results
- The Results Browser
- Conditional search and display
- The Spectre sweep feature
- Sensitivity Analysis
- Stability Analysis

Terms and Definitions

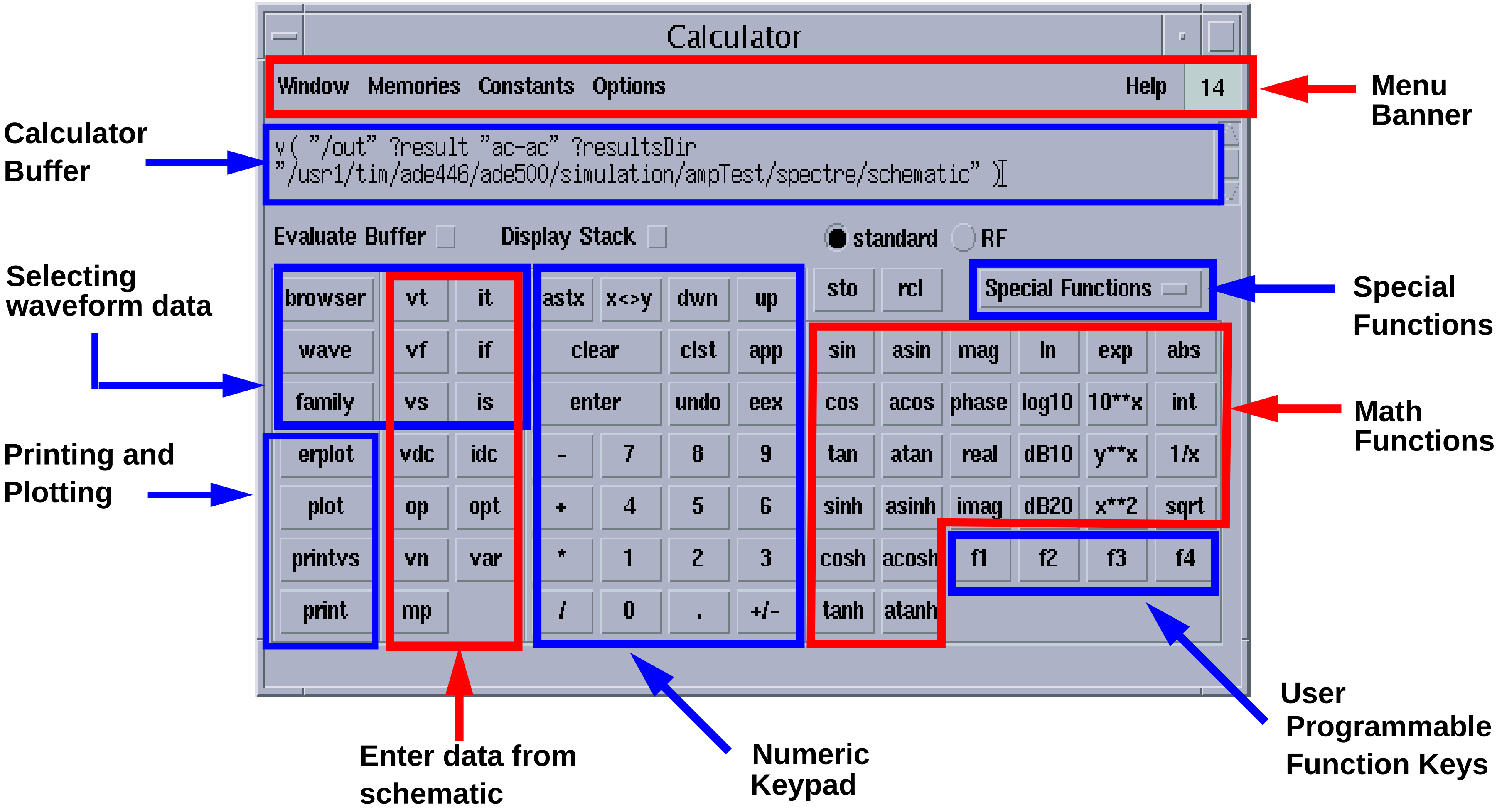
Waveform Calculator	A user interface used to analyze waveforms and simulation data.
vt	Command to enter selected V = F(t) into the calculator buffer.
vf	Command to enter selected V = F(f) into the calculator buffer.
it	Command to enter selected i =F(t) into the calculator buffer.
if	Command to enter selected i= F(f) into the calculator buffer.
wave	Enters data of Waveform Window into the calculator buffer.
evaluate buffer	Command to evaluate data or expression of the calculator buffer.
erplot	Erase Waveform Window and plot data of calculator buffer.
Print Engine	Cadence software tool used for printing specified simulation data.
Results Browser	Cadence software tool used to browse and select simulation data.
Circuit Conditions	A feature that allows circuit conditions to be view on the schematic.
Spectre Sweep	A series of simulations viewed as a curve over a selected range.

The Waveform Calculator

In the Simulation Environment, select **Tools—Calculator**.

In the Waveform Window, select **Tools—Calculator** or click the Calculator Icon.

In the CIW, select **Tools—Analog Environment—Calculator**.



```
v( "/out" ?result "ac-ac" ?resultsDir
"/usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic" )
```

Calculator
Window Memories Constants Options Help 14
Evaluate Buffer □ Display Stack □ ● standard ○ RF
browser vt it astx 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

Menu Banner

Calculator Buffer

Selecting waveform data

Special Functions

Math Functions

Printing and Plotting

User Programmable Function Keys

Enter data from schematic

Numeric Keypad

The Waveform Calculator

Use the Waveform Calculator to:

- Build, print, and plot expressions containing your simulation output data
- Build expressions to be used with labels in the Waveform Window
- Enter expressions, which can contain node voltages, port currents, operating points, model parameters, noise parameters, design variables, mathematical functions, and arithmetic operators, into a buffer.
- Store the buffer contents into a memory and then recall the memory contents back into the buffer.
- Save calculator memories to a file and load those memories back into the calculator.

The *Waveform Calculator User Guide* has more instructions concerning calculator usage and functionality. This is available through the CDSDoc documentation library.

Postprocessing Data with the Waveform Calculator

There are four ways to enter data into the calculator:

1. Import buttons (**vt**, **vf**, **it**, and **if**) allow probing in the schematic window.
2. The **wave** button allows probing in a Waveform Window.
3. Use the Results Browser.
4. Type in a signal name or expression.

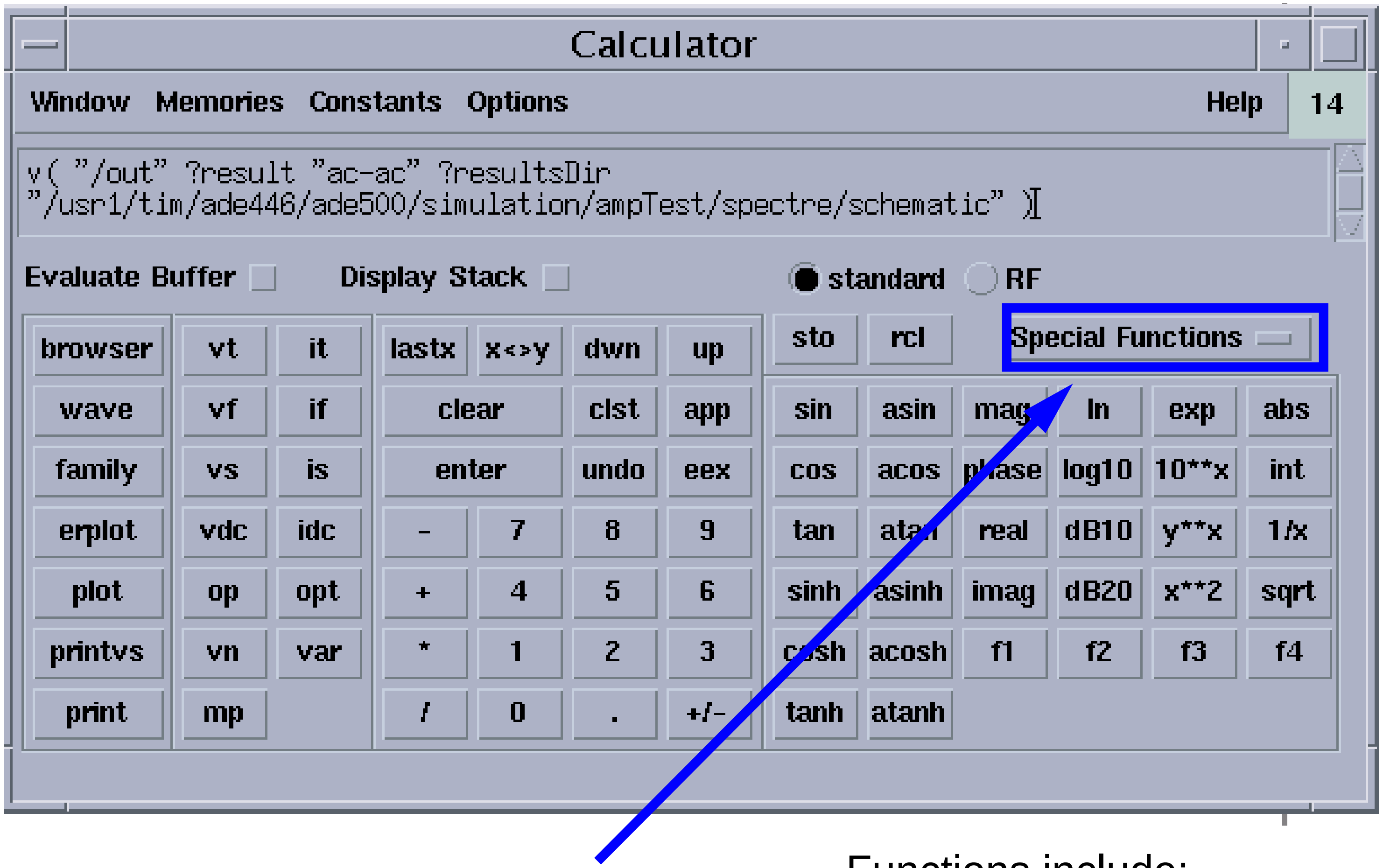
- Plot results or evaluate buffer expressions.
- Use the **printvs** button to print data tables to a file.
- Special functions are available, including Discrete Fourier Transforms (DFTs), rise times, gain margin, phase margin, and slew rates.
- Use the Calculator in Reverse Polish Notation (RPN) or Algebraic Mode.
- A special RF mode is available.

Postprocessing Data with the Waveform Calculator

Enter information into the calculator in four ways. Enter simulation output expressions, such as node voltages, port currents, operating points, model parameters, and noise parameters, into the buffer by probing in the schematic. Also select simulation output directly from Results Browser. The **wave** button selects signals in the Waveform Windows, and imports it into the calculator buffer for processing.

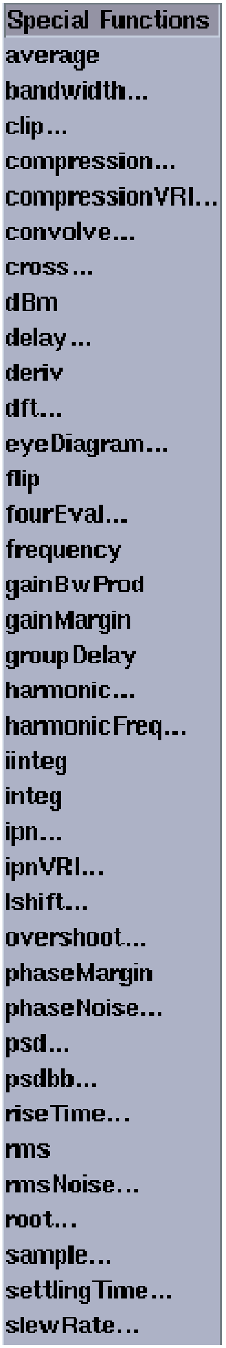
The Results Browser (discussed later in this module) locates the data file with signal information. A left click on the signal name shall enter the signal data into the buffer of the calculator.

Waveform Calculator, Special Functions Key



```
v( "/out" ?result "ac-ac" ?resultsDir
"/usr1/tim/ade446/ade500/simulation/ampTest/spectre/schematic" )
```

Evaluate Buffer □ Display Stack □ ● standard ○ RF



Press “Special Functions” to display 47 waveform processing functions.

Functions include:

- integral
- derivative
- dft
- delay
- eye diagram (New!)
- thd

Waveform Calculator, Special Functions Key

The Waveform Calculator is a special tool for analyzing simulation data, including waveform analysis. The “Special Functions” key provides preprogrammed operations that simplify the process. The 47 different functions include:

- Discrete Fourier Transform
- Total Harmonic Distortion
- Bandwidth Measurement
- Frequency
- Automated Delay Measurement
- A new function for ADE 5.0 is the “eye-diagram”

Use the special functions to simplify measurements of your simulation results. These functions are available for automated measurements using Corners, Monte Carlo, and with OCEAN scripts.

Print Engine

- Printed results are now displayed in a “smart” print window.
- Examples of printed results:
  - All Results—Print commands
  - Print and Report commands from OCEAN
  - Simple or parametric waveforms
  - Generic tabular data (such as Monte Carlo data)
- Tabular data can be further formatted using form-driven interface. (Move columns, sort, and expressions.)

Print Engine

The Print Engine is an attempt at generating a smarter implementation of the “view file” window that displays results in releases prior to 4.4.3.

The print engine works best for tables of numbers. For example:

- The output of the calculator printvs command for several outputs over a set of transient time points is well suited for the print engine. All columns are related (all from the same time point) and functions like sorting, moving columns, and operating on the all data points in a column with expressions makes sense.
- The output of the DC operating point on a transistor, however, is not well suited to the print engine, because it is described by several columns of unrelated numbers. For consistency, this function uses the Print Engine window to display results. However, no further manipulation on the data can be done because the columns are unrelated.

Printing the Results

Specify the results to print:

- Run a simulation.
- Then in the Simulation Window, select **Results—Select**.

  A Select Results window appears. This window lets you select other simulation results. (Make sure the schematic window for the selected design is open.)
- To print the results, select **Results—Print** with one of the following options:
  - DC Node Voltages or DC Operating Points
  - Model Parameters
  - Transient Node Voltages or Transient Operating Points
  - S-Parameter
  - Noise Parameters
  - Noise Summary or PSS Noise Summary
  - Sensitivities
- Select a node or device in the schematic window.

Printing the Results

DC Operating Points: Choose **Results—Print—DC Operating Points**. Move your cursor into the schematic window. The CIW displays prompts to select instances for the operating point output. Click an instance.

Transient Operating Points: Choose **Results—Print—Transient Operating Points**. Move your cursor into the schematic window. The CIW displays prompts to select instances for the transient operating point (OPT) output. Click an instance or node.

Model Parameters: Choose **Results—Print—Model Parameters**. Move your cursor into the Schematic window. The CIW displays prompts to select instances for the model parameter output. Click an instance of a device.

Noise Parameters: Choose **Results—Print—Noise Parameters**. Set a frequency in the Select Frequency Value form that appears. If the form does not appear, press the **F3** key. The default frequency is 1K. Move your cursor into the schematic window. The CIW displays prompts to select instances for the VNP output. Click on an instance or node.

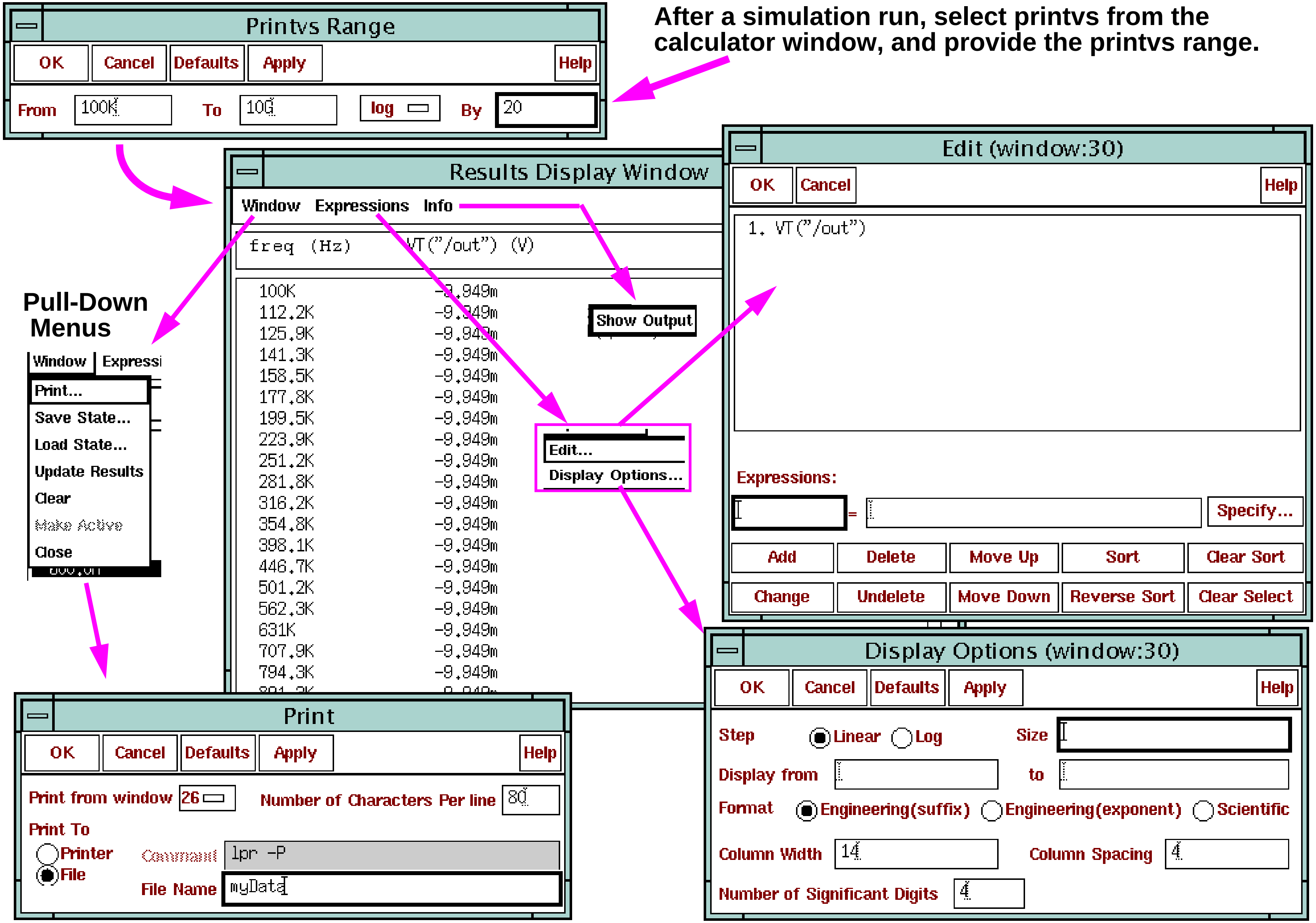
DC Node Voltages: Choose **Results—Print—DC Node Voltages**. Move your cursor into the Schematic window and select nets for the VDC output. Click a node.

Transient Voltages: Choose **Results—Print—Transient Node Voltages**. Enter a time in the Select Time Value form that appears. If the menu does not appear automatically, press the **F3** key. The default time value is zero. Move your cursor into the Schematic window. The CIW displays prompts to select nets for the time value output. Click a node (net8, for example).

Sensitivities: Choose **Results—Print—Sensitivities**. Move your cursor into the Schematic window and select nets for the output. Click on a net or port.

Results Display Window

After a simulation run, select printvs from the calculator window, and provide the printvs range.



Printvs Range

From 100K To 10G log By 20

Results Display Window

```
freq (Hz)         VT("/out") (V)

 100K             -9.949m
 112.2K           -9.949m
 125.9K           -9.949m
 141.3K           -9.949m
 158.5K           -9.949m
 177.8K           -9.949m
 199.5K           -9.949m
 223.9K           -9.949m
 251.2K           -9.949m
 281.8K           -9.949m
 316.2K           -9.949m
 354.8K           -9.949m
 398.1K           -9.949m
 446.7K           -9.949m
 501.2K           -9.949m
 562.3K           -9.949m
 631K             -9.949m
 707.9K           -9.949m
 794.3K           -9.949m
```

Edit (window:30)

1. VT("/out")

Expressions: =

Print

Print from window 26

Number of Characters Per line 80

Print To

○ Printer

● File

Command lpr -P

File Name myData

Display Options (window:30)

Step ● Linear ○ Log Size

Display from to

Format ● Engineering(suffix) ○ Engineering(exponent) ○ Scientific

Column Width 14 Column Spacing 4

Number of Significant Digits 4

Results Display Window

The Results Display Window offers a comprehensive suite of tools to process simulation data.

Run Data Storage Directories

Subdirectories are created below the project directory to define a unique storage location for simulation results.

- The project directory is specified in the Simulation environment.

  Hierarchy Level

  Project Directory/Cell_Name/Simulator_Name/Run_Name/Run_Data

  Example

  ~/simulation/ampTest/spectre/schematic/   (netlist/ or psf/)

- Data storage directories can be managed with UNIX commands.
- Numerical simulation data is written to a binary file in Parameter Storage Format (PSF), which is stored in the *psf/* directory.
- The final netlist and text files related to netlist generation are stored in the *netlist/* directory.

Run Data Storage Directories

Run data is stored in this location:

```
<Project_Directory>/<Cell_Name>/<Simulator_Name>/<Run_Name>/<Run_Data>
```

Names in brackets represent directories on the system as defined below:

`<Project_Directory>`

User-defined directory defined in the Simulation environment.

`<Cell_Name>`

Name of the simulated cell as it appears in the Library Manager.

`<Simulator_Name>`

Name of the simulation engine used.

`<Run_Name>`

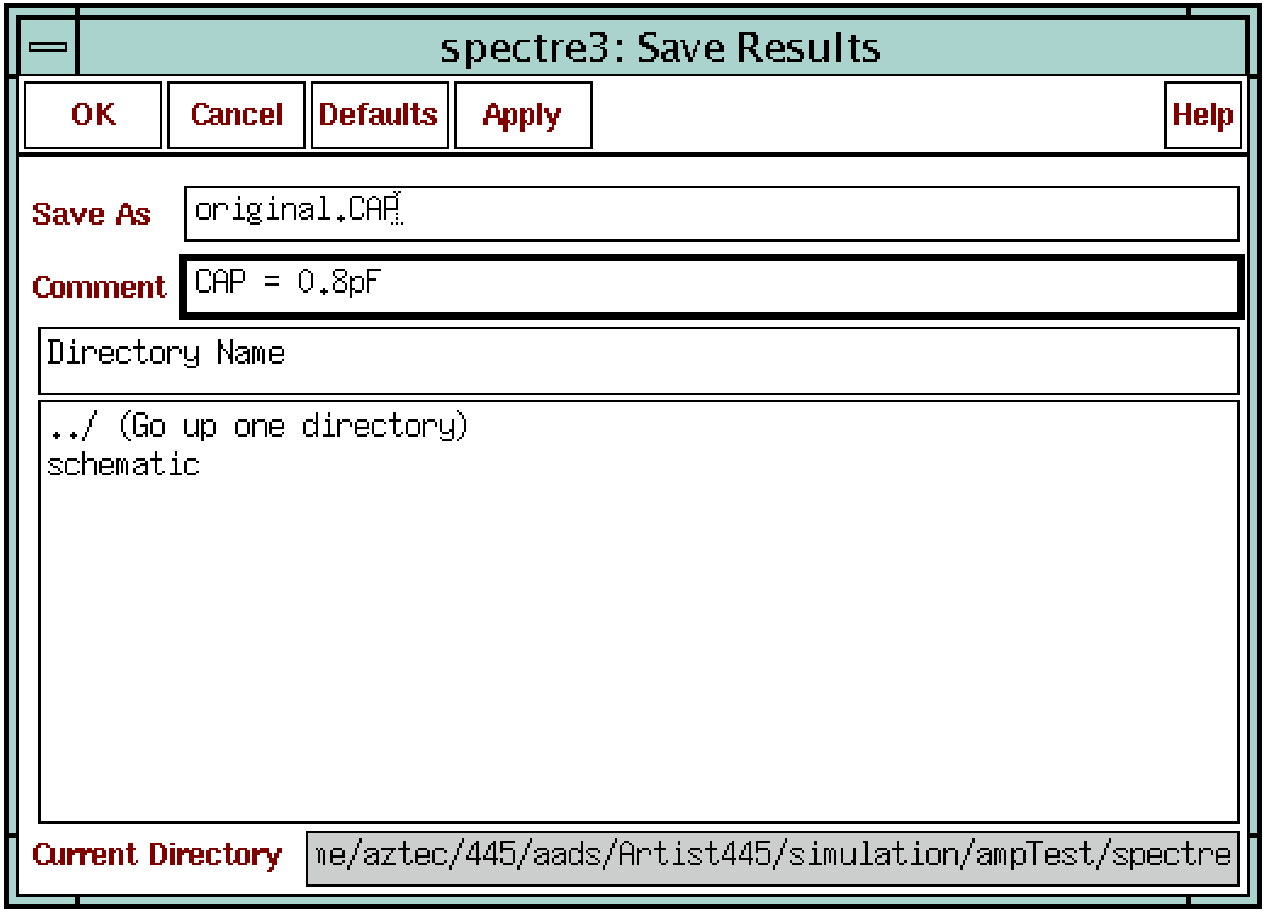
Always */schematic* unless the <*Run_Data*> is moved by the user. The */schematic* directory is overwritten on the next simulation. We discuss <*Run_Data*> saving methods later.

`<Run_Data>`

Run_Data contains two subdirectories. The */psf* directory holds all the numerical run data. This includes waveform data, initial conditions, model parameters, and operating points. The */netlist* directory contains textual run data, including the netlist and simulator output files.

Backing Up Simulation Data Explicitly

The */schematic* directory is overwritten during subsequent simulation runs of the same design. To preserve simulation data, use the **Results—Save** command.



Save As original.CAP

Comment CAP = 0.8pF

Directory Name

```
../ (Go up one directory)
schematic
```

Current Directory ne/aztec/445/aads/Artist445/simulation/ampTest/spectre

Results—Save

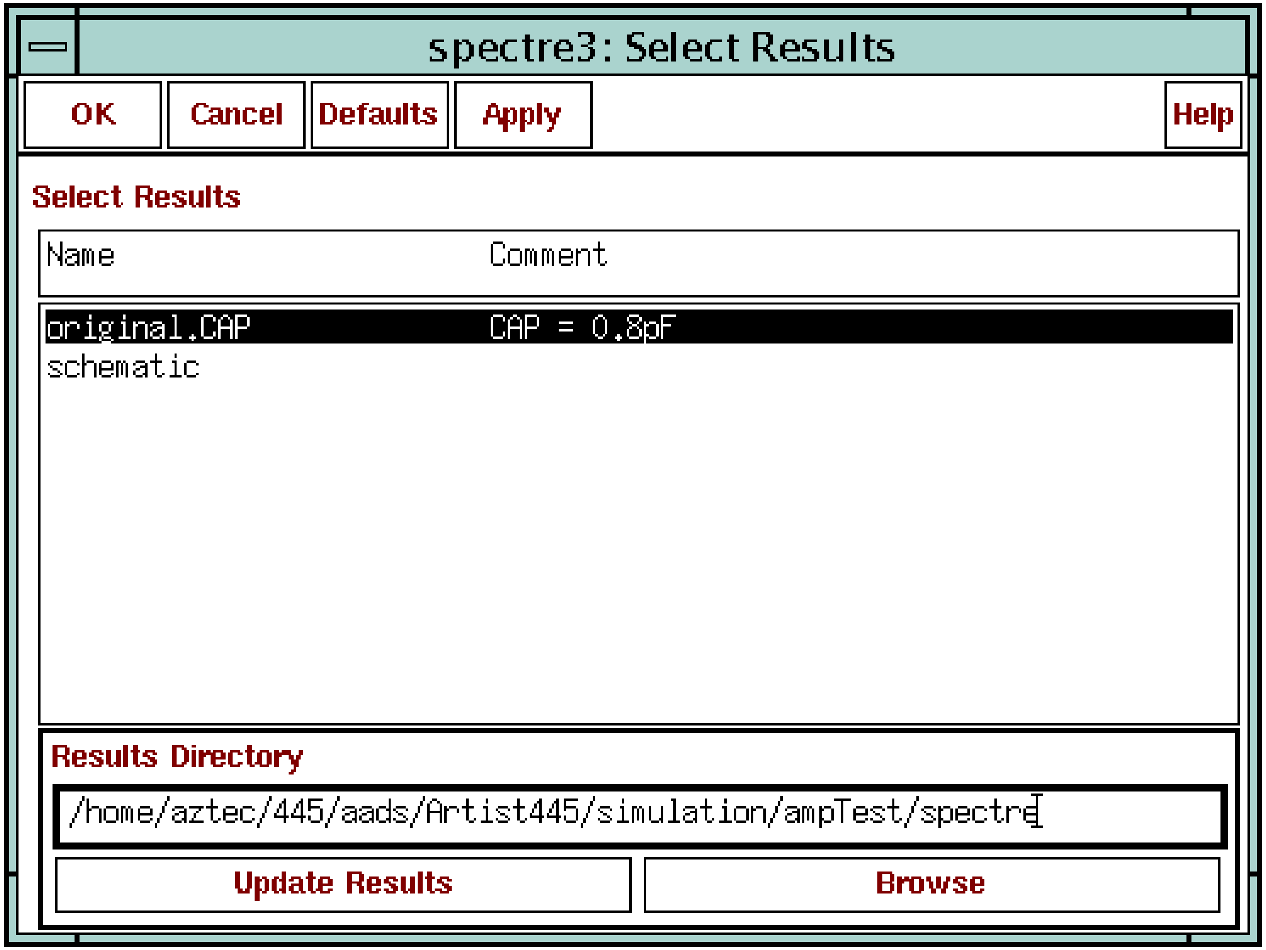
Backing Up Simulation Data Explicitly

If automatic simulation results saving is disabled, then save the simulation results explicitly. To do this, copy the results data directories to user-defined locations. Attach comments to the data directories for use when selecting results in the Simulation environment.

Use this command to back up an automatically saved run directory or save it with a more meaningful name.

Selecting Results

Load results from previous simulations.



Name	Comment
original.CAP	CAP = 0.8pF
schematic	

Results Directory: /home/aztec/445/aads/Artist445/simulation/ampTest/spectre

Results—Select...

Selecting Results

By specifying the location of a *psf/* directory, past simulation data is linked to a schematic window. Probe the schematic to analyze older simulation results and compare them with current data. Use this feature when exiting the Analog Design Environment or logging out to retain the simulation data upon resuming work.

Note: The schematic window used to generate the simulation data must be open when saving the results. Additionally, save the *.artist_states* used to create the simulation results.

Important

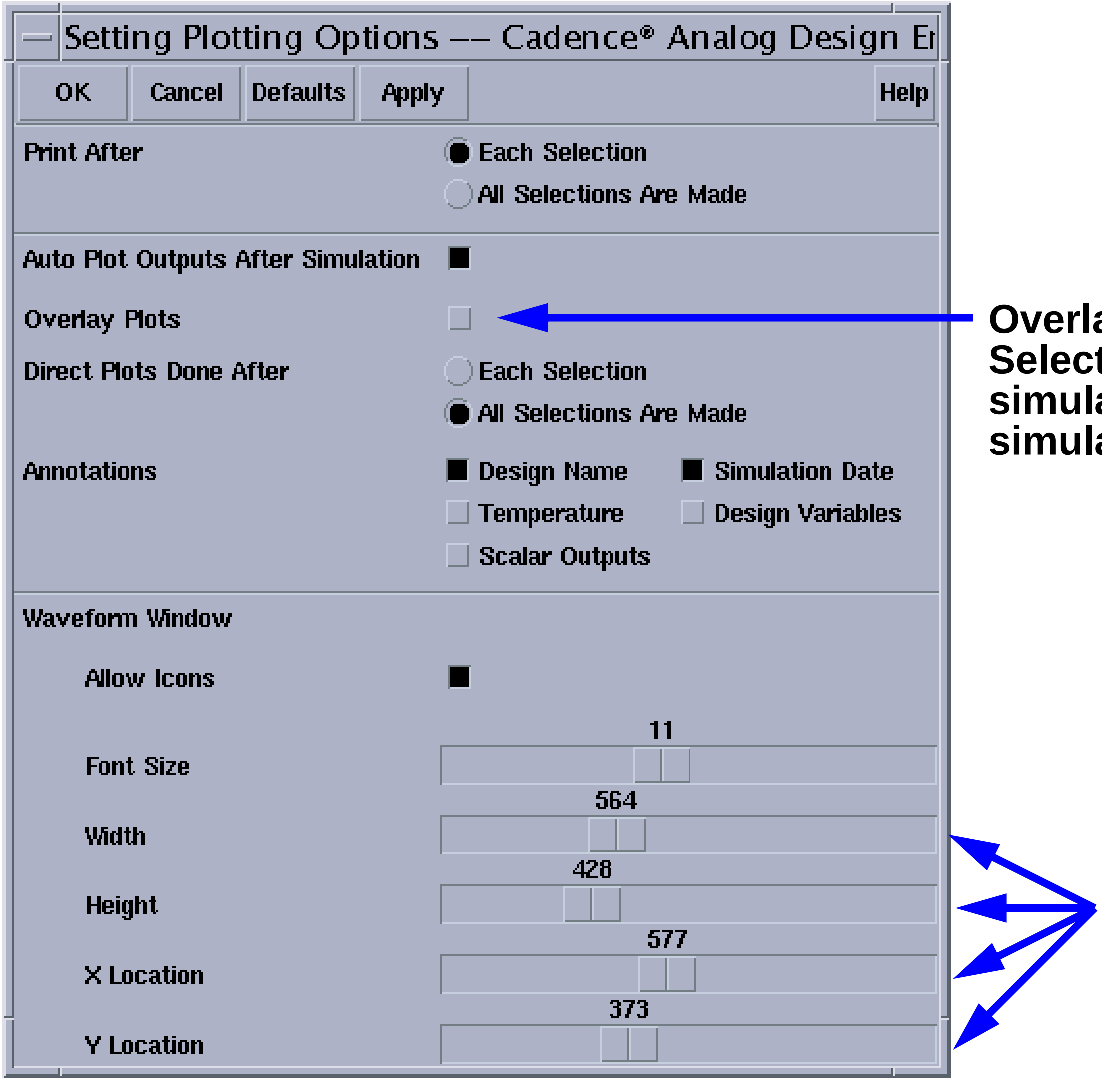
Never edit the schematic in a way that changes the topology. The saved data corresponds to the schematic at the time the simulation is run. If you must change the schematic, consider an alternate cell or view name. Otherwise, the simulation data linked to the edited schematic is not meaningful. Make a copy of the design, and edit the copy, leaving the original design to link simulation results in the future. Failing to follow this practice does not render the saved data useless, even if the topology changes; however, you can no longer click on an internal node of the schematic. Use the Results Browser to access the data for plotting and post-simulation analysis.

Hints for Saving Simulation Data

Copy the */schematic* directory to a directory whose name contains the design name and date of last modification. This helps to keep the results pertinent to the proper design.

PSF is an efficient storage format, and the files in the */psf* directory can be compressed to further save disk space. When deleting files in the */netlist* directory, use caution as some of the data may be needed later on.

Setting Plotting Options



Print After ● Each Selection ○ All Selections Are Made

Auto Plot Outputs After Simulation ■

Overlay Plots □

Direct Plots Done After ○ Each Selection ● All Selections Are Made

Annotations ■ Design Name ■ Simulation Date □ Temperature □ Design Variables □ Scalar Outputs

Waveform Window

Allow Icons ■

Font Size 11

Width 564

Height 428

X Location 577

Y Location 373

Overlay Plots
Select if comparing previous simulation data to the next simulation waveform.

Location and size of Waveform Window

Results—Printing/Plotting Options

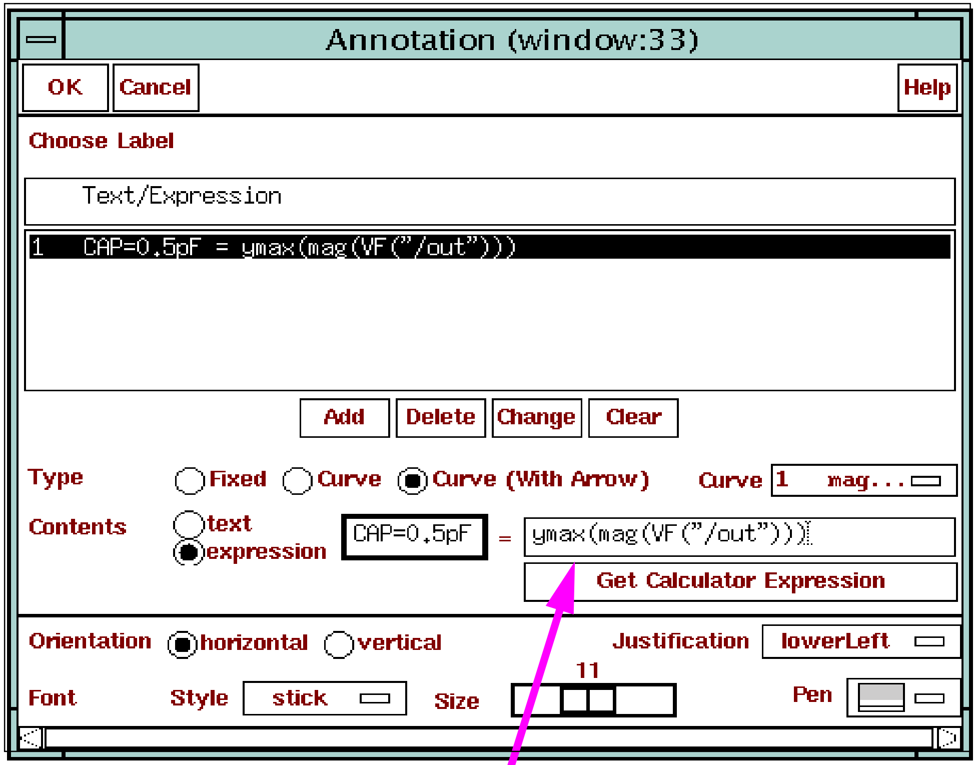
Setting Plotting Options

Plotting Options can be set with the following commands:

- **Auto Plot After Simulation**: Instructs the software to automatically plot the selected signals and waveform expressions contained in the Outputs list box of the Simulation window, after the simulation ends.
- **Overlay Plots**: This is part of the automatic plotting feature of the Analog Design Environment. If Overlay Plot is **OFF** (default), the Waveform Window is always erased before a new waveform is drawn. If Overlay Plots is **ON**, a new waveform is plotted on top of the old waveform. This is useful for comparing simulation results on the same design with parameter changes.
- **Direct Plots Done After**: This refers to the Direct Plot commands. The default is to plot signals after **All Selections Are Made**. Change the setting so that the Waveform Window is updated immediately after **Each Selection** on the schematic.
- **Annotations**: This feature displays information such as Design Name, Temperature, and other parameters shown above, in the Waveform Window. This makes simulation results more complete for archiving purposes.
- **Waveform Window** options: These options affect the physical appearance of the Waveform Window. Turn the icons in the window on or off, change the font size used for labels, and change the size and location of the Waveform Window. These choices have the greatest impact on the Waveform Window that is generated with automatic plotting after the simulation ends.

Annotating Data to the Waveform Window

Annotation—Edit



Annotation (window:33)

OK Cancel Help

Choose Label

Text/Expression

1 CAP=0.5pF = ymax(mag(VF("/out")))

Add Delete Change Clear

Type ○ Fixed ○ Curve ● Curve (With Arrow) Curve 1 mag...

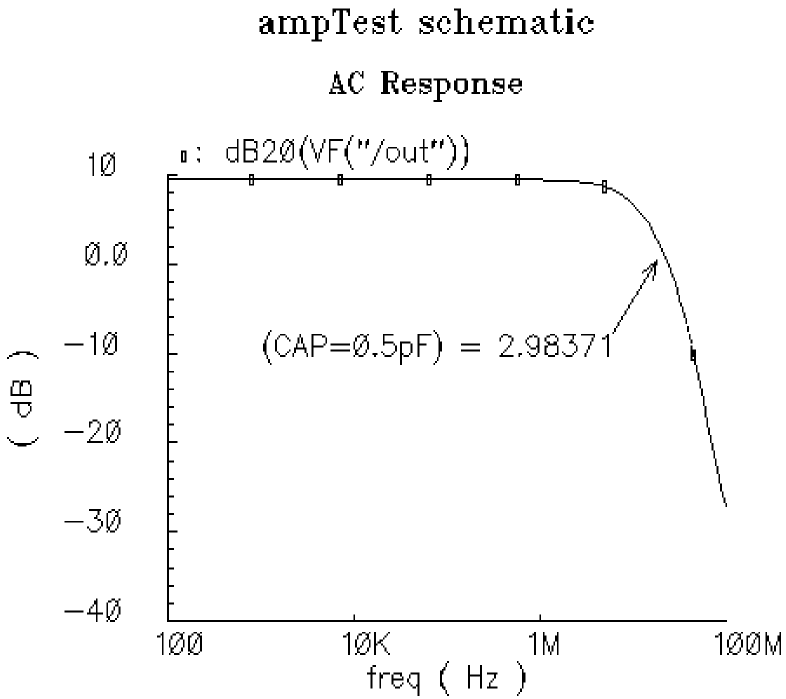
Contents ○ text ● expression CAP=0.5pF = ymax(mag(VF("/out")))

Get Calculator Expression

Orientation ● horizontal ○ vertical Justification lowerLeft

Font Style stick Size 11 Pen

Waveform Window



Build expressions in the Calculator and use them to create labels in the Waveform Window.

Annotating Data to the Waveform Window

Use the calculator to build expressions to use with labels in the Waveform Window.

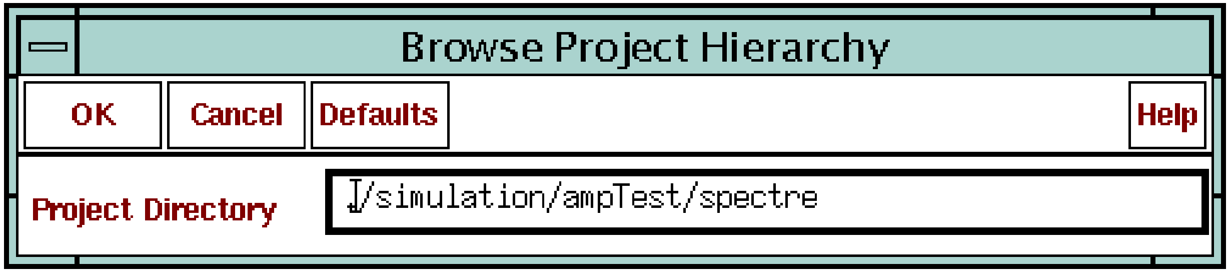
Use these labels to annotate important measurements to waveforms in datasheets or other documentation.

Starting the Results Browser

In the Simulation Environment or Waveform Window, select **Tools—Results Browser**.

In the CIW, select **Tools—Analog Environment—Results Browser**.

In the Calculator, click the **Browser button**.



Project Directory ./simulation/ampTest/spectre

Specify a Project Directory.

The Results Browser accesses data in the Project Directory and below.

Examples of Project Directories

`./simulation/ampTest/spectre`

Accesses all Spectre results for this design.

`./simulation/ampTest`

Accesses all results for this design.

Starting the Results Browser

This Results Browser is a powerful simulation data management tool that is similar in appearance and functionality to a tree browser.

Specify a project directory to display the Results Browser. The project directory is the highest directory the browser can access. Set this high enough to access required simulation data.

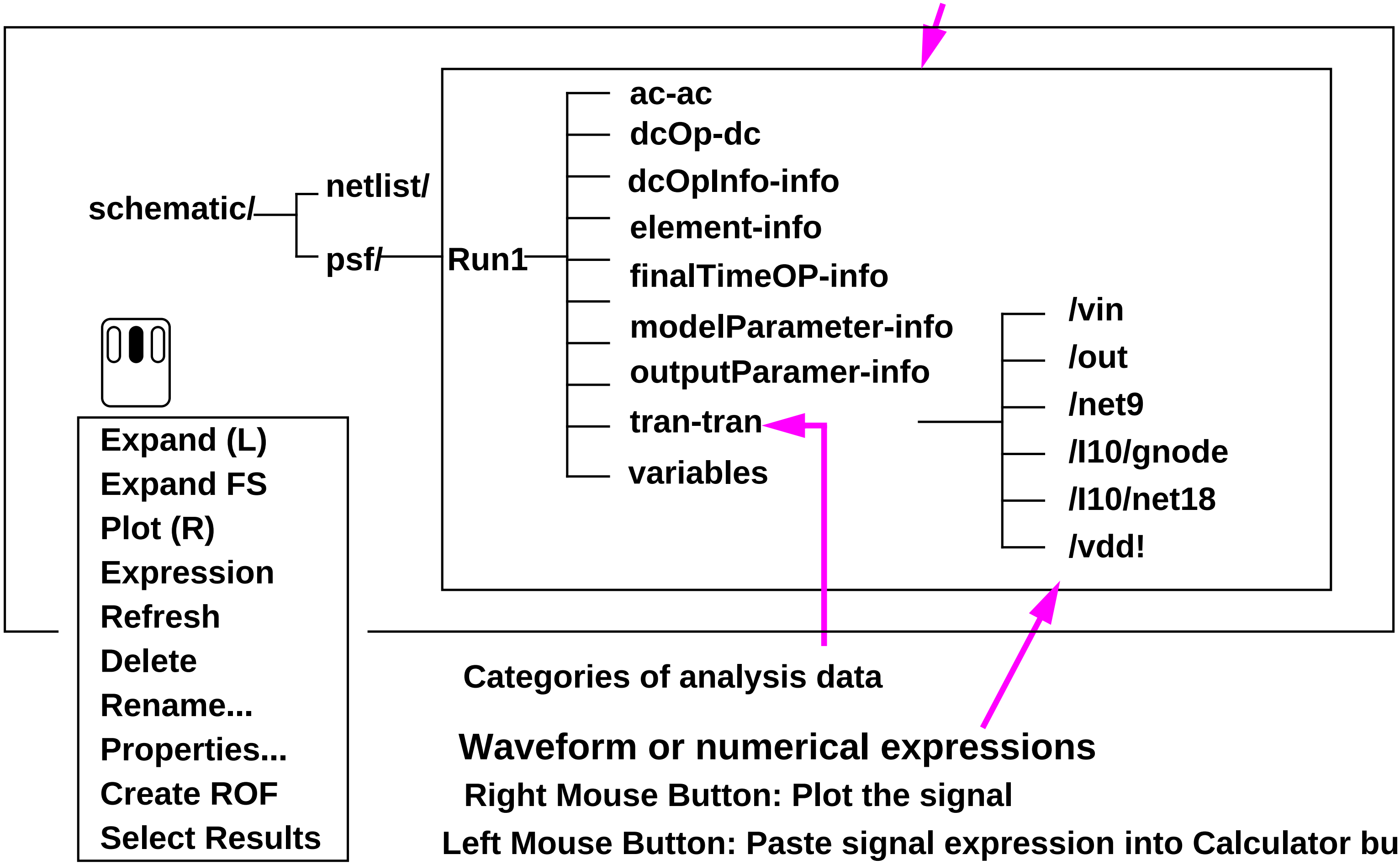
Note, use the Results Browser to load in and view simulation data without having the schematic open or available. This is useful in sending simulation data between two or more locations of a design center in an efficient manner.

The Results Browser

The Results Browser is similar to the Library Manager. An object menu can manage or expand the data structure.

Expanding the *psf* directory

Interpretation of the PSF binary data



schematic/

netlist/

psf/

Run1

ac-ac

dcOp-dc

dcOpInfo-info

element-info

finalTimeOP-info

modelParameter-info

outputParamer-info

tran-tran

variables

/vin

/out

/net9

/I10/gnode

/I10/net18

/vdd!

Expand (L)

Expand FS

Plot (R)

Expression

Refresh

Delete

Rename...

Properties...

Create ROF

Select Results

Categories of analysis data

Waveform or numerical expressions

Right Mouse Button: Plot the signal

Left Mouse Button: Paste signal expression into Calculator buffer

The Results Browser

The Results Browser Object Menu

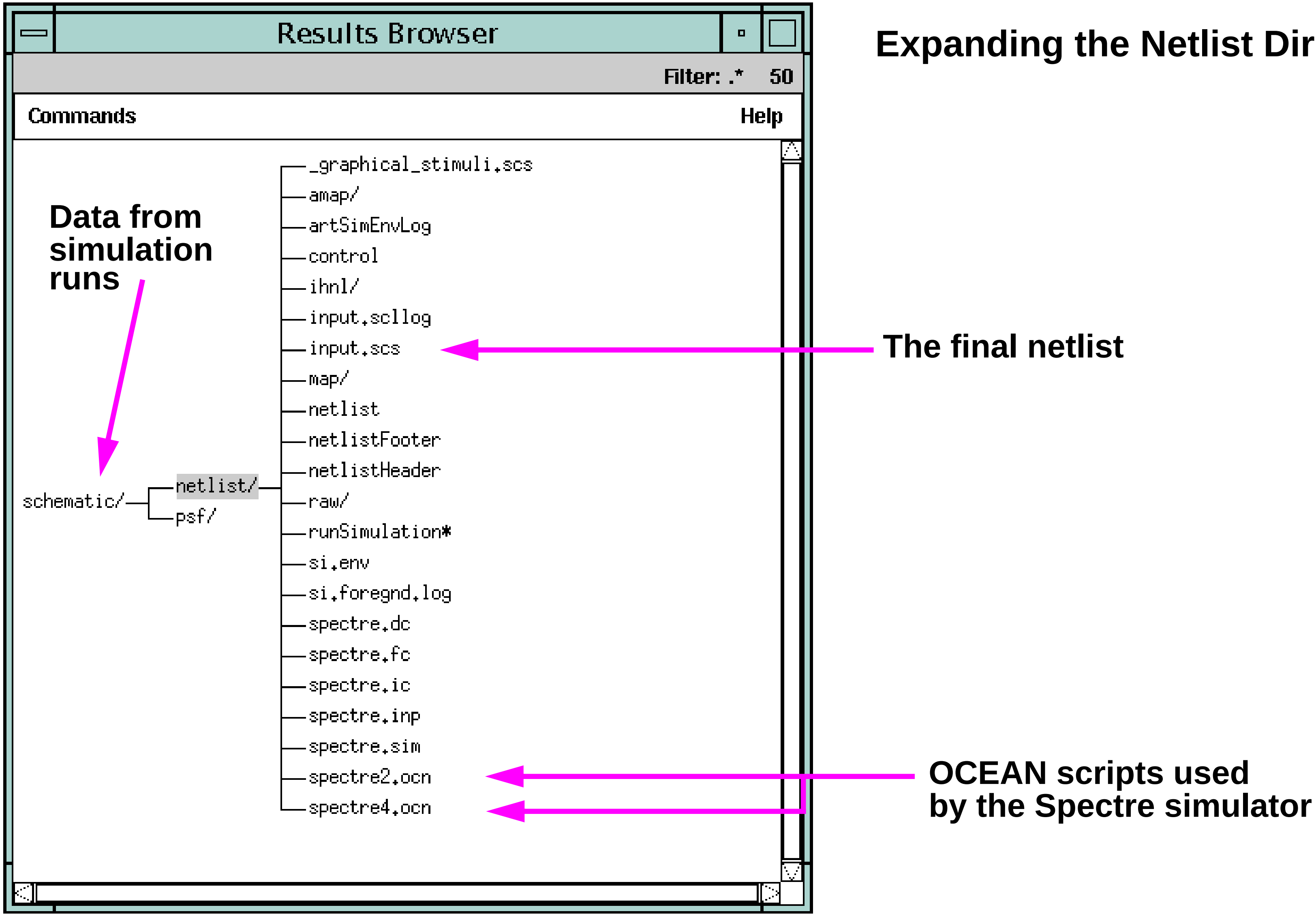
There is an Object Menu (OM) for the Results Browser. For a complete description, see CDSDoc. Unlike the Library Manager, the Results Browser OM does not change depending on the selected text. **Expand** is bound to the left mouse button as indicated by the *(L)* in the OM. Use this command at all levels of the Results browser. **Plot** is bound the right mouse button as indicated by the *(R)* in the OM. Move a data quantity such as a waveform, to the post processing tool with the **Expression** command. Use **Delete** or **Rename** for directories. The **Create ROF** command translates ASCII output data from a standalone simulation engine into the PSF format for use with the Waveform Window. Also **Select Results** from a data directory and link them to a schematic window if the simulation window is open.

Postsimulation Data Structure Categories

element-info contains component parameters for design circuit elements. *modelParameters-info* contains simulation model parameters. *dcOp-dc* contains the node voltages calculated from the DC operating point. *dcOpInfo-dc* contain operating point information for all of the component parameters in the design. *tran-tran* contains waveform data from the transient analysis. *ac-ac* contains waveform data for the AC analysis.

Viewing Presimulation Text Data

Click the names of text files with the **right** mouse button in the Results Browser to view their contents.



Expanding the Netlist Directory

Data from simulation runs

The final netlist

OCEAN scripts used by the Spectre simulator

Viewing Presimulation Text Data

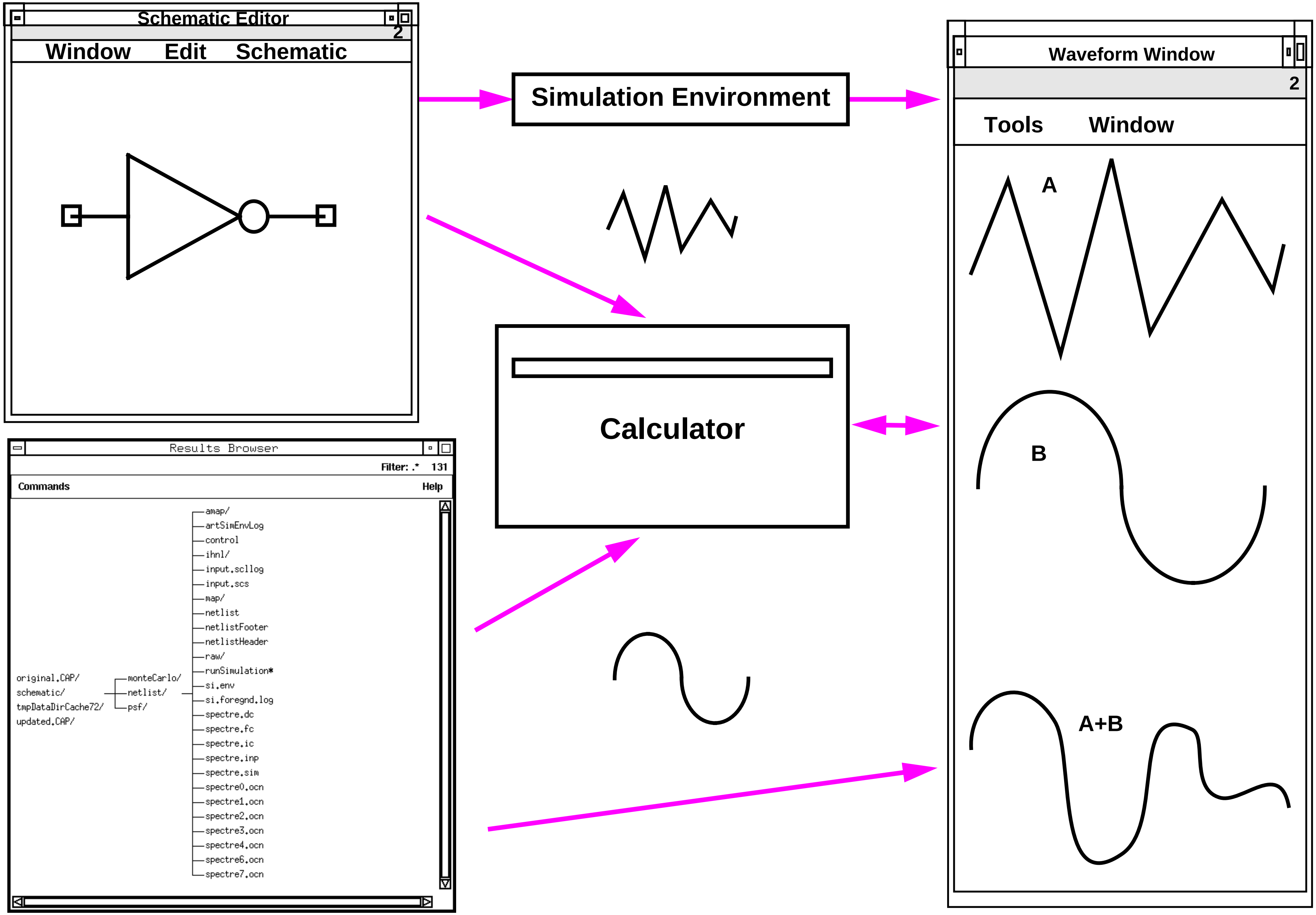
At the *schematic* level, there can be more than one simulation, but each must have a distinct name from the others. There can then be a tree structure under each of these runs similar to the one shown above under *schematic*.

Textual data includes netlists. Expand the textual data entries in the Results Browser as illustrated and click on the name of file for viewing. A window appears containing the text.

In this example, *input.scs* is the netlist used by the simulation engine.

The files names *spectre0.ocn*, *spectre1.ocn* and so on are OCEAN scripts that are automatically created during the simulation session and are discussed later on.

Interactive Postprocessing Tools



```
                                           ┌amap/
                                           ├artSimEnvLog
                                           ├control
                                           ├ihnl/
                                           ├input.scllog
                                           ├input.scs
                                           ├map/
                                           ├netlist
                                           ├netlistFooter
                                           ├netlistHeader
                                           ├raw/
                                           ├runSimulation*
original.CAP/          ┌monteCarlo/       ├si.env
schematic/           ──┼netlist/        ──┤si.foregnd.log
tmpDataDirCache72/     └psf/               ├spectre.dc
updated.CAP/                               ├spectre.fc
                                           ├spectre.ic
                                           ├spectre.inp
                                           ├spectre.sim
                                           ├spectre0.ocn
                                           ├spectre1.ocn
                                           ├spectre2.ocn
                                           ├spectre3.ocn
                                           ├spectre4.ocn
                                           ├spectre6.ocn
                                           └spectre7.ocn
```

Interactive Postprocessing Tools

Take advantage of powerful interactive post processing tools in the Analog Design Environment.

Conditional Search and Display

- Form-driven search, probe, and print for specified device operating conditions.
- Finds and displays breakdown conditions, MOSFETs in linear region, and saturated BJTs.
- User-configurable search criteria
- Includes multiple constraint (Boolean) searches.

Conditional Search and Display

After the simulation is finished, use this form to search through all operating point data for devices with certain conditions. Set up search conditions by using the Results: Circuit Conditions form.

The boolean functions AND and OR are available to look for devices with multiple conditions.

- For Spectre saturation, an instance is highlighted if:
  - BJT: If the operating point parameter region=3
  - MOS/BSIM: If the operating point parameter region=2
- For cdsSpice saturation, an instance is highlighted if the operating point sat=0.
- For Spectre breakdown in a BJT, an instance is highlighted if the operating point parameter region=4
- For cdsSpice breakdown, an instance is highlighted if the operating point bkdwn=0
- For the simulator to calculate breakdown or saturation, the appropriate model parameters need to be set.

Currently, the conditional search capability only works for DC operating conditions.

Setting Up a Conditional Search

1. Run a simulation or invoke **Results—Select Results** and choose other simulation results. This provides the results to search.
2. Choose **Results—Circuit Conditions**.
3. Select the Device Operating Conditions.
4. Set up the User-Defined Conditions.
5. View the Circuit Conditions results using the Place or Print buttons.

Setting Up a Conditional Search

1. Run a simulation or choose which results to search using the **Results—Select** command.

   This feature only works when running a DC operating point analysis.

2. Choose **Results—Circuit Conditions** from the Simulation window to launch the Conditional Search form.

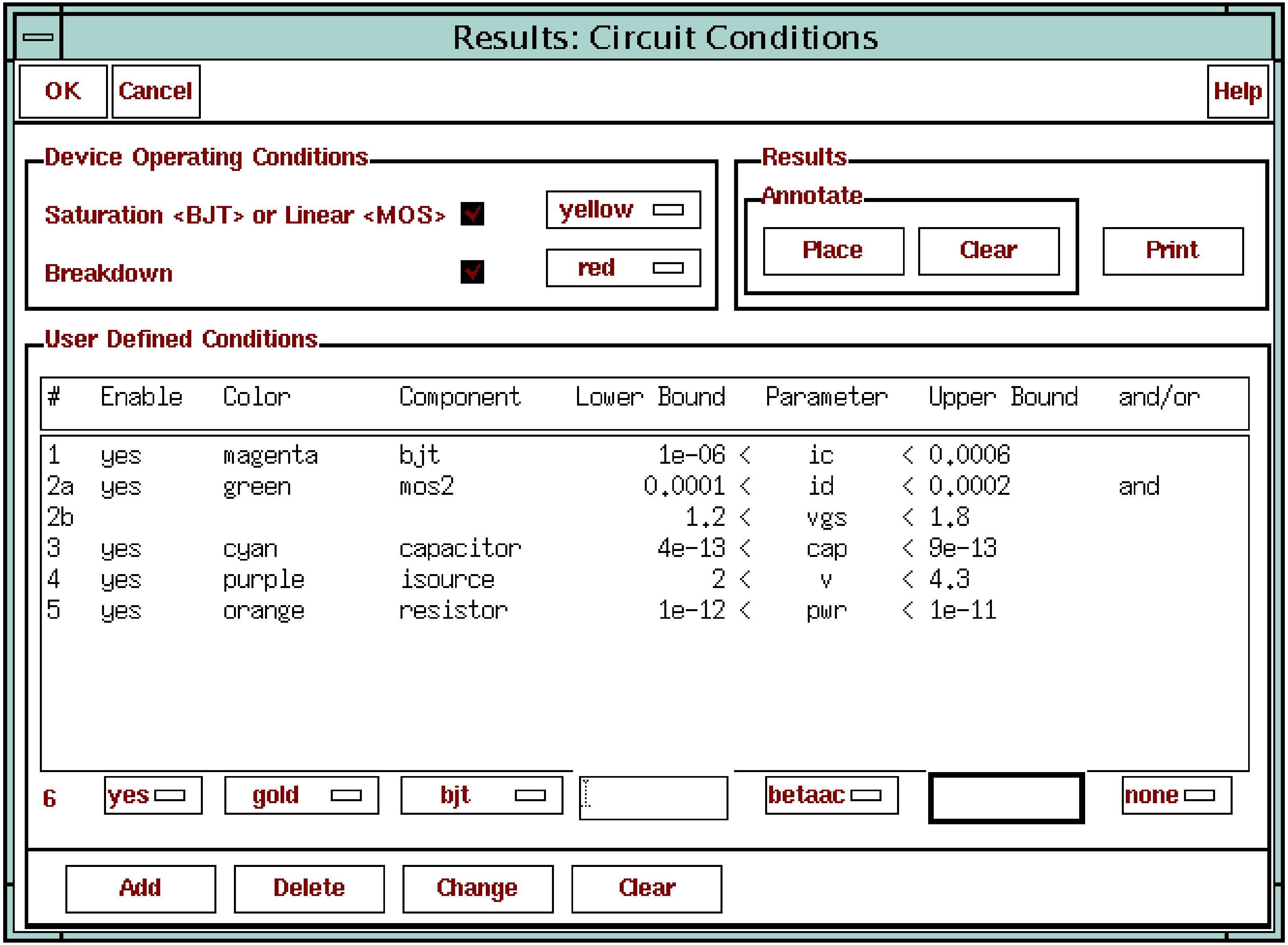
3. Choose device operating conditions to view components in the saturation (for BJT devices), linear (for MOS devices) or breakdown region.

   The appropriate model parameters must be set for the simulator to calculate these conditions. These features might not be available for simulators other than the Spectre or cdsSpice tools.

4. Set up the User Defined Conditions for cyclic and type-in fields to create custom search conditions .

5. To view the results of the conditions, click the **Place** button to highlight the instances that meet the specified conditions on the schematic. Click **Print** to print the values of instances that meet the specified conditions in a print window.

The Circuit Conditions Form



Results: Circuit Conditions

OK | Cancel | Help

Device Operating Conditions

Saturation <BJT> or Linear <MOS> ☑ yellow

Breakdown ☑ red

Results

Annotate: Place | Clear

Print

User Defined Conditions

#	Enable	Color	Component	Lower Bound		Parameter		Upper Bound	and/or
1	yes	magenta	bjt	1e-06	<	ic	<	0.0006	
2a	yes	green	mos2	0.0001	<	id	<	0.0002	and
2b				1.2	<	vgs	<	1.8	
3	yes	cyan	capacitor	4e-13	<	cap	<	9e-13	
4	yes	purple	isource	2	<	v	<	4.3	
5	yes	orange	resistor	1e-12	<	pwr	<	1e-11	
6	yes	gold	bjt			betaac			none

Add | Delete | Change | Clear

The Circuit Conditions Form

The top left section provides buttons for turning the display of devices in the breakdown, saturation, and linear region on and off. Set the probe colors to help differentiate the conditions.

The upper right section of the form is used to place and remove the graphical forms from the schematic.

The lower portion of the form sets up the user-defined search criteria.

1. Pick a component type, such as bjt, mos, vsource, or resistor.
2. Choose the operating point parameter value from the cyclic field.
3. Set a lower and upper bound by typing a value on either side of the parameter. Leave either blank to set only one boundary.
4. Set the Boolean arguments to a condition. The options are: “none”, “and”, and “or”. When *and* is used, both conditions must be met for an instance to be highlighted. When *or* is used, either condition must be met for an instance to be highlighted. Both operators have the same precedence.
5. Set the probe colors. This field shows the color with which an instance that meets the condition will be highlighted.
6. Use the Enable field to turn a condition on or off without deleting it from the form.
7. Use the bottom buttons to **Add**, **Delete**, or **Change** an entry, or completely **Clear** the form.

Sensitivity Analysis

Use the Sensitivity Analysis to:

- View the parameters that most affect the specified outputs
- Tune a design to increase or decrease certain goals
- Determine what parameters to run in an Optimization analysis

The Sensitivity Analysis requires a base analysis to be run first.

- Only runs with AC and DC analysis at the present time.

Sensitivity Analysis

Sensitivity analysis allows a designer to see what parameters in the circuit most affect the specified outputs. It is typically used to tune a design to increase or decrease certain design goals. A designer many times will run a Sensitivity analysis to determine what parameters to optimize in the Optimizer. Sensitivity is a useful tool on its own but also has capabilities of tying in with other tools in the future.

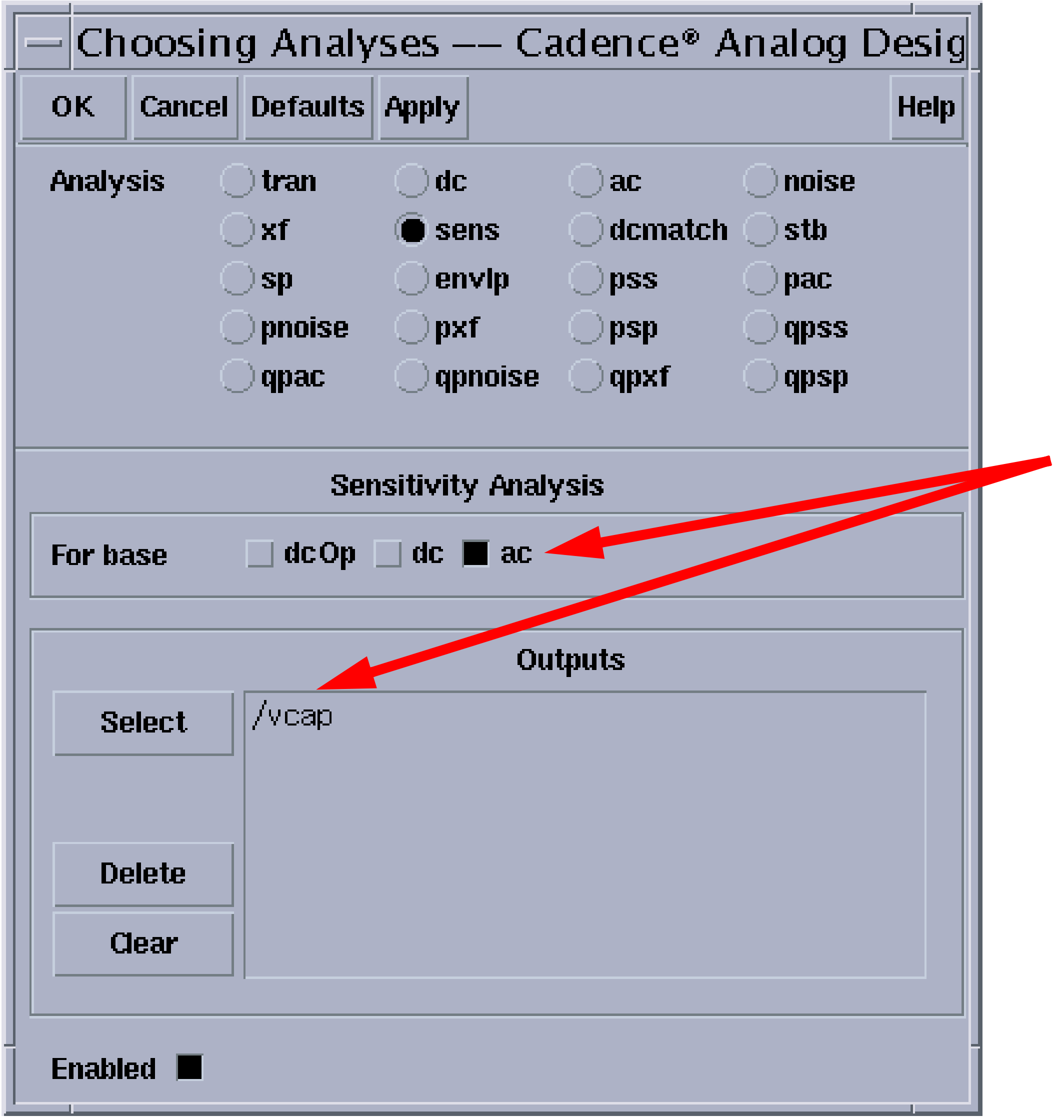
The Sensitivity Analysis project will provide a user interface to run the base Sensitivity Analysis provided by the Spectre tool. This will be created using the Analog Design Environment to add the Sensitivity analysis for the Spectre simulator. There will also be a user interface for printing the Sensitivity results. This will sort the sensitivity results with the most sensitive values being displayed first.

The Sensitivity analysis currently runs with AC and DC analyses, only.

The Spectre simulator generates model parameter sensitivities on a per model basis. If multiple devices in your circuit use the same model, the effect of changing a model parameter within that model is done on all those models at the same time.

Optionally, treat each model as a separate entity per device, so when a model parameter is changed in the model on a device, it is only changed for that single device. To achieve this with the Spectre simulator, create multiples of the same model and place a different model on each device. This functionality will be simplified in future releases.

Set Up Sensitivity Analysis



Analysis ○ tran ○ dc ○ ac ○ noise ○ xf ● sens ○ dcmatch ○ stb ○ sp ○ envlp ○ pss ○ pac ○ pnoise ○ pxf ○ psp ○ qpss ○ qpac ○ qpnoise ○ qpxf ○ qpsp

Sensitivity Analysis

For base □ dcOp □ dc ■ ac

Outputs

/vcap

Enabled ■

Select one of these and run the base analysis in addition to any selection here.

Analyses—Choose

Set Up Sensitivity Analysis

Select **Analyses—Choose** in the Simulation window to display the Choosing Analyses form. Select **sens**, and click which base analysis (AC or DC) to run with the sensitivity algorithm. Also, set up a simulation for the specified base analysis for the Sensitivity Analysis to run. The form options are as follows:

For base Select the base analysis to run for a sensitivity analysis. Default: none.

Select Prompts to select nets or ports from the schematic. The schematic will appear in the foreground of the screen, so it must be open before selecting the outputs. Select only voltages and currents as outputs for Sensitivity analysis. Press **Esc** to end selection.

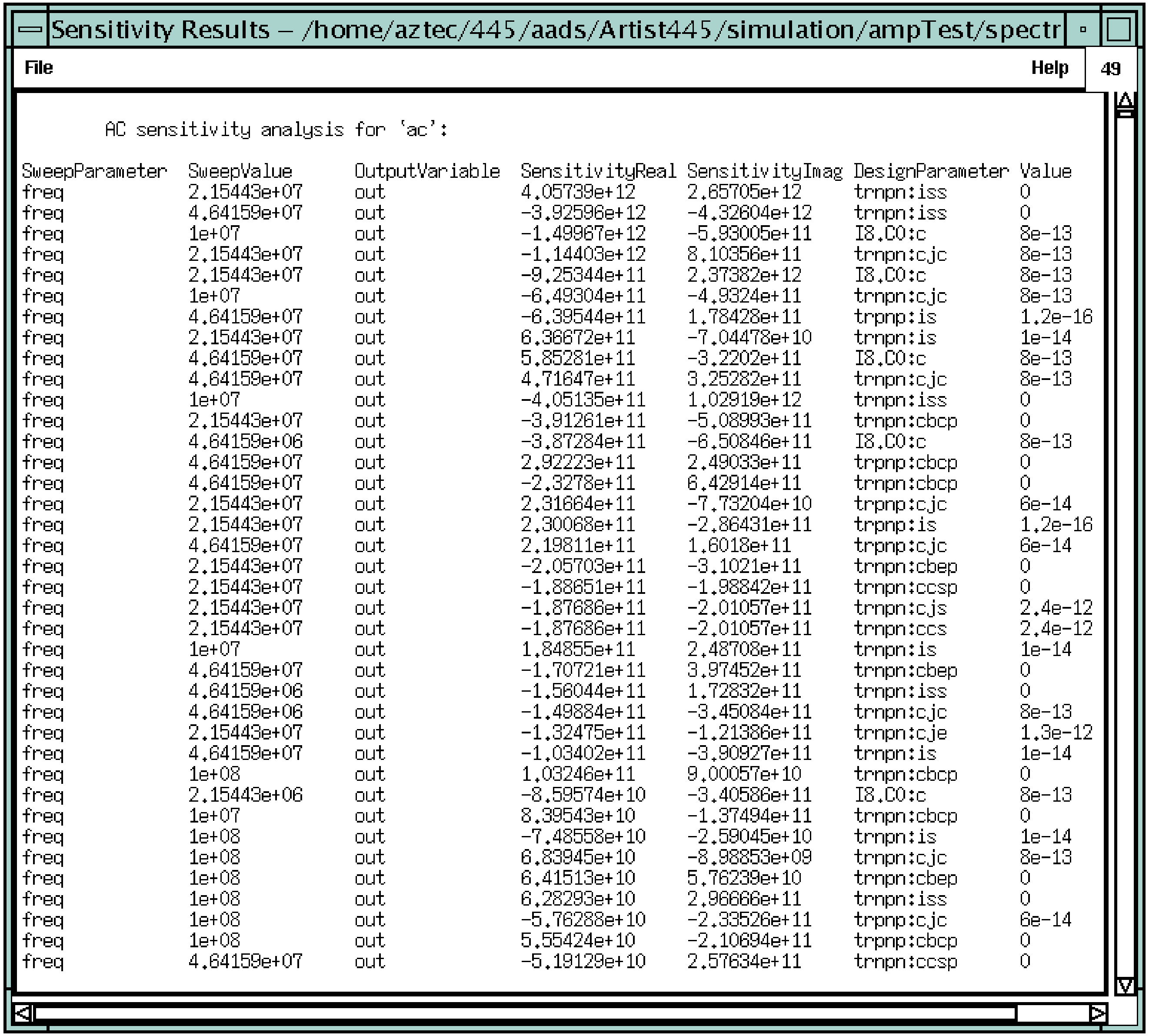
Outputs Provides a List box of the selected outputs. Highlight one or more of these outputs using the mouse. Default: no outputs.

Delete Delete any of the highlighted outputs in the list box.

Clear Removes all outputs from the list box.

Enabled Enables the Sensitivity Analysis in the Simulation window.

Viewing Sensitivity Results


Sensitivity Results – /home/aztec/445/aads/Artist445/simulation/ampTest/spectr
File
Help
49

```
      AC sensitivity analysis for 'ac':

SweepParameter  SweepValue      OutputVariable  SensitivityReal SensitivityImag DesignParameter Value
freq            2.15443e+07     out             4.05739e+12     2.65705e+12     trnpn:iss       0
freq            4.64159e+07     out             -3.92596e+12    -4.32604e+12    trnpn:iss       0
freq            1e+07           out             -1.49967e+12    -5.93005e+11    I8.C0:c         8e-13
freq            2.15443e+07     out             -1.14403e+12    8.10356e+11     trnpn:c.jc      8e-13
freq            2.15443e+07     out             -9.25344e+11    2.37382e+12     I8.C0:c         8e-13
freq            1e+07           out             -6.49304e+11    -4.9324e+11     trnpn:c.jc      8e-13
freq            4.64159e+07     out             -6.39544e+11    1.78428e+11     trpnp:is        1.2e-16
freq            2.15443e+07     out             6.36672e+11     -7.04478e+10    trnpn:is        1e-14
freq            4.64159e+07     out             5.85281e+11     -3.2202e+11     I8.C0:c         8e-13
freq            4.64159e+07     out             4.71647e+11     3.25282e+11     trnpn:c.jc      8e-13
freq            1e+07           out             -4.05135e+11    1.02919e+12     trnpn:iss       0
freq            2.15443e+07     out             -3.91261e+11    -5.08993e+11    trnpn:cbcp      0
freq            4.64159e+06     out             -3.87284e+11    -6.50846e+11    I8.C0:c         8e-13
freq            4.64159e+07     out             2.92223e+11     2.49033e+11     trpnp:cbcp      0
freq            4.64159e+07     out             -2.3278e+11     6.42914e+11     trnpn:cbcp      0
freq            2.15443e+07     out             2.31664e+11     -7.73204e+10    trpnp:c.jc      6e-14
freq            2.15443e+07     out             2.30068e+11     -2.86431e+11    trpnp:is        1.2e-16
freq            4.64159e+07     out             2.19811e+11     1.6018e+11      trpnp:c.jc      6e-14
freq            2.15443e+07     out             -2.05703e+11    -3.1021e+11     trnpn:cbep      0
freq            2.15443e+07     out             -1.88651e+11    -1.98842e+11    trnpn:ccsp      0
freq            2.15443e+07     out             -1.87686e+11    -2.01057e+11    trnpn:c.js      2.4e-12
freq            2.15443e+07     out             -1.87686e+11    -2.01057e+11    trnpn:ccs       2.4e-12
freq            1e+07           out             1.84855e+11     2.48708e+11     trnpn:is        1e-14
freq            4.64159e+07     out             -1.70721e+11    3.97452e+11     trnpn:cbep      0
freq            4.64159e+06     out             -1.56044e+11    1.72832e+11     trnpn:iss       0
freq            4.64159e+06     out             -1.49884e+11    -3.45084e+11    trnpn:c.jc      8e-13
freq            2.15443e+07     out             -1.32475e+11    -1.21386e+11    trnpn:c.je      1.3e-12
freq            4.64159e+07     out             -1.03402e+11    -3.90927e+11    trnpn:is        1e-14
freq            1e+08           out             1.03246e+11     9.00057e+10     trnpn:cbcp      0
freq            2.15443e+06     out             -8.59574e+10    -3.40586e+11    I8.C0:c         8e-13
freq            1e+07           out             8.39543e+10     -1.37494e+11    trnpn:cbcp      0
freq            1e+08           out             -7.48558e+10    -2.59045e+10    trnpn:is        1e-14
freq            1e+08           out             6.83945e+10     -8.98853e+09    trnpn:c.jc      8e-13
freq            1e+08           out             6.41513e+10     5.76239e+10     trnpn:cbep      0
freq            1e+08           out             6.28293e+10     2.96666e+11     trnpn:iss       0
freq            1e+08           out             -5.76288e+10    -2.33526e+11    trpnp:c.jc      6e-14
freq            1e+08           out             5.55424e+10     -2.10694e+11    trpnp:cbcp      0
freq            4.64159e+07     out             -5.19129e+10    2.57634e+11     trnpn:ccsp      0
```

Results—Print—Sensitivities

Viewing Sensitivity Results

To view the simulation results, select **Results—Print—Sensitivities** in the Simulation window. This is the only way to view Sensitivity simulation results.

The results window displays the analysis results, sorted by the magnitude of the real components of the sensitivities (SensitivityReal). The results generated from Spectre are the actual sensitivity values. They have not been modified or normalized in any way.

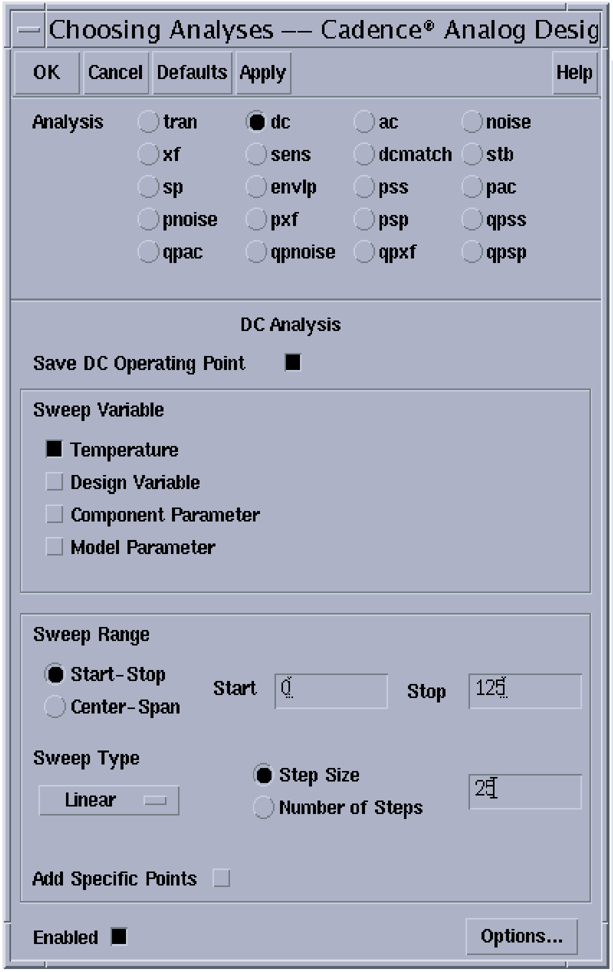
The sensitivities indicate the relative impact a change in a design parameter, such as trnpn:iss (the saturation current of the npn model file used to simulate the npn transistor) has on the output variable selected, the node *out*. This data displayed is valid for the AC analysis that was run in this example. The value of the frequency (SweepValue) at each measurement is displayed.

The magnitude of the sensitivity is not as important as the relative value to other sensitivities. For example, a design parameter with a sensitivity of 4.05739e+12 will impact the output variable by four orders of magnitude greater than a parameter with a sensitivity of 4.05739e+8.

The sensitivities are also important in regards to the base value of the parameter being changed. This can lead to giving Normalized Sensitivity as a possible addition to sensitivity results.

There is a Sensitivity option used to specify the filename for the Spectre sensitivity results. Select **Simulation—Options—Analog** to display the Simulator Options form. Scroll down to the SENSITIVITY OPTIONS section. Optionally, set the value of **sensfile**, which has a default value of *../psf/sens.output*. This file is displayed with data in a sorted format, as shown above.

Spectre Sweep Feature



Analysis ○ tran ● dc ○ ac ○ noise ○ xf ○ sens ○ dcmatch ○ stb ○ sp ○ envlp ○ pss ○ pac ○ pnoise ○ pxf ○ psp ○ qpss ○ qpac ○ qpnoise ○ qpxf ○ qpsp

DC Analysis

Save DC Operating Point ■

Sweep Variable

■ Temperature
□ Design Variable
□ Component Parameter
□ Model Parameter

Sweep Range

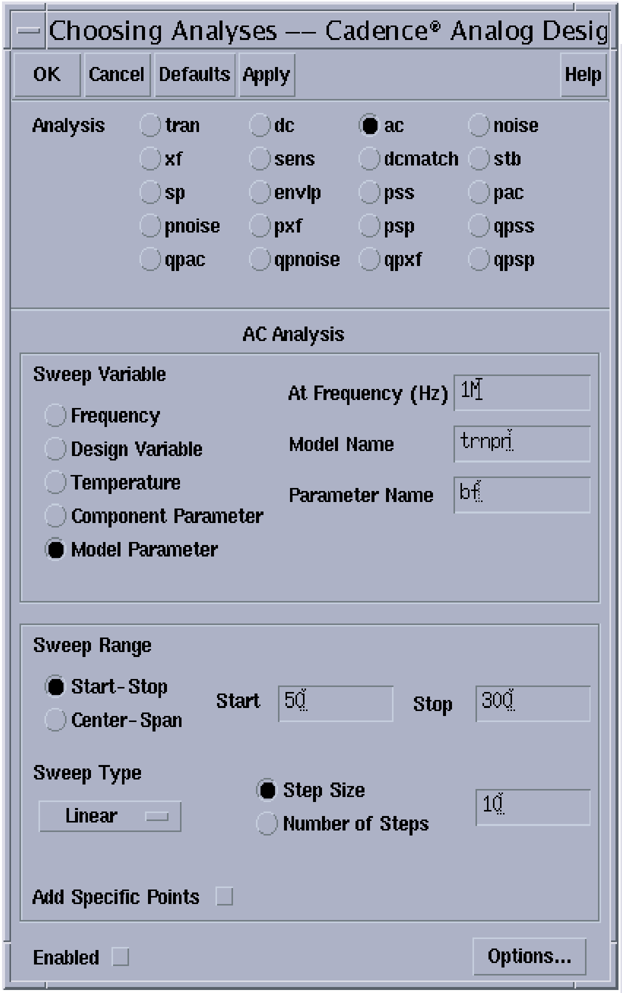
● Start-Stop Start 0 Stop 125
○ Center-Span

Sweep Type Linear ● Step Size 25
○ Number of Steps

Add Specific Points □

Enabled ■

Temperature Sweep
(DC Analysis)



Analysis ○ tran ○ dc ● ac ○ noise ○ xf ○ sens ○ dcmatch ○ stb ○ sp ○ envlp ○ pss ○ pac ○ pnoise ○ pxf ○ psp ○ qpss ○ qpac ○ qpnoise ○ qpxf ○ qpsp

AC Analysis

Sweep Variable

○ Frequency
○ Design Variable
○ Temperature
○ Component Parameter
● Model Parameter

At Frequency (Hz) 1M
Model Name trnpn
Parameter Name bf

Sweep Range

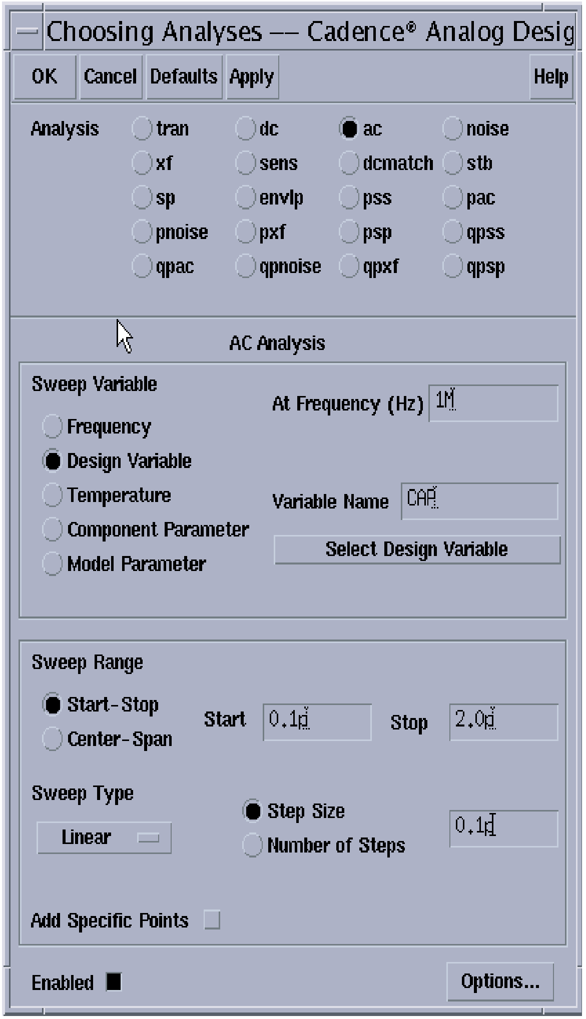
● Start-Stop Start 50 Stop 300
○ Center-Span

Sweep Type Linear ● Step Size 10
○ Number of Steps

Add Specific Points □

Enabled □

Model Parameter Sweep
(AC Analysis)



Analysis ○ tran ○ dc ● ac ○ noise ○ xf ○ sens ○ dcmatch ○ stb ○ sp ○ envlp ○ pss ○ pac ○ pnoise ○ pxf ○ psp ○ qpss ○ qpac ○ qpnoise ○ qpxf ○ qpsp

AC Analysis

Sweep Variable

○ Frequency
● Design Variable
○ Temperature
○ Component Parameter
○ Model Parameter

At Frequency (Hz) 1M
Variable Name CAP

Sweep Range

● Start-Stop Start 0.1p Stop 2.0p
○ Center-Span

Sweep Type Linear ● Step Size 0.1p
○ Number of Steps

Add Specific Points □

Enabled ■

Design Variable Sweep
(AC Analysis)

Spectre Sweep Feature

The Spectre simulator interface offers a powerful yet simple way to sweep parameters.

The following sweeps are available:

- Frequency (ac, noise, xf (transfer), sp (s-parameter) analyses)
- Design Variable (dc, ac, noise, xf, sp analysis)
- Temperature (dc, ac, noise, xf, sp Analyses)
- Component Parameters (dc, ac, noise, xf, sp analyses)
- Model Parameters (dc, ac, noise, xf, sp analyses)

For simple sweeps, the Spectre simulator provides a quick method of calculating and viewing data over a given range.

Introduction to Stability Analysis

- Stability analysis is a small-signal analysis that is available for Spectre only.
- It can be used with Spectre standalone, or with the Analog Design Environment, to:
  - — Verify the stability of feedback circuits.
  - — Check circuit stability with margin information.
  - — Easily measure loop gain, phase margin, and gain margin.
- Results can be printed in the Results Display Window and plotted in the Waveform Window.
- Two algorithms are available depending on the probe parameter specified: loop-based and device-based.
  - — The loop-based algorithm produces accurate stability information for circuit design in which a critical wire can be identified to break all feedback loops.
  - — The device-based algorithm produces accurate stability information for a circuit design in which a critical controlled source can be identified such that nulling this source renders the whole network passive.

Introduction to Stability Analysis

Loop gain calculation for a specific feedback loop or active device at the given dc operating point.

Gain Margin

The gain margin is defined to be the amount of magnitude in decibels of the loop gain below the 0 dB level at the frequency for which the phase is 0 degrees.

Loop Gain

The loop gain, a function of frequency, is defined as the gain around the feedback loop when the loop is virtually broken at any point. It is the negative of Bode’s return ratio.

Phase Margin

The phase margin is defined to be the amount of phase in degrees of the loop gain above 0 degrees at the frequency for which the gain is 0 dB.

Two algorithms: loop-based and device-based, are available in Spectre for small-signal stability analysis. Both algorithms are based on the calculation of Bode’s return ratio. Loop gain waveform, gain margin, and phase margin are the analysis output.

Loop-Based Algorithm

- Calculates the true loop gain.
- Invoked when a probe parameter points to a current probe or zero-DC-valued voltage source.
- Current probe or zero-DC-valued voltage source is placed on the feedback loop.
- Provides accurate stability information for single loop circuits and for multiloop circuits with a critical wire.
- Multiloop circuit can be stable if all individual loops have reasonable stability margins.
- Determines the stability of the whole network as long as all nested loops are stable.

Loop-Based Algorithm

The probe parameter must be specified to perform stability analysis. When the probe parameter points to a current probe or voltage source instance, the loop-based algorithm is invoked; when it points to a supported active device instance, the device-based algorithm is invoked.

The gain margin and phase margin are automatically determined from the loop gain waveform by detecting zero-crossing in the gain plot and phase plot.

The loop-based algorithm requires a probe component—current probe or zero-DC-valued voltage source—being placed on the feedback loop to identify and characterize the particular loop of interest. The introduction of the probe component does not change any of the circuit characteristics. There is no special requirement on the polarity configuration of the probe component.

The loop-based algorithm provides accurate stability information for single-loop and multiloop circuits, provided that a probe component can be placed on a critical wire to break all loops.

For a general multiloop circuit, such a critical wire may not be available. The loop-based algorithm can be only performed on individual feedback loops to ensure they are stable. Although the stability of all feedback loops is a necessary condition for the whole circuit to be stable, the multiloop circuit tends to be stable if all individual loops are associated with reasonable stability margins.

The Device-Based Algorithm

- Calculates the loop gain around a particular active device.
- Invoked when a probe parameter points to a supported active device instance.
- Used for designs in which local feedback loops cannot be neglected.
- Can be used to ensure all local loops are stable.
- Local feedback loops are not accessible from the schematic or netlist level to insert the probe component.
- Nulling dominant controlled source renders the active device to be passive.

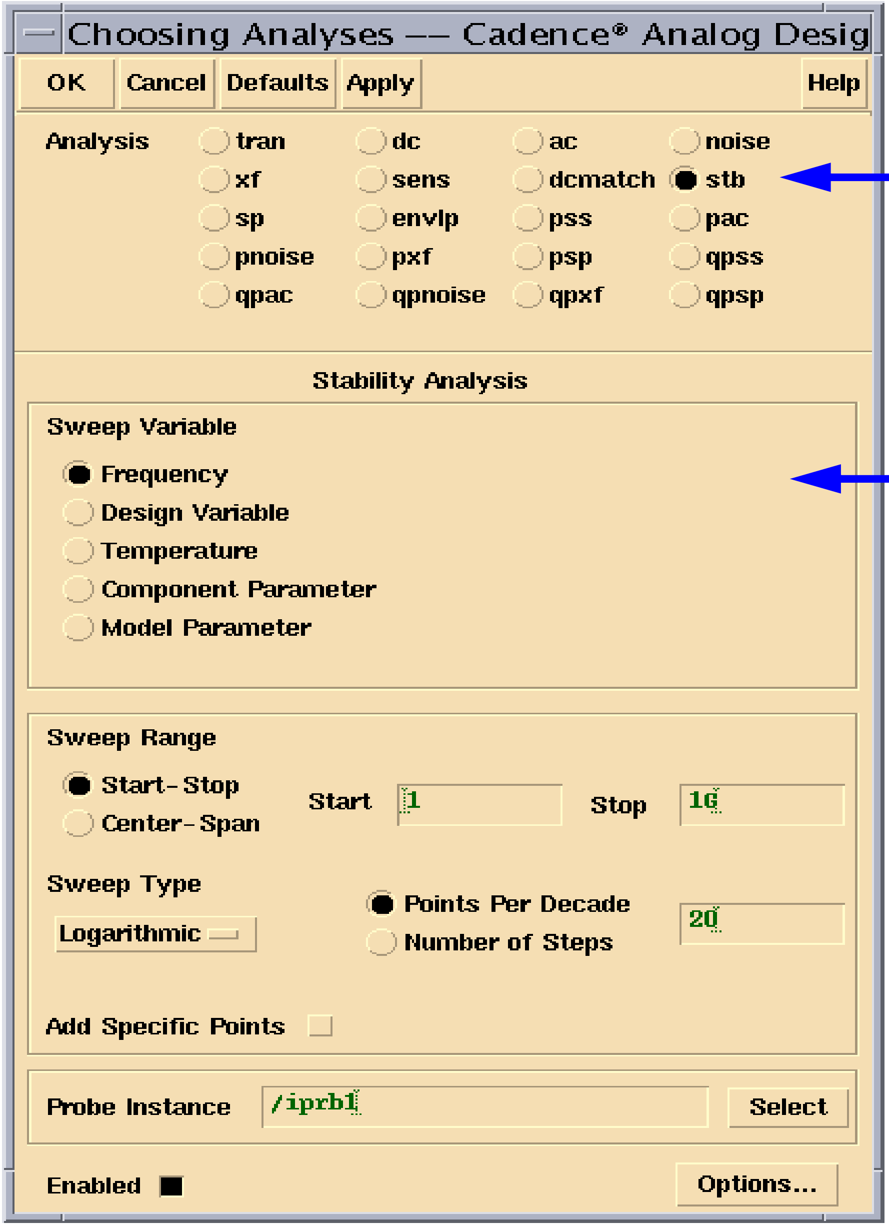
The supported active device and its dominant gain source are summarized below. The device-based algorithm produces accurate stability information for a circuit in which a critical active device can be identified such that nulling the dominant gain source of this device renders the whole network to be passive. Examples are multistage amplifier, single-transistor circuit, and S-parameter characterized microwave component.

The Device-Based Algorithm

Component	Dominant Controlled Source	Description
b3soipd	gm	Common-source transconductance
bjt	gm	Common-emitter transconductance
bsim1,2,3,3v3	gm	Common-source transconductance
btasoi	gm	Common-source transconductance
cccs	gain	Current gain
ccvs	rm	Transresistance
ekv	gm	Common-source transconductance
gaas	gm	Common-source transconductance
hbt	dice_dvbe	Intrinsic dIce/dVbe
hvmos	gm	Common-source transconductance
jfet	gm	Common-source transconductance
mos0,1,2,3	gm	Common-source transconductance
tom2	gm	Common-source transconductance
vbic	dic_dvbe	Intrinsic dIc/dVbe
vccs	gm	Transconductance
vcvs	gain	Voltage gain

Starting Stability Analysis

In the simulation window, select: **Analyses—Choose...** .



Analysis
○ tran ○ dc ○ ac ○ noise
○ xf ○ sens ○ dcmatch ● stb
○ sp ○ envlp ○ pss ○ pac
○ pnoise ○ pxf ○ psp ○ qpss
○ qpac ○ qpnoise ○ qpxf ○ qpsp

Stability Analysis

Sweep Variable
● Frequency
○ Design Variable
○ Temperature
○ Component Parameter
○ Model Parameter

Sweep Range
● Start-Stop
○ Center-Span
Start 1 Stop 1G

Sweep Type
Logarithmic
● Points Per Decade 20
○ Number of Steps

Add Specific Points □

Probe Instance /iprb1

Enabled ■

In the Choosing Analyses form select the **stb** button.

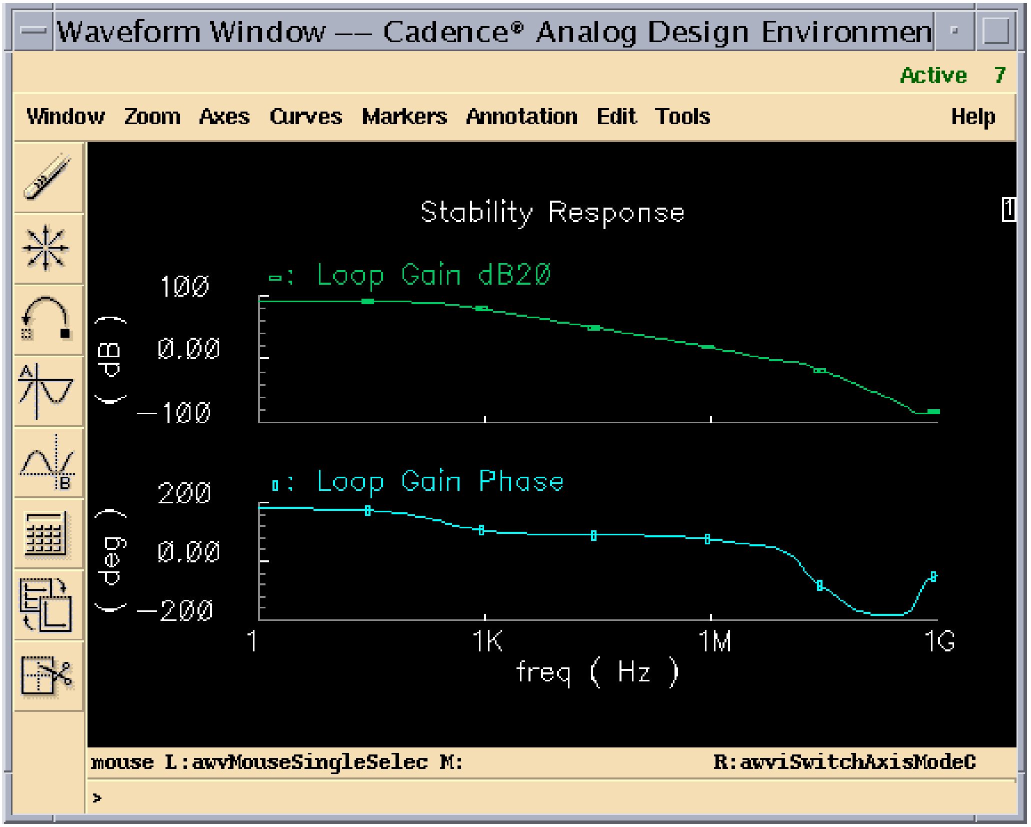
Sweep variables allowed in stability analysis.

Starting Stability Analysis

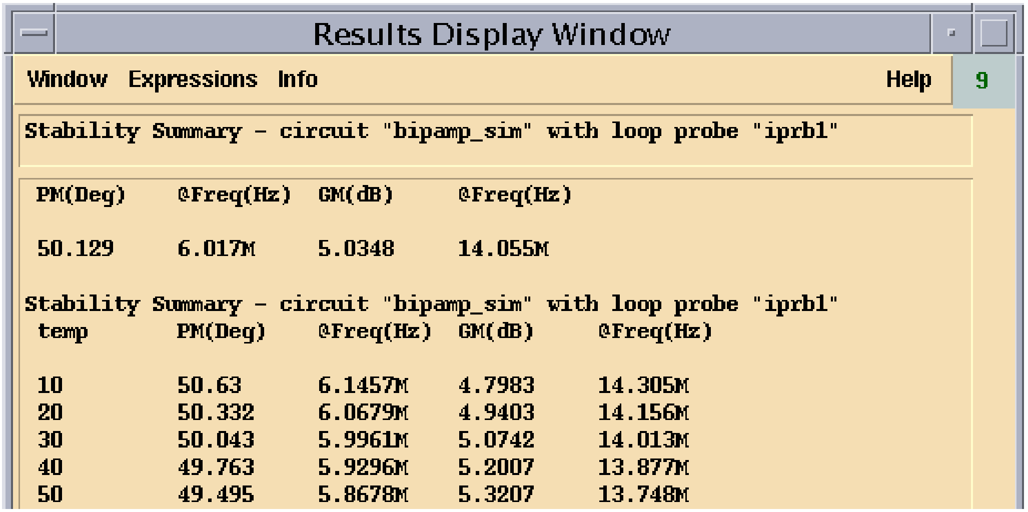
Start the stability analysis from the simulation widow by clicking the **Choose Analyses** icon or selecting **Analyses—Choose** from the menu bar. When the Choosing Analyses form appears, select the **stb** button. The stability analysis is a small-signal ac analysis. As such, frequency is a sweep variable. In addition, design variables, temperature, and component parameters can be swept.

Stability Analysis Results

Results — Direct Plot — Main Form



Results — Print—Stability Summary



```
Stability Summary - circuit "bipamp_sim" with loop probe "iprb1"

 PM(Deg)     @Freq(Hz)  GM(dB)      @Freq(Hz)

 50.129      6.017M     5.0348      14.055M

Stability Summary - circuit "bipamp_sim" with loop probe "iprb1"
 temp        PM(Deg)    @Freq(Hz)   GM(dB)      @Freq(Hz)

 10          50.63      6.1457M     4.7983      14.305M
 20          50.332     6.0679M     4.9403      14.156M
 30          50.043     5.9961M     5.0742      14.013M
 40          49.763     5.9296M     5.2007      13.877M
 50          49.495     5.8678M     5.3207      13.748M
```

Stability Analysis Results

After successful stability analysis, you can get a plot of loop gain by selecting **Results—Direct Plot—Main Form...** from simulation window and selecting “Loop Gain” button and clicking “Plot” button in the “Direct Plot Form”. In this form, click on “Add To Outputs” button to add any function or expression (Loop Gain, Phase Margin, Gain Margin, etc.) to “Outputs” section of the simulation window.

You can use markers in the Waveform Window to manually verify your gain margin and phase margin results.

After successful stb analysis, you can view stability summary by selecting **Results—Print—Stability Summary...** from the simulation window. A Results Display Window and Stability Summary window will open.

In the Stability Summary window, choose “Phase Margin”, “Gain Margin”, or “Both” and click “Apply” or “OK”. This displays phase margin, gain margin, and the frequency values at which these values were measured. This information is very important to know if your circuit is stable or not. You can find out whether your circuit is stable or not by doing poles and zeroes analysis; however, poles and zeroes analysis will not provide the margin information that the stability analysis provides.

Labs

Lab 5-1 The Waveform Calculator

Lab 5-2 Managing Simulation Results

Lab 5-3 Managing Simulation Data with the Results Browser

Lab 5-4 Viewing Circuit Conditions

Lab 5-5 Using the Spectre Sweep Features

Lab 5-6 Stability Analysis

Labs

Lab Reference Materials

Any import key in the calculator, such as *vt*, *it*, *wave*, when pressed will prompt to probe in the design entry window. When finished graphically probing, press the **Esc** key with the cursor in the design entry window to cancel the probing function. Failure to cancel the probing function before starting a different one, will temporarily disable the system. This can be solved by cancelling the current probing functions using the **Esc** key.

This condition violates the *Nest Limit* of the environment, and a warning appears in the CIW. To avoid this, change the *Nest Limit* with the **Options—User Preferences** form through the CIW.

Currently, the conditional search capability only works for *DC operating* conditions.

Lab Reference Materials

Module 6: SKILL and OCEAN

Topics in this module

- Overview of SKILL and OCEAN
- SKILL in the DFII Environment
- Basic SKILL statements
- Introduction to OCEAN
- Types of commands
- Sample OCEAN script
- Data access commands
- Plotting commands
- Aliases
- Running OCEAN interactively
- Creating OCEAN scripts in the Analog Design Environment
- Loading OCEAN scripts

Terms and Definitions

OCEAN	Acronym for Open Command Environment for ANalysis; Text-based command language for running simulations.
setup commands	Commands that control input data to the host simulator.
run commands	Commands that run the simulator.
data access commands	Commands that are used to get simulator results.
OCEAN aliases	Abbreviated text that activates specific OCEAN commands.

Overview of SKILL and OCEAN

Introduction

This chapter provides some basic instruction in the use and applications of **SKILL** programming language.

This chapter provides instruction on SKILL because:

- SKILL is used as the fundamental program language of DFII and the Analog Design Environment.
- The OCEAN Command Language is based on SKILL and many SKILL and OCEAN commands are similar and interchangeable.

This chapter also provides basic instruction on the use and applications of the **OCEAN** Command Language. This information on OCEAN is used to:

- Run long simulations in batch mode.
- Run simulations from remote non-graphic terminals
- Setup and run a large set of simulations, obtain data from the simulation runs, and evaluate the simulation data automatically.
- Use the Analog Design Environment to create OCEAN scripts to run simulations at a later time.

Overview of SKILL and OCEAN

In this chapter you will be introduced to the SKILL program language and to the OCEAN command language. The introduction to SKILL is to demonstrate that SKILL is the fundamental program language of the DFII Environment. Most of the user interface windows, features, and forms within DFII are written in SKILL.

This chapter also provides instruction on using OCEAN. The OCEAN syntax is based on SKILL, and so many of the SKILL and OCEAN commands are identical. The OCEAN command language is included with the installation of Analog Design Environment. OCEAN provides an alternate method to set up simulations, run the simulation, access the simulation data, and then process the data.

Introduction to SKILL

- SKILL is a graphics-based program language based on Lisp.
- DFII and **most** features and applications of Analog Design Environment are written in SKILL code.
- Features of the Analog Design Environment and related tools can be customized using SKILL.
- The OCEAN command language is based on SKILL.
- Many SKILL and OCEAN commands are interchangeable between the two domains.
- Training in SKILL is available from Cadence Education Services.
- Both classroom and internet training (iLS) are available.
- There are SKILL development tools available in your installation of Analog Design Environment.

Introduction to SKILL

SKILL is a graphical base programming language with thousands of commands. DFII and **most** features and applications of Analog Design Environment are written in SKILL code.

In the classroom environment for this class some students ask:

- “How can I change this feature to do *this* or *that*?” or
- “How do I change the appearance of the user interface?”

Very often the answer to such questions is “SKILL”.

SKILL Programming Language Class is taught by Cadence Education Services as a 5-day class. An Internet Learning Series (iLS) version of the class is also provided through Cadence Education Services.

Using SKILL Commands

There are numerous ways to execute SKILL commands and programs.

- The command line of the CIW accepts SKILL commands.
- The command line of the CIW executes SKILL programs.
- The buffer of the Waveform Calculator will evaluate mathematical expression written in SKILL.
- There is a SKILL environment that is started by typing “skill” in a unix window.

SKILL support

- The Finder utility is a SKILL dictionary for locating the syntax and usage SKILL commands.
  - Start the Finder from the CIW by selecting: **Tools—SKILL Development**, then select the **Finder** button.
- Other SKILL support features include SKILL Development Tool and CDSDoc.

Using SKILL Commands

Basic SKILL Statements

- SKILL commands can be entered into and evaluated on the command line of the Command Interpreter Window.
- You can enter valid SKILL syntax into the CIW. For example, on the command line enter:

  2 + 2

  4 ... SKILL response
- Now enter:

  pi=3.14159

  3.14159 ... SKILL response

  then enter:

  x= 1/(2 * pi * 1M * 1p) ... the reactance of a 1pf capacitor at 1MHz

  159155.1 ... SKILL response
- SKILL evaluates the command line, assigns values to variables, and evaluates expressions.

Basic SKILL Statements

SKILL easily handles setting of variables and expressions. SKILL can also be used to set up arrays.

Parentheses and Double Quotes

There are common SKILL syntax characters. These are also used in OCEAN scripts:

- Parentheses [()]

  Example: *path("./simulation1/schematic/psf")*

  No space! This very important.

- Double quotes [""]

  Example: ***path("./simulation2/schematic/psf")***

Parentheses and Double Quotes

Parentheses

Parentheses surround the arguments to the command. The command name is followed immediately by the left parenthesis, with no intervening space as in this example:

```
path( "./simulation1/schematic/psf" "./simulation2/schematic/psf" )
```

This is correct. There is no space between the word path and the first parenthesis.

```
path  ( "./simulation1/schematic/psf" "./simulation2/schematic/psf" )
```

This example is **incorrect**. The space after the command name causes a syntax error.

Double Quotes

Use double quotes to surround string values. A string value is a sequence of characters, such as “abc”. In the following example, the directory names provided to the path command are strings, which must be surrounded by double quotes:

```
path( "./simulation1/schematic/psf" "./simulation2/schematic/psf" )
```

In *OCEAN*, a SKILL convention indicates when an argument must be a string. For the prefix *t_*, substitute a string value (surrounded by double quotes) for the argument as in this example:

```
desVar( t_desVar1 g_value1 t_desVar2 g_value2)
```

This example includes two string values that must be supplied: **t_desVar1** and **t_desVar2**.

More information is in the *OCEAN Reference* manual.

Single Quote and Question Mark

- Single Quotes

  Example: *analysis(‘tran ....)*

- Question Mark

  Example: *analysis(‘tran ?stop 1u)*

Single Quote and Question Mark

Single Quote

A single quote indicates that an item is a symbol. Symbols in SKILL correspond to constant enumerated values in the *C* language. In the context of OCEAN, there are predefined symbols used to avoid errors. The simulator being used also has predefined symbols.

In example below, *tran* is a predefined symbol and must be preceded by a single quote. Determine the valid symbols for a command by checking the valid values for the command's arguments.

```
analysis( 'tran .... )
```

Another example is the predefined *save* command. The *'v* symbol indicates that the item to save is the voltage on a net.

```
save( 'v "net1" )
```

In addition, use the *selectResult(s_resultsName)* command, where *s_resultsName* = **‘dc**, **‘tran**, or **‘ac**.

Question Mark

A question mark indicates an optional keyword argument, which is the first part of a keyword parameter. For a keyword parameter, the first component is the keyword, which has a question mark in front of it. The second component is the value being passed, and immediately follows the keyword.

In the example, *analysis('tran ?stop 1u),* all the keyword/value pair arguments to the analysis command are optional, except the **‘tran** keyword.

Introduction to OCEAN

Open **C**ommand **E**nvironment for **AN**alysis

- OCEAN is a product included in the Analog Design Environment.
- Use OCEAN for the following tasks:
  - To create scripts to run batch mode simulations.
  - To run parametric, Corners, Monte Carlo, and Optimization analyses.
  - To run long simulations without starting the graphical user interface.
  - To run simulations from a non-graphical, remote terminal.
- OCEAN is based on the SKILL programming language.

  Scripts are created automatically within the Analog Design Environment. These scripts can be saved, modified, and used to run batch simulations.
- After the design has been debugged in the Analog Design Environment, use OCEAN to test your circuit under a variety of conditions.

Introduction to OCEAN

The Open Command Environment for ANalysis (OCEAN) sets up, simulates, and analyzes circuit data. OCEAN is a text-based process that runs from a UNIX shell by starting the **ocean** executable, or from the Command Interpreter Window (CIW). Type in OCEAN commands in an interactive session, or create scripts containing your commands, then load those scripts into OCEAN. Use OCEAN with any simulator integrated into the Analog Design Environment software.

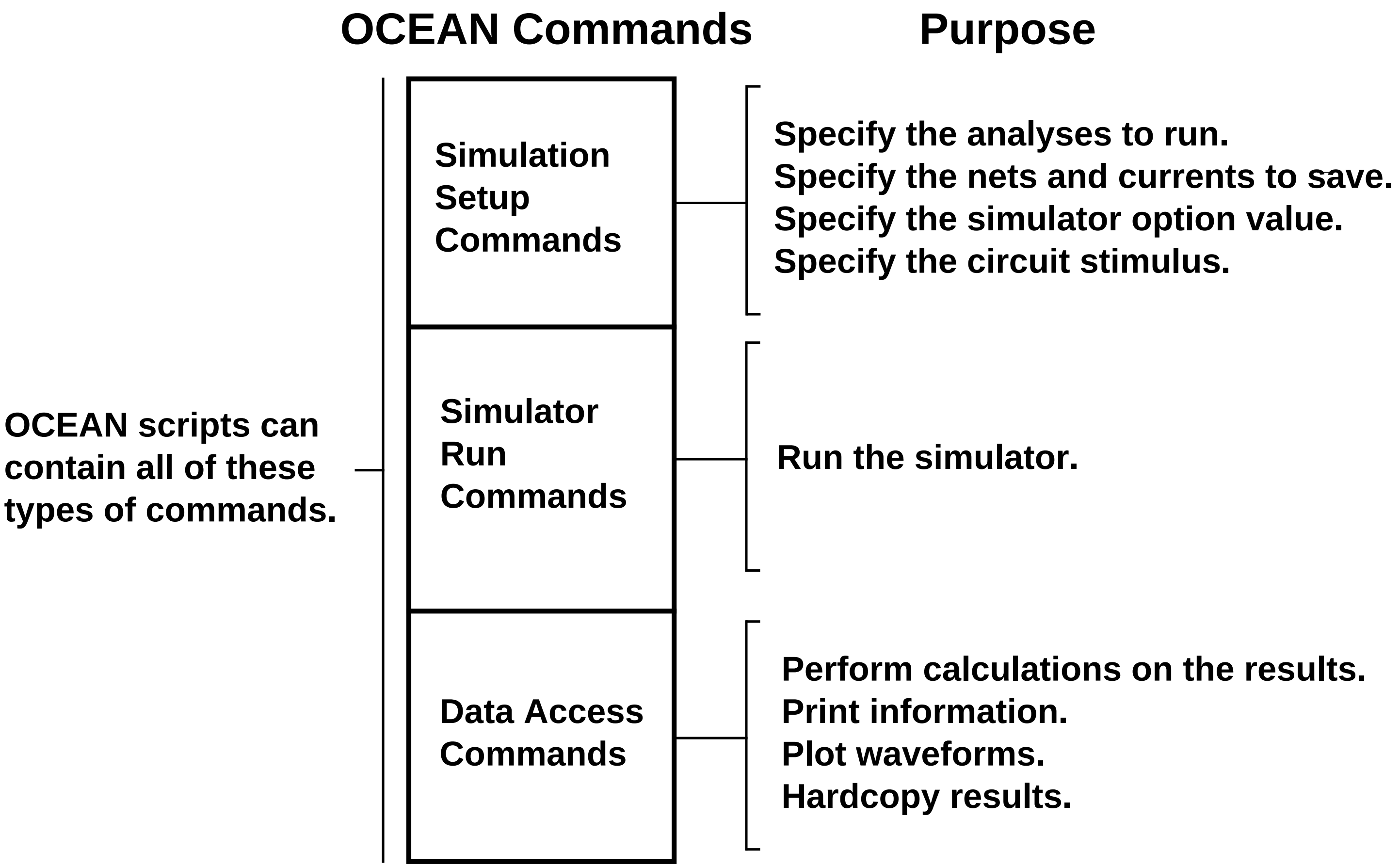
Typically, you use the Analog Design Environment when creating your circuit (in the Composer software) and when interactively debugging the circuit. After the circuit has the required performance, use OCEAN to run your scripts and test the circuit under a variety of conditions. After making changes to your circuit, run your scripts again.

Use OCEAN to:

- Create scripts that can run repeatedly to verify circuit performance.
- Run longer analyses such as parametric, Monte Carlo, Optimization, and Corners analyses more effectively.
- Run long simulations in OCEAN without the graphics interface of the Analog Design Environment.
- Run simulations from a nongraphic, remote terminal.

OCEAN is based on the SKILL programming language and uses SKILL syntax. Use all the SKILL language commands in OCEAN. These commands include *if* statements, *case* statements, *for* loops, *while* loops, *read* commands, *print* commands, and so on.
The most frequently used SKILL commands relevant to OCEAN are listed in the *OCEAN Reference* manual.

Types of OCEAN Commands



OCEAN Commands

Purpose

Simulation Setup Commands

Specify the analyses to run.
Specify the nets and currents to save.
Specify the simulator option value.
Specify the circuit stimulus.

Simulator Run Commands

Run the simulator.

Data Access Commands

Perform calculations on the results.
Print information.
Plot waveforms.
Hardcopy results.

OCEAN scripts can contain all of these types of commands.

Types of OCEAN Commands

Create OCEAN scripts to accomplish the full suite of simulation and data access tasks that you can perform in the Analog Design Environment.

An OCEAN script can contain three types of commands:

- Simulation setup commands
- Simulator run commands
- Data access commands

All the parameter storage format (PSF) information created by the simulator is accessible through the OCEAN data access commands. (The data access commands include all of the Analog Design Environment calculator functions.)

Sample OCEAN Script

```
simulator( 'spectre )
design("./simulation/ampTest/spectre/schematic/netlist/netlist")
resultsDir( "./simulation/ampTest/spectre/schematic" )
modelFile(
     ( "./Models/myModels.scs" “”)
)
analysis('ac ?start "100"  ?stop "150M"  ?dec "20"   )
desVar(   "CAP" .5p )
temp( 27 )
run()
selectResult( 'ac )
plot(getData("/out") )
```

Note: Specify the **resultsDir()** to set the directory where simulation data is saved. In the example above, resultsDir() points to the default location. When using the default, the resultsDir() expression is not needed.

Sample OCEAN Script

You can annotate the sample OCEAN script as follows:

```
simulator( 'spectre ) ->Choose simulator.
design("./simulation/ampTest/spectre/schematic/netlist/netlist")
->Specify schematic to be simulated. The netlist was created in Analog Design
Environment.
resultsDir( "./simulation/ampTest/spectre/schematic" )
->Specify location of psf directory that holds simulation results. In this
example, the default is specified as an argument to resultsDir(), so the command
is not necessary.
modelFile( ‘( "./Models/myModels.scs" ““)) ->Set Library Model File in
Simulation Environment.
analysis('ac ?start "100"  ?stop "150M"  ?dec "20"  ) ->Specify AC Analysis.
desVar(   "CAP" .5p ) ->Set the value of the design variable, CAP.
temp( 27 ) ->Set the simulation temperature.
run() ->Run the simulation.
selectResult( 'ac ) ->Select the AC analysis results.
plot(getData("/out") ) -> Generic command to plot the data. Try using
plot(vm("/out")) to plot the magnitude of the voltage at node “/out”.
```

Note: There are “easy” commands available for common analyses. For example, enter ***tran(0 100n 1n)*** for a transient analysis, ***ac(1 10000 “linear” 100)*** for an AC analysis, and ***dc(“r1” “r” 0 5 1)*** for a DC analysis. See the *OCEAN Reference Manual* for more examples and the general form of these commands.

OCEAN Help

Online help is available for all the OCEAN commands. In an OCEAN session, type the following:

```
ocnHelp( 'commandName )
```

For example, enter:

```
ocnHelp('analysis)
```

An explanation of the command and examples of use are returned.

For a list of all types of OCEAN commands, enter:

```
ocnHelp()
```

Additional help and samples of OCEAN scripts are located at:

```
<install_dir>/tools/dfII/samples/artist/OCEAN
```

Detailed documentation on OCEAN and OCEAN scripts is accessible through CDSDoc.

OCEAN Help

Online help is available for OCEAN commands. Enter **ocnHelp(‘analysis)** and note:

```
PROTOTYPEanalysis( s_analysisType

  [?<analysisOption1> g_analysisOptionValue1] ...
  [?<analysisOptionN> g_analysisOptionValueN]) => undefined/nil

DESCRIPTIONSpecifies the analysis to be simulated. Include as many analysis
  options as you want. Analysis options vary, depending on the
  simulator being used. To include an analysis option, replace
  <analysisOption1> with the name of the desired analysis option
  and include another argument to specify the value for the option.
  With an ac analysis, the first option/value pair might be
  [?from 0].
  Note: Some simplified commands are available for basic SPICE
  analyses, See the ac, dc, tran, and noise commands. Use
  ocnhelp( `analysis ) for more information on the analysis types
  for the simulator choosen.

EXAMPLE(S)
  analysis( 'ac ?start 1 ?stop 10000 ?lin 100 )

  For the Spectre simulator, specifies that an ac analysis is
  to be performed.

  analysis( 'tran ?start 0 ?stop 1u ?step 10n )
  Specifies that a transient analysis is to be performed.
```

Data Access Commands

Open simulation results and analyze data.

- Simulation does not have to have been run in the current session.
- Examples of data access commands include:

```
openResults( “./simulation/opamp/spectre/schematic/psf” )
results()
selectResult( ‘tran)
ocnPrint( v( “/net56”))
i( “/R1” )
plot(v(“/out”))
pv( “/Q19” “ib”)
```

- List of commands includes:

```
dataTypes, getData, i, noiseSummary, ocnPrint, openResults,
outputParams, outputs, pv, report, results, selectResults,
sweepNames, sweepValues, v
```

- Common arithmetic operators and all Calculator functions are available.

Data Access Commands

There are special commands available in OCEAN that allow you to access data after a simulation is run. These include: *dataTypes*, *getData*, *i*, *noiseSummary*, *ocnPrint*, *openResults*, *outputParams*, *outputs*, *pv*, *report*, *results*, *selectResults*, *sweepNames*, *sweepValues*, *and v*. A full list of commands and their usage is found in the *OCEAN Reference Manual*. Some of these commands function as follows:

```
openResults( “./simulation/opamp/spectre/schematic/psf” )
```

— Opens simulation results stored in PSF format in psf directory.

```
results()
```

— Returns a list of the type of results that can be selected.

```
selectResult( ‘tran)
```

— Selects Transient results.

```
ocnPrint( v( “/net56”))
```

— Prints the text data of the waveform net56.

```
i( “/R1” )
```

— Returns the current through R1.

```
plot(v(“/out”))
```

— plots node ‘out” in a waveform window.

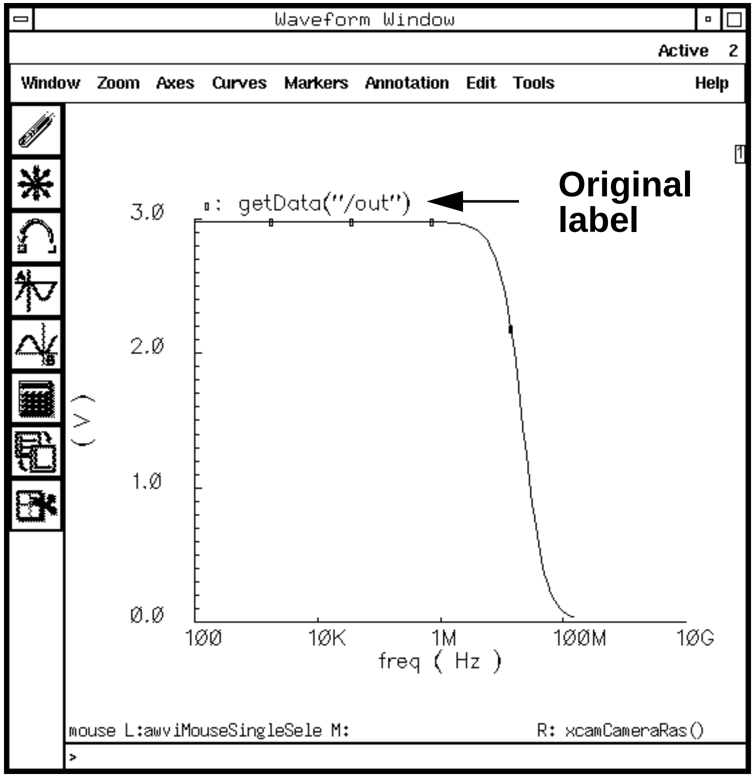
```
pv( “Q19” “ib”)
```

— Returns the value of the ib parameter for the Q19 component.

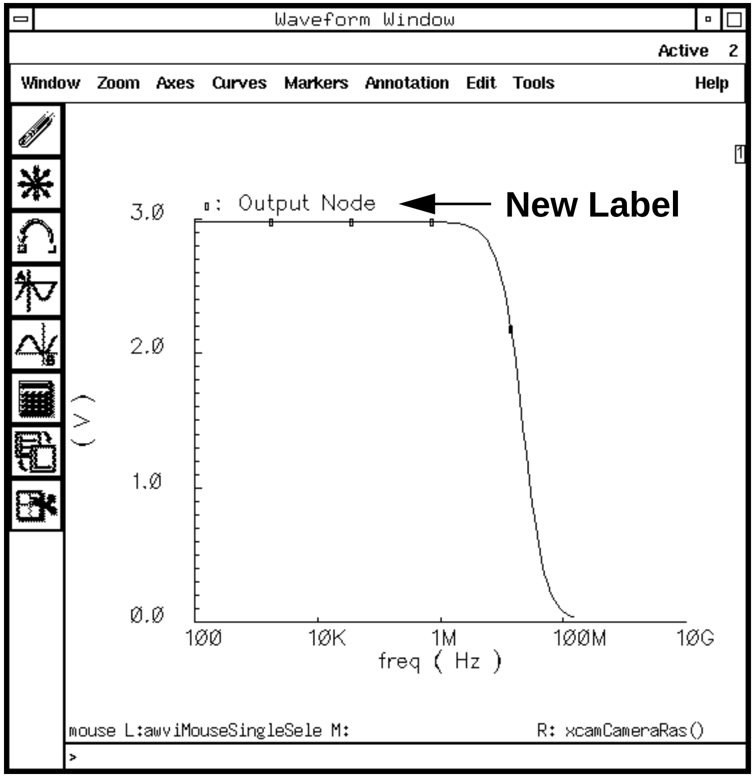
Plotting Commands

Use any Waveform Window SKILL command for plotting in OCEAN.

- By default, the Waveform Window is in overlay mode.
  Use the ***clearAll()*** command to erase the window.
- Use ***graphicsOn()*** and ***graphicsOff()*** to turn plotting on and off.
- Arithmetic operators and all Calculator functions are available.


Waveform Window
Active 2
Window Zoom Axes Curves Markers Annotation Edit Tools Help
3.0
2.0
1.0
0.0
(V)
▫: getData("/out")
100 10K 1M 100M 10G
freq (Hz)
mouse L:awviMouseSingleSele M: R: xcamCameraRas()
>

Original label


Waveform Window
Active 2
Window Zoom Axes Curves Markers Annotation Edit Tools Help
3.0
2.0
1.0
0.0
(V)
▫: Output Node
100 10K 1M 100M 10G
freq (Hz)
mouse L:awviMouseSingleSele M: R: xcamCameraRas()
>

New Label

Examples: **plot(getData("/out"))** **plot(getData("/out") ?expr list("Output Node"))**

Plotting Commands

Use any Waveform Window SKILL command for plotting in OCEAN. Plot any data that is in PSF format.

The *graphicsOff()* command disables the redrawing of the current Waveform Window. Use this command to freeze the Waveform Window display, send several plots to window, and then use *graphicsOn()* to unfreeze the window and display all of the plots at once.

Plotting commands include: *plot*, *newWindow*, *addSubwindow*, *clearSubwindow*, *clearAll*, *deleteSubwindow*, *deleteWaveform*, *displayMode*, *plotStyle*, *xLimit*, *yLimit*, *currentWindow*, *currentSubwindow*, *addWaveLabel*, *addWindowLabel*, *removeLabel*, *hardCopyOptions*, and *hardCopy*.

Details of plot commands are in the *OCEAN Reference* manual.

The example above illustrates how to replace the command, *plot(getData("/out")*, with the following expression:

```
plot(getData("/out") ?expr list("Output Node"))
```

This expression gives a specific name to a waveform that is plotted in the Waveform Window.

Also replace the expression with *plot(vm("/out"))*.

Available OCEAN Aliases

Aliases are available in OCEAN to simplify your scripts.

Alias	Syntax	Description
vm	mag(v(t_net))	Magnitude of voltage on the net
vdb	db20(v (t_net))	Power gain in decibels from net “in” to net “out”
vp	phase(v(t_net))	Phase of voltage on net
vr	real(v(t_net))	Real part of complex voltage
vim	imag(v(t_net))	Imaginary part of complex voltage
im	mag(i(t_component))	Magnitude of AC current
ip	phase(i(t_component))	Phase of AC current
ir	real(i(t_component))	Real part of complex number representing AC current
iim	imag(i(t_component))	Imaginary part of complex number representing AC current

Available OCEAN Aliases

The aliases shown above provide shortcuts to commonly used pairs of commands. All of these aliases operate on results previously selected with *selectResult*. However, an alias can be used on a different set of results as follows:

```
vm( t_net [?result  s_resultName] )
```

where *resultName* is the name of the data type for the particular analysis required.

Use an alias on results stored in a different directory as follows:

```
vm( t_net [?Resultsdir  t_resultsDir] [?result  s_resultName] )
```

where *resultsDir* is the name of a different directory containing PSF results, and *resultName* is the name of a data type contained in that directory. (If specifying another directory with *resultsDir*, specify the particular results with *resultName*.)

Running OCEAN Interactively

Run OCEAN from a UNIX prompt or from the CIW.

- In UNIX:

  Enter ***ocean*** from the UNIX prompt to start.

  This loads *awd.exe* and loads and reads the *.oceanrc* startup file. The *.oceanrc* file contains OCEAN commands that include alias definitions, and user-defined procedures and characterization scripts.

  An **ocean>** prompt appears in the UNIX window. Enter commands at the prompt.

  The *.cdsinit* file is not loaded.

  Type ***exit*** to quit OCEAN.

- In the CIW, through the Analog Design Environment:

  — Enter OCEAN commands at any time.

  — The *.oceanrc* is not loaded automatically. If it exists, load it by entering:

  ```
  load ".oceanrc"
  ```

Running OCEAN Interactively

Run OCEAN from a UNIX prompt or from the CIW. The primary method is to run it from a UNIX shell.

When entering ***ocean*** at a UNIX prompt, a series of messages are printed. This includes a note that the ***awd.exe*** code is loaded. This code drives the Waveform Display tool that drives the Waveform Window. Other messages indicate that context files are loaded to activate the Simulation Environment.

The *.oceanrc* file contains OCEAN commands that include alias definitions, and user-defined procedures and characterization scripts. For example, an *.oceanrc* file might contain:

```
alias h ocnHelp
alias d ocnDisplay
alias sr selectResults
alias op openResults
alias p ocnPrint
installDebugger()
```

Once this file is loaded, you can enter:

```
h( 'commandName)
```

instead of typing

```
ocnHelp('commandName)
```

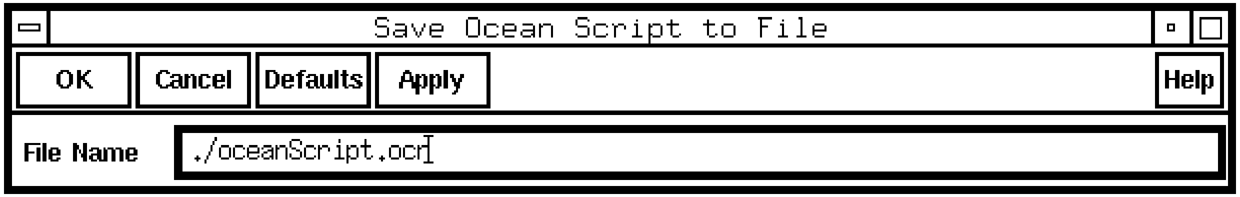
Using OCEAN in the Analog Design Environment is straightforward, because commands are directly entered into the CIW.

Creating OCEAN Scripts in ADE

It is easy to create OCEAN scripts in the Analog Design Environment.

- OCEAN commands of tasks performed in the Analog Design Environment are automatically saved in the *./simulation/design/simulator/schematic/netlist/simulatorX.ocn* files.

- In the Simulation Environment form, select **Session—Save Script...**



File Name ./oceanScript.ocn

  — Specify the file name of the script to create.

  — Only enabled analyses are saved.

  — The command saves the design, library model file, design variables, data directory, simulator options, plot set, etc.

Creating OCEAN Scripts in ADE

Automatically create an OCEAN script within the Analog Design Environment. The following example illustrates the difference between the script that is automatically created in the *./simulation/design/simulator/schematic/netlist/simulatorX.ocn* location, and the one created by the **Session—Save Script...** command.

For example, walk through the following design flow:

1. Start the Analog Design Environment software.
2. Specify DC analysis.
3. Select nets on schematic to save.
4. Run simulation.
5. Turn off DC analysis.
6. Select transient analysis.
7. Run simulation.
8. Save OCEAN script.

The *simulatorX.ocn* file in the *netlist* directory will contain the entire design session, including the DC analysis. The *oceanScript.ocn* (default name) will not have the DC analysis, because it was turned off before the script was created.

Loading OCEAN Scripts

Load OCEAN scripts from a UNIX window or from the CIW.

- In UNIX:

  ```
  ocean> load( “script_name.ocn”)
  ```

  or

  ```
  load “script_name.ocn”
  ```

- In UNIX, to run an OCEAN script and then have OCEAN quit, enter:

  ```
  unix> ocean  <  script_name.ocn
  ```

- In the CIW, enter:

  ```
  load( “script_name.ocn”)
  ```

  or

  ```
  load “script_name.ocn”
  ```

  This command loads and runs the OCEAN script.

Loading OCEAN Scripts

Once an OCEAN script exists, it is easy to load it and run it through the CIW or in a UNIX window.

The netlist specified with the ***design*** command needs to have already been created. It is recommended to use the netlist generated from the Analog Design Environment. Use netlists from other sources as long as they are in the same format as those created by the ADE simulation environment.

Any models, include files, stimulus files, or PWLF files must be in the locations set by the **path** command.

The scripts that are created automatically in the *../simulation/design/simulator/schematic/netlist/simulatorX.ocn* location or with the **Session—Save Ocean Script** command have the proper syntax to run in OCEAN.

Labs

Lab 6-1 Using an OCEAN Script to Run a Simple Simulation

Lab 6-2 Measuring PSRR and CMRR with OCEAN

Lab 6-3 Introduction to SKILL

Lab 6-4 SKILL Development Tools

Labs

Module 7: Parametric Analysis

Topics in this module

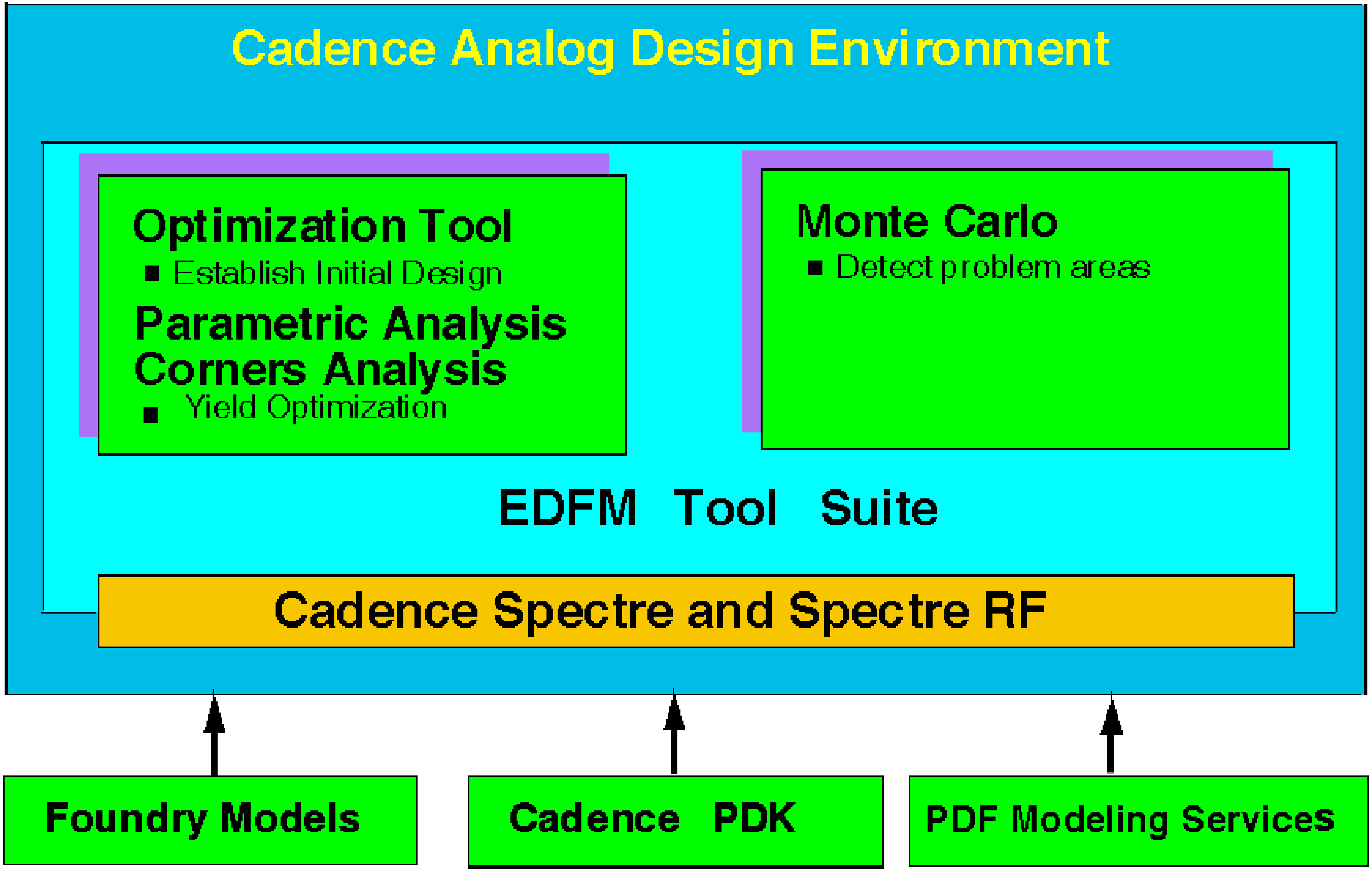
- Introduction to EDFM design tools
- Parametric analysis flow and methodology
- Parametric analysis environment
- Parametric plots
- Accessing data from parametric analysis
- Model parameters
- Running parametric analysis in OCEAN

Terms and Definitions

parametric analysis	Set of simulations sweeping one variable while stepping another.
inclusion list	A list of data points to be added to defined set.
exclusion list	A list of data points to be removed from a defined set.
EDFM	Electronic Design For Manufacturability
PDF	process description file
PDK	process design kit

Introduction to EDFM Design Tools

Parametric Analysis is one of set of software products called **EDFM** tools.



Cadence Analog Design Environment

Optimization Tool

- Establish Initial Design

Parametric Analysis

Corners Analysis

- Yield Optimization

Monte Carlo

- Detect problem areas

EDFM Tool Suite

Cadence Spectre and Spectre RF

Foundry Models

Cadence PDK

PDF Modeling Services

Introduction to EDFM Design Tools

EDFM is a set of design tools used to develop manufacturable products. In general, a circuit simulator is a software design tool for running a single analysis. The simulator performs calculations using the device models to predict behavior at each step of the simulation run. The simulator performs these calculations as directed by the input files. A simulator is used to design circuits, but it is also used run experiments and verify lab results.

Alternatively, **EDFM** tools are used to **complete** the design process. Instead of a single simulation run to verify circuit operation and performance at specific conditions, an EDFM analysis is a set of multiple simulations. The EDFM analysis verifies operation and performance over a specified range, or over multiple ranges, or over a set of specified of conditions.

EDFM tools feature the following properties:

- Multiple simulations.
- A graphical user interface for setup.
- An automated flow to alter the simulation input between runs.
- Specified ranges for parameters or design variables.
- Multiple pdf files.
- Special post processing tools to display results.
- EDFM tools are selected under the **Tools** menu of the Analog Design Environment.

EDFM Tool Usage

The following table provides on overview of the EDFM tool usage. The table shows the existing common usage for the EDFM tool set. The table also shows what is now considered the “best usage”. Note that each tool shows usage in more than one domain. Parametric analysis is suitable for design exploration and for problem detection.

TABLE OF DESIGN TOOL USAGE

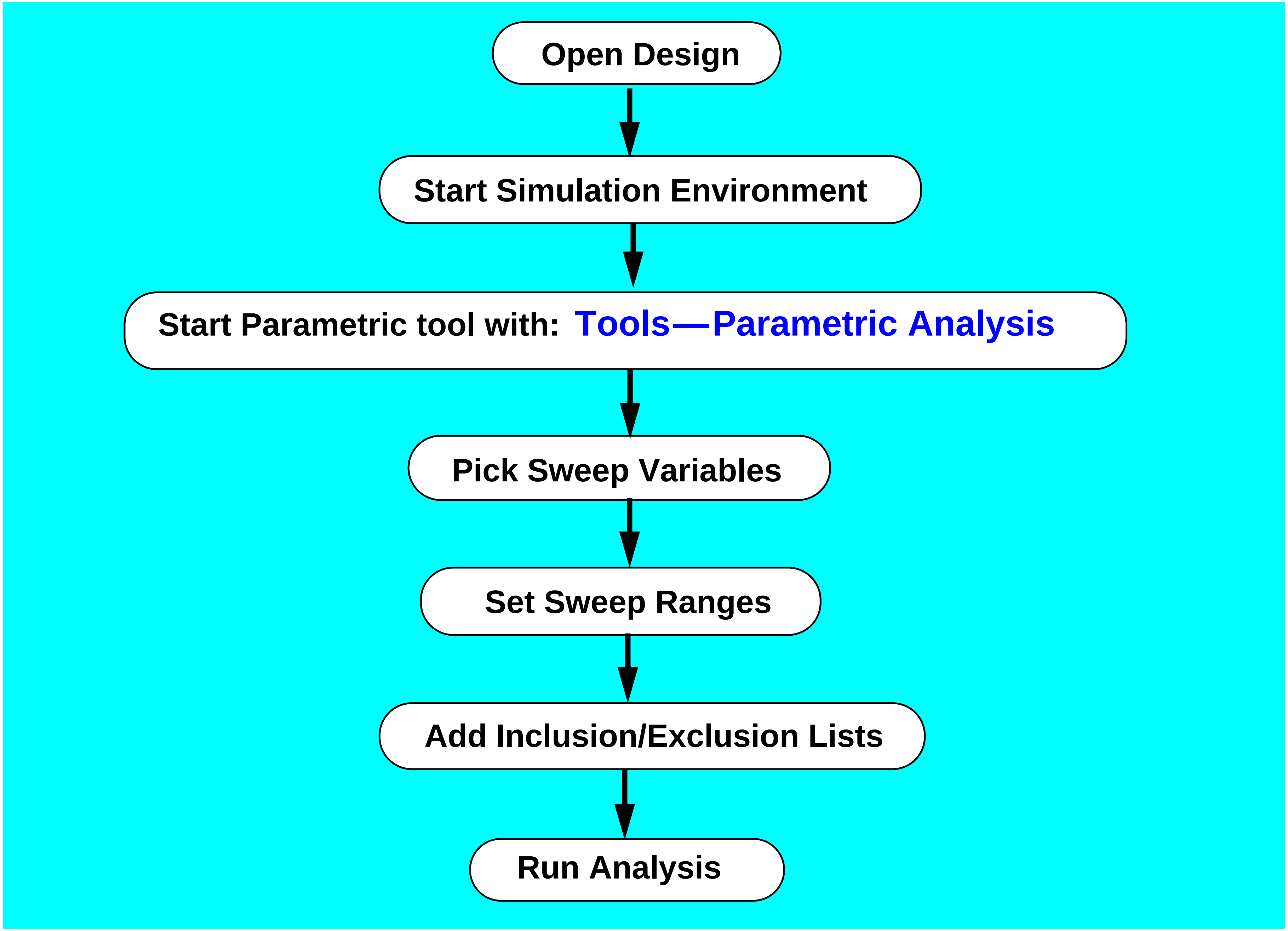
EDFM Tool	**Initial Design**	**Design Exploration**	**Problem Detection**	**Improve & Center**	**Final Verification**
Circuit Optimizer	High	Medium			
Parametric Analysis		High	High		
Corners		High	Medium		High
Monte Carlo			High	Medium	Medium
Yield Analysis				High	High

EDFM Tool Usage

Important

The Table of Design Usage is **intended** to show **“Best Use”** as well as the current usage. The Best Use **is intended to show** the best application of each tool. **It is not a claim** that the historical use or **any** use of a **specific tool to a specific application is flawed!**

Overview of Parametric Analysis and Flow



Open Design

Start Simulation Environment

Start Parametric tool with: **Tools—Parametric Analysis**

Pick Sweep Variables

Set Sweep Ranges

Add Inclusion/Exclusion Lists

Run Analysis

Overview of Parametric Analysis and Flow

Parametric Analysis is a flow where multiple simulations are performed over specified ranges of design variables, device parameters, or circuit conditions. The Analog Design Environment Parametric Analysis Tool provides an easy-to-use interface to run multiple simulations.

Note: The Parametric Analysis tool is somewhat analogous to a semiconductor parameter analyzer (SPA). The SPA is used to measure the device parameters, or device behavior as inputs are swept. The Parametric Analysis tool simulates device behavior as simulation inputs are swept.

In the **EDFM** realm, Parametric Analysis is used to **explore the design space.**

The flow diagram above shows the major steps for performing a Parametric Analysis.

- Open the design
- Start the simulation environment and verify proper circuit operation.
- Start the Parametric Analysis tool.
- Select the sweep variables.
- Select sweep ranges and number of steps for the sweeps.
- Add or delete specific points in the sweep.
- Run the simulation and evaluate the results.

Parametric Analysis Methodology

1. Set up the simulation environment.
   - The environment remains fixed during parametric analysis.
   - Limit output data if results storage space is limited.

2. Start the Parametric Analysis tool by selecting **Tools—Parametric Analysis** from the simulation environment. The tool behaves as a:
   - Number list generator
   - Simulation engine controller

3. Generate lists of design variable values.
   - A simulation is run for each variable value or *point*.
   - A parametric analysis on a variable is called a *sweep*.

4. Plot results by probing in the design window.
   - A family of curves is displayed.

Parametric Analysis Methodology

The Parametric Analysis tool runs a simulation many times while changing the value of design variables for each simulation. Set up the Simulation environment as if running a single simulation, then start the Parametric Analysis tool from the simulation environment.

Open the Parametric Analysis tool and invoke the Design Variables form. Design variables must be declared on this form, before they can be selected for a sweep.

Variables that are part of parameterized models can be added to the Parametric Analysis environment. The DC temperature variable (temp) is already available to be swept.

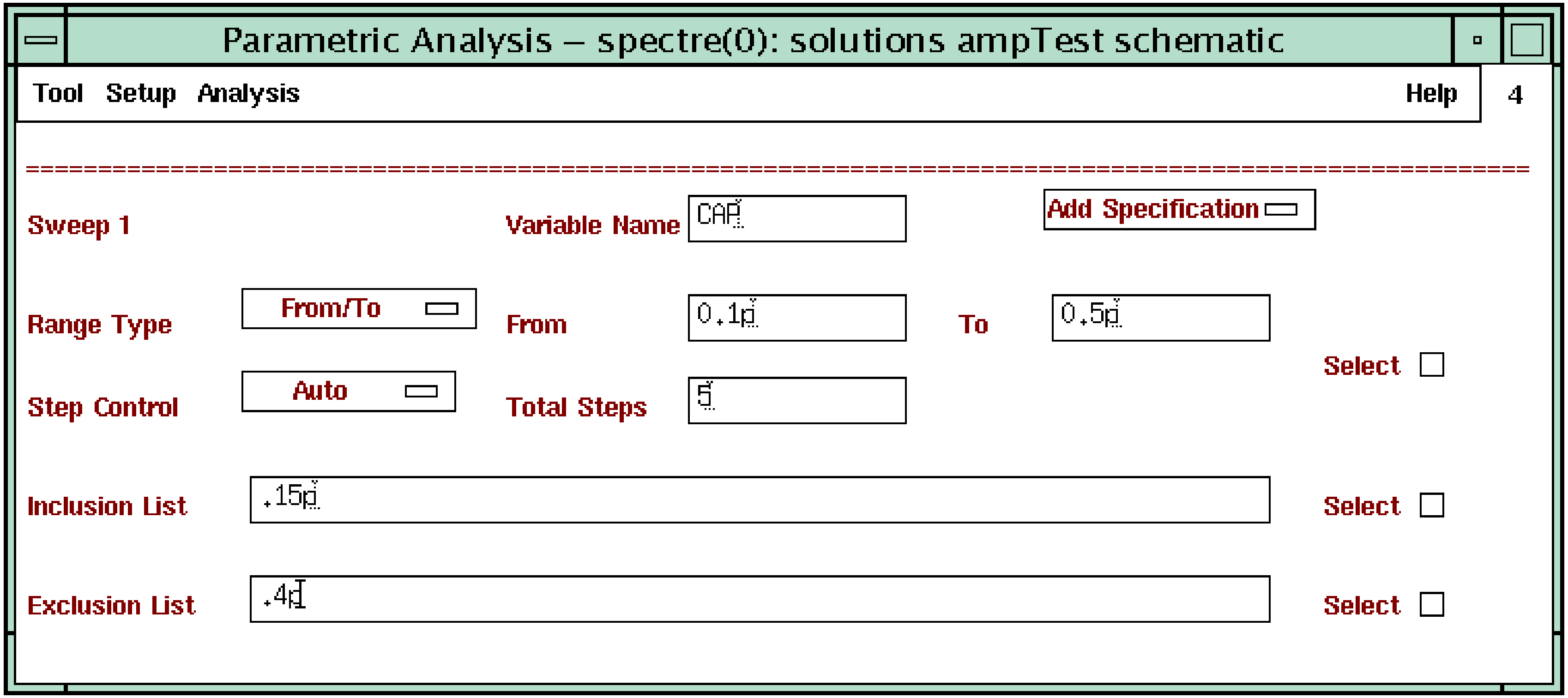
For any variable, pick a starting and ending value then generate intermediate numbers. This is called a range of points. Any number of ranges can be generated and they can overlap.

For any variable add and delete points in a range with inclusion and exclusion lists.

Other features:

- Select a multiple sweep feature
- Pick multiple variables
- Create independent ranges
- Create inclusion and exclusion lists for each variable

The Parametric Analysis Environment



Parametric Analysis – spectre(0): solutions ampTest schematic

Sweep 1

Variable Name CAP

Add Specification

Range Type From/To

From 0.1p

To 0.5p

Select □

Step Control Auto

Total Steps 5

Inclusion List .15p

Select □

Exclusion List .4p

Select □

Tools—Parametric Analysis

The Parametric Analysis Environment

In the Parametric Analysis form, define and view simulation data points, as well as save any parametric analysis settings in a file to load again at a later time.

In addition, set up multiple variable sweeps, which are nested so that the value of the Nth variable is swept at every value of the *(N-1)*th variable where *N* is the position of the variable in the analysis. For example, take two swept variables, *A* and *B*, *A* will be set to an initial value, and simulations will be run for all values of *B*. Next, A will be changed to its next value, and simulations will be run again for all values of *B*. The process continues for all values of *A* and *B*.

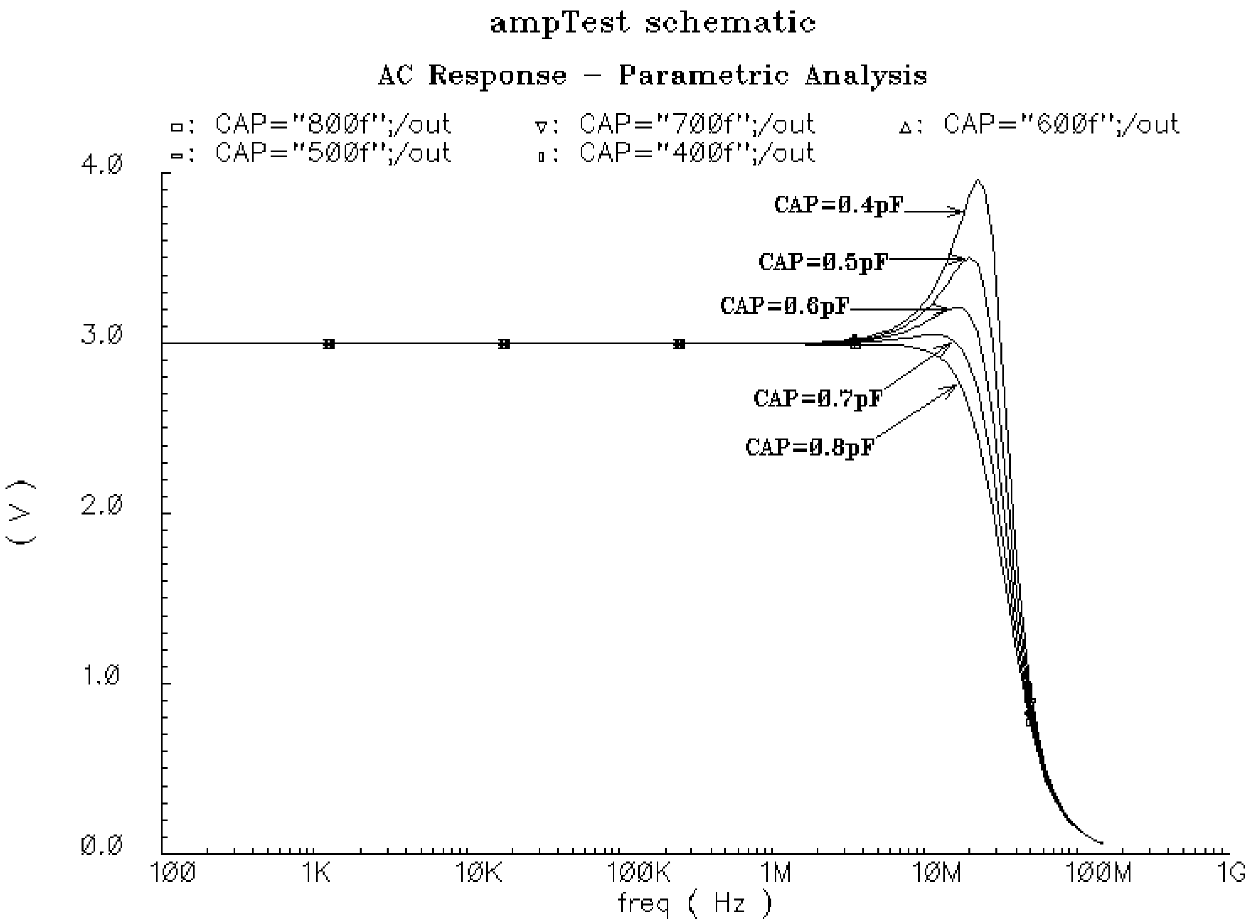
The **Analysis** pull-down menu commands provide start, pause, and continue of an analysis. Take note that interrupting a parametric analysis stops the analysis between simulations. When interrupting a standard simulation in the Simulation environment, the simulation is aborted.

Additionally, delete some or all of the specifications in an analysis and then restore them after deletion.

Generating Ranges

Edit the sweep ranges for the different variables selected, or generate new ones. Choose to sweep any design variable(s) that are declared in the Design Variables form in the Simulation environment. Add multiple ranges which may overlap, and vary the steps in individual ranges. Add inclusion and exclusion fields to specify additional simulation data points or prohibit simulations at particular points.

Parametric Plots



- Curve identifiers match the curve colors.
- Put your cursor on a curve to indicate the associated value(s) of variables.
- Use the calculator to process families of curves.
- Once plotted, each curve is independent for postprocessing.

Parametric Plots

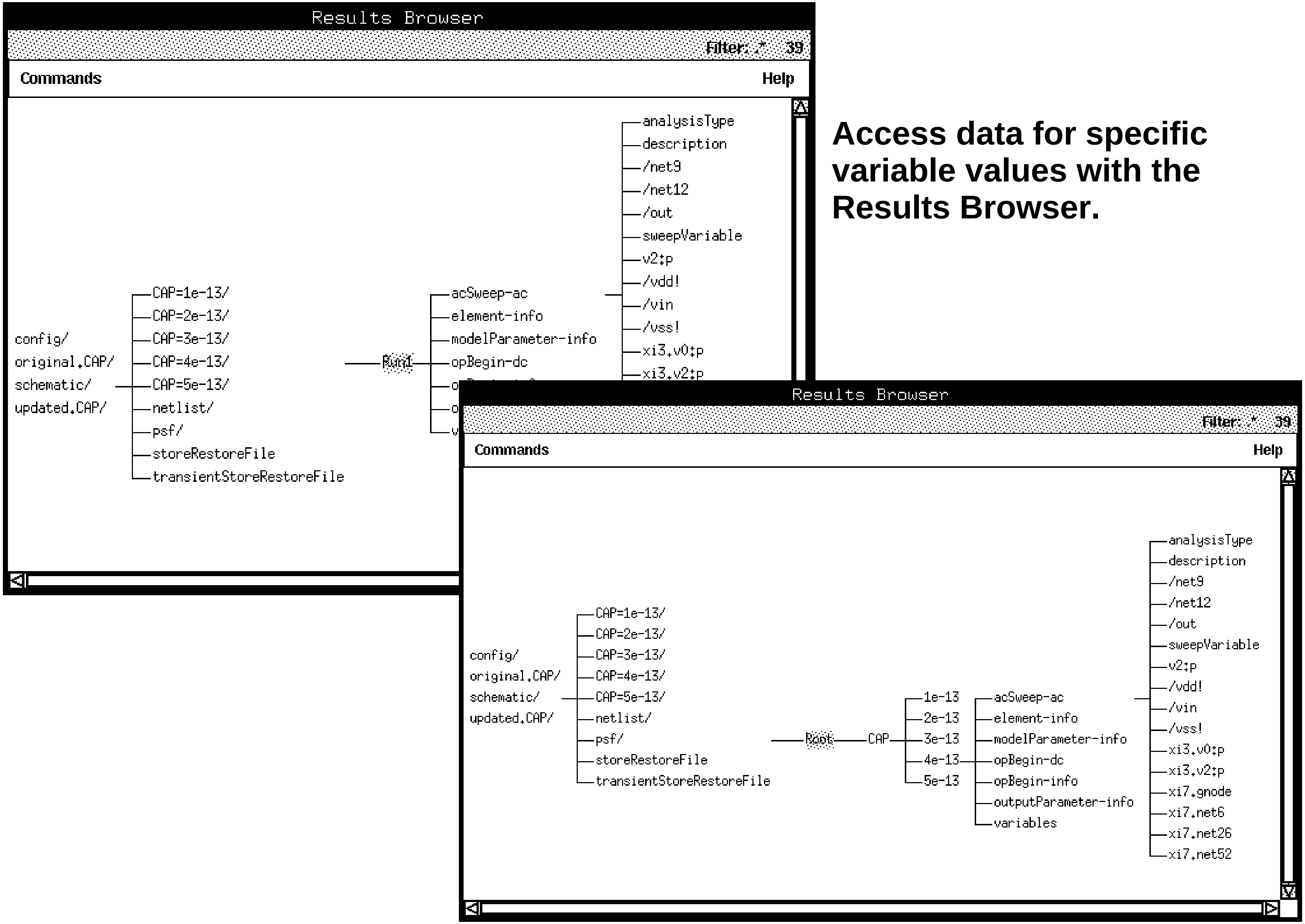
After Parametric Analysis has finished, families of curves are plotted for the selected signals in the Waveform Window. The Waveform Window displays multiple curves labeled with the design variable values for that simulation. Putting your cursor over a particular colored curve will show the value(s) of the swept variable(s) associated with that curve.

Parametric analysis data is also available via the Results Browser. Access individual simulation data for each value of the swept variable by using this method. All functionality of the Results Browser is available for this data.

Once plotted, the different colored curves are independent and can be moved, deleted, or analyzed separate from other curves in the “family.”

Accessing the Parametric Analysis Data Structure

Access data for specific variable values with the Results Browser.

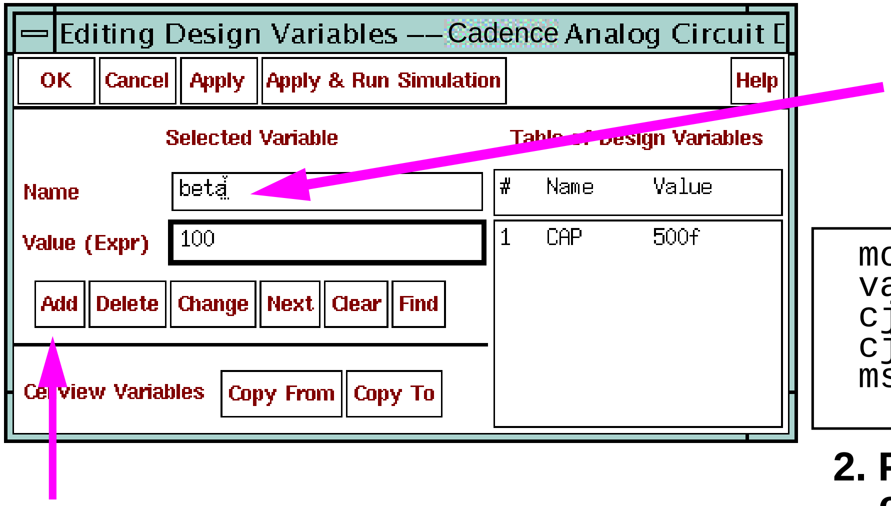


Accessing the Parametric Analysis Data Structure

It is possible to use the Results Browser to access data for any simulation run for a specific value of any number of swept variables.

Use the Results Browser to plot the data or move it into the calculator buffer for postprocessing.

Parametric Analysis on Model Parameters



Name: beta

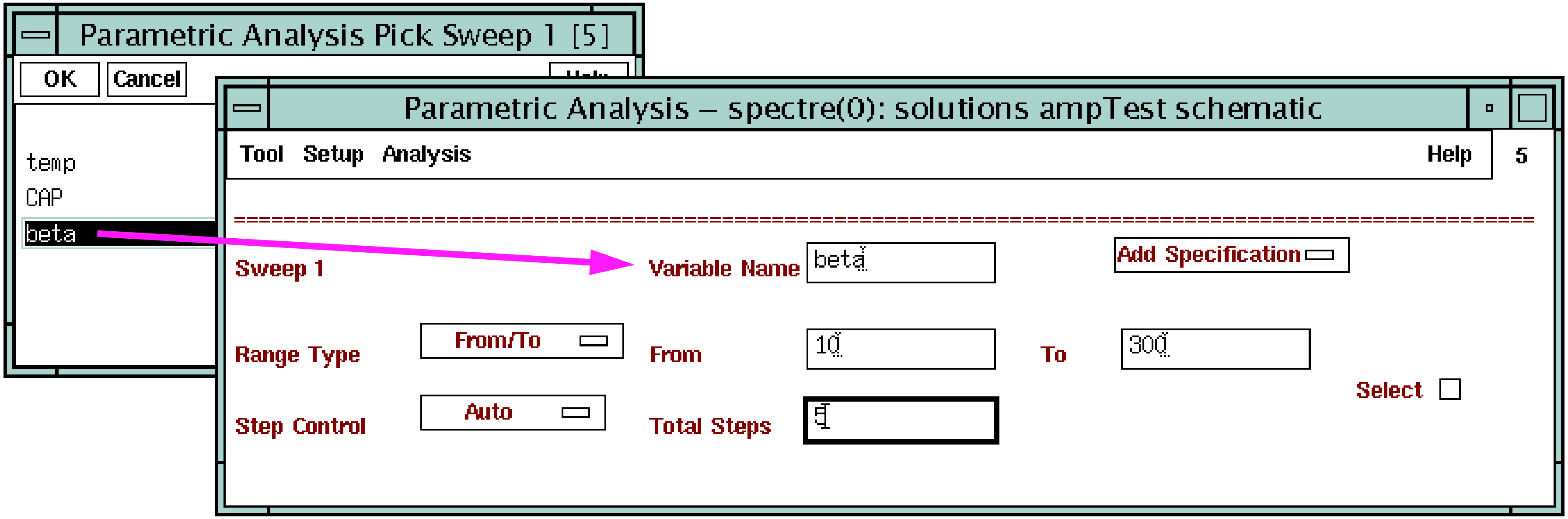
Value (Expr): 100

#	Name	Value
1	CAP	500f

1. Add model parameter as a Design Variable.

```
model trnpn type=npn is=10e-15 bf=beta  \
va=58.7 ik=5.63e-3 rb=665 rbm=86 re=3.2 \
cje=0.25e-12 pe=0.76 me=0.34 tf=249e-12 \
cjc=0.34e-12 pc=0.55 mc=0.35 ccs=2.4e-12 \
ms=0.35 ps=0.53 rc=169
```

2. Point to a model file that contains the new variable.


Parametric Analysis Pick Sweep 1 [5]
OK
Cancel
Help
temp
CAP
beta
Parametric Analysis – spectre(0): solutions ampTest schematic
Tool
Setup
Analysis
Help
5
Add Specification

Sweep 1

Variable Name: beta

Range Type: From/To

From: 10

To: 300

Select □

Step Control: Auto

Total Steps: 5

3. Start Parametric Analysis and choose the new sweep variable.

Parametric Analysis on Model Parameters

Parametric Analysis in OCEAN

- The parametric analysis commands are not saved to the OCEAN script when selecting **Session—Save Ocean Script**.
- Use **Tool—Save Script** to save the parametric analysis.
- Nest parametric analyses for multiple variables, as shown in this example.

Example

```
paramAnalysis( “rl” ?start 200 ?stop 600 ?step 200 paramAnalysis(
“rs” ?start 300 ?stop 700 ?step 200 ))

paramRun()
```

Runs a parametric analysis on rl for each value of rs, yielding a nested parametric analysis.

Parametric Analysis in OCEAN

OCEAN scripts must be manually modified to run a parametric analysis. (This will be done automatically in future releases.) Remove or comment out the run() statement in your script, and add the paramAnalysis() and paramRun() commands. For example, add the following to your OCEAN script:

```
paramAnalysis("CAP" ?start 0.1p ?stop 0.5p ?step 0.1p)
paramRun()
```

This example will run a parametric analysis on the CAP variable.

Set up multiple parametric analyses, and only simulate the variables desired as in this example:

```
paramAnalysis( "rs" ?start 200 ?stop 1000 ?step 200 )
paramAnalysis( “rl” ?start 200 ?stop 600 ?step 200 )
paramRun( ‘rs)
```

This example runs a parametric analysis on the rs design variable only.

Nest parametric analyses with multiple variables by setting up an OCEAN script with syntax similar to the following example:

```
paramAnalysis( “rl” ?start 200 ?stop 600 ?step 200 paramAnalysis(
“rs” ?start 300 ?stop 700 ?step 200 ))
paramRun()
```

This example runs a parametric analysis on rl for each value of rs, yielding a nested parametric analysis.

Summary

In this module we discussed:

- An overview of EDFM tools, their uses and properties.
- Parametric Analysis flow and methodology
- The Parametric Analysis environment
- Plotting results
- Accessing data from Parametric Analysis
- Model parameters
- Running Parametric Analysis in OCEAN

Summary

Labs

Lab 7-1 Running Parametric Analysis

Labs

Module 8: Corners Analysis

Topics in this module

- Introduction to the Corners Analysis tool
- Corners Analysis Window
- Adding a new process
- Modeling styles used for Corners Analysis
- Results

Terms and Definitions

process corner	Model parameters assumed for the extreme limits of a process.
corner	Selected set of model parameters and operating conditions.
Corners Analysis	Software tool to run multiple simulations at specified corners.
pcf	Process Customization File, defines corners for a given process.
dcf	Design Customization File, defines design variables.
model style	Method of compiling data to describing model variations.

Corners Analysis Tool

- The Corners Analysis tool provides useful performance information on a design by **performing multiple simulations under specified conditions**.
- With the Corners Analysis tool, a design is simulated at specified “corners” where the process parameters, supply voltages, temperature, and other operating conditions are altered.
- The Corners Analysis tool is one of five EDFM software tools that operate within the Analog Design Environment.
- As an EDFM tool, the Corners Analysis is intended to automatically set up, run, and analyze multiple simulations. It is used to explore the design space and locate problem areas.

Corners Analysis Tool

In a theoretical manufacturing process, process variables can have exact values and these exact values can be used to calculate the yield for the process. However, in a real manufacturing process, process variables are subject to a manufacturing tolerance—they fluctuate randomly around their ideal values. The combined random variation for all the components results in an uncertain yield for the circuit as a whole.

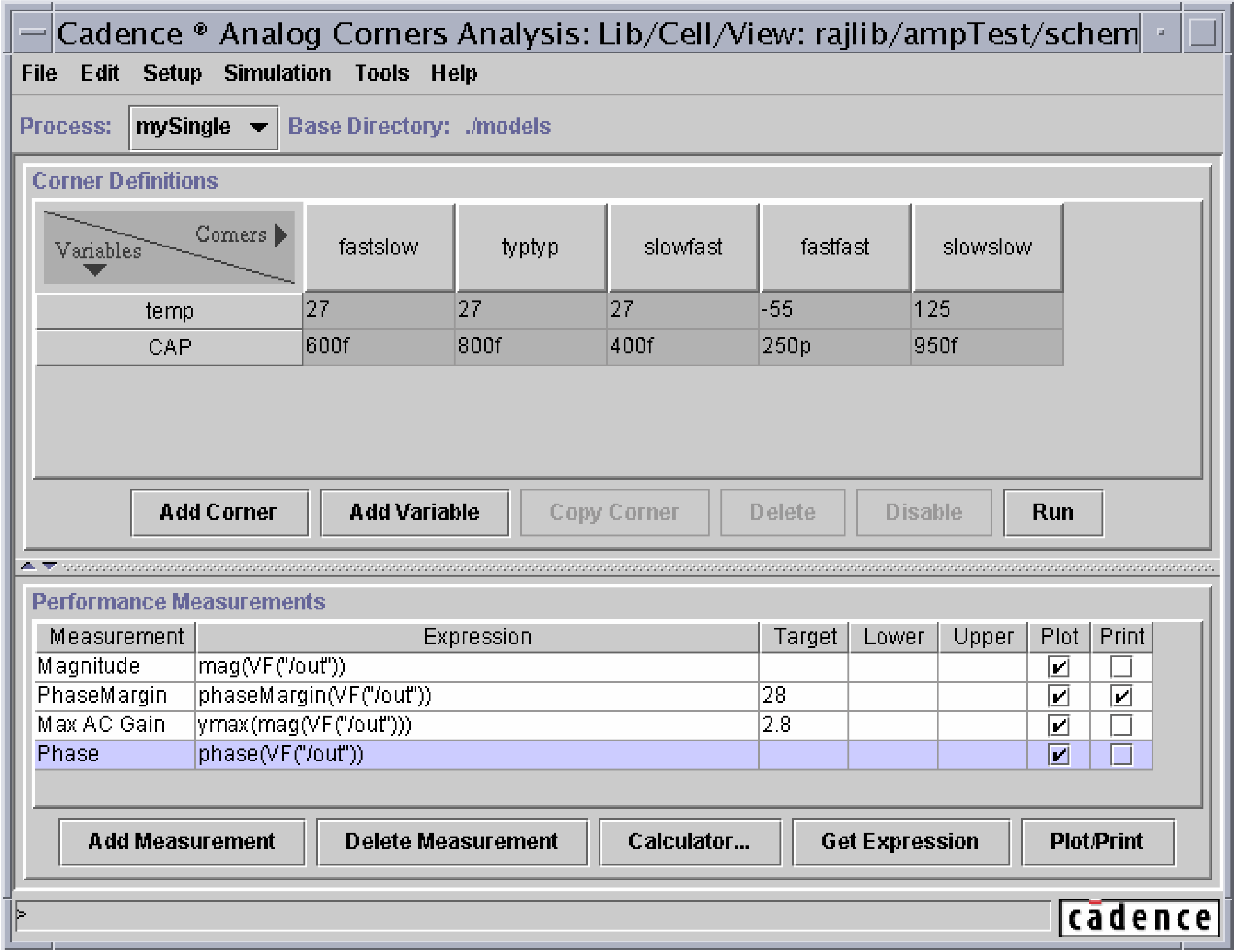
A design methodology that is used to guarantee both reliability and yield is to simulate the circuit under worst-case conditions. In actuality, there are numerous worst-case conditions. As such, circuit simulation is performed using worst-case device models at the specified limits of temperature, supply voltage, and frequency. The so called worst-case models, along with the worst-case design limits, are often called “the worst-case corners”. Hence the name used for this design tool.

The fabrication process is often considered to occupy a specified region of multi-dimensional fabrication universe. Within this region, the process produces worst-case extremes of device parameters. These worst-case extremes are represented by worst-case models. For example, for typical CMOS process, the following table establishes the worst-case corners:

process corner	K’n	K’p	Vtn	Vtp	Ln	Lp	Vdd	Temp
worst-case slow	min.	min.	max.	max.	max.	max.	min.	max.
worst-case fast	max.	max.	min.	min.	min.	min.	max.	min.
worst-case one	max.	min.	min.	max.	min.	max.	min.	
worst-case zero	min.	max.	max.	min.	max.	min.	max.	

Corners Analysis Window

Select **Tools—Corners** from the Simulation window.



Process: mySingle Base Directory: ./models

Variables \ Corners	fastslow	typtyp	slowfast	fastfast	slowslow
temp	27	27	27	-55	125
CAP	600f	800f	400f	250p	950f

Measurement	Expression	Target	Lower	Upper	Plot	Print
Magnitude	mag(VF("/out"))				☑	☐
PhaseMargin	phaseMargin(VF("/out"))	28			☑	☑
Max AC Gain	ymax(mag(VF("/out")))	2.8			☑	☐
Phase	phase(VF("/out"))				☑	☐

Corners Analysis Window

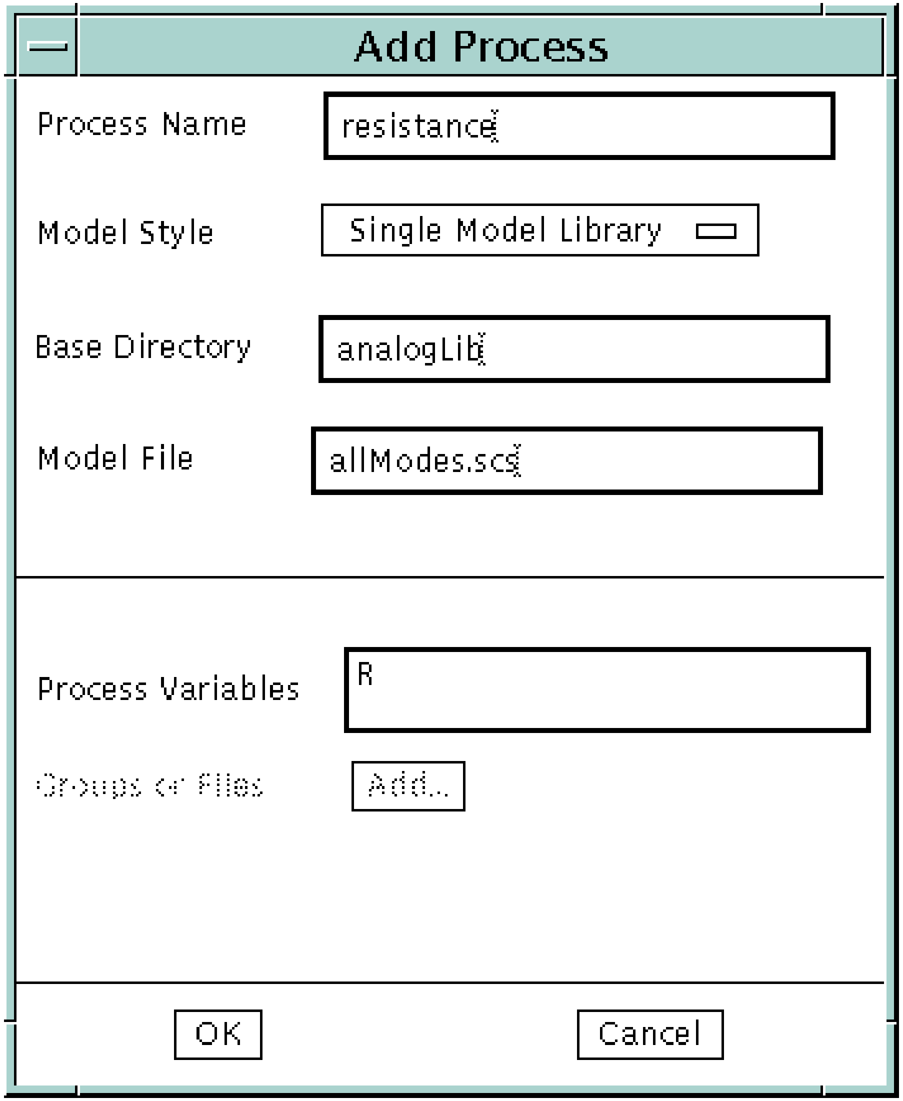
Corners Analysis looks at the performance outcomes generated from the most extreme variations expected in the process, voltage, and temperature values (the *corners*). With this information, the circuit performance specifications can be determined, even when the random process variations combine in their most unfavorable patterns.

There are different kinds of files associated with setting up a Corners Analysis.

- Process customization files (pcf's) define processes, groups, variants, and corners that are shared by an entire organization. pcf's are usually created by a process engineer or process group.
- Design customization files (dcf's) contain definitions that are used for a particular design or for several designs within a design group. dcf's are usually created by designers, who use the dcf's to add design-specific information to the general information provided in pcf's.

Use the **Add Corner** button, or select **Edit—Add Corner** from the menu to add a new process corner. Use the **Add Variable** button, or select **Edit—Add Variable** from the menu to add a new variable. Use the **Add Measurement** button, or select **Edit—Add Measurement** from the menu to add a new measurement.

Adding a New Process



Add Process

Process Name: resistance

Model Style: Single Model Library

Base Directory: analogLib

Model File: allModes.scs

Process Variables: R

Select **Setup—Add Process** from the Corners Window.

Adding a New Process

Implementing Modeling Styles

Model styles are specified in a .scs file. There are several ways to implement models:

- Single-Model Library Style
- Multiple-Model Library Style
- Single-Numeric Style
- Multiple-Numeric Style
- Multiple-Parametric Modeling

Two additional types of files used are:

- Process Customization File (pcf)—ends in *.pcf*

  Adds the name of a new process to the Corners tool and defines the basic set of corners

- Design Customization File (dcf)—ends in *.dcf*

  Adds design specific variables and measurements to the Corners tool

Load them explicitly, or with the *.cdsinit* file:

```
loadPcf( "./CORNERS/multipleModelLib.dcf" )
loadPcf( "./CORNERS/singleModelLib.pcf" )
```

Implementing Modeling Styles

There are different modeling styles for corners.

Single-Model Library Style

Recommended style:

- All models for all corners are located in a single model file
- The model file is located in the base directory and can have any name
- Can also use the .LIB syntax for this style
- Works with *altergroup* function in the Spectre simulator

```
Path: ./models/fab6


Filename: mylibfile.scs
```

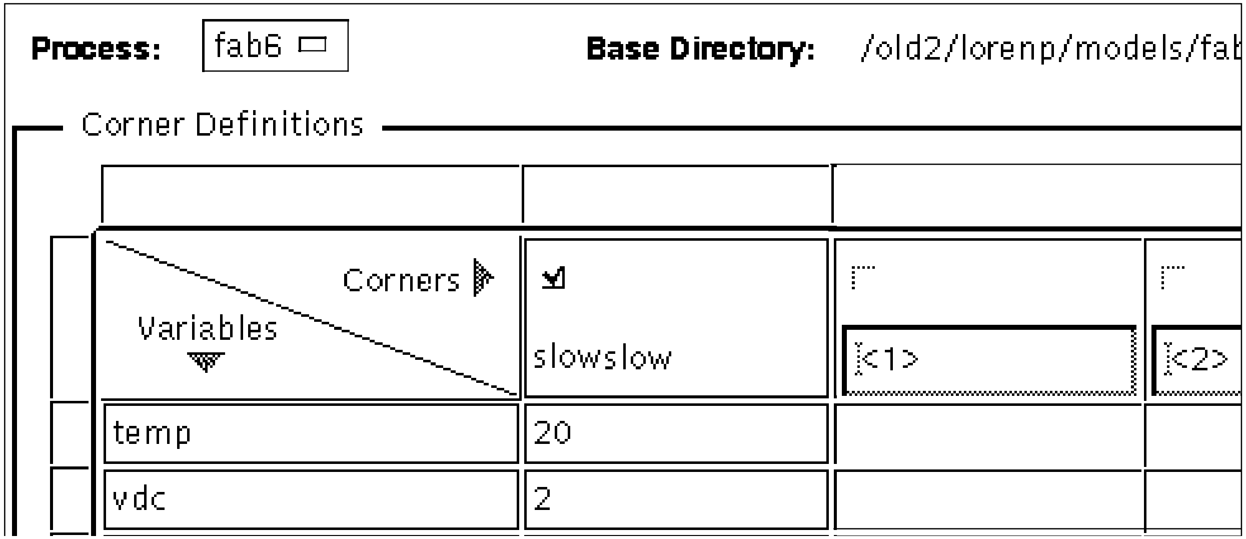
```
library processA
section slowslow
 model npn2 npn tf=120n
 model npn9 npn tf=320n
 model nmosR nmos tox=120n
 model nmos8 nmos tox=320n
endsection
section nom
 model npn2 npn tf=100n
 model npn9 npn tf=300n
 model nmosR nmos tox=100n
 model nmos8 nmos tox=300n
endsection
section fastfast
 model npn2 npn tf=80n
 model npn9 npn tf=380n
 model nmosR nmos tox=80n
 model nmos8 nmos tox=380n
endsection
endlibrary
```

Single-Model Library Style

The following code illustrates how you can refer to this modeling structure in a pcf.

```
corAddProcess("fab6" "~lorenp/models/fab6/" ’singleModelLib)

corSetModelFile("fab6" "mylibfile.scs")

corAddProcessVar("fab6" "vdc")

corAddCorner("fab6" "slowslow"
  ?runTemp 20
  ?nomTemp 27
  ?vars ’( ("vdc" 2) )
)
```

The Corners window produced by this pcf looks like this.



Process: fab6 ▭ **Base Directory:** /old2/lorenp/models/fab

Corner Definitions

Corners / Variables	☑ slowslow	<1>	<2>
temp	20		
vdc	2		

Multiple-Model Library Style

Similar to Single-Model Library Style

Example: models might be located in the files:

```
./models/fab6/path1/npn.scs
./models/fab6/path3/nmos.scs
```

```
Path: ~/john/models/fab6/path1
Filename: npn.scs
```

```
library npn
section slow
model npn2 bjt tf=120n
model npn8 bjt tf=80n
endsection

section nom
model npn2 bjt tf=100n
model npn8 bjt tf=60n
endsection

section fast
model npn2 bjt tf=80n
model npn8 bjt tf=50n
endsection
endlibrary
```

```
Path: ~/john/models/fab6/path3
Filename: nmos.scs
```

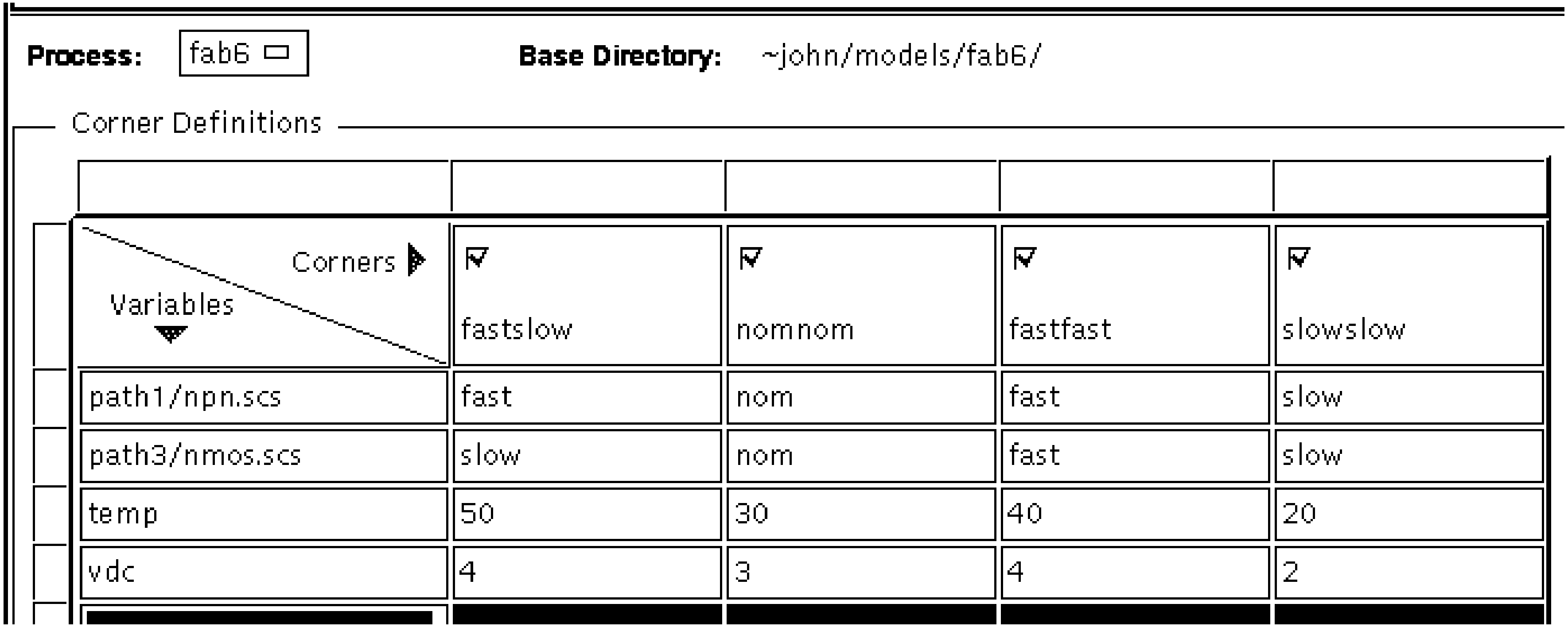
```
library nmos
section slow
model nmosR mos3 tox=120n
model nmos2 mos3 tox=140n
endsection

section nom
model nmosR mos3 tox=100n
model nmos2 mos3 tox=115n
endsection

section fast
model nmosR mos3 tox=80n
model nmos2 mos3 tox=90n
endsection
endlibrary
```

Multiple-Model Library Style

The following code illustrates how to refer to this multiple-model library structure in a pcf.

```
corAddProcess("fab6" "~john/models/fab6/" 'multipleModelLib)
corAddModelFileAndSectionChoices("fab6" "path1/npn.scs" '( "slow" "nom" "fast") )
corAddModelFileAndSectionChoices("fab6" "path3/nmos.scs" '( "slow" "nom" "fast") )
corAddProcessVar("fab6" "vdc")
corAddCorner("fab6" "slowslow" ?sections '( ("path1/npn.scs" "slow") ("path3/nmos.scs" "slow") )
?runTemp 20 ?nomTemp -27 ?vars '( ("vdc" 2) ) )
corAddCorner("fab6" "nomnom" ?sections '( ("path1/npn.scs" "nom") ("path3/nmos.scs" "nom") )
?runTemp 30 ?nomTemp 27 ?vars '( ("vdc" 3) ) )
corAddCorner("fab6" "fastfast" ?sections '(("path1/npn.scs" "fast") ("path3/nmos.scs" "fast") )
?runTemp 40 ?nomTemp -27 ?vars '( ("vdc" 4) ) )
corAddCorner("fab6" "fastslow" ?sections '(("path1/npn.scs" "fast") ("path3/nmos.scs" "slow") )
?runTemp 50 ?nomTemp -27 ?vars '( ("vdc" 4) ) )
```

The Corners window produced by this pcf looks like this:



Process: fab6 **Base Directory:** ~john/models/fab6/

Corner Definitions

Corners / Variables	☑ fastslow	☑ nomnom	☑ fastfast	☑ slowslow
path1/npn.scs	fast	nom	fast	slow
path3/nmos.scs	slow	nom	fast	slow
temp	50	30	40	20
vdc	4	3	4	2

Other Styles

Not heavily tested or recommended—provided for backward compatibility

- Single Numeric

  Each corner is located in a separate file. If there are four corners, there are four model files with the same name.

- Multiple Numeric

  Each model is defined in a separate file. All model parameters are defined with numeric values.

- Multiple Parametric

  With this style, each model is defined in a separate file. There is a corresponding parameter file for every model associated with each corner.

Other Styles

(Single Numeric) Path	Filename	File Contents

`./models/fab6/allslow/`	`models`	`.model npn2 npn tf=120n`
		`.model nmos8 nmos tox=320n`

`./models/fab6/allnom/`	`models`	`.model npn2 npn tf=100n`
		`.model nmos8 nmos tox=300n`

(Multiple Numeric) Path	Filename	File Contents

`./models/fab6/npn/slow/`	`npn2.scs`
	`npn9.scs`
		`model npn2 bjt tf=120n`
		`model npn9 bjt tf=320n`

`./models/fab6/npn/nom/`	`npn2.scs`
	`npn9.scs`
		`model npn2 bjt tf=100n`
		`model npn9 bjt tf=300n`

(Multiple Parametric) Path	Filename	File Contents

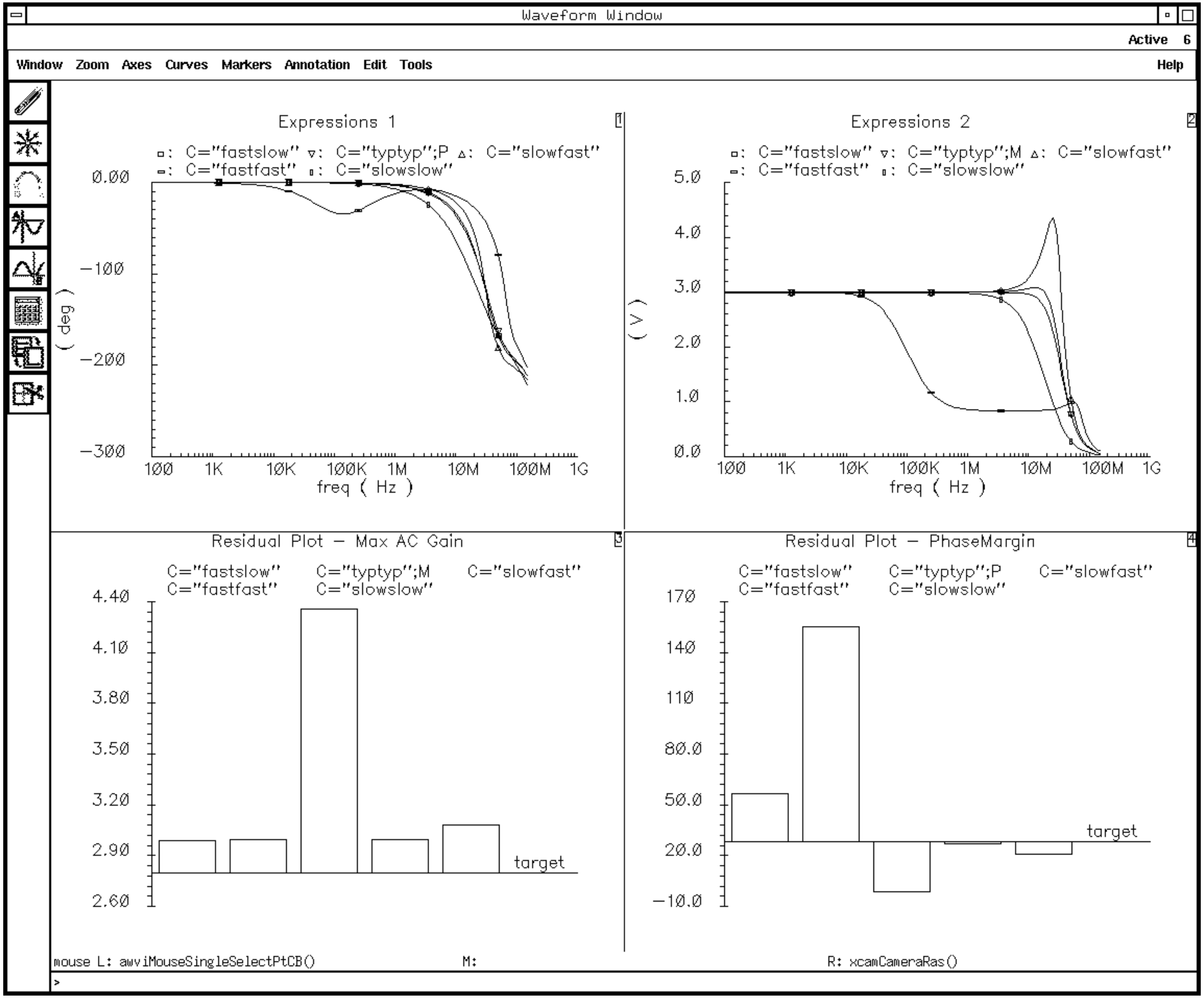
`./models/fab6/npn`	`npn2.scs`	`include "npn2.param"`
		`model npn2 bjt tf=TF2`

	`npn9.scs`	`include "npn9.param"`
		`model npn9 bjt tf=TF9`

`./models/fab6/npn/slow/`	`npn2.param`	`parameter TF2=120n`
	`npn9.param`	`parameter TF9=320`

This chart shows a partial list to the models associated with each of the above styles.

Corners Results


Waveform Window
Active 6
Window Zoom Axes Curves Markers Annotation Edit Tools Help
Expressions 1
□: C="fastslow" ▽: C="typtyp";P △: C="slowfast"
▭: C="fastfast" ▯: C="slowslow"
0.00
−100
−200
−300
(deg)
100 1K 10K 100K 1M 10M 100M 1G
freq (Hz)
Expressions 2
□: C="fastslow" ▽: C="typtyp";M △: C="slowfast"
▭: C="fastfast" ▯: C="slowslow"
5.0
4.0
3.0
2.0
1.0
0.0
(V)
100 1K 10K 100K 1M 10M 100M 1G
freq (Hz)
Residual Plot − Max AC Gain
C="fastslow" C="typtyp";M C="slowfast"
C="fastfast" C="slowslow"
4.40
4.10
3.80
3.50
3.20
2.90
2.60
target
Residual Plot − PhaseMargin
C="fastslow" C="typtyp";P C="slowfast"
C="fastfast" C="slowslow"
170
140
110
80.0
50.0
20.0
−10.0
target

```
mouse L: awviMouseSingleSelectPtCB()              M:                              R: xcamCameraRas()
>
```

Corners Results

When the analysis finishes, the Corners tool plots or lists the results according to choosing text or graphic outputs in the Performance Measurements pane.

To view a different set of outputs from those selected before running the analysis, set up new outputs in the Performance Measurements section. Next, select **Tools—Plot** or **Print Outputs** from the menu or click the **Plot/Print** button. The Corners tool evaluates and displays the new measurements or plots, as long as the data needed for these new calculations was saved during simulation.

There are two kinds of graphical output:

- Residual plots for scalar data

  The residual plots (bar graphs) above shows that three of the corners fail for the Phase Margin specifications. This result implies that yield for the manufactured circuit will be less than 100 percent if the circuit is produced in its current form. For greater yield, the circuit designer might want to redesign the circuit so that it performs acceptably for all corners.

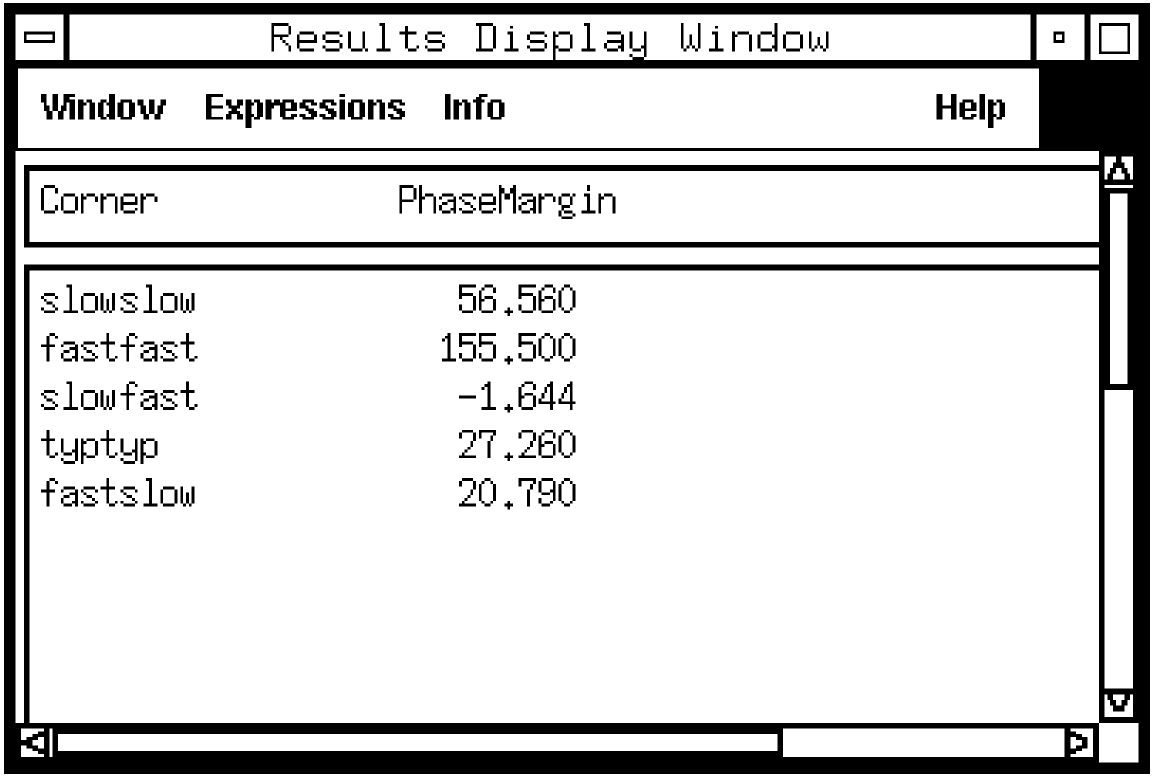
- Family of curves for waveform data

  The plot above shows how the magnitude and phase vary as a function of frequency for each of the corners.

All these features are also available in OCEAN.

Corners Results Window

Textual output



```
Corner                PhaseMargin

slowslow                   56.560
fastfast                  155.500
slowfast                   -1.644
typtyp                     27.260
fastslow                   20.790
```

Corners Results Window

The window above shows the result of the request for **Textual** output for the *Phase Margin* measurement. It gives a clear indication of the successes and failures of this performance measurement with each corner.

Labs

Lab 8-1 Using the Corners Analysis Tool

Labs

Module 9: Monte Carlo Analysis

Topics in this module

- Overview
- The Monte Carlo tool
- Statistical modeling
- Process variation methodology
- Setting up Monte Carlo analysis
- Running Monte Carlo analysis
- Filtering output data

Terms and Definitions

Monte Carlo Statistical analysis with multiple simulations using random number generation to synthesize process variations.

process variations Variation of device parameters inherent in fabrication due to acceptable tolerances.

mismatch Variation in device parameters within the same circuit. Contributes to input offset in opamps and comparators.

output filtering Process by which data beyond expected statistical limits are eliminated.

Overview of Monte Carlo Analysis

The features of the Monte Carlo Analysis Tool include:

- An EDFM analysis tool
- Multiple simulation tool using a large number of samples
- Uses a statistical approach to evaluate the performance of a design
- Provides broad coverage in exploring the “Design Space”
- Emulates the variations in model parameters due to process tolerances
- Very useful in finding “problem areas”
- Used to predict process yield

The Monte Carlo Tool is based on the “Monte Carlo” analysis technique. This analysis uses the random selection of model parameters based on statistical distributions.

Overview of MonteCarlo Analysis

Background:

The manufacturing variations in components affect the production yield of any design that includes them. Monte Carlo analyses these relationships in detail.

Monte Carlo Analysis

Monte Carlo is a specific type of statistical analysis in which variables are randomly selected and applied to the specific application, and the results are then evaluated. If applied properly, a Monte Carlo analysis can accurately predict the yield of a product fabricated with a specified process.

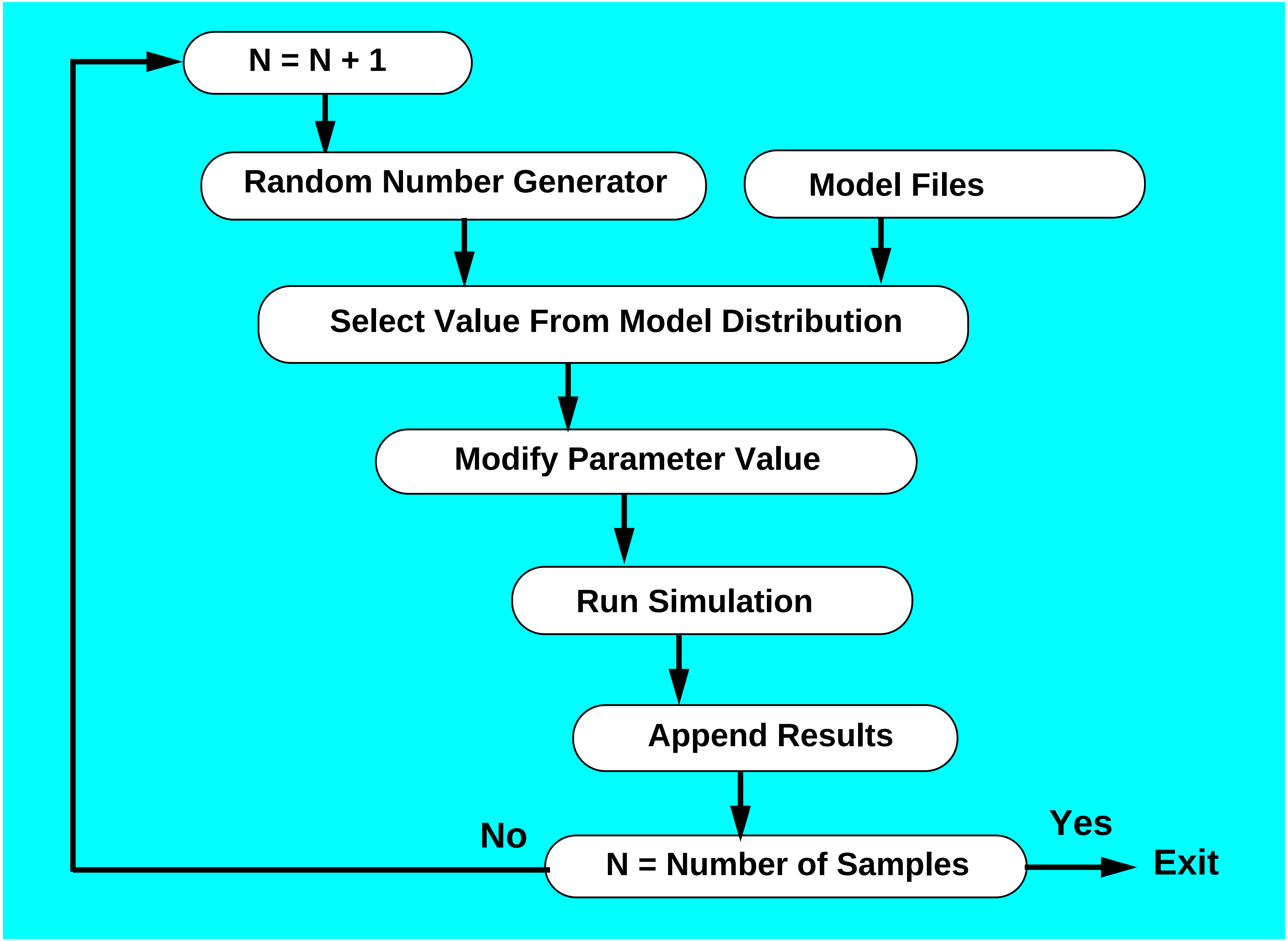
EDFM

Monte Carlo Analysis is one of the **EDFM** tools available with the Analog Design Environment. It is often used to detect problem areas within the design space. It can also be used to explore the design space.

Multiple Simulation Tool

Monte Carlo is a multiple simulation analysis tool. In Monte Carlo, the inputs to the simulation (process variables and mismatches) are automatically altered between simulation runs. These variables are altered randomly using random number generators. The Spectre® Direct simulator provides special features for running Monte Carlo Analysis that facilitate higher performance.

Simulation Using Process Distributions



Simulation Using Process Distributions

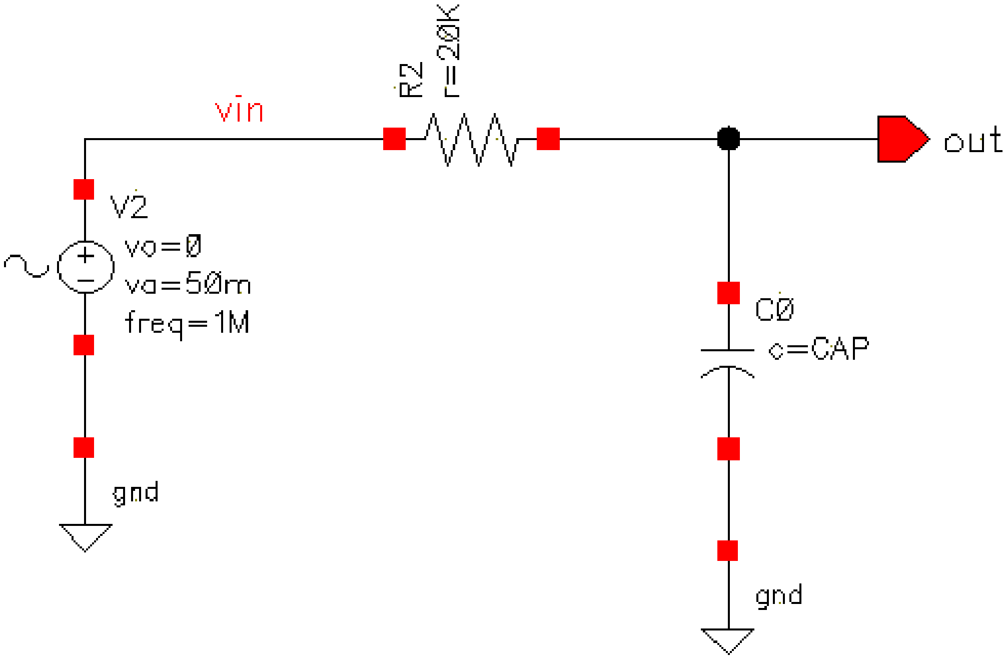
Background

Manufacturing always produces a distribution in model parameters. The distribution of each parameter is evaluated for mean, standard deviation, and type of distribution. The statistical properties are included in the model file.

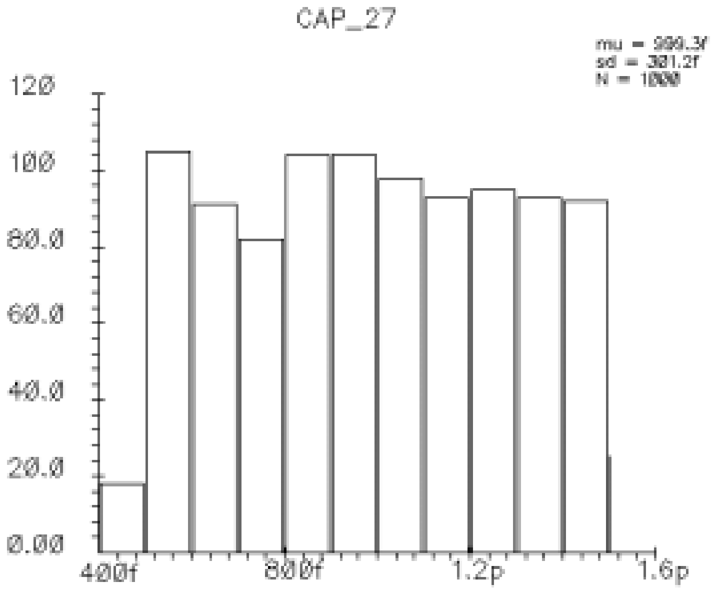
When the device is manufactured, process variations will occur. The actual device parameters are not known until fabrication has been completed. The outcome of a specific fabrication cycle cannot be predicted. However, over a large number of fabrication cycles, the distributions in device parameters will occur.

Using the above flow, the analysis emulates the variation in the device parameters from processing many fabrication lots.

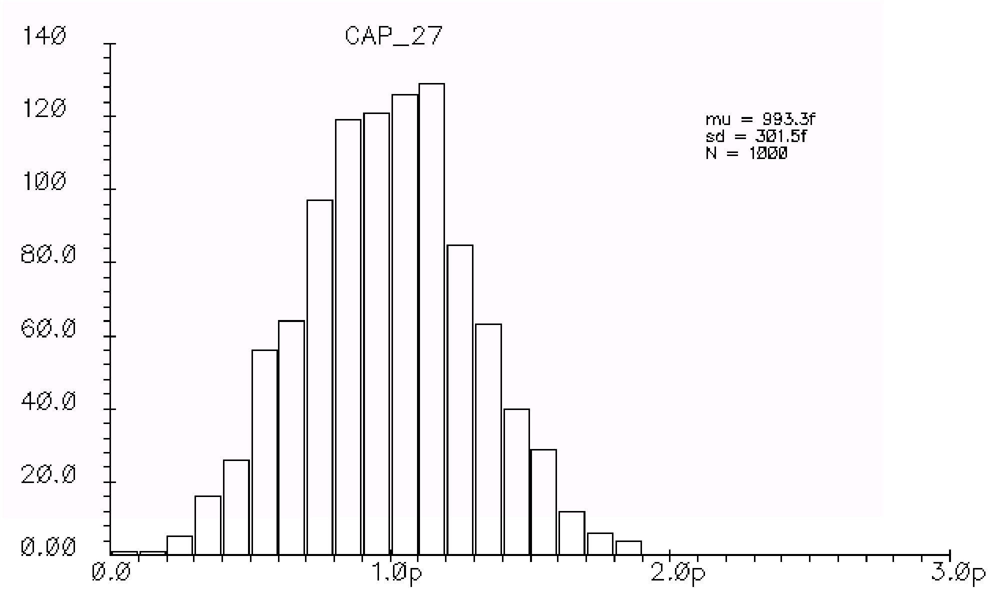
Example of Monte Carlo Using Simple LPF



Uniform Distribution



Normal or Gaussian Distribution



Example of Monte Carlo Using Simple LPF

The circuit above is simulated using the Monte Carlo Analysis Tool. The capacitor C0 has a nominal capacitance value of 1.0pf. This is a simplified example of a low pass filter in which only the CAP value is altered between simulation runs.

The histograms for C0 are obtained by selecting

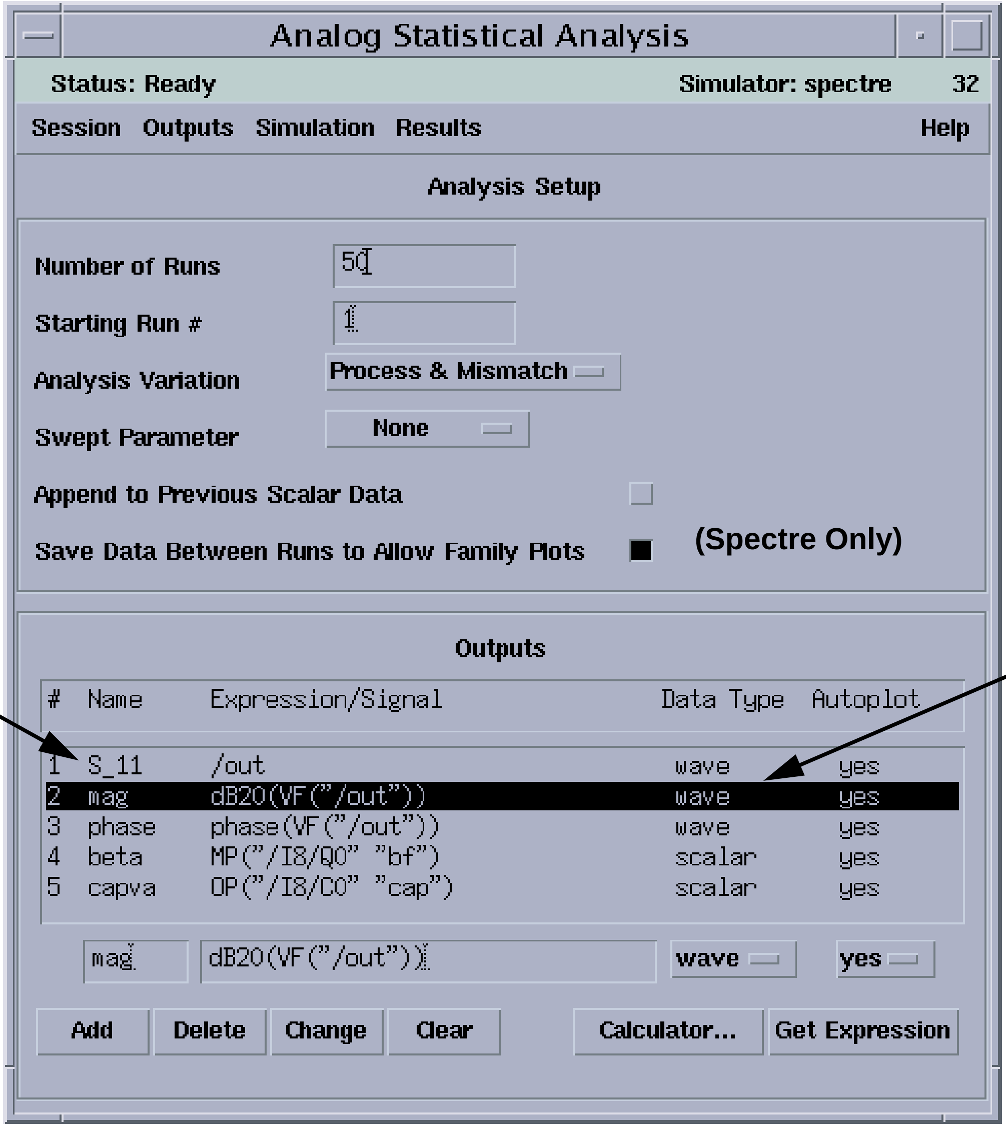
Results—Plot—Histogram

in the Analog Statistical Analysis user interface.

The first simulation run is 1000 samples using a “unif” distribution for CAP in the model file spectreLib5.scs. The Spectre circuit simulator simulates the above circuit 1000 times in which the value for CAP is changed randomly with a uniform distribution.

The second simulation run is 1000 samples using a “gauss” distribution for CAP in the model file spectreLib5.scs. The Spectre circuit simulator simulates the above circuit 1000 times in which the value for CAP is changed randomly with a normal distribution.

Monte Carlo Analysis Environment



Status: Ready Simulator: spectre 32

Number of Runs: 50

Starting Run #: 1

Analysis Variation: Process & Mismatch

Swept Parameter: None

Append to Previous Scalar Data □

Save Data Between Runs to Allow Family Plots ■ (Spectre Only)

#	Name	Expression/Signal	Data Type	Autoplot
1	S_11	/out	wave	yes
2	mag	dB20(VF("/out"))	wave	yes
3	phase	phase(VF("/out"))	wave	yes
4	beta	MP("/I8/Q0" "bf")	scalar	yes
5	capva	OP("/I8/C0" "cap")	scalar	yes

mag dB20(VF("/out")) wave yes

Starts with outputs from the Simulation window

wave, scalar, or unknown

Select **Tools—Monte Carlo** in the Simulation Window.

Setup, expression definition, and statistical analysis forms all in a single window.

Monte Carlo Analysis Environment

Number of Runs: Select the number of Monte Carlo runs (default = 100).

Starting Run #: Select starting iteration # (default = 1). Define the starting seed of the statistical analysis that sets the first pseudo-random seed used in the random variation functions (i.e., *gauss()*, etc.). Consequently, to run several sets of analyses, collecting the results from all, each subsequent set should start with an iteration of the # of previous runs + 1. This is used in conjunction with the Append to Previous Scalar Data option.

Analysis Variation: Select an Analysis Variation type (default = "Process Only"). The default choices for the Spectre® tool are (i) Process Only, (ii) Process and Mismatch, and (iii) Mismatch Only.

Swept Parameter: Select a stepped/swept parameter (default = "None"). This selection determines if an inner loop is performed for each set of random variables (i.e., each iteration of Monte Carlo). The choices are: "None", "Temperature", or any one of the design variables in the design. Only one parameter may be swept.

Append to Previous Scalar Data: Choose whether the scalar output data from the run should be appended to previously saved scalar data (default = "no").

Save Data Between Runs to Allow Family Plots: Choose whether the raw output data (PSF files) should be saved between iterations (default = "no"). Selecting this option plots families of waveforms representing variation of entire waveforms. Select specific nodes or terminal, rather than all voltages and currents, to save on the disk space usage.

Perform Nominal Run to Check Expressions: Choose whether a nominal simulation is run, prior to starting the Monte Carlo process, to validate the expressions defined in the Outputs section (default = "no"). This option is highly recommended because it avoids the possibility of running the analysis and not getting the desired results.

Support of Spectre Direct

- Spectre includes commands for statistical modeling and Monte Carlo looping within the simulator.
- Simplified methodology for specifying statistics, correlation, and mismatch.
- All runs from a single netlist.
- 5 to 10 times speed improvement over Spectre implementation in the cdsSpice socket.
- Online help available with the following command in an xterm window:

```
spectre -h montecarlo
```

Note: The following are only available in Spectre Direct Monte Carlo simulations:

- Distributed processing
- Save/plot any data or families of curves

Support of Spectre Direct

Monte Carlo with the Spectre Direct simulator runs from a single netlist. The statistical variations take place in the simulator, whereas in the socket mode, every single Monte Carlo run requires a new final netlist and a restart of the Spectre software. This is a huge time saver.

Spectre Direct Statistical Modeling

Several files are used for modeling of process variations:

- **Model file**: Contains variables altered by other include files.
- **Variables include file**: used to set global variables in prior versions, as in this example:

  ```
  simulator lang=spectre
  parameters PiRho=2500 PbRho=200 stat=1 npnbeta=145.5
  pnpbeta=200 initstat=1
  ```

- **Distribution include file**: Defines statistical distributions for all variables.
  - Contains statistics blocks for process and mismatch variations
  - Types of distributions available: **Gaussian**, **Uniform**, **Lognormal**
  - Specify process and mismatch correlations
  - Support of *altergroup* statement in Spectre simulator

Spectre Direct Statistical Modeling

The Monte Carlo tool now works both in socket mode and in native Spectre (direct) mode. When you run Monte Carlo in Spectre Direct mode, the statistical variations are specified in several include files or a single file.

Specifying the statistical expressions in the CDF is not currently supported in Spectre direct.

Example using several files:

```
simulator lang=spectre
include "processSimple.scs"
include "process.scs"
statistics {
  process {
    vary RSHSP dist=gauss std=5
    vary RSHPI dist=lnorm std=0.15
    vary SPDW dist=gauss std=0.25
    vary SNDW dist=gauss std=0.25
  }
  correlate param=[RSHSP RSHPI] cc=0.6
  mismatch {
    vary XISN dist=gauss std=1
    vary XBFN dist=gauss std=1
    vary XRSP dist=gauss std=1
  }
}
```

File: *monteProcess.scs*

```
// Provides the designer with "geometrical" device interface
simulator lang=spectre
// Call to the geometrical TNSA model
inline subckt TNSA (C B E S)
  parameters WE=1u LE=1u MULT=1 dIS=0 dBF=0
  TNSA (C B E S) TNSA_PR WE=WE LE=LE \
    MULT=MULT dIS=dIS dBF=dBF     // call TNSA_PR
ends TNSA

// Call to the geometrical RPLR model
inline subckt RPLR (A B)
  parameters Rnom=1 WB=10u MULT=1 dR=0
  RPLR (A B) RPLR_PR Rnom=Rnom WB=WB \
    MULT=MULT dR=dR              // call RPLR_PR
ends RPLR
```

File: *process.scs*

In addition, there is a file that will contain the models for the “primitives” (*TNSA_PR* and *RPLR_PR* devices) that are used by the inline subcircuits in the *process.scs* file. The file *monteProcess.scs* is a distribution include file.

Statistical Modeling

```
simulator lang=spectre
parameters PiRho=2500 PbRho=200 npnbeta=250  pnpbeta=80 CAP=.8p

statistics {
       process {
               vary PbRho     dist=lnorm std=40
               vary npnbeta   dist=gauss std=20
               vary pnpbeta   dist=gauss std=25
               vary CAP       dist=gauss std=30 percent=yes
       }
       mismatch {
               vary PbRho dist=lnorm std=3.75
               vary npnbeta dist=gauss std=4
               vary pnpbeta dist=gauss std=6
       } }
inline subckt trnpn (C B E)
model npnstat bjt type=npn is=10e-15 bf=npnbeta \
 va=58.7 ik=5.63e-3 rb=565 rbm=86 re=3.2 cje=1.3e-12 pe=0.76 \
 me=0.34 tf=249e-12 cjc=0.8e-12 pc=0.55 mc=0.35 ccs=2.4e-12 \
 ms=0.35 ps=0.53 rc=169 vsubfwd=10

trnpn (C B E) npnstat
ends trnpn

inline subckt trpnp (C B E)
model pnpstat bjt type=pnp is=1.2e-16 bf=pnpbeta nf=1 vaf=60 ikf=4e-5 \
 ne=1.5 br=80 nr=1 var=5 ikr=5e-5 isc=0 nc=1.5 rb=100 re=15 rc=30\
 cje=30e-15 vje=.72 mje=.45 tf=5e-10 ptf=40 cjc=60e-15 vjc=.72\
 mjc=.45 xcjc=.9 tr=5e-10 cjs=0 vjs=.99 mjs=.99

trpnp (C B E) pnpstat
ends trpnp ...(continued)
```

Statistical Modeling

In the example above, the model names *trnpn* and *trpnp* were maintained so you do not need to change the names of the models on the devices in the schematic.

Additional example of a Monte Carlo distribution include file:

```
simulator lang=spectre
statistics {
     process {
     vary vth0_nfet_x dist=gauss std=vth0_nfet_t
     vary u0_nfet dist=gauss std=u0_nfet_t
     vary rdsw_nfet dist=gauss std=rdsw_nfet_t
     vary cgso_nfet dist=gauss std=cgso_nfet_t
     vary cgdo_nfet dist=gauss std=cgdo_nfet_t
     vary cgsl_nfet dist=gauss std=cgsl_nfet_t
     vary cgdl_nfet dist=gauss std=cgdl_nfet_t
                 }
mismatch {
     vary stat dist=gauss std=0.1
                }
               }
```

Other Features in Monte Carlo

- Save data between runs to plot families of curves for all iterations.

  This can consume lots of disk space, so select signals appropriately.
- Outputs selected in Simulation window prior to starting the Monte Carlo tool are automatically placed in the Output Expressions section
  - No need to manually re-enter data to be measured/saved
  - Expressions can be added/edited/deleted
- Can run specific iterations again
- Can save the OCEAN script for batch processing
- Can save and load Monte Carlo setups

A nominal run not required prior to entering expressions.

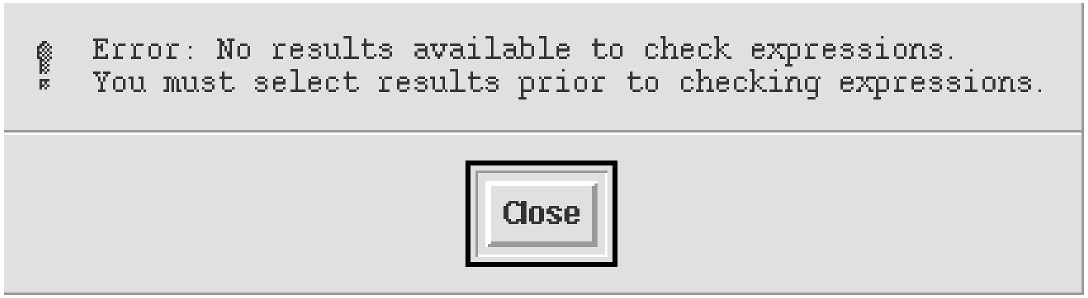
- Use the **Check Expressions** capability.

  This helps to determine whether the output expressions and signals—defined or selected—are valid.

Other Features in Monte Carlo

Previous versions of Monte Carlo analysis required you to run a nominal simulation prior to entering Monte Carlo expressions. This prerequisite has been removed and replaced by the **Check Expressions** function.

Use **Simulation—Check Expressions** to validate the entries in the Outputs section. If data exists, Check Expressions will use this for the validation. If no data is currently available, a message box pops up.

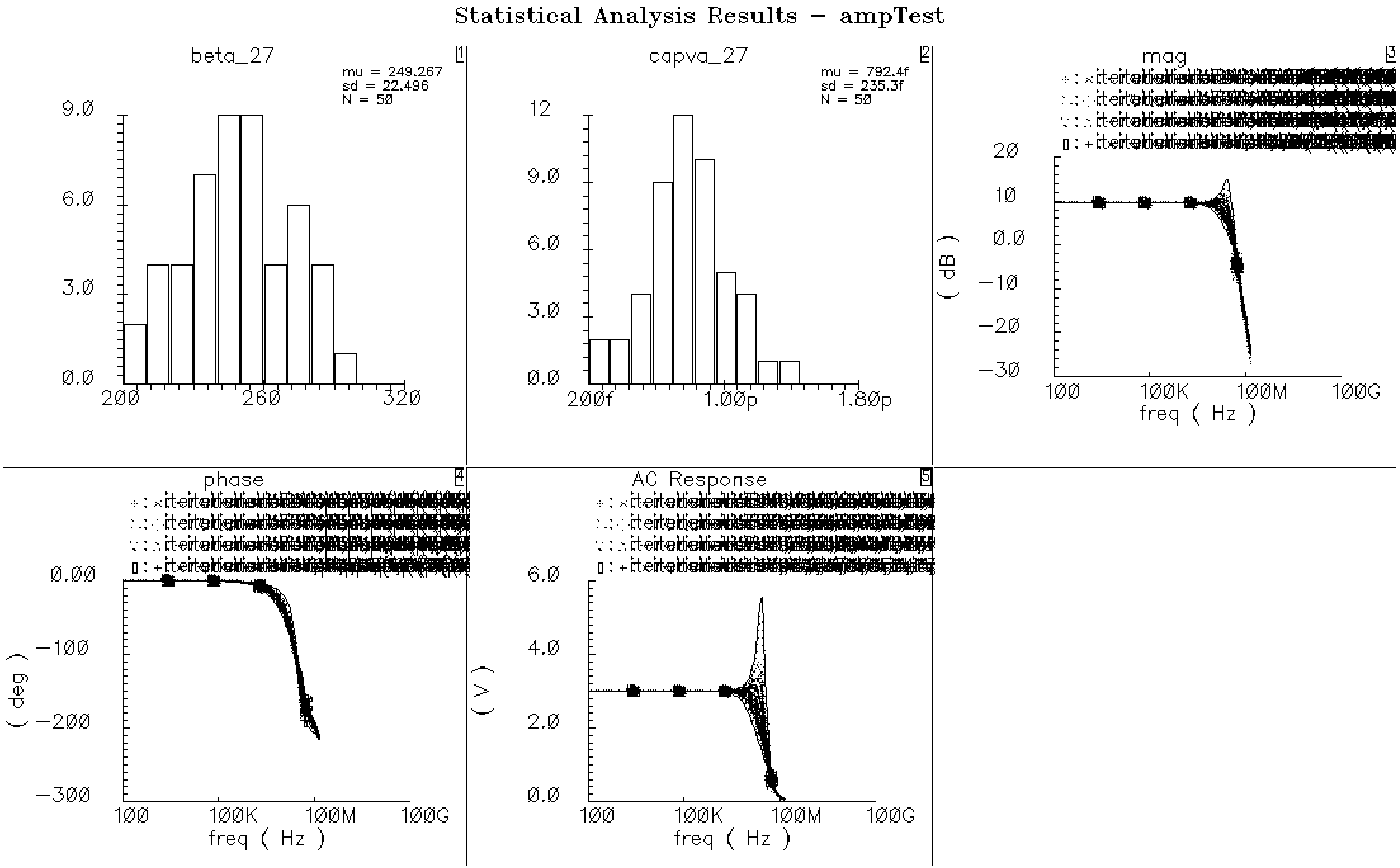


Error: No results available to check expressions.
You must select results prior to checking expressions.

A nominal simulation will be run automatically by the Monte Carlo tool.

With this feature, it is possible to use restored results with **Check Expressions**.

Monte Carlo Results Analysis

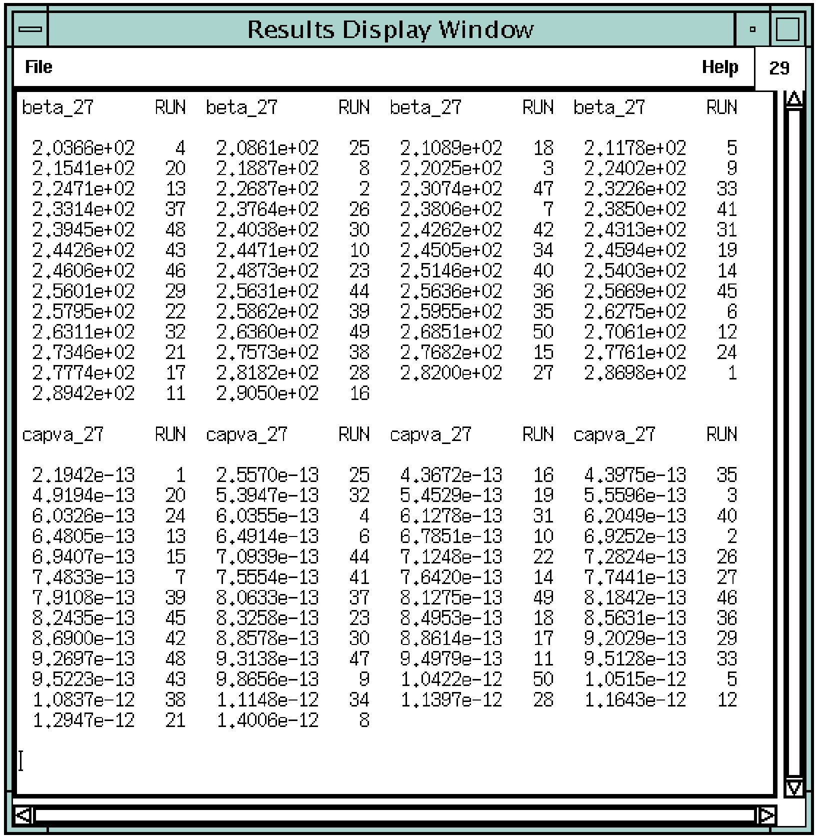


Monte Carlo Results Analysis

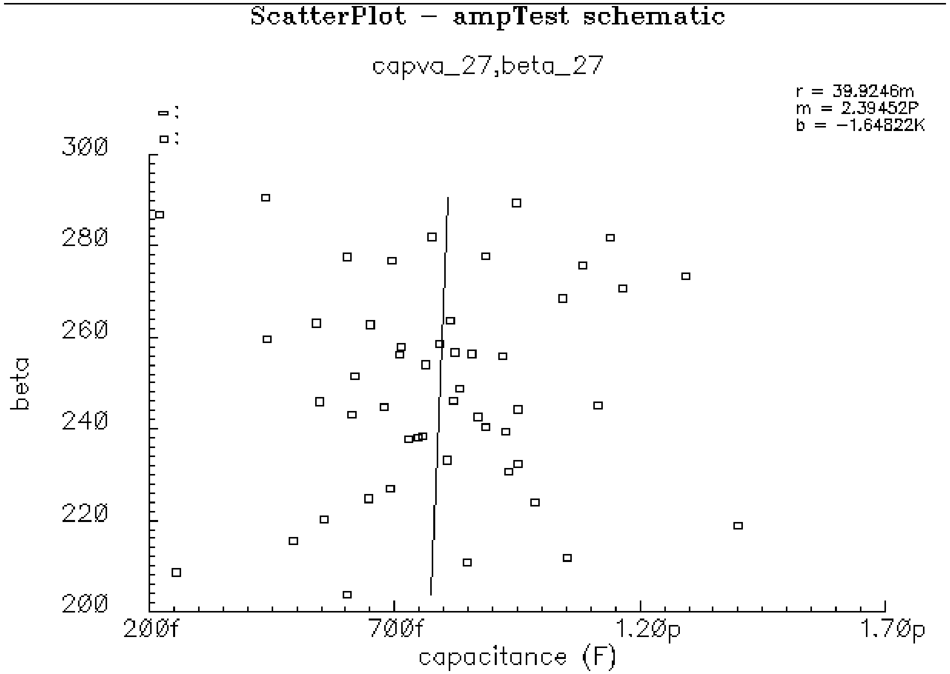
The Monte Carlo environment provides many ways to evaluate the simulation results.

- The graphical results as shown above and as demonstrated in the lab activity.
  - Printed results
  - Histograms
  - Filtered data
  - Scatter plots
  - Yield

Monte Carlo Results Analysis (continued)



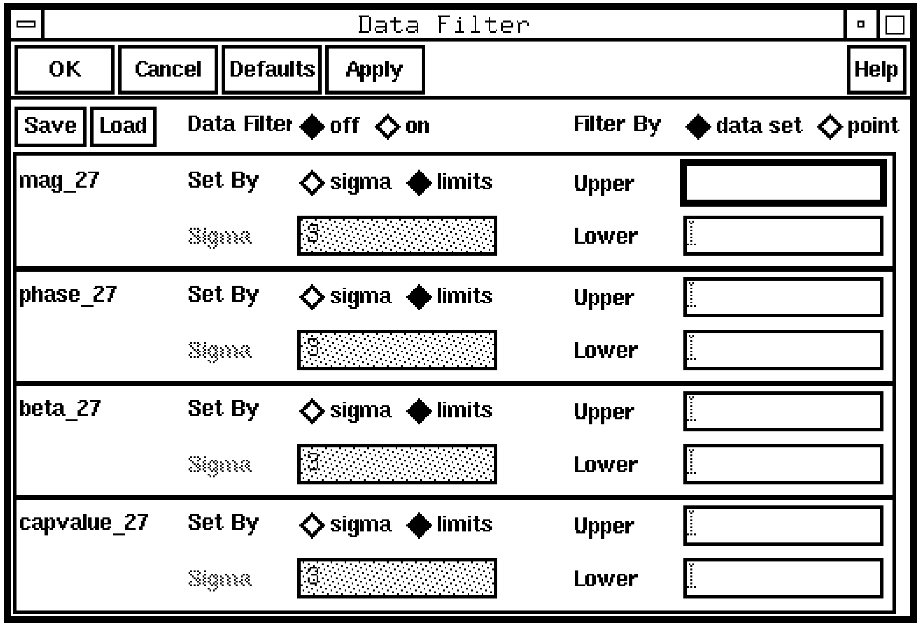
```
beta_27        RUN  beta_27        RUN  beta_27        RUN  beta_27        RUN

 2.0366e+02     4    2.0861e+02    25    2.1089e+02    18    2.1178e+02     5
 2.1541e+02    20    2.1887e+02     8    2.2025e+02     3    2.2402e+02     9
 2.2471e+02    13    2.2687e+02     2    2.3074e+02    47    2.3226e+02    33
 2.3314e+02    37    2.3764e+02    26    2.3806e+02     7    2.3850e+02    41
 2.3945e+02    48    2.4038e+02    30    2.4262e+02    42    2.4313e+02    31
 2.4426e+02    43    2.4471e+02    10    2.4505e+02    34    2.4594e+02    19
 2.4606e+02    46    2.4873e+02    23    2.5146e+02    40    2.5403e+02    14
 2.5601e+02    29    2.5631e+02    44    2.5636e+02    36    2.5669e+02    45
 2.5795e+02    22    2.5862e+02    39    2.5955e+02    35    2.6275e+02     6
 2.6311e+02    32    2.6360e+02    49    2.6851e+02    50    2.7061e+02    12
 2.7346e+02    21    2.7573e+02    38    2.7682e+02    15    2.7761e+02    24
 2.7774e+02    17    2.8182e+02    28    2.8200e+02    27    2.8698e+02     1
 2.8942e+02    11    2.9050e+02    16

capva_27       RUN  capva_27       RUN  capva_27       RUN  capva_27       RUN

 2.1942e-13     1    2.5570e-13    25    4.3672e-13    16    4.3975e-13    35
 4.9194e-13    20    5.3947e-13    32    5.4529e-13    19    5.5596e-13     3
 6.0326e-13    24    6.0355e-13     4    6.1278e-13    31    6.2049e-13    40
 6.4805e-13    13    6.4914e-13     6    6.7851e-13    10    6.9252e-13     2
 6.9407e-13    15    7.0939e-13    44    7.1248e-13    22    7.2824e-13    26
 7.4833e-13     7    7.5554e-13    41    7.6420e-13    14    7.7441e-13    27
 7.9108e-13    39    8.0633e-13    37    8.1275e-13    49    8.1842e-13    46
 8.2435e-13    45    8.3258e-13    23    8.4953e-13    18    8.5631e-13    36
 8.6900e-13    42    8.8578e-13    30    8.8614e-13    17    9.2029e-13    29
 9.2697e-13    48    9.3138e-13    47    9.4979e-13    11    9.5128e-13    33
 9.5223e-13    43    9.8656e-13     9    1.0422e-12    50    1.0515e-12     5
 1.0837e-12    38    1.1148e-12    34    1.1397e-12    28    1.1643e-12    12
 1.2947e-12    21    1.4006e-12     8
```



Monte Carlo Results Analysis (continued)

The above figure shows a text window displaying iteration versus value and a scatter plot. The scatter plot shows correlation between the capvalue and beta parameters.

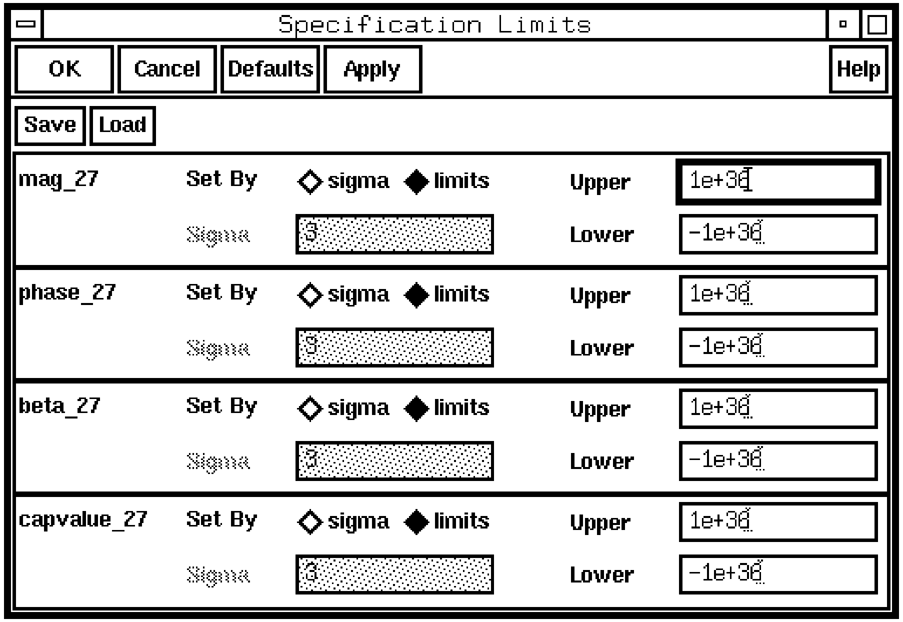
Filtering Output Data



Data Filter ● off ○ on   Filter By ● data set ○ point

	Set By	Sigma	Upper	Lower
mag_27	○ sigma ● limits	3		
phase_27	○ sigma ● limits	3		
beta_27	○ sigma ● limits	3		
capvalue_27	○ sigma ● limits	3		

Results—Filter



	Set By	Sigma	Upper	Lower
mag_27	○ sigma ● limits	3	1e+36	-1e+36
phase_27	○ sigma ● limits	3	1e+36	-1e+36
beta_27	○ sigma ● limits	3	1e+36	-1e+36
capvalue_27	○ sigma ● limits	3	1e+36	-1e+36

Results—Specification Limits

Filtering Output Data

Summary

In this module we discussed:

- Overview of Monte Carlo analysis
- The Monte Carlo environment
- Statistical modeling
- Process variation methodology
- Setting up Monte Carlo analysis
- Running Monte Carlo analysis
- Filtering output data

Summary

Labs

Lab 9-1 Monte Carlo Analysis

Labs

Module 10: Optimization Analysis

Topics in this module

- Introduction
- Optimization design flow
- Optimization computational flow
- Optimization options form
- Algorithms used in optimization
- Adding goals
- Curve fitting
- Iteration history
- Plotting options
- Optimization using OCEAN

Terms and Definitions

optimization Process to increase circuit or system performance by changing selected design parameters.

goals A design target of circuit performance, such as bandwidth.

LSQ Least Squares Fit. A model fit that minimizes the squared error of all data points.

CFSQP C version Feasible Sequential Quadratic Programming, an optimization algorithm.

curve fitting The optimization capability to define the target goals over a specified range.

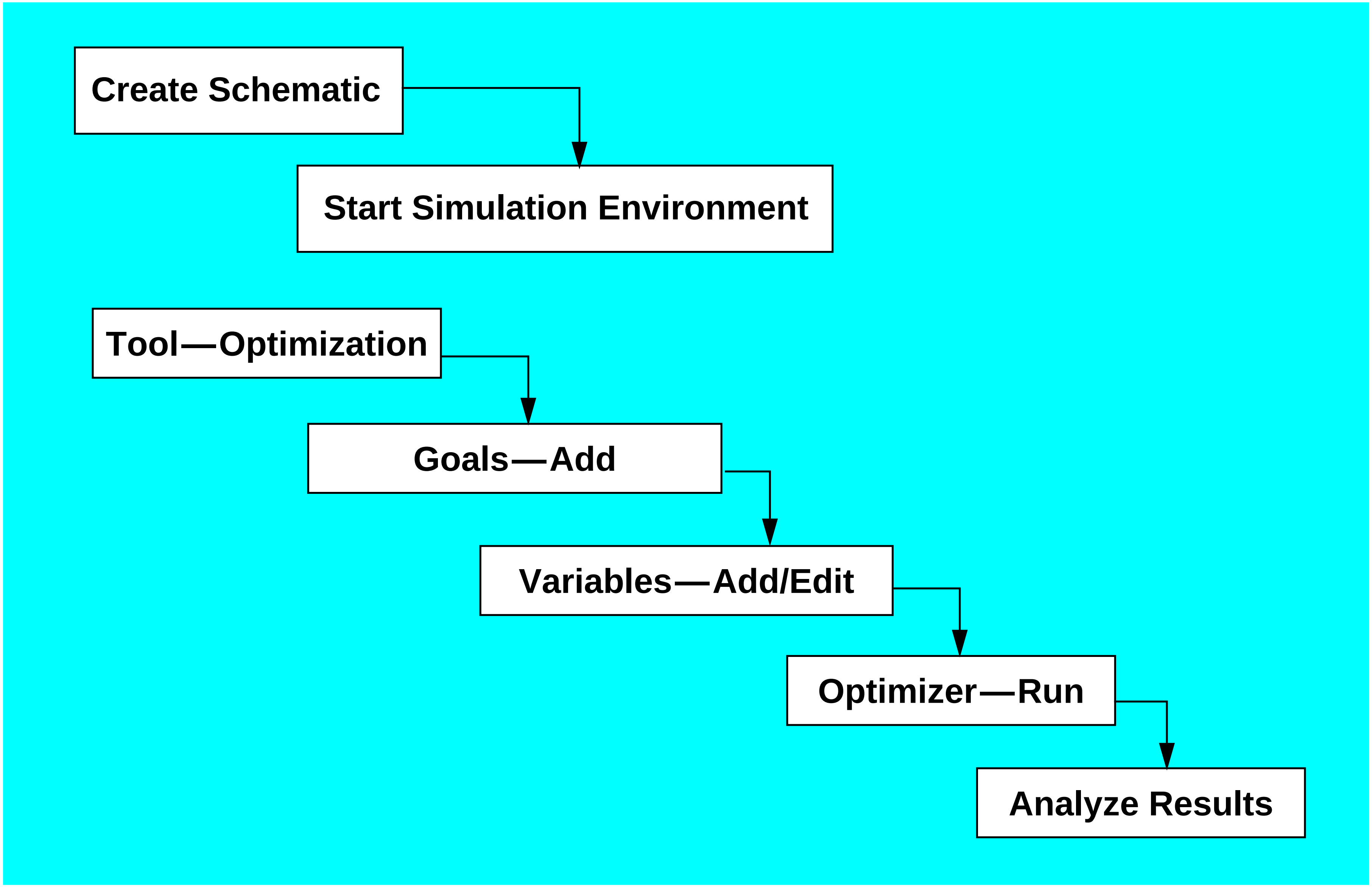
Introduction to the Circuit Optimizer Tool

- The Cadence Analog Circuit Optimizer is a software design tool used to find the optimum values of circuit components.
- The **Optimizer** is one of five **EDFM design tools**. As such, it is intended to set up and run multiple simulations. The Optimizer is unique in that the number of simulation runs is unknown. It is also unique in that the simulation input files are generated by the previous simulation results. The Optimizer is used to
  - **explore the design space**,
  - **establish circuit component values** used for the other EDFM tools,
  - **optimize circuit performance**, and
  - **develop accurate behavioral models** for the circuit.
- Use the Optimizer to automatically determine component values based on design goals.
  - The Optimizer eliminates the need for the tedious tasks of searching for component values to obtain specified performance (or to prove it is not feasible).

Introduction to the Circuit Optimizer Tool

The Cadence Analog Circuit Optimizer is a design tool intended to eliminate the tedious tasks of searching for component values to obtain a specified performance. The Optimizer performs the repetitive tasks of opening the simulation output files, analyzing the simulation results, altering component values, and then submitting the next set of simulation runs.

Optimization Analysis Flow



Create Schematic

Start Simulation Environment

Tool—Optimization

Goals—Add

Variables—Add/Edit

Optimizer—Run

Analyze Results

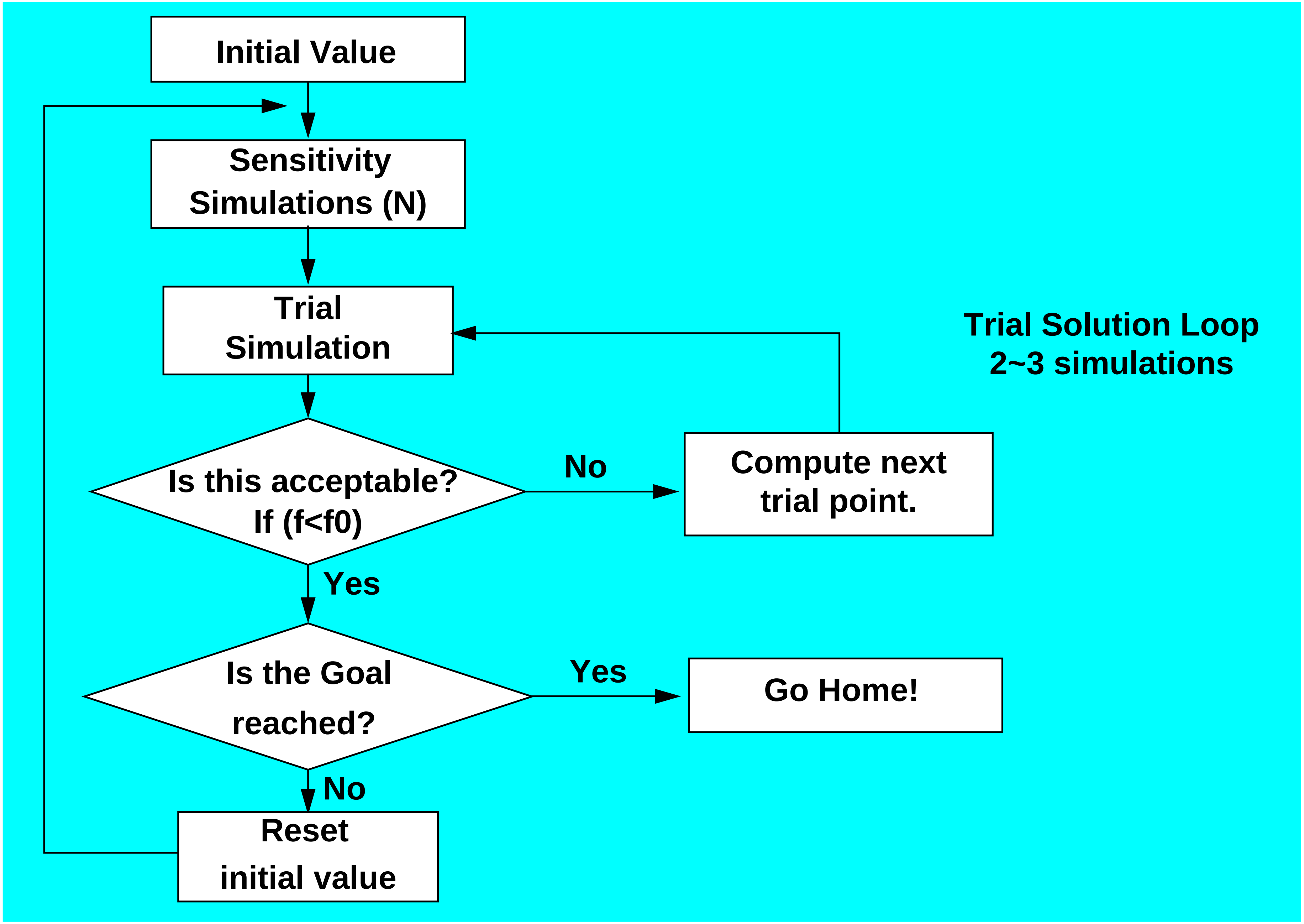
Optimization Analysis Flow

Apply optimization profitably in a wide range of activities.

- When using a top-down design approach, optimize a circuit block so that its performance characteristics match the desired characteristics defined in an analog AHDL module.
- Using the opposite approach, optimize a macro or behavioral module so that it accurately describes the behavior of a circuit block. Then, instead of simulating with the circuit block, simulate with the macro or behavioral model, which usually runs much faster.
- Optimize model parameters so that they match measured device data under various conditions.
- Use optimization to address radio frequency (RF) design problems, such as impedance matching.
- To increase circuit yield, use optimization to achieve better design center values.
- Use optimization to match the frequency response of a filter to the specifications for the filter.
- Use optimization to balance design tradeoffs.

The product is called the Resolve Optimizer.

Optimization Computational Flow



Optimization Computational Flow

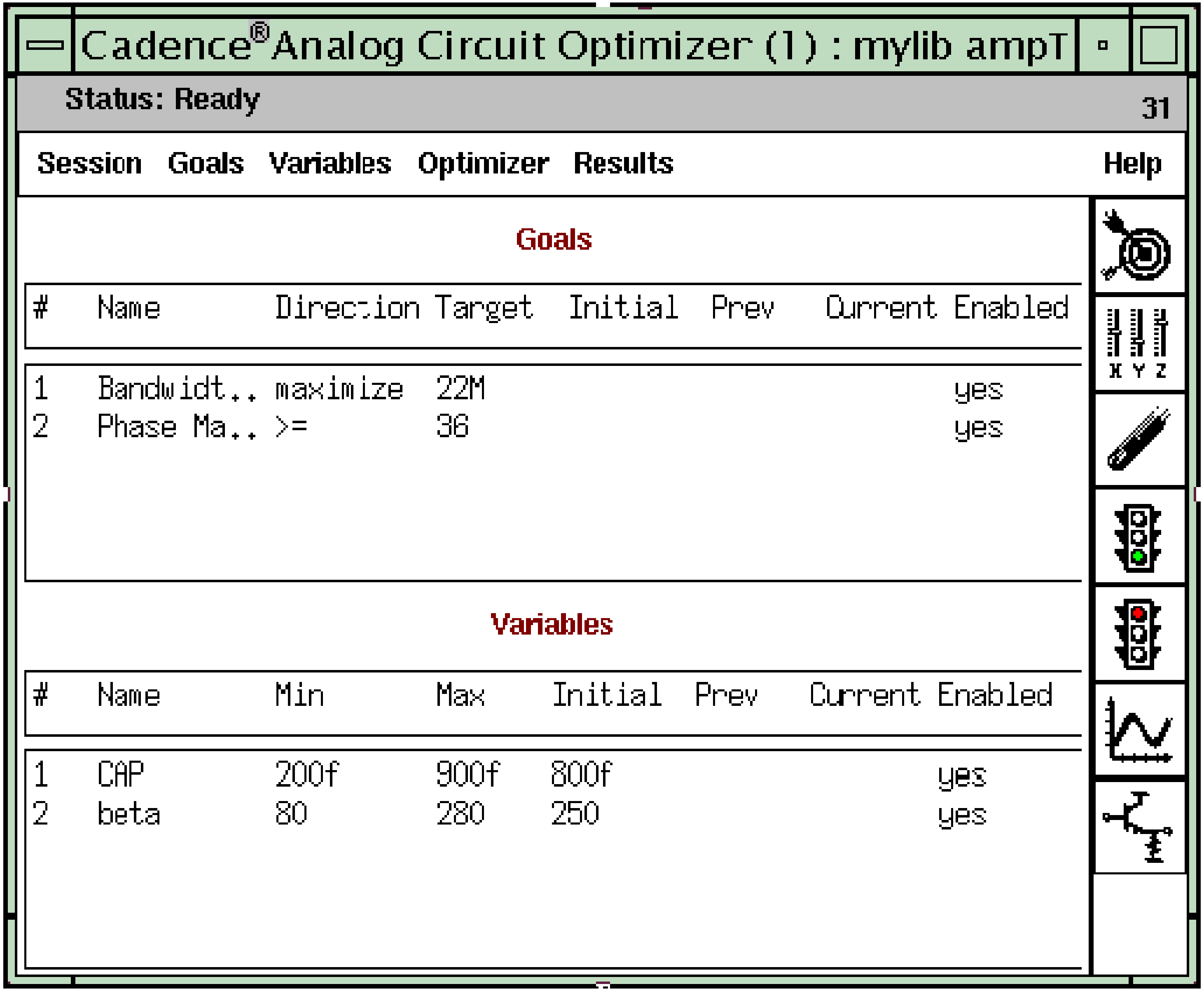
The Optimizer works in the following way. First, the system takes the initial values of the variables and runs a simulation to determine the values of the objectives/constraints. This simulation would also help to identify the functional objective and constraint.

Then, the system would automatically choose a better algorithm (LSQ and CFSQP) for the problem based on the input information and the information from the first simulation run. Next, the system determines the sensitivity of the objectives/constraints to each design variables by using finite differences. This involves separate perturbation on each design variable (typically by 0.5%) and simulating. When all the sensitivities have been computed, the optimizer determines a new set of values for the design variables. This process is repeated for all variables (N). At this time, the system does a trial simulation by setting the design variables to those previously determined by the algorithm. If this trial simulation is successful (that is the values obtained for the objectives are sufficiently better than the previous results), then the new values become the initial values for the second iteration. If the trial simulation is unsuccessful (that is the new values obtained are not better than the previous ones), then a new set of values will be computed and another simulation will take place. This completes the first iteration, and the second and subsequent iterations will follow the same steps.

Optimization stops using the CFSQP algorithm when the gradient is close to zero. Optimization stops for the LSQ algorithm when:

- The values of the design variables change very little or not at all.
- The goals do not change much.
- The gradient is close to zero.

Analog Circuit Optimization Option Form



Goals

#	Name	Direction	Target	Initial	Prev	Current	Enabled
1	Bandwidt..	maximize	22M				yes
2	Phase Ma..	>=	36				yes

Variables

#	Name	Min	Max	Initial	Prev	Current	Enabled
1	CAP	200f	900f	800f			yes
2	beta	80	280	250			yes

Tools—Optimization

Analog Circuit Optimization Option Form

The optimizer is accessed from the simulation environment with **Tools—Optimization**.

- Session

  Save or load the optimization setup and quit the optimization environment.

- Goals

  Add, edit, delete, enable, and disable your goals.

- Variables

  Add, edit, delete, enable, and disable your variables.

- Optimizer

  Run, step, and stop the engine.

- Results

  Update the design variables after you have attained the objectives.

Optimization Algorithms

CFSQP—C version Feasible Sequential Quadratic Programming

- CFSQP is suited for a wide variety of optimization problems, including constrained and unconstrained, minimizing and maximizing, and sequentially related goals.

  Example: Low-noise amplifier design, in which the goals are to specify gain, minimize noise, and maintain a phase margin greater than 45 degrees.

LSQ—Least Square

- Only optimization algorithm used in prior versions
- Good for measured, noisy, unconstrained data.

  Example: To design a filter with an output waveform that matches measured frequency response data (curve fitting).

- Primarily used for new curve fitting feature.

Algorithm selection is automatic, unless manually specified.

Optimization Algorithms

The CFSQP is an implementation of two algorithms based on Sequential Quadratic Programming (SQP), modified so as to generate feasible iterates. If the initial guess provided is not feasible for some inequality constraint, CFSQP first generates a feasible point for these constraints. Subsequently, the successive iterates generated by CFSQP all satisfy these constraints. When solving problems with many sequentially related constraints (or objectives), such as discretized semi-infinite programming (SIP) problems, CFSQP gives the option to use an algorithm that efficiently solves these problems, greatly reducing computational effort. You are provided the options on both goals and constraints to compute the gradients or have CFSQP estimate values by using finite differences. The default stopping criteria is that the gradient is close to zero.

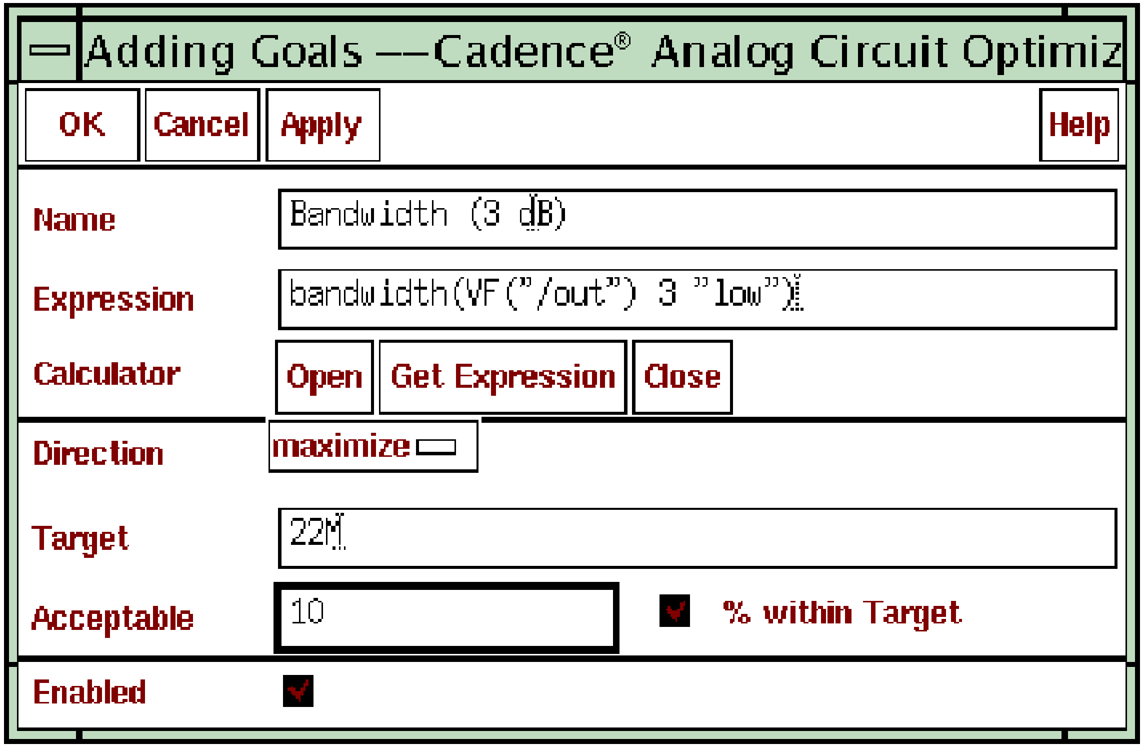
The LSQ is a special type of algorithm for curve fitting problem (with measurement data and Least Square Norm for deviation measurement). It is designed to handle the nonconstructive optimization problem with special form of objective functions. When dealing with constraints, it uses penalty functions.There are three stopping criteria for LSQ engine:

- Design variables do not change much.
- The goals do not change much.
- The gradient is close to zero.

Automatic Selection (can be overridden): If the objective is to minimize, maximize, or match a specific scalar value, then the CFSQP algorithm will be chosen. If there are no constraints (simple bounds on design variables) and the objective is to match measured data (curve fitting), then the LSQ engine is invoked to solve the problem.

Adding Goals

Goals—Add



Name	Bandwidth (3 dB)	
Expression	bandwidth(VF("/out") 3 "low")	
Calculator		
Direction	maximize	
Target	22M	
Acceptable	10	☑ % within Target
Enabled	☑	

Goals can be scalar or functional:

- Scalar—a fixed numerical value.
- Functional—a waveform or more than one point of data.

Data Fields	Scalar Goal	Functional Goal
Expression	Scalar	Waveform
Acceptable	Scalar	Scalar/Waveform
Target	Scalar	Scalar/Waveform

Adding Goals

This form is started from the Analog Circuit Optimization window. Specify the goals that the optimizer can work toward. These can be arbitrary expressions of design variables and other circuit quantities, as well as simulation results such as voltages and currents.

Name The name of the function.

Expression The expression can be entered manually or by using the Calculator.

Calculator Open the Calculator and use the Get Expression button.

Direction Match your direction to a given target value, maximize the expression, or minimize the expression. The Target and Acceptable values are used for tradeoff purposes (for example, decrease one objective to improve another.)

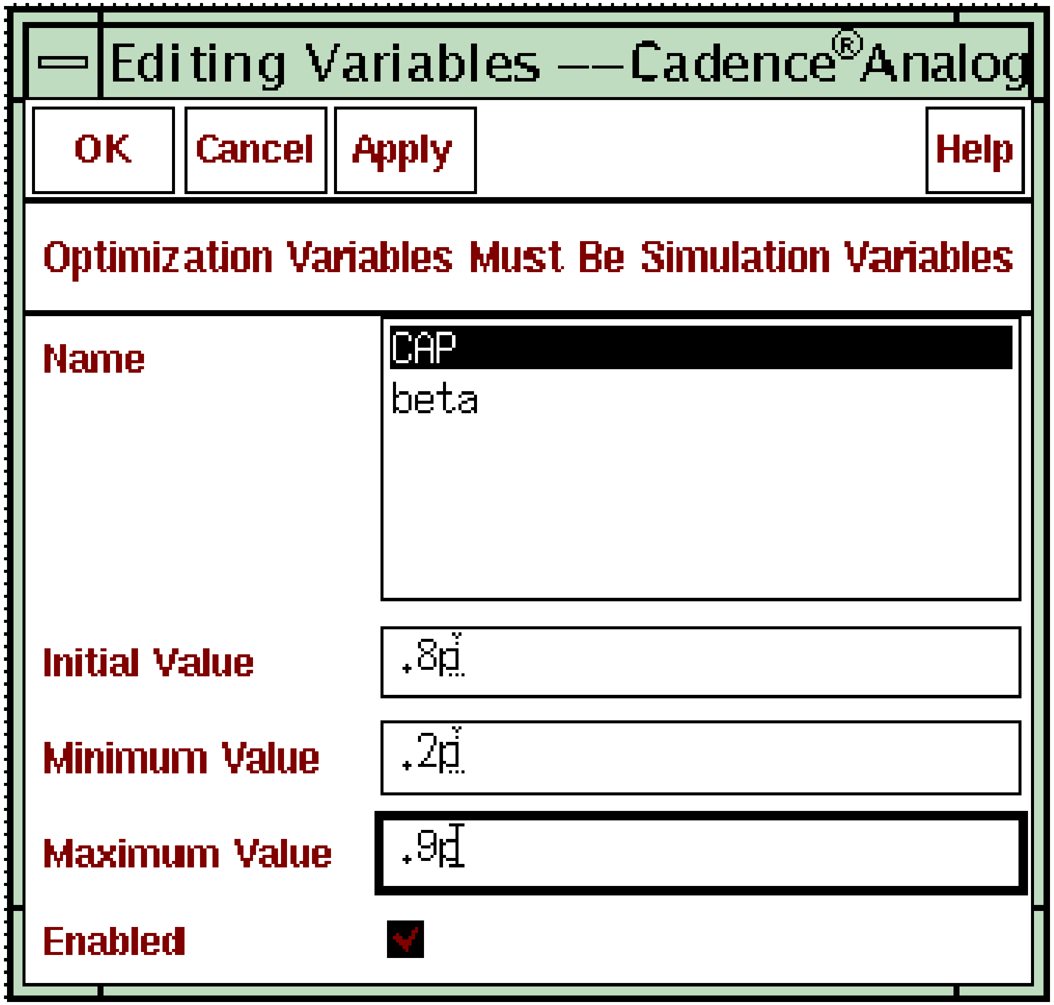
Target Select your target to be <= or >= than a certain value. The values for the Target and Acceptable fields are used by the algorithm to attach weights to the constraints. Also, specify the percentage (%) within the target.

Enabled Turns on/off this goal during optimization.

Use a functional goal to optimize in curve fitting applications.

Design Variables Menu

Variables—Add/Edit



Optimization Variables Must Be Simulation Variables

Name: CAP, beta

Initial Value: .8p

Minimum Value: .2p

Maximum Value: .9p

Enabled ☑

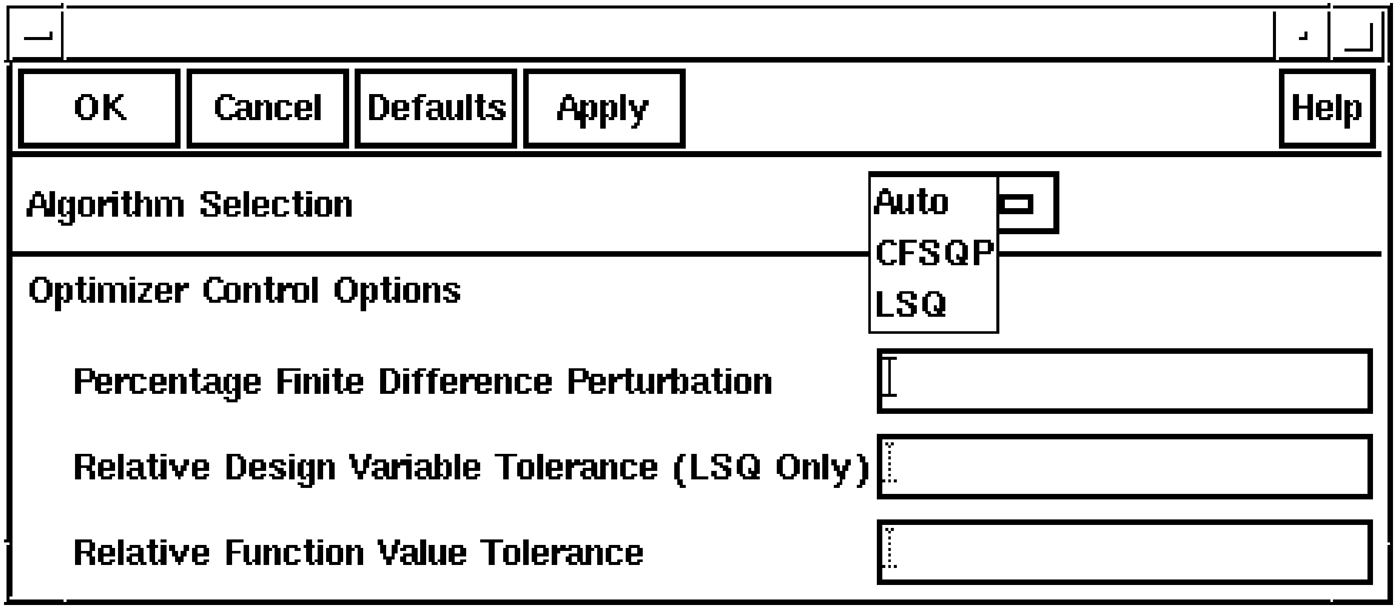
Design Variables Menu

After setting up the objectives/constraints form, specify which design variables to use. These are the variables that the optimizer can change in the process of meeting the design targets.

Optimization design variables must be Simulation environment variables. Typical variables can be component parameters and device/model parameters.

Name	The name of the design variable in the design.
Initial Value	The initial value that the algorithm will use to start the optimization step.
Minimum Value	The minimum value for the design variable.
Maximum Value	The maximum value for the design variable.
Enabled	Turns on/off this design variable during optimization.

Options Menu


OK
Cancel
Defaults
Apply
Help
Algorithm Selection
Auto
CFSQP
LSQ
Optimizer Control Options
Percentage Finite Difference Perturbation
Relative Design Variable Tolerance (LSQ Only)
Relative Function Value Tolerance

Sessions—Options

- **Algorithm Selection**: Specify the optimization algorithm.
- **Percentage Finite Difference Perturbation**: Set the step length.
- **Relative Design Variable and Function Value Tolerance**: Specify the relative percentage change in values for the stopping criteria for LSQ.

Note: These are expressed as absolute numbers rather than percentages (for instance, 0.05 is equivalent to 5%.)

Options Menu

Algorithm Selection: Generally, there is no need to pick which algorithm to use for the problem. The optimizer decides which algorithm to use based on the type of optimization to be done. However, this option provides the ability to force the optimizer to use a particular algorithm.

Percentage Finite Difference Perturbation: As is the case for algorithm selection, it is not necessary to set up the step: the optimizer uses its default value. This is suggested for most designers. For advanced designers who have better knowledge of the effect of the step length, the Finite Difference Perturbation field provides a way to specify the step length that is appropriate. Caution should be taken in using this because some problems are very sensitive to the step length used.

Relative Design Variable Tolerance (LSQ Only) and Relative Function Value Tolerance: The Relative Design Variable Tolerance and Relative Function Value Tolerance fields are designed to stop the algorithm by specifying stopping criteria to use rather than using the default settings. These fields are entered as absolute numbers instead of percentages. For example, if 0.05 is specified in the Relative Design Variable Tolerance field, that means if the relative change in the design variables is smaller than 5 percent, the algorithm would stop. Likewise, if 0.05 is specified in the Relative Function Value Tolerance field, the algorithm will stop when the relative change in each function value is smaller than 5 percent.

Curve Fitting

- Optimizing circuit output to match a desired waveform (*functional* goal).
- Waveform can be a waveform object or a file of x-y data.
- Many applications
  - Real circuit used to match an AHDL model
  - Filter responses
  - Impedance matching in RF applications
  - Design scaling and migration
  - Design centering in order to increase circuit yield
  - Filter frequency response can be optimized to meet the required performance
  - Balance design tradeoffs

Curve Fitting

Curve Fitting is the capability to optimize a circuit behavior to a set of waveform targets and constraints. Each objective consists of multiple points generated by the simulation analysis. This kind of curve fitting is usually used to optimize device model parameters or to improve circuit performance. Examples:

- Optimizing a circuit model to an AHDL module. In a top-down design, the final circuit parameters will be calculated based on the behavioral model performance. Place an AHDL instance as well as circuit instance and optimize the output from both. Select the output from AHDL instance as a target and from the circuit as an objective.
- The opposite case is to create a macro or behavioral module from a circuit model. In this case, it is preferred to save the circuit behavior in a file and use it as a target for the behavioral model. This is useful because the circuit doesn’t have to be resimulated (which is usually much slower to simulate) for every new AHDL or macro model parameter set.
- Device model parameters extraction. Model parameters are used as design variables that are optimized to produce device model characteristics that match measured device data at different conditions. Use this to migrate to a new device model that has better convergence characteristics, faster simulation time, and uses less memory.
- RF applications such as impedance matching.
- Design scaling and migration, and design centering in order to increase circuit yield.
- Filter frequency response and balance design tradeoffs.

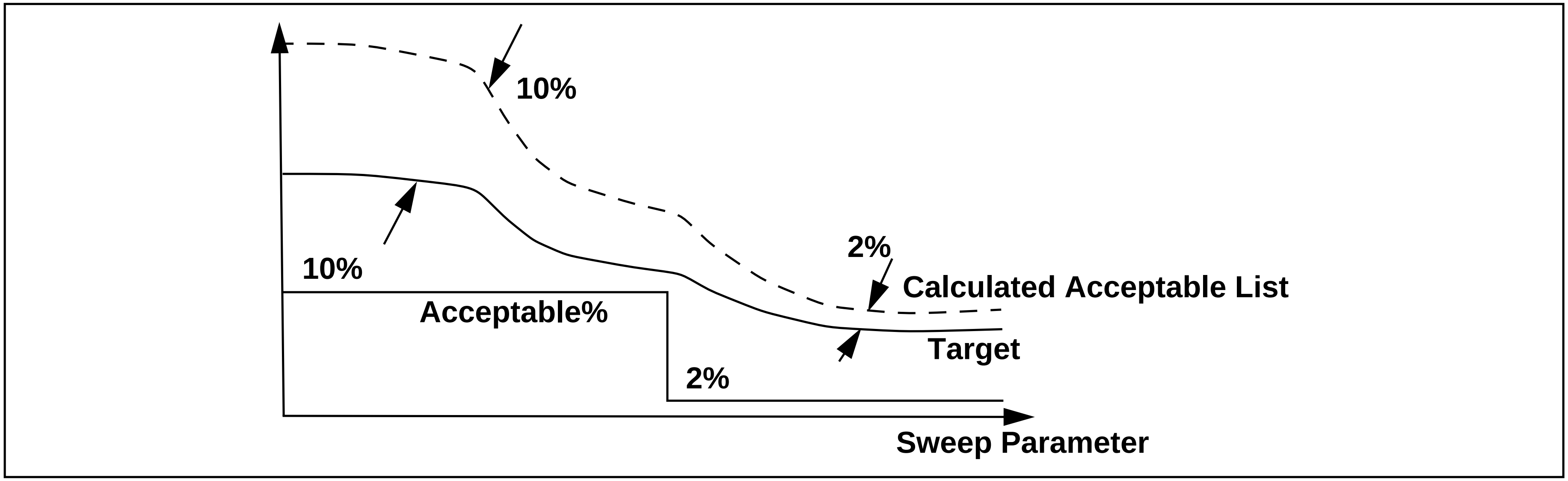
Curve Fitting (continued)

- When setting goals, use the Acceptable Percentage option:

Acceptable 5 ■ % within Target

A scalar entry gives the same weight to all calculated points.

- Use a table or a list to weight different parts of a curve to match.



Target	1	4	9	16	25
Acceptable%	10	10	2	2	2
Calculated Acceptable	1.1	4.4	9.18	16.32	25.5

Curve Fitting (continued)

This feature allows the acceptable value to be a percentage of the Target field at each separate point in the curve. The percentage field is useful to keep final optimization results consistent at very small and very large numbers in the Target list. Specify the percentage change around the Target. For example, if the Objective is to be at least 99% of the Target, specify Acceptable as 1% within Target.

The Acceptable field can be a scalar or a list. In the case of a list, each Target point will have its own Acceptable value. This will emphasize the importance of some areas in the curve over other less important regions.

The closer the Acceptable to the Target, the higher the Objective weight is. This is because the optimizer is using the following scaling factor to evaluate the Objective with respect Target:

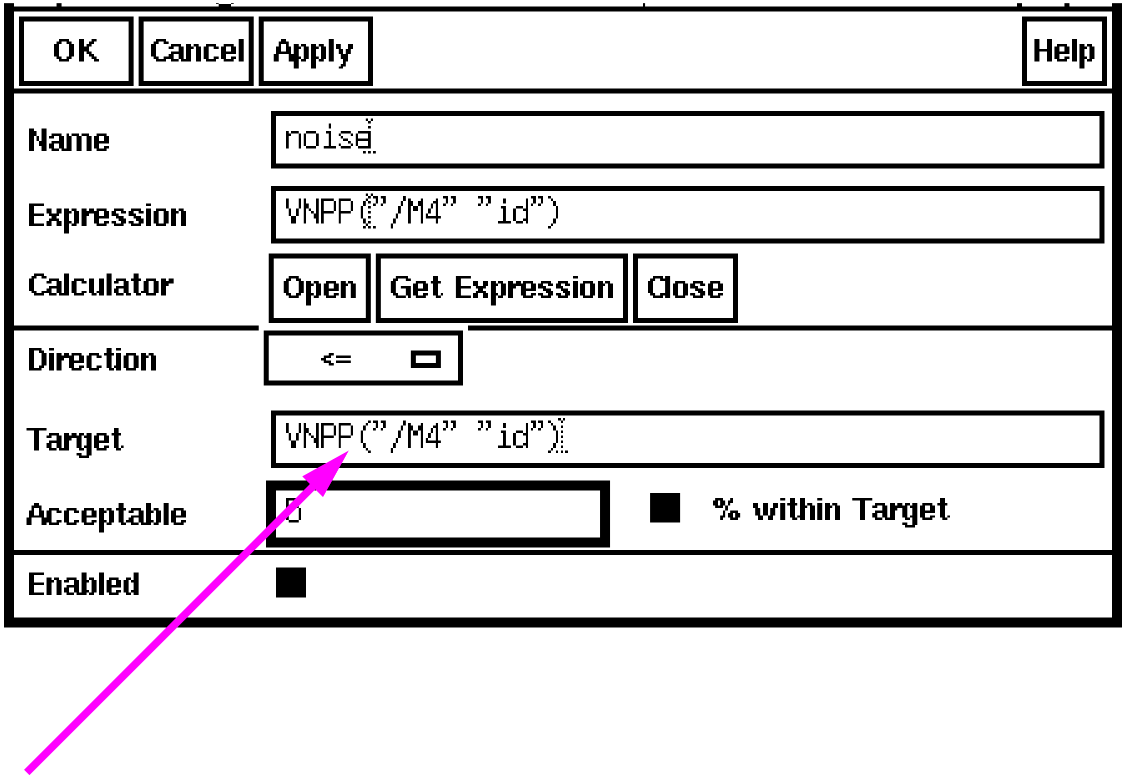
```
    (Objective - Target) / (Acceptable - Target)
```

When using absolute values for the Acceptable field, these values are used directly. On the other hand, specify the Acceptable as a percentage of the target as shown in the table above.

When specifying the Acceptable percentage field as scalar, all the points will have the same weight. But when defining an Acceptable percentage list, the resulting weighting will have the shape of the square wave shown above.

Curve Fitting to User-Defined Waveforms

Goals—Add



Name: noise

Expression: VNPP("/M4" "id")

Calculator

Direction: <=

Target: VNPP("/M4" "id")

Acceptable: 5 ■ % within Target

Enabled ■

Click the **wave** button in the calculator and select a waveform, or enter an expression or table object.

Curve Fitting to User-Defined Waveforms

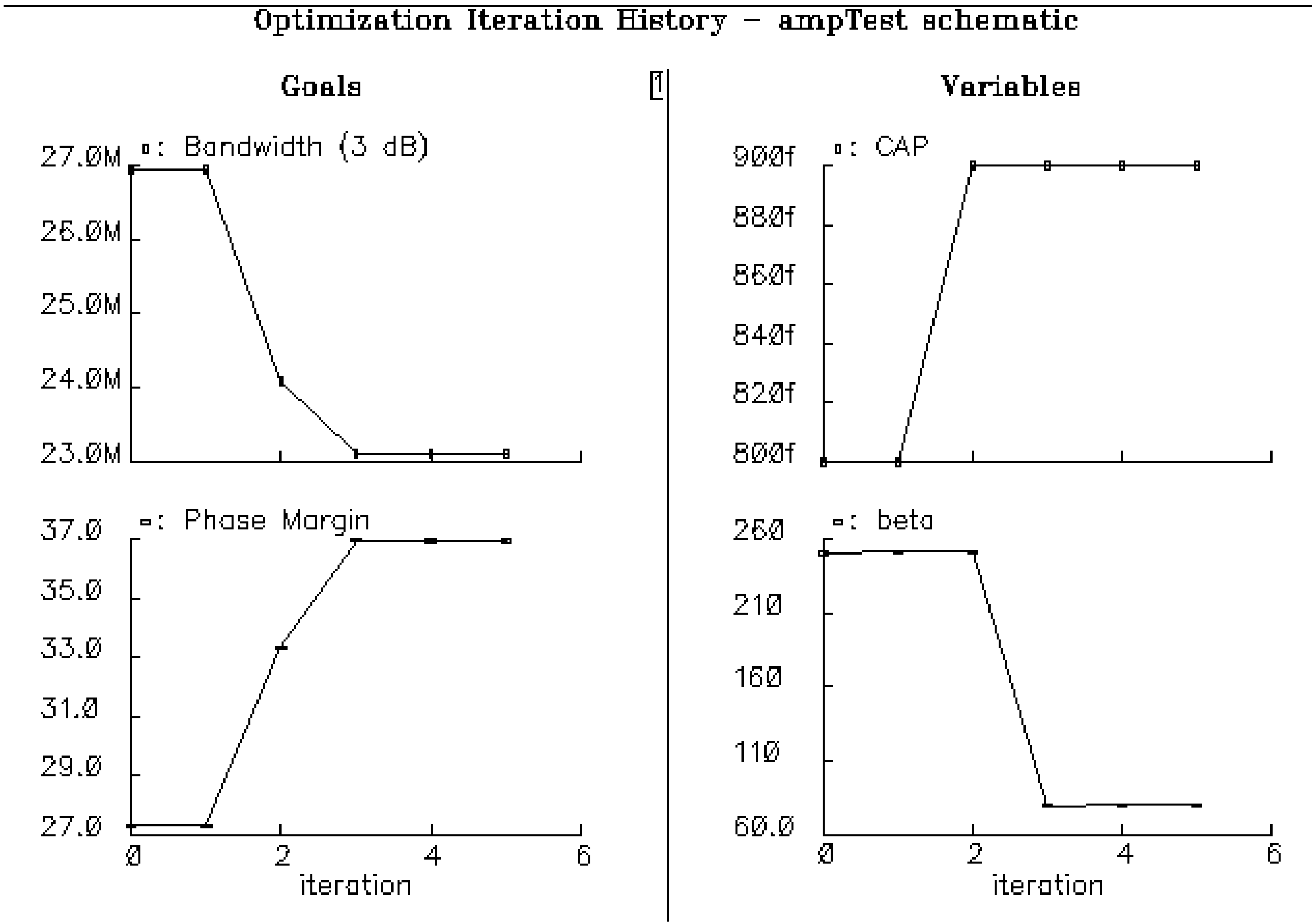
Define the target, goal, and acceptable waveforms. This data has to be converted to waveform variables to be used in conjunction with other tools and input forms. A file of x-y value pairs can be obtained through the print engine, a third party tool, or manually entered x-y data file. The expression for the waveform in the **Target** field above can be obtained from the calculator by clicking on the **wave** button and then the desired waveform. When running OCEAN, the waveform can be stored as a SKILL variable and then entered with the **goals** command.

The source of these waveforms can be any of the following:

- A user-defined SKILL list.
- A calculator SKILL expression.
- A tabular input data file.
- A waveform generated from previous results.

These variables will be used directly or in expressions and assigned to target, acceptable or constraints fields.

Iteration History



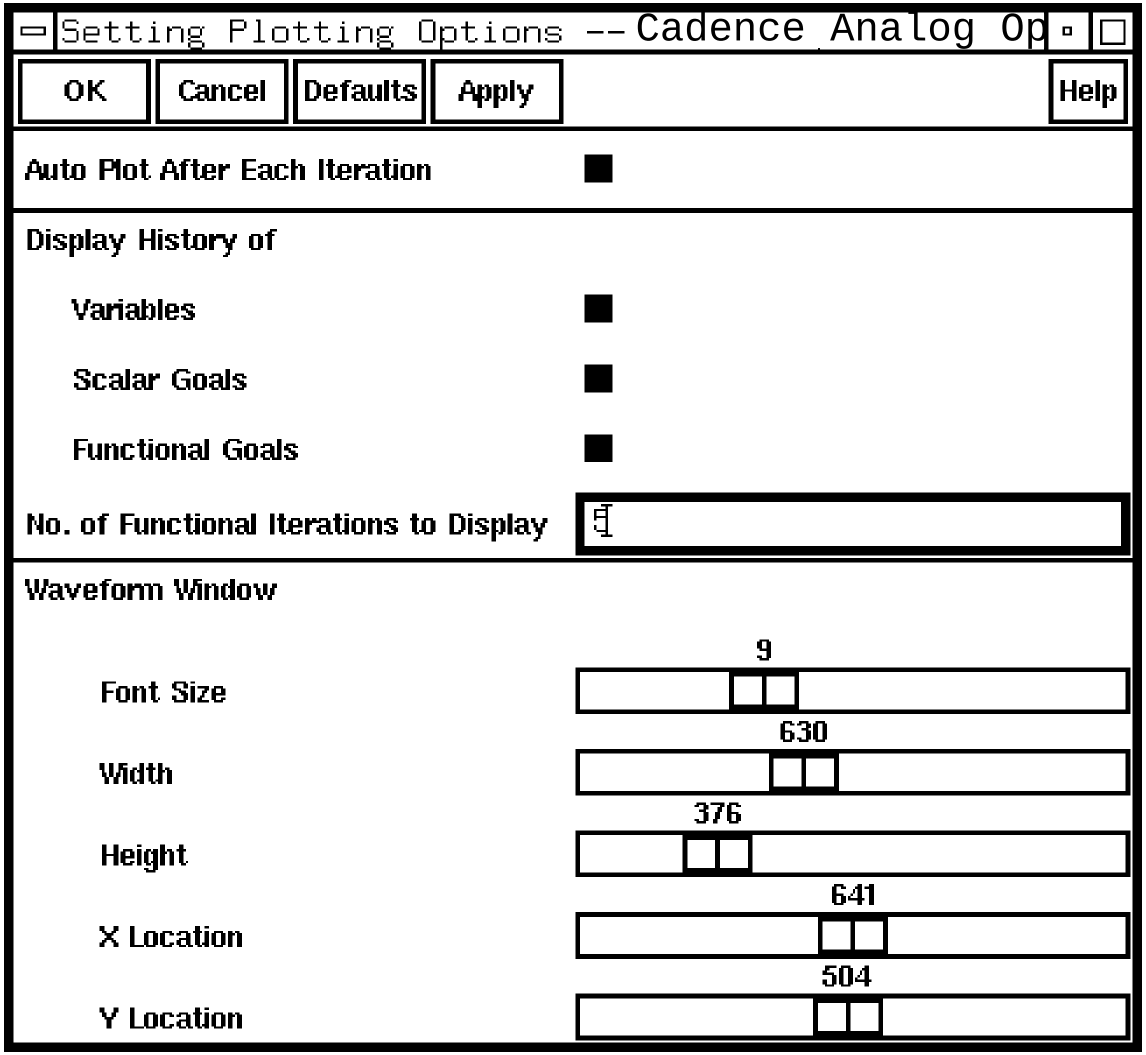
Iteration History

A Waveform Window appears during the optimization run that displays the history of the different values of the variables, as well as the goals that the simulator attempts to satisfy.

After the simulation ends, the final values of the goals and design variables appear in the Optimization window.

Specify which items to display in the Waveform Window during optimization. The options are shown in the graphic on the next page.

Plotting Options



Setting Plotting Options -- Cadence Analog Op

Auto Plot After Each Iteration ■

Display History of

Variables ■

Scalar Goals ■

Functional Goals ■

No. of Functional Iterations to Display 5

Waveform Window

Font Size 9

Width 630

Height 376

X Location 641

Y Location 504

Results—Set Plot Options

Plotting Options

Specify to plot the Variables, Scalar Goals, and Functional Goals during the optimization run.

- Variables: The values of the design variables that are modified during optimization
- Scalar Goals: The values of the scalar goals set by the user
- Functional Goals: The waveforms that are set as goals by the user

The **No. of Functional Iterations to Display** field specifies how many iterations of the optimization loop are plotted in the Waveform Window.

OCEAN Interface

- Select **Session—Save Ocean Script** to create an OCEAN script for batch optimization jobs.

- The following OCEAN commands have been added to allow optimization simulations:

```
optimizeAlgoControl
optimizeGoal
optimizeRun
optimizeVar
optimizePlotOption
```

Use *ocnHelp()* for more information on these commands.

OCEAN Interface

Labs

Lab 10-1 Running Optimization Analysis

Labs

Module 11: Component Description Format (CDF)

Topics in this module

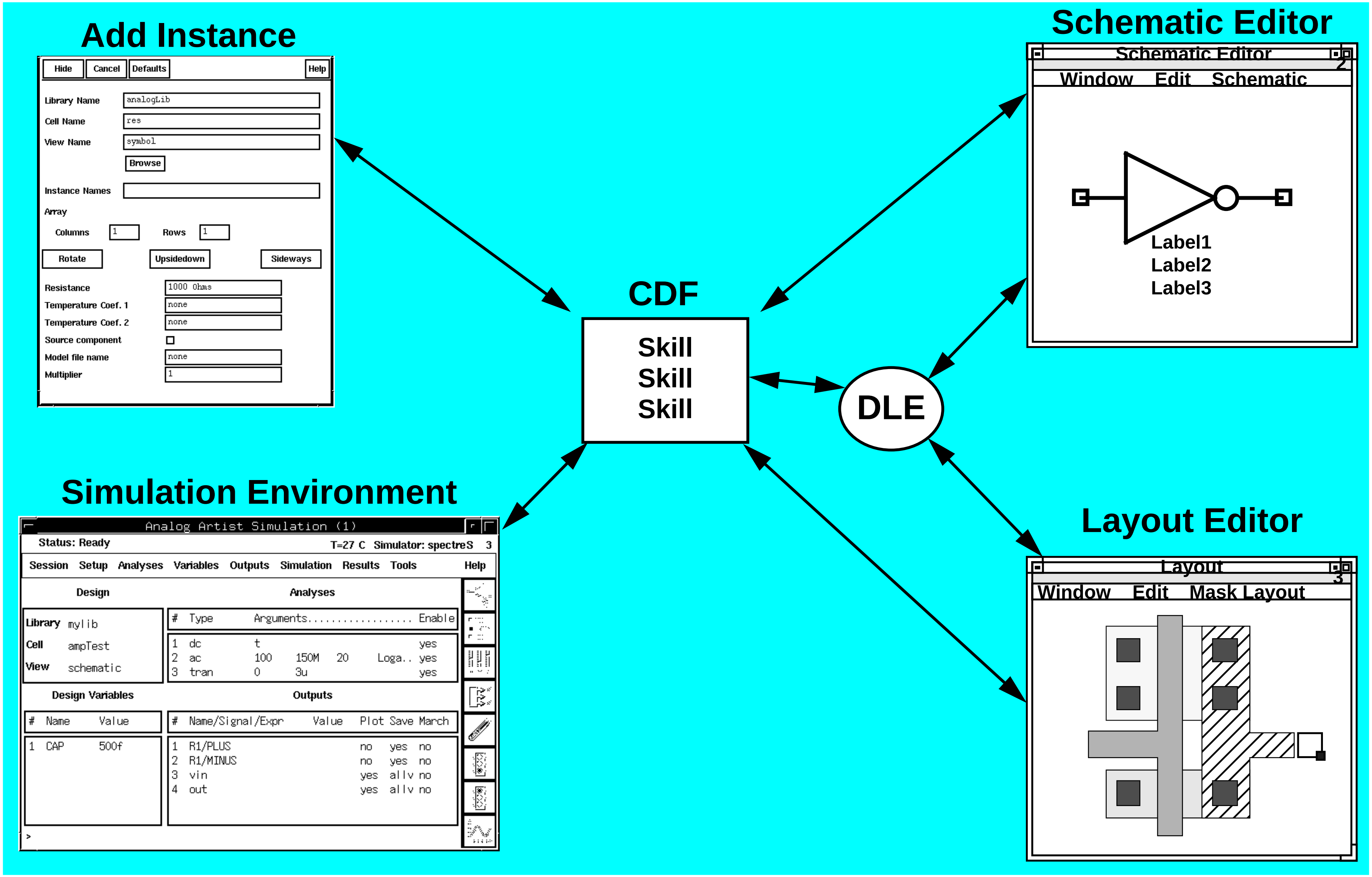
- Overview of CDF
- Types of CDF and CDF levels
- CDF user interface form
- Editing a component’s parameters using CDF
- Editing simulation information using CDF

Terms and Definitions

CDF	Acronym for Component Description Format
library CDF	Defines component properties common to all cells in the library.
cell CDF	Defines component properties unique to the cell in the library.
base-level CDF	Lowest level CDF, always saved to memory.
user-level CDF	User-defined property that masks a base-level CDF when applied to schematic components.
effective CDF	Combination of user- and base-level CDF used for simulation.
label display	Labels attached to components in schematics.
prompt	A displayed text that indicates a value is to be entered.

CDF Overview

CDF is encoded and attached to design database objects. CDF defines component behavior within the SKILL language.



Add Instance

Library Name	analogLib
Cell Name	res
View Name	symbol
Instance Names	
Array	
Columns	1
Rows	1
Resistance	1000 Ohms
Temperature Coef. 1	none
Temperature Coef. 2	none
Source component	□
Model file name	none
Multiplier	1

CDF

Skill
Skill
Skill

DLE

Schematic Editor

Simulation Environment

Analog Artist Simulation (1)
Session Setup Analyses Variables Outputs Simulation Results Tools Help

Status: Ready T=27 C Simulator: spectreS 3

Design

Library	mylib
Cell	ampTest
View	schematic

Analyses

#	Type	Arguments.................				Enable
1	dc	t				yes
2	ac	100	150M	20	Loga..	yes
3	tran	0	3u			yes

Design Variables

#	Name	Value
1	CAP	500f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	R1/PLUS		no	yes	no
2	R1/MINUS		no	yes	no
3	vin		yes	allv	no
4	out		yes	allv	no

>

Layout Editor

CDF Overview

CDFs are attached to components and libraries in the Design Framework II environment. It is a textual database written in the SKILL extension language. The Analog Design Environment references CDFs to gain information about design components. Some examples are:

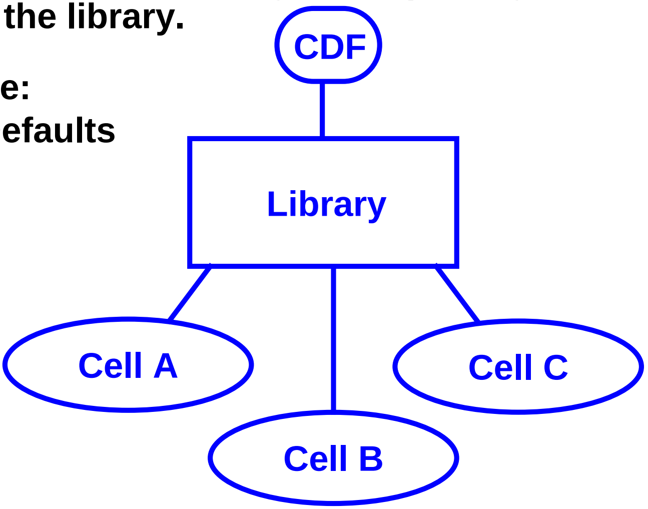
- The Add Instance form reads the CDF to display a component’s parameters. The CDF determines default value, type, and units. CDF can be used to limit available choices for component parameters (*parameterization*).
- Label displays in the schematic editor are controlled by the CDF. In fact, the Label Display form works by indirectly editing CDF.
- The simulation environment uses the CDF to generate the simulation netlist.
- The Layout Editor references the CDF to translate schematic components into layout components with the proper dimensions and electrical characteristics. As changes are made in the schematic, the layout can be continually updated.

Types of CDF

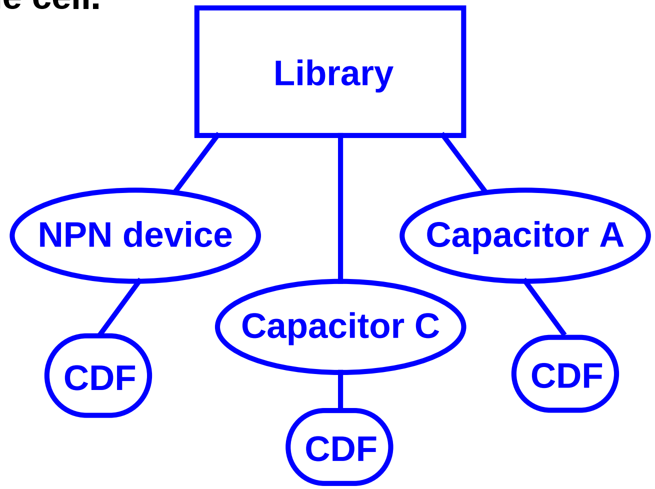
CDF is a part of a library or cell and is always copied with the database object.

CDF attached to a library is adopted by all cells in the library.

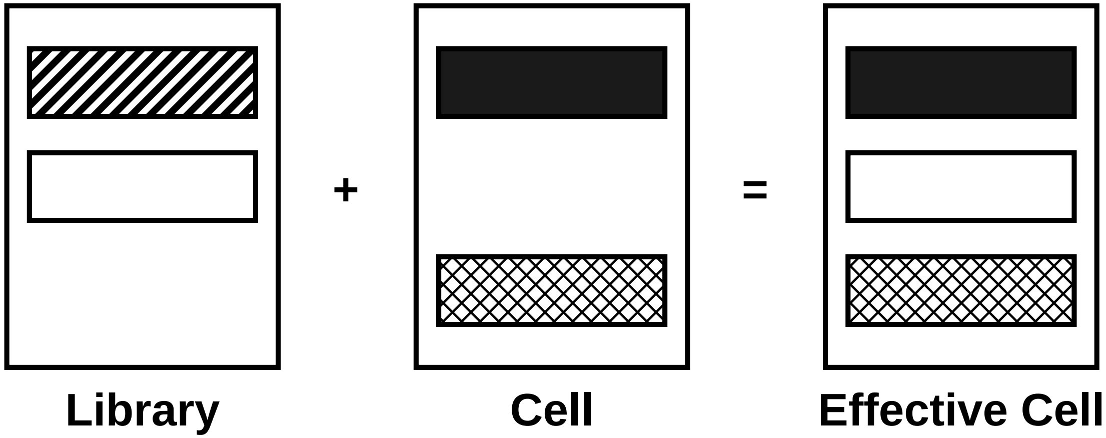
Example:
Label Defaults



CDF attached to a cell in a library is specific to the cell.



Cell CDF masks the *Library* CDF to produce an *Effective* result.



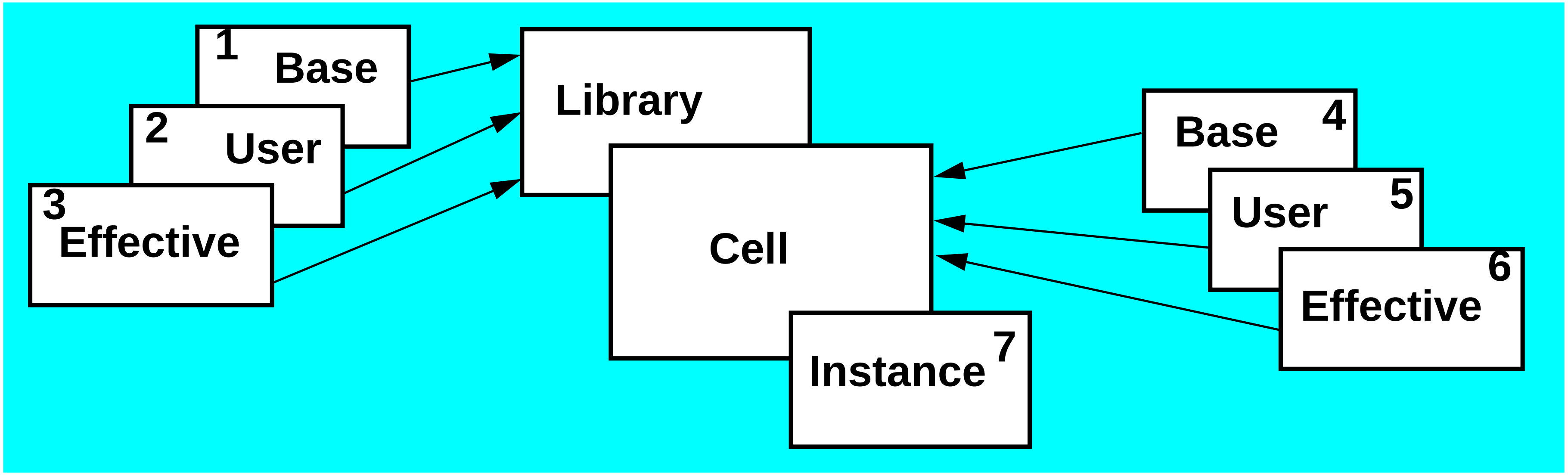
The *Effective Cell* CDF exists only in virtual memory.

Types of CDF

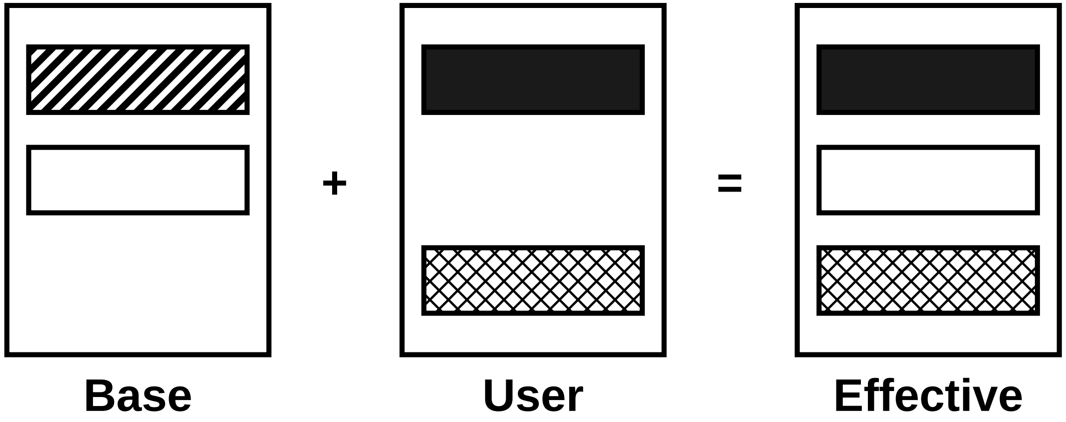
When attaching a CDF to a library, all components in the library inherit the description. When attaching a CDF to a component, the description is attached to the component's cell. Library-level CDF data is shared by all cells in the library. Cell-level CDF data creates or overwrites library level data.

Library CDF is the lowest level. *Cell* CDF overlaps the *Library* CDF to produce the *Effective Cell* CDF as illustrated. Note that *Cell* CDF overrides *Library* CDF adding additional quantities and redefining *Library* CDF quantities. *Effective Cell* CDF is the CDF used by the system. View *Effective* CDFs to see the combined *Library* and *Cell* CDFs.

Levels of the CDF



Base CDF is stored on disk. *User* CDF is a user-defined mask in virtual memory. The *Effective* CDF is the result of the user mask redefining the *Base* CDF.



The highest *Effective* level of CDF is used by the system. The highest level of CDF is the overlapping summation of all the CDFs at lower levels.

Levels of the CDF

A dual hierarchy of CDF exists. Library and cell CDFs have another level of hierarchy to describe them. *Base* CDF is the lowest level. *User* CDF masks the *Base* CDF to produce the *Effective* CDF as illustrated. Note that *User* CDF overrides *Base* CDF adding additional quantities and redefining *Base* CDF quantities. *Effective* CDF is the overlapping sum of all the existing *Base* and *User* level CDFs beneath it. It is the CDF used by the system.

The illustration shows the seven overlapping levels of CDF. The number indicates the precedence of the CDF. For an existing CDF, the highest numbered *Effective* CDF is the one used by the system. *Instance* CDF stores instance-specific quantities, like component parameter values or instance identifiers, and is always the highest level of CDF stored with the design database.

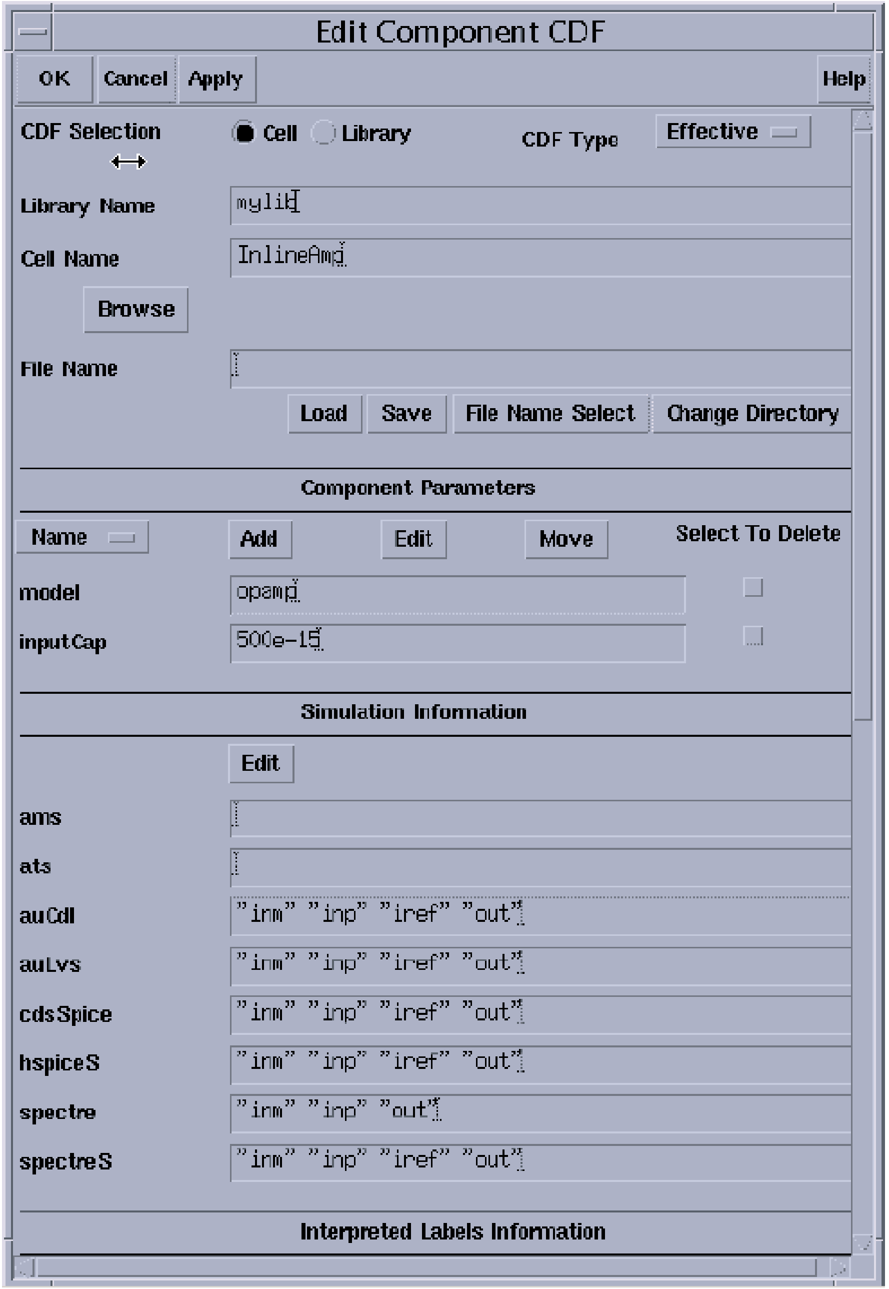
Effective and *User* CDFs, by default, are not saved beyond the current design session. However, you can save these settings to a file and attach them to other schematic objects in the future. Write permission is not needed to attach user-level CDF information to an object. Instance CDF (level 7) is edited when updating an instance’s properties in the schematic window. It is saved permanently when the design is saved.

Base CDF can be permanently saved to disk using the CDF user interface. The creator of a base-level CDF must have write permission to the object to which the description is being attached. We will discuss the CDF user interface next.

The *Label Display* form updates *User* CDF implying labels must be explicitly saved if they are to be retrieved after the current design session. To make label display information on components permanent, update component labels using the *Label Display* form and immediately copy the *Effective* CDF to the *Base* CDF. Label display settings are then saved to disk as part of the component or library description.

The CDF User Interface Form

In the CIW, select **Tools—CDF—Edit**.



Edit Component CDF

CDF Selection ● Cell ○ Library

CDF Type: Effective

Library Name: mylib

Cell Name: InlineAmp

File Name:

Component Parameters

Name

Select To Delete

model: opamp □

inputCap: 500e-15 □

Simulation Information

ams:

ats:

auCdl: "inm" "inp" "iref" "out"

auLvs: "inm" "inp" "iref" "out"

cdsSpice: "inm" "inp" "iref" "out"

hspiceS: "inm" "inp" "iref" "out"

spectre: "inm" "inp" "out"

spectreS: "inm" "inp" "iref" "out"

Interpreted Labels Information

NOTE:
This is a long form!
Form also includes entry fields for “Interpreted Labels Information” and “Other Information”

The CDF User Interface Form

Use the CDF user interface to create, view, or edit a CDF without having to write or edit a textual SKILL CDF. The CDF user interface form has header information, including the *library* and *cell* names. A CDF Type cyclic field selects the *Base*, *User*, or *Effective* CDF. This cyclic field is set to *Effective* when the form first appears. In this mode, the *User*-level CDF is generated indirectly when changing the effective results.

A file saving feature saves the settings of the CDF forms, but not the CDF itself. Saved settings can also be loaded back into the CDF. However, the CDF is generated for a component, or edits take effect, only when clicking **Apply** or **OK** in the CDF user interface form.

The four sections of the CDF appear below the header information. Under each section are relevant command buttons, such as **Add, Edit**, and **Move** in the first section, called *Component Parameters*. The remaining sections cover *Simulation Information*, *Interpreted Labels Information*, and *Other Information.* Avoid creating a CDF from scratch, since similar CDF information can be copied and edited from a sample component.

To view or edit a textual CDF file, type the following in the CIW:

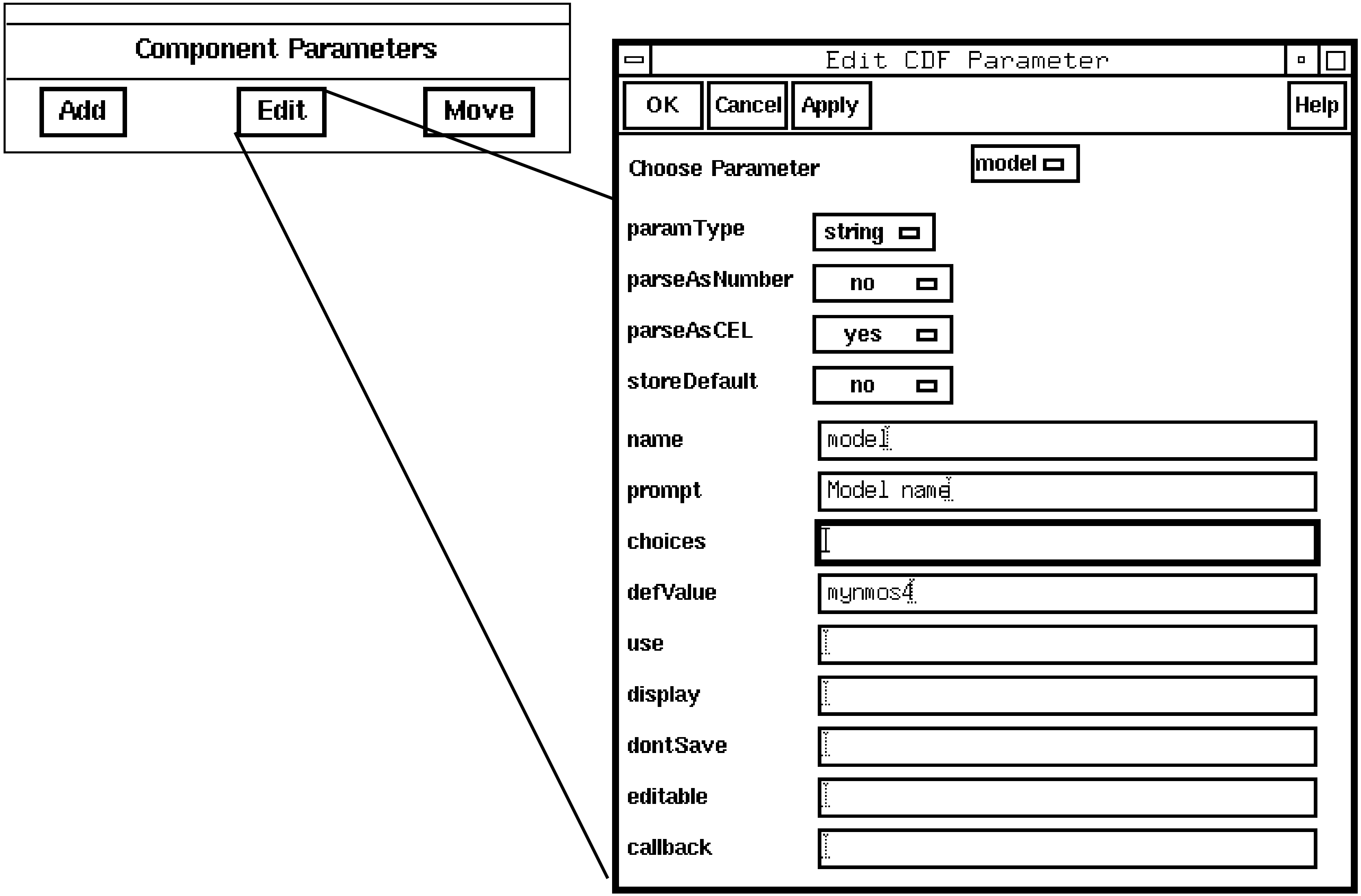
```
  cdfDump(“libName” “filename” ?cellName “cellName” ?level ‘<base or
user> ?edit <t or ’nil>)
```

To write the CDF of the *npn* cell in *analogLib* to a file named “test” that opens for edit, type:

```
  cdfDump(“analogLib” “test” ?cellName “npn” ?level ‘base ?edit t)
```

This file can be edited and loaded back into the CDF. See the online *Component Description Format User Guide* for more information.

Editing Component Parameters in the CDF



Choose Parameter	model
paramType	string
parseAsNumber	no
parseAsCEL	yes
storeDefault	no
name	model
prompt	Model name
choices	
defValue	mynmos4
use	
display	
dontSave	
editable	
callback	

- Click **Add** to add new component parameters.
- Click **Move** to change the order of listed parameters.

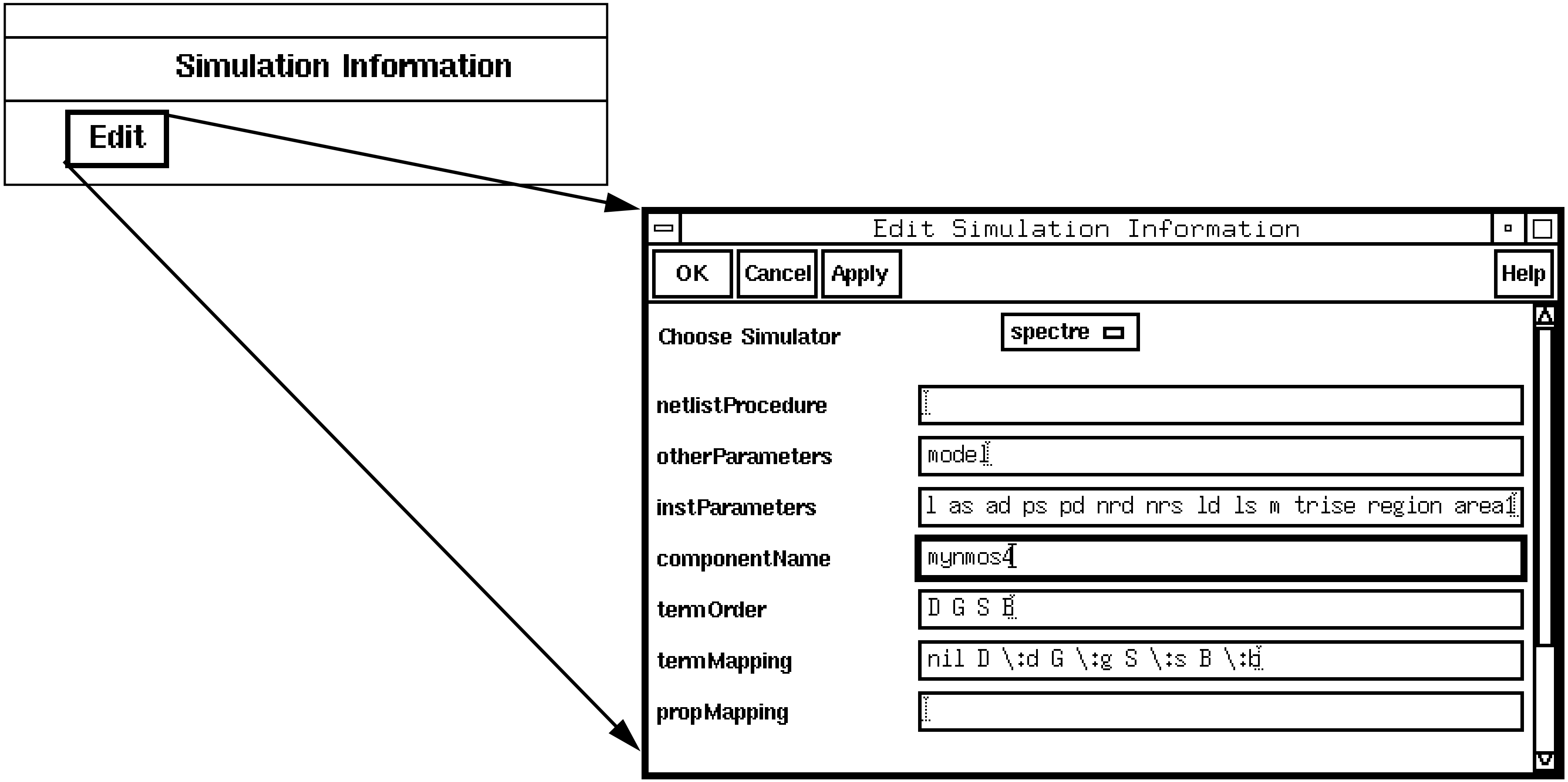
Editing Component Parameters in the CDF

A separate form is used to edit component parameters in the CDF. Each entry field has a complete definition in the *Component Description Format User Guide*.

Here are some field definitions that highlight CDF capabilities:

- Change the *paramType* field to make this component appear as a radio button, cyclic field, text field, etc. on any form that uses CDF.
- For most analog applications, the *parseAsCEL* field should be **yes**, and the *storeDefault* field should be **no**. For details, refer to the *Component Description Format User Guide*.
- If there were multiple choices for this parameter, they could be entered in the *choices* field. Examples would be the cyclic or radio parameter types.
- The component parameter has a *name* and a *prompt* as it would appear on a form that uses CDF. For example, the *Add Instance* form.
- Set a default parameter value in the *defValue* field. Some parameterized component types require a default value in the CDF.
- The *callback* field contains the name of a SKILL routine loaded into the Analog Design Environment that can alter this component parameter's values based on the values of the other component parameters. An example is size-dependent models, which will be modified in a layout or schematic based on a parameter passed to the placed instance. Callbacks are discussed in depth in the *Component Description Format User Guide*, and are **not** recommended for use with the Spectre Direct simulation environment.

Editing Simulation Information in the CDF



Choose Simulator: spectre

netlistProcedure:

otherParameters: model

instParameters: l as ad ps pd nrd nrs ld ls m trise region area1

componentName: mynmos4

termOrder: D G S B

termMapping: nil D \:d G \:g S \:s B \:b

propMapping:

This is the Spectre simulation information for a 4-terminal NMOS with a model parameter *area1* (in addition to others) that is passed to the model file.

There is a Simulation Information section for each simulator.

Editing Simulation Information in the CDF

This illustration is an example of the CDF Simulation Information that you can edit. The parameters and their values are specific to the chosen simulator. In this case, the Choose Simulator cyclic field is set to *spectre*.

Editing Simulation Information in the CDF (continued)

Set the *instParameters* field for the Spectre simulator to pass information from the design to a parameterized model by the following:

1. Create a parameterized model file (inline subcircuit):

   ```
   inline subckt mynmos4 (d g s b)
   parameters area1 = 10
   mynmos4 (d g s b) trnmos1 l=sqrt(area1)*1e-6 w=10*sqrt(area1)*1e-6
   model trnmos1 mos2 type=n vto=(area1/100)*0.775 tox=400e-10 nsub=8e+15\ ...
   ```

2. Define CDF component parameters or add a User Property that passes to the model file. Example: *area1*
3. List the component parameters in the *instParameters* field of the Simulation Information Section. Example: *instParameters* =area1
4. Set the *componentName*=Model Name, **OR** set a value for the **model** component parameter and set *otherParameters*=model.

Set or reset the model parameter value when placing a component instance.

The netlister sets the value of each parameter for each instance that uses the model file.

Editing Simulation Information in the CDF (continued)

Edit the CDF Simulation Information section for every simulation engine integrated into your system. Consult the *Component Description Format User Guide* for details on form settings. Information in this form is used by the system for netlisting purposes. The examples in this book cover the use of the Spectre Direct simulator.

The *instParameters* field can contain the names of a cell’s CDF parameters to be passed to the cell’s model file. To accomplish this, first write a model file (inline or standard subcircuit, or standard model file) that contains parameters, as in this example:

```
inline subckt mynmos4 (d g s b)
parameters area1 = 10
mynmos4 (d g s b) trnmos1 l=sqrt(area1)*1e-6 w=10*sqrt(area1)*1e-6
model trnmos1 mos2 type=n vto=(area1/100)*0.775 tox=400e-10 nsub=8e+15 \
...
```

The parameter *area1* is set in the model. Next, add this component parameter to the CDF and to each parameter to be passed into a model. Finally edit the CDF simulation information and list the names of the component parameters in the *instParameters* field. In addition, set the *componentName* field to be the name of the model file (inline or standard subcircuit, or standard model file) to point to, or create, a component parameter named **model**. Set the value of **model** to the name of the model file. Consult the *Component Description Format User Guide* for details.

Values for CDF component parameters are requested when the component is placed in a design, and can overwrite the default values set in the model file. Assign numbers or variables to the parameters. If choosing the latter, the model parameters can be swept in a parametric analysis.

Summary

In this module we discussed:

- Overview of CDFs
- That CDF information is used to provide data between DFII design tools
- The types of CDFs and CDF Levels
- How to use the CDF user interface form
- Editing a component’s parameters using CDF
- Editing simulation information using CDF

Summary

Labs

Lab 11-1 The CDF User Interface

Lab 11-2 CDF Effects in Simulation

Labs

Module 12: Macromodels, Subcircuits, and Inline Subcircuits

Topics in this module

- Overview of Macromodels, Subcircuits, and Inline Subcircuits
- Macromodels
- Subcircuits
- Library requirements to use Subcircuits
- Inline Subcircuits
- Inline Subcircuit example: Parasitic Devices
- Advantages of Inline Subcircuits
- Generalized Binning

Terms and Definitions

macromodel	A behavioral model using a simplified schematic or text file.
subcircuit	A behavioral model using a text file.
inline subcircuit	Special subcircuit with one component having the same name as the instantiated subcircuit.
high-level description	A simplified behavioral description replacing the schematic.
generalized binning	Process of modeling parameters into regions.

Overview

Macromodels and **subcircuits** provide high-level descriptions of a cell block.

The **macromodel** is a behavioral representation of a circuit that is either a schematic or a text file.

When the **macromodel** is described with a text file, it is called a **subcircuit**.

Inline subcircuits are special-case subcircuits where one device has the same name as the instantiated subcircuit. Use inline subcircuits to:

1. Override model parameters based on the instance parameter.
2. Place virtual components such as a parasitic BJT.
3. Select a component based on instance parameter.

Overview

Macromodels and **subcircuits** provide a high-level description of a complex block or cell. The high level simulates faster than the transistor-level circuit description of the cell. As such, the macromodels or subcircuits are used to test functionality of circuit blocks within a hierarchy. This facilitates design and verification of the system with a top-down or with a mixed-hierarchy design methodology. The system or chip-level debug and verification proceeds rapidly using the high-level descriptions. The final transistor-level description, the *schematic* view, is used during final verification.

The **Inline subcircuit** is a special feature of the Spectre® simulator. The inline subcircuit provides customization of component parameters within the subcircuit.

Advantages of Inline Subcircuits

Advantages of Inline Subcircuits versus a conditional if statement in the Spectre environment:

- No need to redefine the instance name, such as mospar.
- Component shows up in postprocessing tools (Results Browser) with its instantiated name, rather than a hierarchical name.
- Allows annotation of operating points in the Analog Design Environment.
- Supports binning of models.
- Supports parasitic modeling.

  For example, add parasitics to a component by changing a model card to an inline subcircuit.

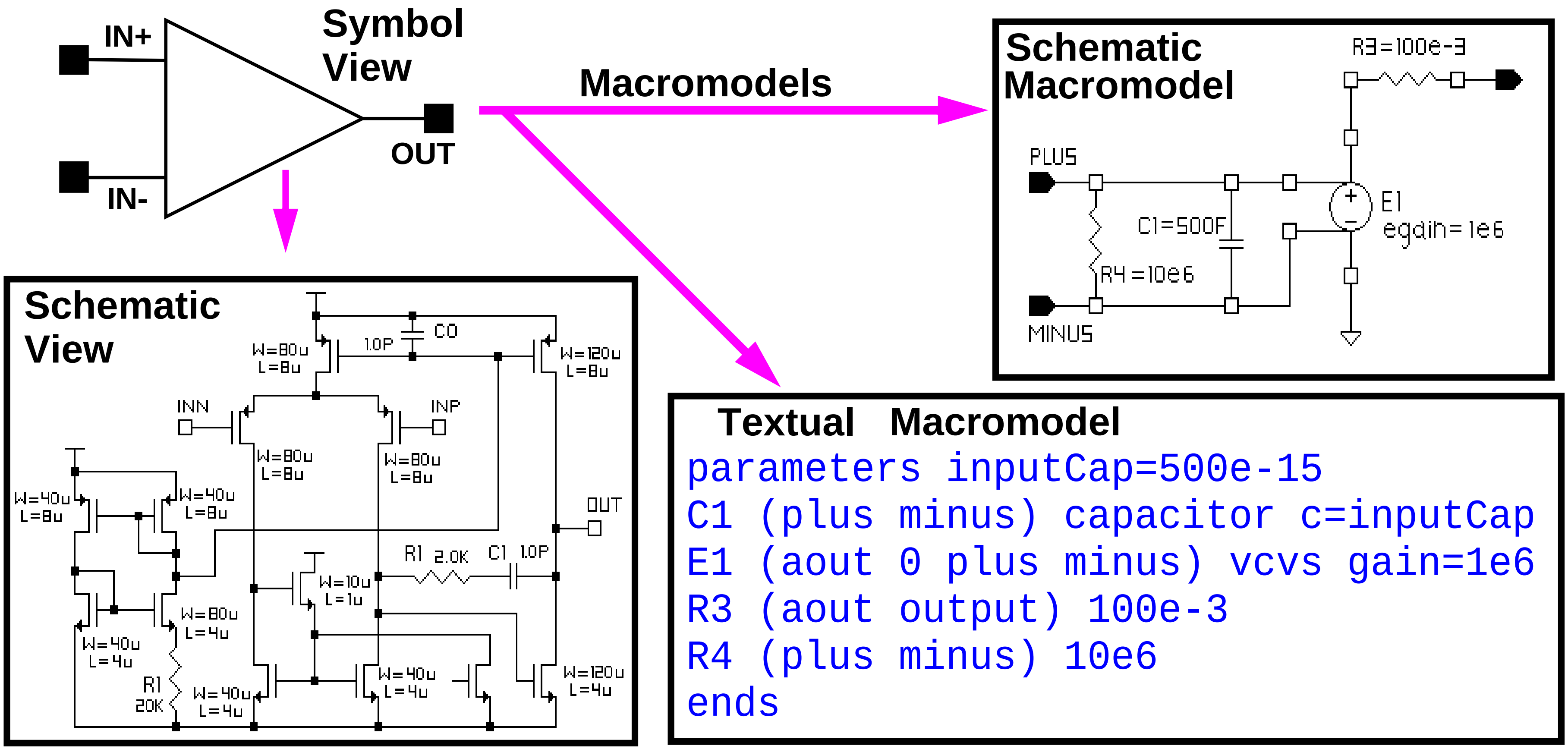
Advantages of Inline Subcircuits

Annotate operating points in the Analog Design Environment using inline subcircuits. (This cannot be done with regular subcircuits.)

Separate models into bins, to take advantage of foundry database modeling.

Macromodels and Subcircuits

A graphic instance can be described by a **schematic,** a **macromodel,** or a user-defined text file called a **subcircuit** file. The file is a subcircuit description.



Textual Macromodel

```
parameters inputCap=500e-15
C1 (plus minus) capacitor c=inputCap
E1 (aout 0 plus minus) vcvs gain=1e6
R3 (aout output) 100e-3
R4 (plus minus) 10e6
ends
```

A **macromodel** description or **subcircuit** file in the final simulation netlist will **simulate faster** than the schematic circuit description.

The parameter **inputCap** allows the capacitor **C1** to have the value passed into the subcircuit when the symbol is instantiated in a schematic.

Macromodels and Subcircuits

Macromodeling is a procedure that describes the functionality of a component that is a combination of many primitive devices. An example is an operational amplifier. Within the Analog Design Environment, there are two ways to create a macromodel.

1. Describe the functionality of a component with a schematic.
2. Describe the functionality of a component with a text file called a subcircuit file.

When creating macromodels, there are a number of things to keep in mind:

- What will the symbolic graphic look like?
- What special properties will the component need to use a subcircuit file?
- Will parameters pass from the component in the design to the subcircuit file?
- How is a subcircuit file created?

Library Requirements to Use Subcircuits

Cells referencing a user-defined subcircuit file require cellviews with:

- A symbol view.

  Create this using the symbol editor. Add symbol pins and a shape.

- A cellview identifying the cell as a primitive component.

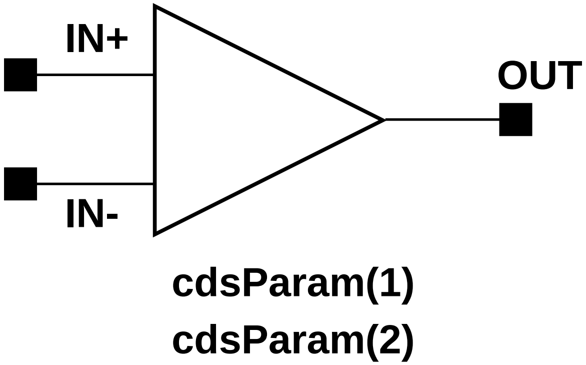
  Copy the symbol cellview to one that has the name of the simulator you are using, such as Spectre, Spectre S, cdsSpice.

- A cell CDF that contains the name of the subcircuit file and defines any parameters that are passed to the file.
- Pins on the subcircuit text that correspond to those on the symbol.
- Any other parameters to be defined.
- A subcircuit file can be defined in the Library Model File.

Library Requirements to Use Subcircuits

Creating the Graphic Symbol

Use the symbol editor to create any shape, which can have any number of input and output pins. When placing symbol pins on the graphic, name them and assign a direction. These names are important since references are made to them later when defining the CDF for the macromodel cell. Add interpreted labels to the symbol to provide annotation capability after simulation or display schematic information as labels. An operational amplifier can have a symbol resembling the following:



Creating the Primitive Cellview

Using the Library Manager, create a cellview for your cell that has the name of the target simulator. It is a copy of the *symbol* cellview. Edit the symbol view and save the information to a cellview named *spectre*, *cdsSpice*, or *spectreS* with a **Save As** command in the symbol editor. The simulator cellview tells your system that the cell is a primitive device that might use a subcircuit file to describe its behavior.

Inline Subcircuits

Inline subcircuits are an enhancement available only with the Spectre Direct simulator.

- Customization of component parameters

  Allows access to model parameters from the instance line

- Virtual Components

  Example: Parasitic BJT

  Example: Parasitic Estimation

- Automatic selection of components based on any instance parameter

Inline Subcircuits

An inline subcircuit is a special case of a subcircuit where one of the devices instantiated within the subcircuit takes on the name of the instance instantiation.

The “inline” component is denoted by giving it the same name as the inline subcircuit itself. When the subcircuit is flattened, the inline component does not take on a hierarchical name such as X1.M1, but rather takes on the name of the subckt call itself, such as X1. Any noninline components in the subckt take on the regular hierarchical name, just as if the concept of inline subckts never existed.

Use inline subcircuits to override model parameters from the instance line. This is a nice alternative to CDF callbacks, because ADE capabilities like parametric analysis will recognize the change to the netlist.

Also, if you implement complex subcircuits as single instances, you will not need to track the often cryptic hierarchical component names. Binning on any model parameter is also possible. The inline subcircuit also allows conditional binning.

For more details, enter these commands in an xterm window:

```
  spectre -h subckt
  spectre -h if (for conditional binning)
```

Inline Subcircuit Example: Parasitic Devices

```
inline subckt bjtpar (c b e )
  parameters we=10u le=10u // emitter width, length
  model npnmod bjt bf=1e3*(we*le/1e-10) is=1e-6*(we+le...)
...
  bjtpar (c b e s) npnmod // the inline component
  qpar (s c b) parpnp // parasitic device
  model parpnp bjt type=pnp.. // model for parasitic
ends bjtpar

// instantiate with:
q0 (1 2 3) bjtpar we=5u le=2u
```

Model parameters based on instance parameters

Inline subckt masquerades as a simple BJT.

Inline Subcircuit Example: Parasitic Devices

This example of an inline subcircuit shows how a designer can model a parasitic bjt device.

The parameters assignments within the subcircuit (we, le) can be overridden on the instance line, so that the model can be parameterized for each device.

In the postprocessing tools, the inline component for element q0 will be known as q0 and the parasitic element as q0.qpar.

In the traditional implementation of subcircuits, the instance is known by its full hierarchical name. With inlines, only the parasitic device is called by its hierarchical name.

The parasitic model (parpnp) can be specified within the subcircuit (as it is in this case), or it can be included as any other model with the include syntax of the Spectre simulator.

Important Points: Inline Subcircuits can promote model parameters to instance parameters, and can add parasitic devices to a component in a manner that is transparent to the user.

Inline Subcircuit Example: Parasitic Estimation

```
inline subckt mospar (d g s b)
  parameters mym=1 min=0 drainwidth=(0.4+0.25+0.3)*1e-6
+sourcewidth =(0.4+0.25+0.3)*1e-6 l=1u
if (min==0)
{mospar d g s b nch ad=mym*l*drainwidth as=
mym*l*sourcewidth
} else if (min==1)
{mospar d g s b nch ad=mym*l*drainwidth as= l*sourcewidth
} else
{mospar d g s b nch ad=l*drainwidth as= mym*l*sourcewidth
}
  model nch bsim3v3
ends mospar
```

Geometry selection based on instance parameter

Inline Subcircuit Example: Parasitic Estimation

The above example calculates the area of a MOS source and drain according to flags that define how the device will appear in the physical design.

In this case, there is a flag, *mym*, that defines the number of stripes, and a flag, *min*, that will minimize the source, the drain, or neither.

Generalized Binning

```
inline subckt NPNmod (c b e s)
parameters area=5e-12
  if ( area < 100e-12 ) {
  NPNmod (c b e s) npn10x10
  } else if ( area < 400e-12 ) {
  NPNmod (c b e s) npn20x20
  } else { NPNmod (c b e s) npn_default }
  model npn_default bjt is=3.2e-16 va=59.8
  model npn10x10 bjt is=3.5e-16 va=61.5
  model npn20x20 bjt is=3.77e-16 va=60.5
ends NPNmod
```

Model selection based on Instance Parameter

```
q1 (1 2 0 0) NPNmod area=350e-12 // gets npn20x20 model
q2 (1 3 0 0) NPNmod area=25e-12 // gets npn10x10 model
q3 (1 3 0 0) NPNmod area=1000e-12 // gets npn_default model
```

Generalized Binning

Model binning is selecting an appropriate model based on certain ranges of specified parameters. It is the process of partitioning a device with different sizes into different models.

With Spectre, model binning can be based on any parameter, and for any device. Other simulators are only able to bin on wmin, wmax, lmin and lmax. See the warnings about conditional instantiation in the *SpectreRF User Guide*.

Rules for General Purpose Model Binning

Within the subckt NPNmod, the inline device “NPNmod” is referenced three times (each with a different model). Allowing multiple “instances” or “references” to the same-named device will only be possible under the following, strict topological conditions:

- Reference to same-named device is only possible in a structural “if” statement which has both an “if” part and an “else” part.
- Both the “if” part and the “else” part must either be a simple one-statement block, or another structural “if” statement to which these same rules apply.
- Both the “if” part and the “else” part must evaluate to a single device instance, whose instance name, terminal list, and type of primitive are identical.

For example, multiple references to the same-named device will only be possible if there can be one single instance of this device after all expressions have been evaluated. In addition, each instance must be connected to the same nodes, and represent the same device (no topology change). The only thing that can change as the conditional expression changes is the actual model bound to the device.

Using Inlines with the Analog Design Environment

To use inline subckts in the Analog Design Environment with a primitive device:

- Write the text of the inline subckt.
- Place the component in the schematic.
- Do one of the following:
  - Using the CDF editor, create a parameter called model. Set a default value for this parameter in the Edit Component CDF form or set the value at the time you place the component in a schematic.
  - Set the base CDF Spectre simInfo field componentName to the name of the inline subckt.
  - Add the model parameter as a User Property to the Edit Object Properties form with a value that is the name of the inline subckt.
- Add any additional parameters needed as instanceParams in the simInfo section of the CDF.

Using Inlines with the Analog Design Environment

It is easy to use an inline subcircuit in the Analog Design Environment.

The value of **model** is the name of the inline subcircuit model.

There is additional information to the steps listed above in the following lecture on Macromodels and Subcircuits.

Values for CDF component parameters are requested when the component is placed in a design, and can overwrite the default values set in the model file. Assign numbers or variables to the parameters. If choosing the latter, the model parameters can be swept in a parametric analysis.

Labs

Lab 12-1 Creating a Parasitic Transistor Model

Lab 12-2 Using Subcircuit Cells

Lab 12-3 Adding a Subcircuit Representation

Labs

Module 13: Inherited Connections

Topics in this module

- Applications of inherited connections
- Setting net expressions and using the netSet property
- Netlist and Run with inherited connections

Terms and Definitions

inherited connection Selectable connection to a default or instantiated net.

default connection A connection to a net when a connection is not instantiated.

net expression A property added to a net that is programmable.

netSet Property added to component to override a default connection.

substrate connection A terminal of a component that is in contact to the substrate.

library duplication A parallel design library required for different applications.

global connection Connection to signal or supply node not requiring symbol pin.

Applications of Inherited Connections

Use inherited connections to:

- Defer where a net is attached until a cell that contains that net is used in the design hierarchy.
- Change the attach point on the fly by changing an instance property.
- Affect many points down a hierarchy with the change of only one parameter.
- Provide a programmable net capability.

  Inherited Connections enhances the link between the physical and logical tool flows.

Inherited Connections help to solve:

- Library Duplication—A cell can be designed with the power supplies left as inherited connections.
- Substrate Connections—Inherited connections provides a solution that works in both the physical and logical design space.
- Inherited Terminals—With inherited connections, add extra terminals to the layout views that are not hard coded.

Applications of Inherited Connections

With inherited connections, the circuit designer has the ability to defer where a net is attached until a cell that contains that net is used in the design hierarchy. The attachment point can be changed on the fly by changing an instance property. Because the capability works hierarchically, there is the ability to affect many points down a hierarchy with the change of only one parameter. The inherited connections concept is to provide a programmable net capability. Inherited connections enhances the link between the physical and logical tool flows.

Inherited connections help to solve

Library Duplication—A cell can be designed with the power supplies left as inherited connections. In the past, users would have multiple copies of the same cell, with the only difference being the connection of the power supplies.

Substrate Connections—Prior to inherited connections, the problem of resolving substrate connections required ingenuity, patience, and a large effort on design resources. Inherited connections provides a unique solution to the problem that is easily incorporated into the design flow.

Inherited Terminals—The physical implementation of a cell usually has more terminals than the logical implementation. Inherited connections can add extra terminals to the layout views that are not hard coded. There is no need to search for terminals in a layout and change them.

Features of Inherited Connections

- Inherited Connections allow you to create global signals and override their names for selected branches of the design hierarchy.
- Inherited Connections enable you to:
  - — Use multiple power supplies in a design.
  - — Add overridable substrate connections.
  - — Parameterize power and ground symbols.
- This override information can be accessed by other Cadence® tools across the design flow.
- Sample library and tutorial

```
<instal_dir>/tools/dfII/samples/tutorials/inhconn
```

Features of Inherited Connections

The inherited connections solution provides a long-needed answer to the problem of multiple power supplies. To implement separate power supplies (analog and digital, for example, or +3V and +5V) in a hierarchical design, assign net expressions to those global signals whose defaults will be overridden, and then use *netSet* properties to specify the new values of the signals. (Global signals are electrical signals that pass through more than one level of a multilevel design, at the same power level and with the same name.)

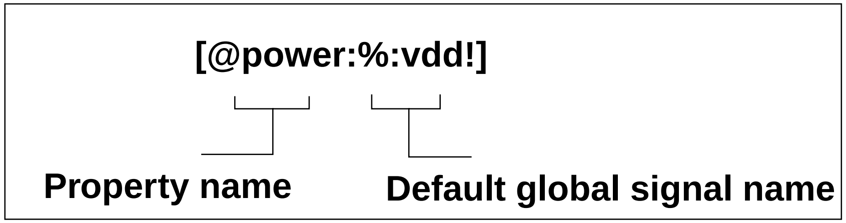
Net expressions assigned to signal names can be overridden with the specifications of global signals. The *netSet* properties redefine the value of the property assigned to a signal or terminal. Redefining the signal eliminates the problem of global nets being merged into a single, electrically equivalent signal, which occurs when the signal traverses the design hierarchy.

In summary, inherited connections allow global signals to be inherited through a design hierarchy. Override default values by setting *netSet* properties on instances where a net expression has been assigned to the signal at the level above. In this way, the *netSet* property values filter down through the hierarchy below that instance.

Defining Inherited Connections

- Add a net expression label to either a pin or a wire to define an inherited connection.
- The net expression defines a default global signal name for the connection and the name of the property overrides the global signal name.

Example Net Expression

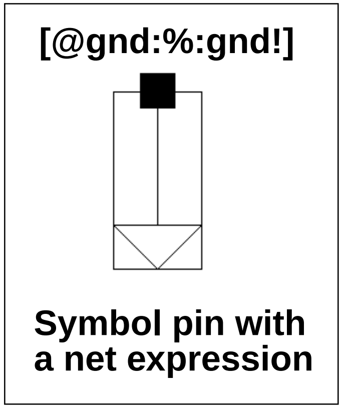


[@power:%:vdd!]

Property name

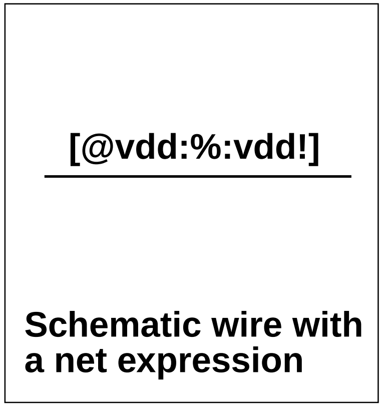
Default global signal name

Example Net Expression Labels



[@gnd:%:gnd!]

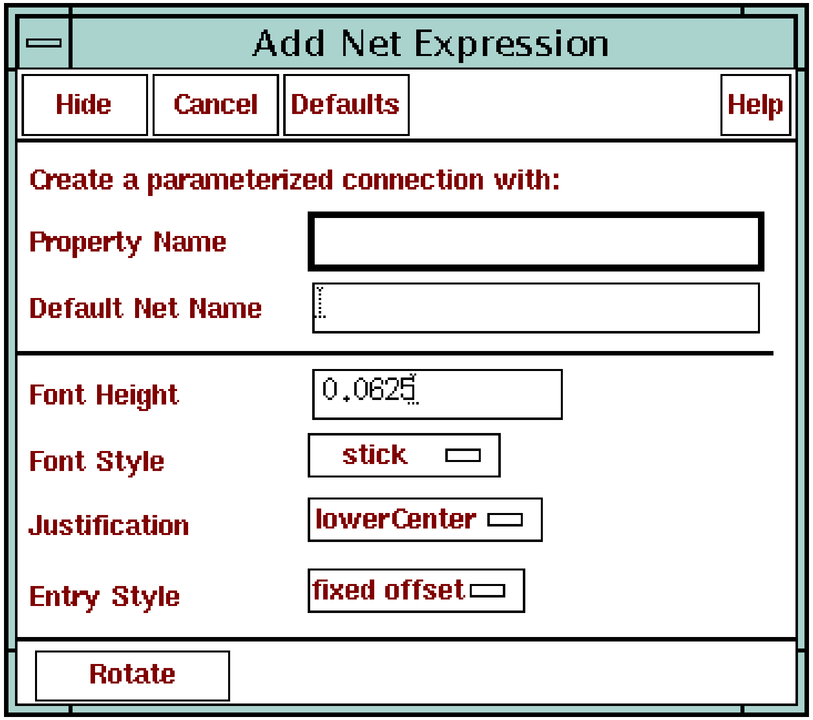
Symbol pin with a net expression



[@vdd:%:vdd!]

Schematic wire with a net expression

Add—Net Expression



Create a parameterized connection with:

Property Name	
Default Net Name	
Font Height	0.0625
Font Style	stick
Justification	lowerCenter
Entry Style	fixed offset

- The *basic* library contains sample power and ground symbols.

Defining Inherited Connections

The default global signal name specifies what the pin or wire is connected to by default. When a net expression is attached to a wire, the default name in the expression names the net. When a net expression is attached to a pin, the default name in the expression names the signal that the pin will be connected to by default.

Net expressions associated with symbol pins do not name the pin. The inherited terminal is created when the net expression is associated with the pin in the symbol editor. The schematic extractor processes expressions associated with symbol pins only when the extractor encounters an instance of the symbol in a schematic.

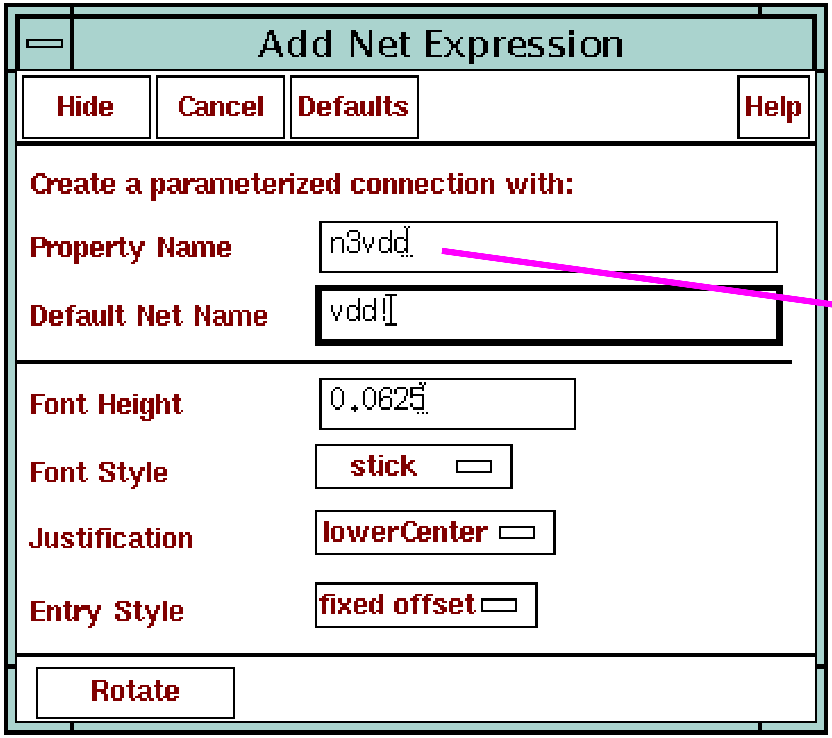
Check the schematic, and the Net Expression from the symbol pin is propagated into the schematic.

Basic Library Samples

The **basic** library contains sample parameterized power and ground supply symbols called *vcc_inherit*, *vdd_inherit*, *gnd_inherit*, and *vss_inherit*.

Inherited Supply Symbol	Net Expressions
basic vcc_inherit symbol	[@vcc:%:vcc!]
basic vdd_inherit symbol	[@vdd:%:vdd!]
basic gnd_inherit symbol	[@gnd:%:gnd!]
basic vss_inherit symbol	[@vss:%:vss!]

Setting a Net Expression



Create a parameterized connection with:

Property Name: n3vdd

Default Net Name: vdd!

Font Height: 0.0625

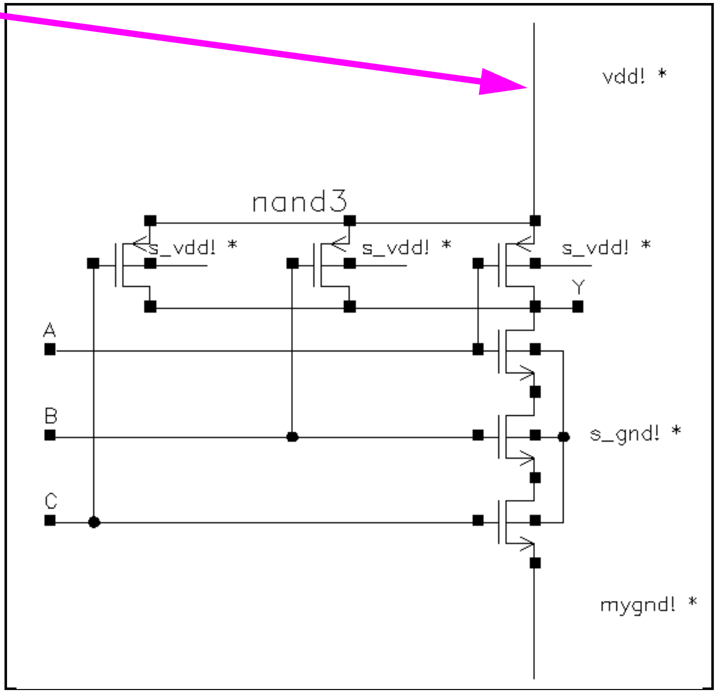
Font Style: stick

Justification: lowerCenter

Entry Style: fixed offset

Add—Net Expression

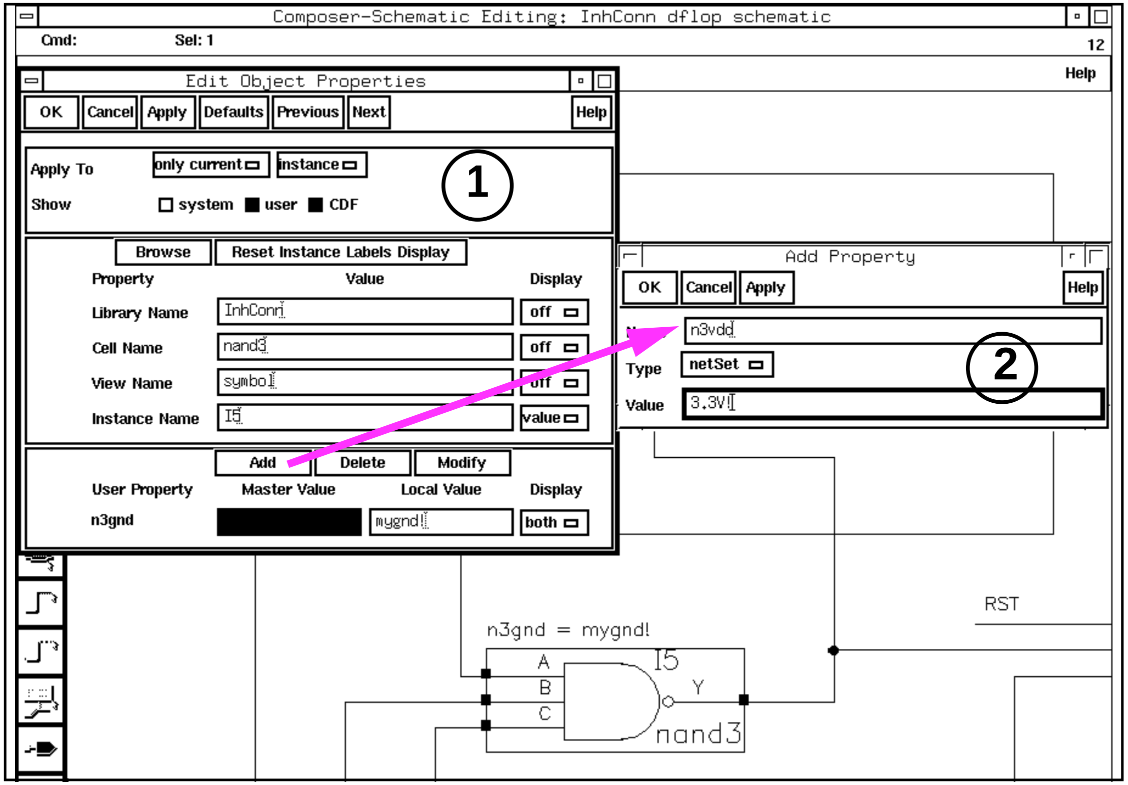
Select the wire.



Setting a Net Expression

The Default Net Name in a net expression must denote a scalar, global name, such as *vdd!* or *mygnd!*.

Override Default with the *netSet* Property



Composer-Schematic Editing: InhConn dflop schematic

Cmd: Sel: 1 12

Edit Object Properties

Apply To: only current, instance

Show: □ system ■ user ■ CDF

Property	Value	Display
Library Name	InhConn	off
Cell Name	nand3	off
View Name	symbol	off
Instance Name	I5	value

User Property	Master Value	Local Value	Display
n3gnd	■	mygnd!	both

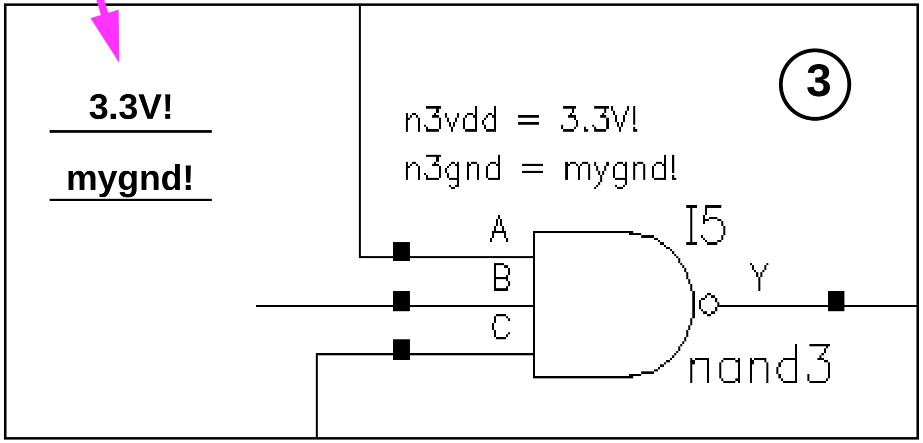
Add Property

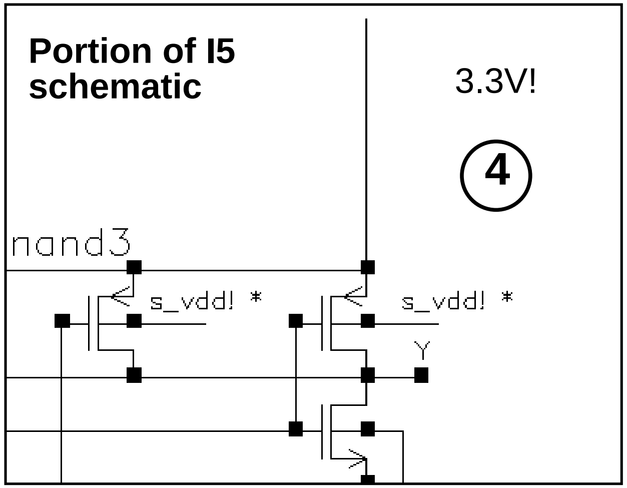
Name: n3vdd

Type: netSet

Value: 3.3V!

All override global names must be assigned at the level of hierarchy where used.





Portion of I5 schematic

Override Default with the *netSet* Property

Override the default Net Expression setting with the *netSet* property. To add the *netSet* property:

1. In the schematic window, click on an instance and select **Edit—Properties—Objects**.

   The Edit Object Properties form appears.

2. Click the **Add** button, and fill in the Add Property form as follows:

   — Set the *Type* to **netSet**.

   — Enter a *Name* and *Value*.

   — **OK** the form.

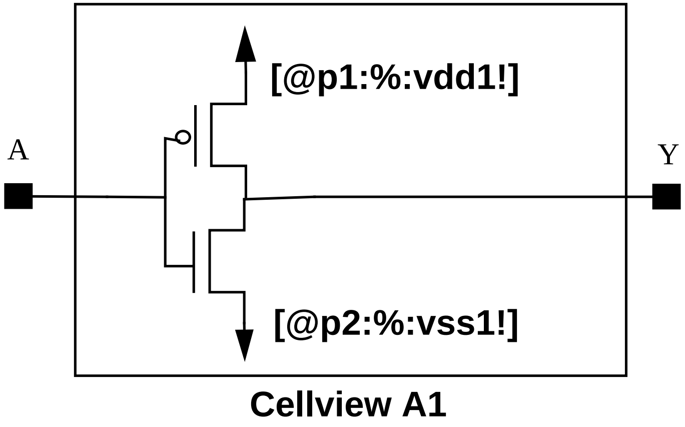
3. Set the visibility of the new property in the Edit Object Properties form to **both**, and **OK** the form.

   When using inherited connections, assign all the override global names at each level of the hierarchy, because the net expression evaluator looks up one level for its connections. In this case, the globals used are *3.3V!* and *mygnd!*.

   The schematic updates and displays the label for your property. **Check and Save** your schematic for the property to go into effect.

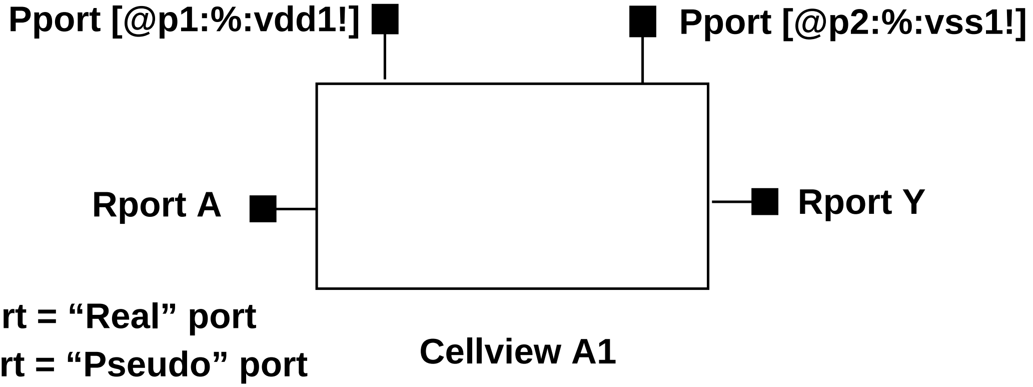
4. Descend into the cell for which you added a *netSet* property. The new value is displayed.

Netlisting with Inherited Connections



Cellview with Inherited Connections

When netlisted, two pseudo ports are created, because it has two inherited connections for the supplies.



Rport = “Real” port

Pport = “Pseudo” port

Netlisting with Inherited Connections

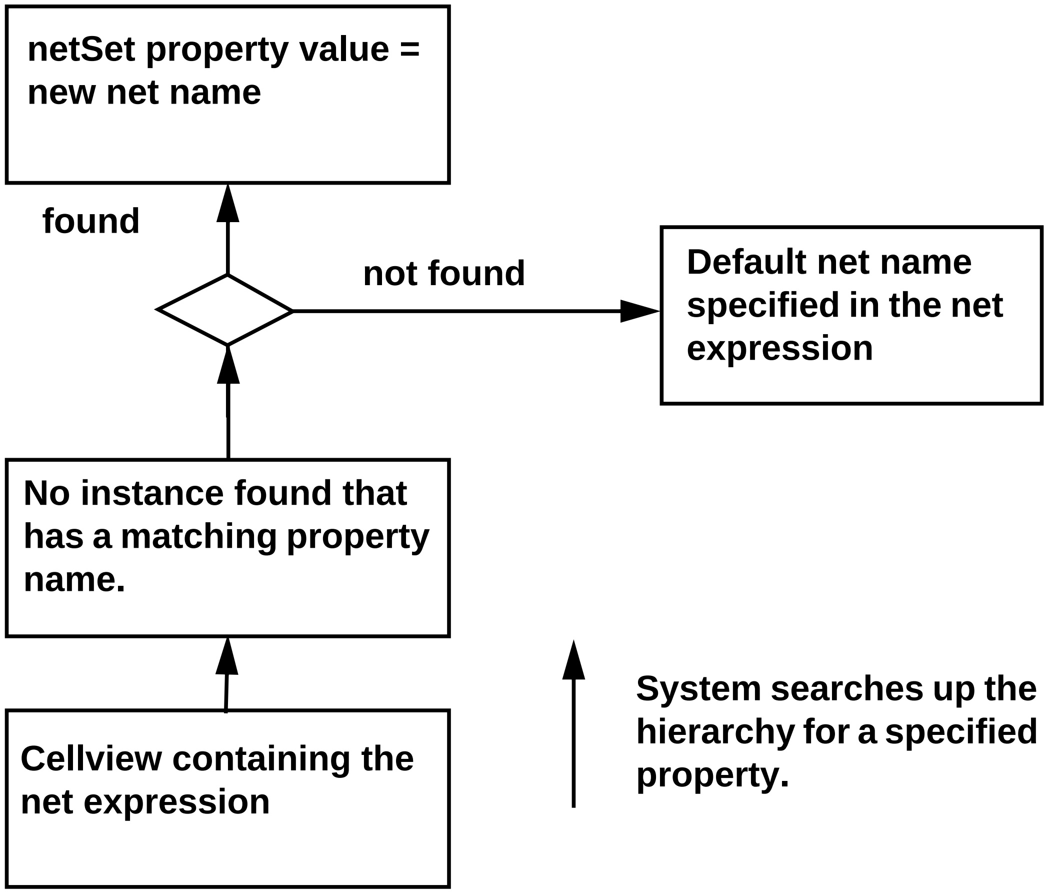
Rport denotes a “real” port, which is a physical pin on the symbol cellview. *Pport* denotes a “pseudo” port that is created by the netlister. The name of the pseudo port is the net expression of the inherited connection.

Below is an example of how the *nand3* gate shown in previous slides appears in the Spectre® direct netlist. In this example, all pseudo-ports begin with the *inh_* prefix.

```
...
// Library name: InhConn
// Cell name: nand3
// View name: cmos_sch
// Inherited view list: schematic symbol cmos_sch verilog behavioral
subckt nand3 A B C Y inh_n3gnd inh_nsub inh_psub inh_n3vdd
    M8 (net37 C inh_n3gnd inh_nsub) nmos w=nw l=nl
    M6 (Y A net34 inh_nsub) nmos w=nw l=nl
    M7 (net34 B net37 inh_nsub) nmos w=nw l=nl
    M3 (Y A inh_n3vdd inh_psub) pmos w=pw l=pl
    M0 (Y C inh_n3vdd inh_psub) pmos w=pw l=pl
    M4 (Y B inh_n3vdd inh_psub) pmos w=pw l=pl
ends nand3
// End of subcircuit definition.
...
I5 (net2 net18 CLK net5 mygnd! inh_nsub inh_psub 6) nand3
```

The I5 statement denotes the instantiation of the *nand3* gate in the netlist.

Evaluating Net Expressions



netSet property value = new net name
found
not found
Default net name specified in the net expression
No instance found that has a matching property name.
Cellview containing the net expression
System searches up the hierarchy for a specified property.

Note: Place the *netSet* property on any instances at any level above the cellviews with net expressions.

Evaluating Net Expressions

Net expressions are evaluated for each hierarchical path. The system uses the property name specified in the net expression to search upward from the cellview containing the net expression (one instance at a time) to the top of the design. The first instance that is found that has a matching property name terminates the search. If the property is of type *netSet* and is a legal net name, then its value is used as the connecting net rather than the default net name specified in the expression.

The *netSet* property can be placed on any instance and it will affect all net expressions with matching property names at all levels below that instance unless overridden by another *netSet* property in a lower-level instance.

Labs

Lab 13-1 Inherited Connections

Lab 13-2 Using Inherited Connections with the ampTest Design

Labs

Module 14: The Hierarchy Editor

Topics in this module

- Applications of the Hierarchy Editor
- Overview
- Creating a configuration
- The Hierarchy Editor window
- Selecting views with the Hierarchy Editor
- The tree view of a Hierarchy
- Opening a configuration
- Synchronizing a configured schematic

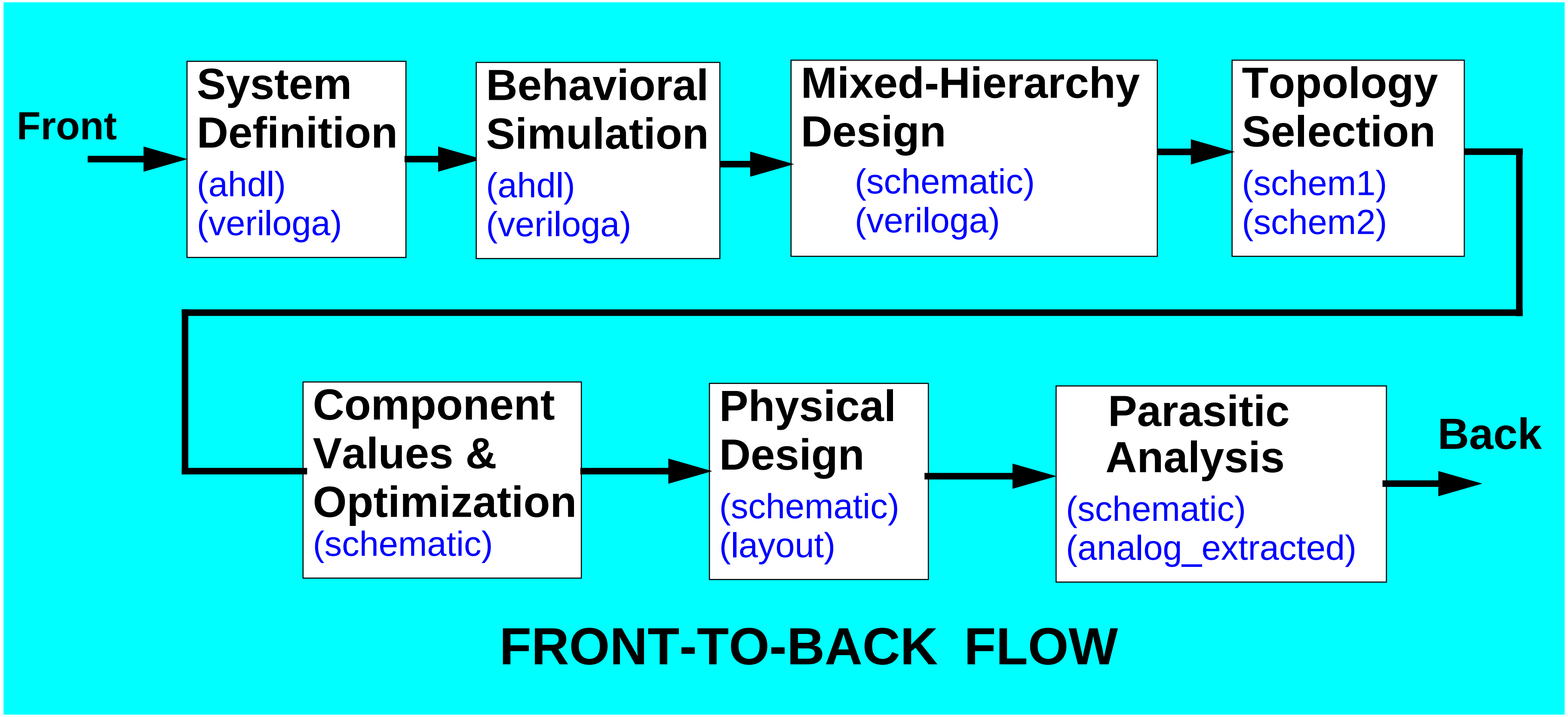
Terms and Definitions

Hierarchy Editor	A Cadence tool for viewing and editing a design hierarchy.
configuration	A specialized cell view for using the Hierarchy Editor.
configured schematic	The schematic of the configuration cell view.
view found	The cell view within hierarchy of the current configuration.
view to use	The cell view to use within the updated configuration.

Applications for the Hierarchy Editor

The **Hierarchy Editor** is used to **select** the **view** to be used **within a system** hierarchy.

In the design of a circuit used within a system, there are numerous phases of the design flow. Each phase of the design has cell views associated to that step. The Hierarchy Editor is a tool that selects the corresponding cell view for display and for netlisting the design. The Hierarchy Editor is used extremely useful in system level design, mixed signal design, and for parasitic analysis.



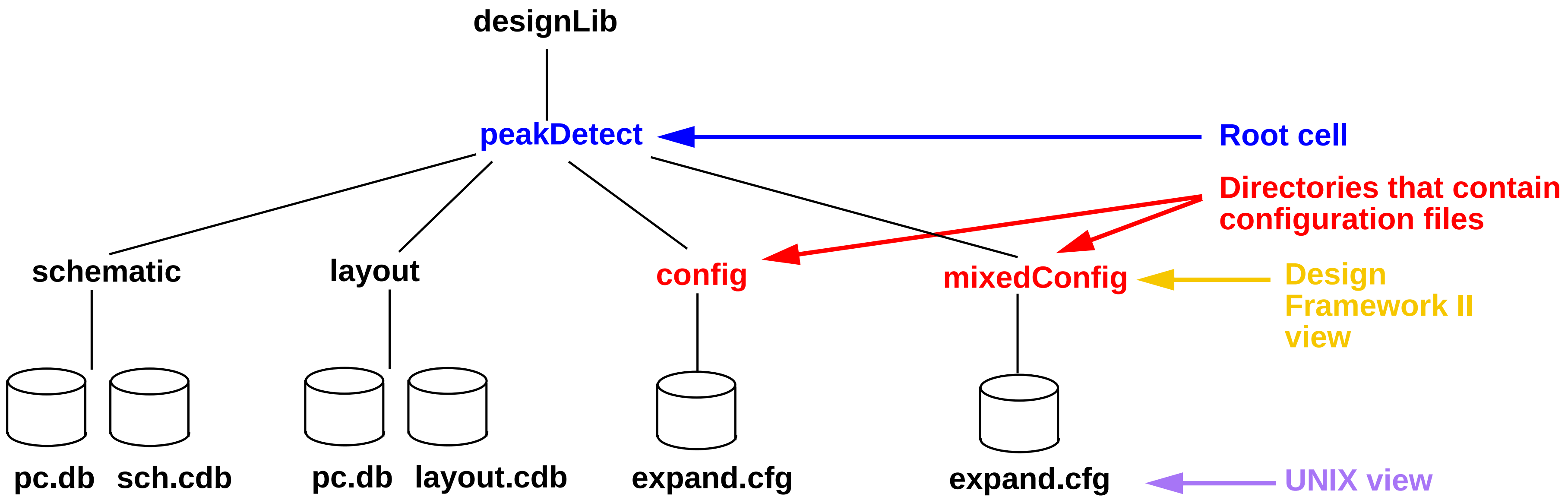
Applications for the Hierarchy Editor

As shown in the diagram above, the design of a structure within the hierarchy takes on numerous views. The Hierarchy Editor is used to select view for netlisting and simulation.

- Initially, the design is **behavioral** to determine the specifications needed for the cell block to work. Here, the view is perhaps *ahdl*, *veriloga*, or *spectre*.
- The design then progresses to **mixed level** or **mixed-hierarchy** design, where the cell view is perhaps a *veriloga* view with more specified behavior, or a *schematic* view using ideal components. The Hierarchy Editor is used to verify proper behavior and performance as the views are selected.
- In **topology selection**, the view is schematic. However, the design sometimes requires some research to determine a feasible circuit topology. As such, numerous schematics with unique view names are used. The Hierarchy Editor is used to select between the available views (*schem1*, *schem2*, etc.). The performance of the various topologies can be compared.
- The circuit *schematic* is then **optimized** using EDFM tools in the Analog Design Environment.
- The circuit is then reduced to a physical design to produce a *layout* view.
- The layout is extracted and the Hierarchy Editor is used to select the *analog_extracted* view for the **parasitic analysis** simulation.

Overview of the Hierarchy Editor

The Hierarchy Editor (HED) is a graphical tool for creating configurations.



Configuration: A set of rules that defines how cellviews in a design are partitioned and netlisted.

Overview of the Hierarchy Editor

Use the Hierarchy Editor to do the following:

- Prepare and organize complex designs for netlisting.
- Browse a design hierarchy by viewing the instance bindings.
- Speed the navigation of a large hierarchical design to open a cellview in the context of a design hierarchy.
- Facilitate copying a hierarchy by specifying a configuration.
- Simplify debugging of a hierarchy (finding, controlling, removing incorrect views).
- Run a simulation on a configuration.

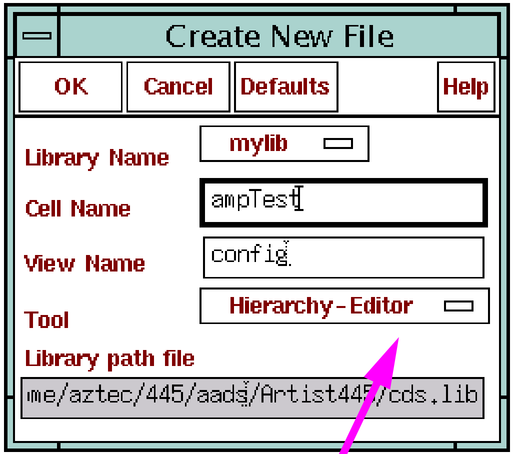
The configuration is contained in a file named *expand.cfg* for each config cellview. A cell can have different configuration files for different purposes, such as mixed-signal simulation netlisting, LVS netlisting, etc.

Run the Hierarchy Editor on a standalone basis or through the Design Framework II environment.

Because the Hierarchy Editor is a separate tool from Design Framework II, you should always save the changes to the configuration files and have your Design Framework II applications read in the information again to access the new information.

Creating a Configuration

1 **File—New—Cellview**



Library Name: mylib

Cell Name: ampTest

View Name: config

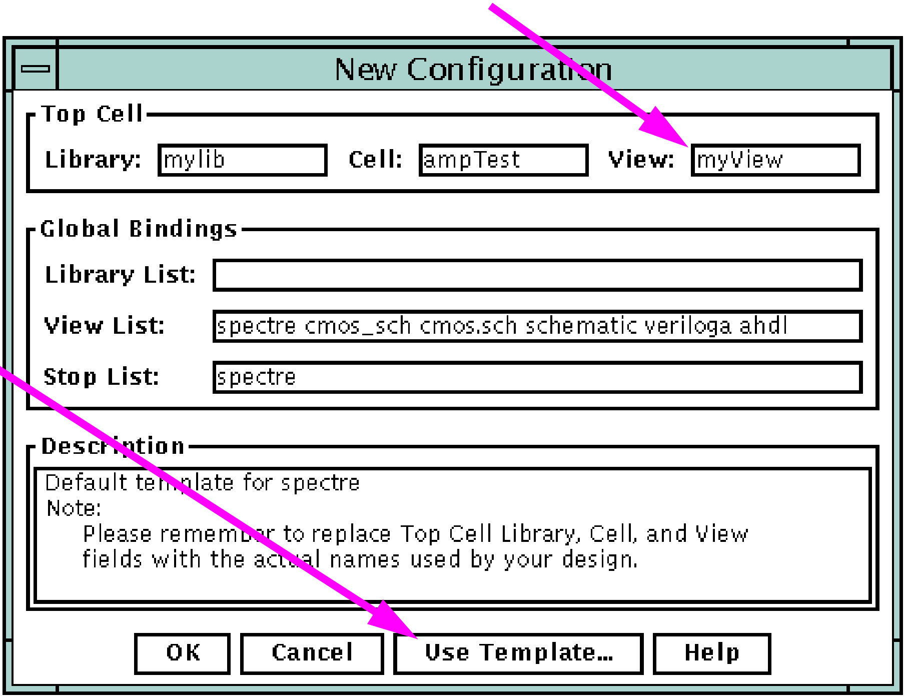
Tool: Hierarchy-Editor

Library path file: me/aztec/445/aads/Artist445/cds.lib

Set to Hierarchy-Editor

2 **Change myView to schematic, OK the form.**

3 **Select template**



Top Cell

Library: mylib Cell: ampTest View: myView

Global Bindings

Library List:

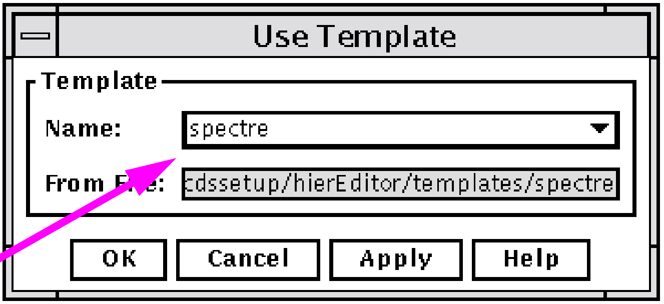
View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Description

Default template for spectre
Note:
Please remember to replace Top Cell Library, Cell, and View fields with the actual names used by your design.

4 **Select simulator name for template desired.**



Template

Name: spectre

From File: cdssetup/hierEditor/templates/spectre

Select Spectre, then OK in form

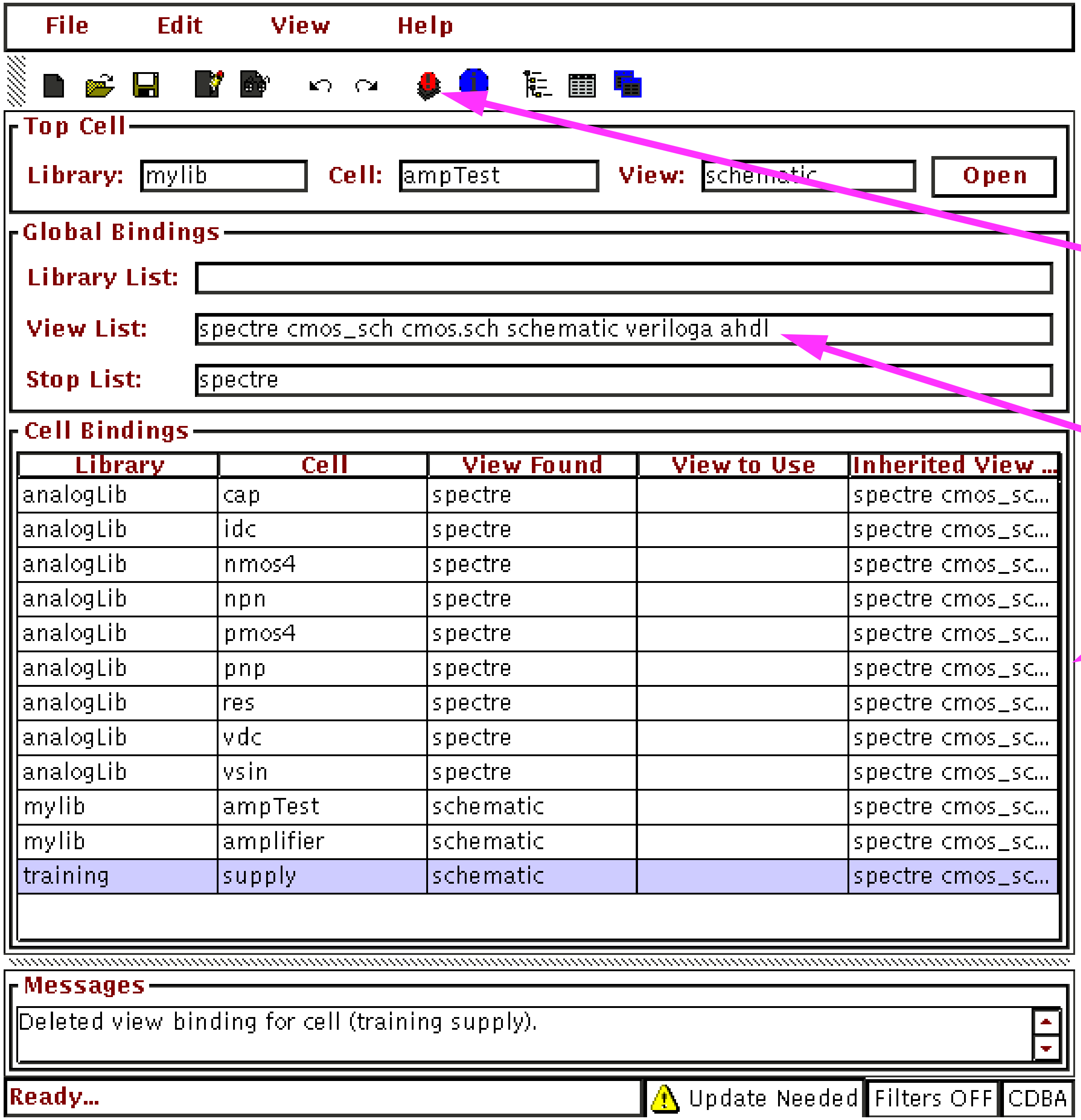
Creating a Configuration

Create a configuration file with the **File—New—Cellview** command. Set the Tool field to **Hierarchy-Editor** in the Create New File form.

The New Hierarchy form appears in front of the Hierarchy Editor window. Set the Template Name field to the simulator that you are using.

More information about the Hierarchy Editor and Configuration files can be found in CDSDoc.

The Hierarchy Editor Window



Top Cell

Library: mylib Cell: ampTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Cell Bindings

Library	Cell	View Found	View to Use	Inherited View ...
analogLib	cap	spectre		spectre cmos_sc...
analogLib	idc	spectre		spectre cmos_sc...
analogLib	nmos4	spectre		spectre cmos_sc...
analogLib	npn	spectre		spectre cmos_sc...
analogLib	pmos4	spectre		spectre cmos_sc...
analogLib	pnp	spectre		spectre cmos_sc...
analogLib	res	spectre		spectre cmos_sc...
analogLib	vdc	spectre		spectre cmos_sc...
analogLib	vsin	spectre		spectre cmos_sc...
mylib	ampTest	schematic		spectre cmos_sc...
mylib	amplifier	schematic		spectre cmos_sc...
training	supply	schematic		spectre cmos_sc...

Messages

Deleted view binding for cell (training supply).

Ready...

When this icon is RED, select it to incorporate the selections.

View Lists can be replaced with a constant, such as $analog.

The Hierarchy Editor Window

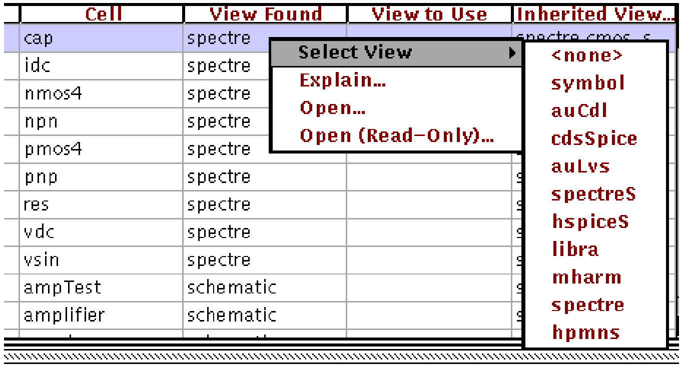
The table format is the default display structure for the HED. A tree structure is also available, and both are helpful in setting up the views for simulation.

Define what views to include in the hierarchy at three different levels:

1. The global level, using a global view list and stop list.
2. The cell level, using cell bindings or Inherited View Lists, which affect the cell as well as structures below the cell in the hierarchy.
3. The instance level, using instance bindings or instance based Inherited View Lists, which affect the instance as well as structures below it in the hierarchy.

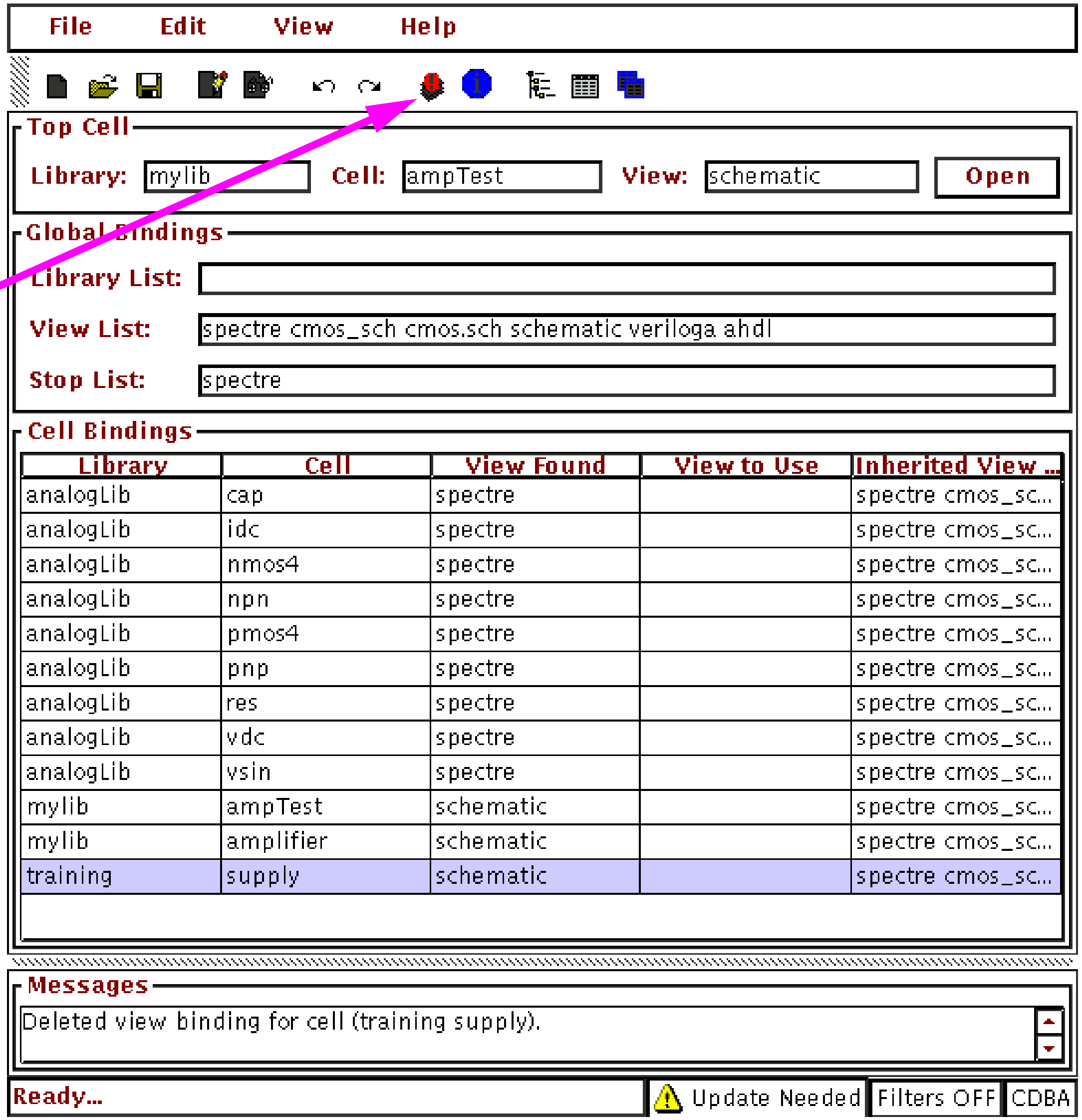
Import the View List, Stop List, and all other Cell Bindings and Instance Bindings from another configuration to the current one that is open.

Selecting Views with the HED



Cell	View Found	View to Use	Inherited View...
cap	spectre		spectre cmos_s...
idc	spectre		
nmos4	spectre		
npn	spectre		
pmos4	spectre		
pnp	spectre		s
res	spectre		s
vdc	spectre		s
vsin	spectre		s
ampTest	schematic		s
amplifier	schematic		s

The banner (not shown) prompts to save the new configuration.



Top Cell

Library: mylib Cell: ampTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Cell Bindings

Library	Cell	View Found	View to Use	Inherited View ...
analogLib	cap	spectre		spectre cmos_sc...
analogLib	idc	spectre		spectre cmos_sc...
analogLib	nmos4	spectre		spectre cmos_sc...
analogLib	npn	spectre		spectre cmos_sc...
analogLib	pmos4	spectre		spectre cmos_sc...
analogLib	pnp	spectre		spectre cmos_sc...
analogLib	res	spectre		spectre cmos_sc...
analogLib	vdc	spectre		spectre cmos_sc...
analogLib	vsin	spectre		spectre cmos_sc...
mylib	ampTest	schematic		spectre cmos_sc...
mylib	amplifier	schematic		spectre cmos_sc...
training	supply	schematic		spectre cmos_sc...

Messages

Deleted view binding for cell (training supply).

Ready... | ⚠ Update Needed | Filters OFF | CDBA

Click the Update icon to put changes into effect.

The new Cell Binding

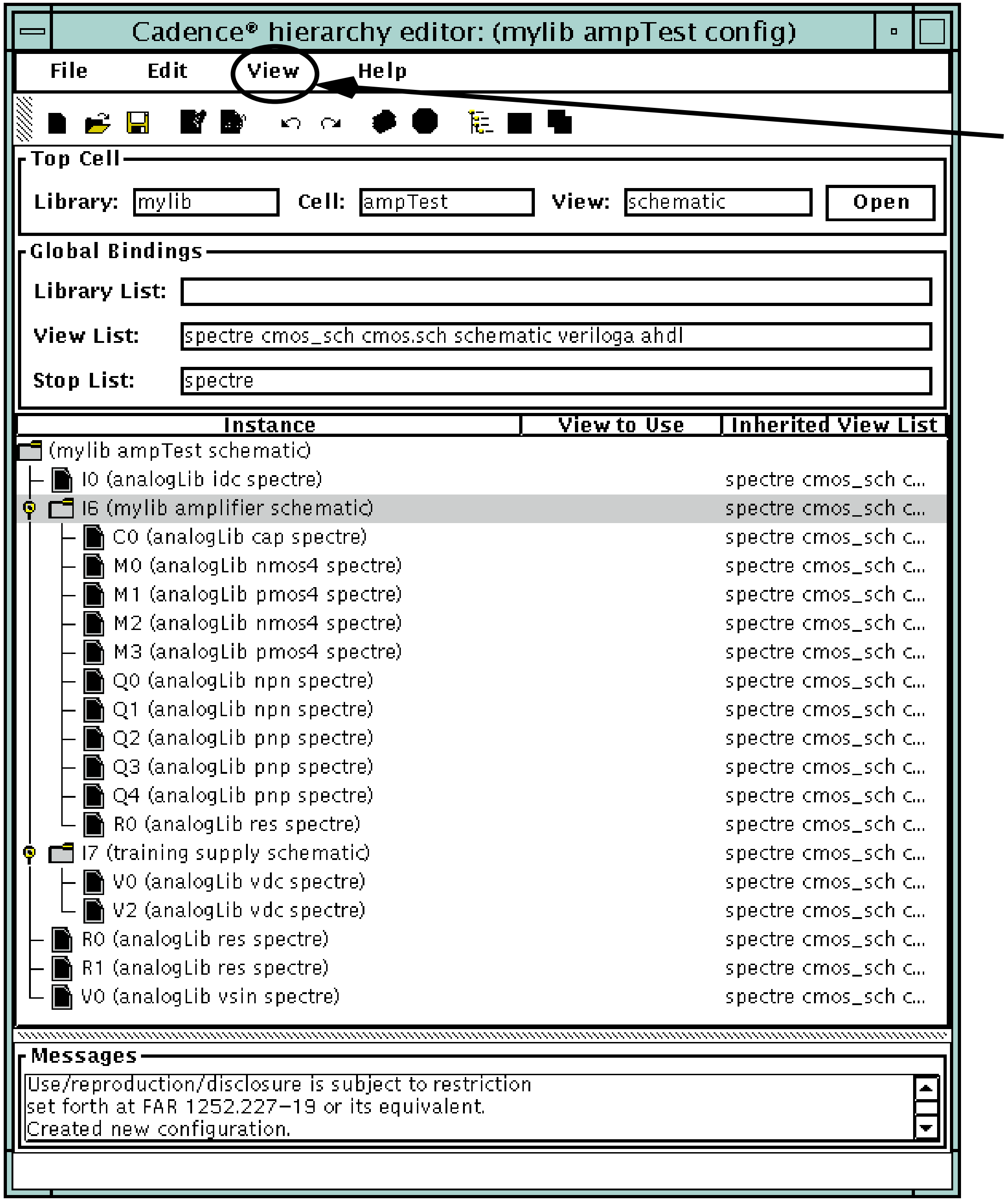
Selecting Views with the HED

Select the desired view for simulation for each cell in the design.

When a change is made to the configuration file in the HED, the *Save Needed* message appears at the top of the configured schematic window. Select **File—Save** to save your changes, or respond to the dialog box that appears after you click the **Update** button.

In addition, when a change is made to the configuration, the Update button will highlight in red in the HED. Click the **Update** button to put your changes to the configuration into effect.

HED Tree Format



Top Cell

Library: mylib   Cell: ampTest   View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Instance	View to Use	Inherited View List
(mylib ampTest schematic)		
I0 (analogLib idc spectre)		spectre cmos_sch c...
I6 (mylib amplifier schematic)		spectre cmos_sch c...
C0 (analogLib cap spectre)		spectre cmos_sch c...
M0 (analogLib nmos4 spectre)		spectre cmos_sch c...
M1 (analogLib pmos4 spectre)		spectre cmos_sch c...
M2 (analogLib nmos4 spectre)		spectre cmos_sch c...
M3 (analogLib pmos4 spectre)		spectre cmos_sch c...
Q0 (analogLib npn spectre)		spectre cmos_sch c...
Q1 (analogLib npn spectre)		spectre cmos_sch c...
Q2 (analogLib pnp spectre)		spectre cmos_sch c...
Q3 (analogLib pnp spectre)		spectre cmos_sch c...
Q4 (analogLib pnp spectre)		spectre cmos_sch c...
R0 (analogLib res spectre)		spectre cmos_sch c...
I7 (training supply schematic)		spectre cmos_sch c...
V0 (analogLib vdc spectre)		spectre cmos_sch c...
V2 (analogLib vdc spectre)		spectre cmos_sch c...
R0 (analogLib res spectre)		spectre cmos_sch c...
R1 (analogLib res spectre)		spectre cmos_sch c...
V0 (analogLib vsin spectre)		spectre cmos_sch c...

Messages

```
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Created new configuration.
```

View tree from here.

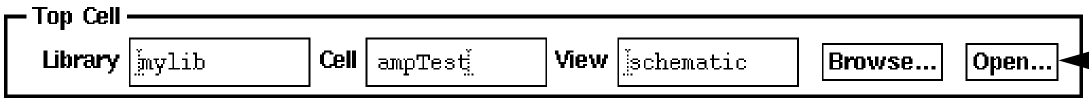
HED Tree Format

Click the Show Tree icon or select **View—Tree** to display the Tree format.

The Tree format shows the position of the cells and cellviews in the hierarchy. Cells and instances subject to inherited view lists appear in dark blue print. Cells and instances subject to cell and instance bindings appear in light blue print. Cells and instances without chosen views appear in red print.

Use the tree format to display the current configuration. Also, use the tree format to change cell/instance bindings, and to change inherited view lists.

Opening a Configured Schematic



Top Cell

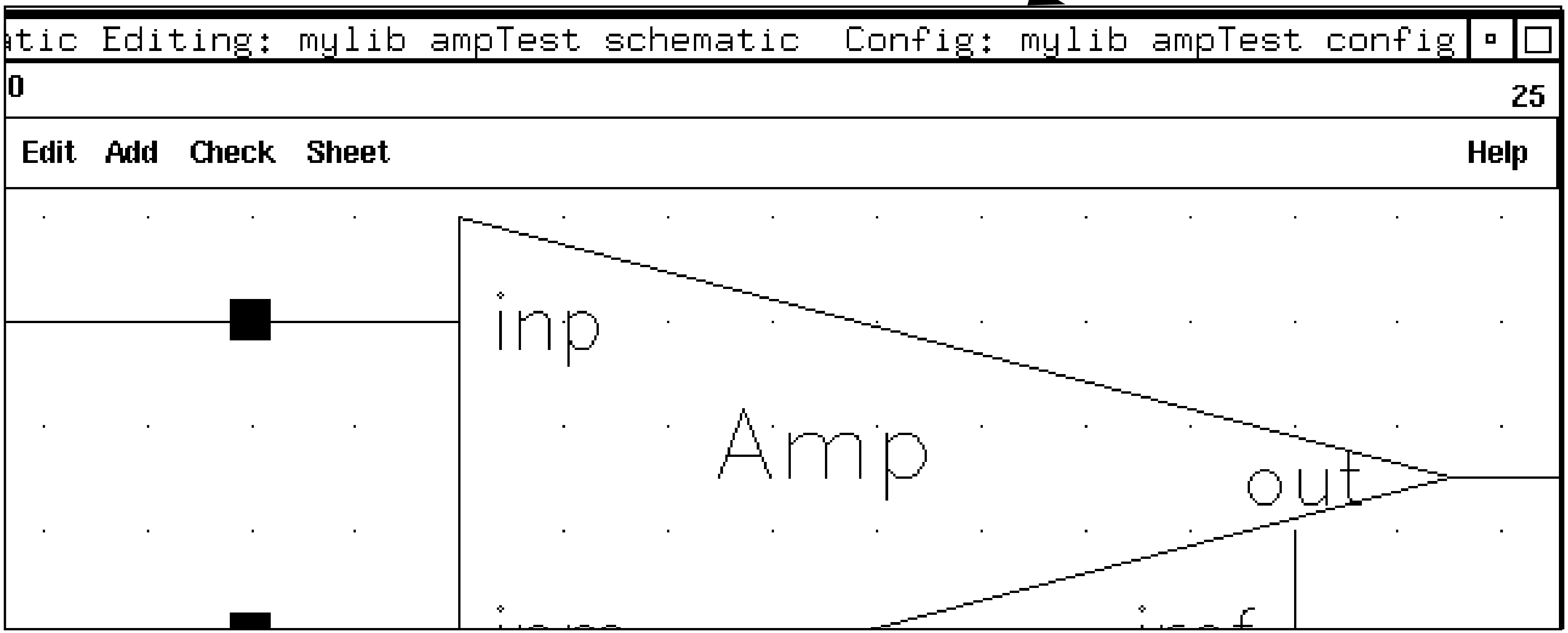
Library: mylib

Cell: ampTest

View: schematic

Click Open

New Information in Schematic Title Bar



atic Editing: mylib ampTest schematic Config: mylib ampTest config

Open a Configured Schematic:

- From the **Open** button in the Hierarchy Editor
- Using the **File—Open** command in the CIW as you open the *config* view.
- Using the Library Manager when you open the *config* view.

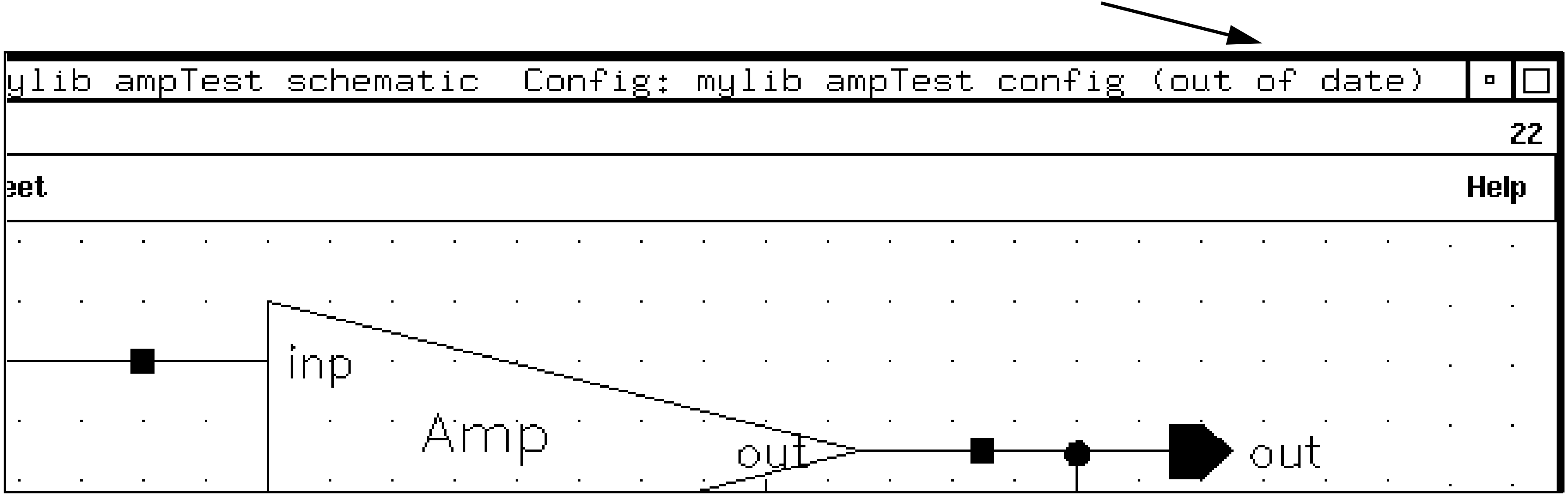
Opening a Configured Schematic

All applications that run based on configurations must be executed from a schematic window that was opened from the config file (referred as the configured schematic). This applies to Showing Views Found, Netlisting, and Simulation.

It is necessary to start the Analog Design Environment from the configured schematic to use your configuration to set the partition and cell bindings for simulation.

Synchronizing the Configured Schematic

The configured schematic might not match the current configuration file.



ylib ampTest schematic Config: mylib ampTest config (out of date)

To synchronize the configuration schematic to the configured file:

- Click the Update button after modifying the configuration.
- Click OK in the Cellviews Need Saving form that appears.

Synchronizing the Configured Schematic

The Hierarchy Editor is a separate tool from the Design Framework II environment. If changes are made in the HED, the configured schematic that was opened prior to the modifications will contain out of date information. The schematic window banner will have the “(out of date)” label.

To synchronize the configured schematic to the updated configuration file, click the **Update** button and then click **OK** in the Cellviews Need Saving form that appears.

Summary

In this module we discussed:

- Applications for using the Hierarchy Editor
- Overview
- How to create a configuration
- Selecting views in the Hierarchy Editor and saving the new configuration
- Opening the configuration to view the hierarchy
- Running a simulation on the configured schematic

Summary

Labs

Lab 14-1 Creating a Configuration File with the Hierarchy Editor

Lab 14-2 Running a Simulation with Subcircuits

Lab 14-3 Rerunning Simulation with the Schematic View

Labs

Module 15: Overview of Parasitic Simulation

Topics in this module

- Background of parasitic simulation
- Brief overview
- Design flow
- Supported layout software
- Diva and Assura
- What happens in parasitic simulation
- An actual parasitic simulation flow
- Summary

Terms and Definitions

CDF	*Component Description Format* is information stored on a cell that defines parameters, simulation rules, and display information.
Selected Parasitic Simulation	Selected parasitic simulation limits the nets that are simulated with parasitic information.
Source/Loads	Components in the schematic that are for simulation purposes only.
LVS	Layout versus schematic checking is used to verify that the layout matches the schematic.
Test Fixture Schematic	A schematic containing a symbol representing the design schematic with all required simulation sources and loads attached to I/O pins of the symbol.
“LVS-able” block	A subblock with a layout and a schematic in the same cell that can be verified with LVS.

Background

In previous chapters, a large amount of time and effort is applied to designing a circuit that realizes specific design goals. During the design phase of the circuit, there was no allowance for the parasitic capacitance and for the resistance of interconnecting the components of the schematic. Such “interconnect parasitics” typically cause a 10 to 20 percent increase in the delay times of a circuit and a corresponding reduction in bandwidth.

To ensure the product meets the specific design goals, the following options are available:

- “Over design” or “guard band” the design by using larger devices and increasing power consumption.
- Carefully measure and analyze the interconnect parasitics after layout has been completed and resimulate the design.
- Run a parasitic extraction simulation on the layout and resimulate the design.

The first and second options have disadvantages. The first consumes more power and is at risk for not meeting the design goals. The second option is work intensive, and is also at risk for errors to occur.

Parasitic extraction requires less time, effort, and is more reliable.

Background

Most semiconductor devices must be interconnected using conductive materials such as metal, polysilicon, salicided polysilicon, and so on. The conductive interconnect has some resistance. In addition, the conductive material often must traverse over a substrate, or power bus. As such, capacitance exists to a power supply potential. In addition, the interconnect sometimes must cross over other signals, which creates a “point to point” capacitance. As such, a circuit design that has been simulated prior to layout must be significantly altered after layout. Such conditions increase delay and reduce bandwidth.

Bandwidth can be increased by increasing the transconductance of the devices in the circuit. However, this requires additional power. As such, using a “guardband” in the design comes at a cost of larger devices, increased power consumption, and risk.

The layout of the circuit can be evaluated by carefully measuring the area and the effective number of squares of the interconnect. The measurements can then be converted into parasitic resistors and capacitors elements. These elements can then be added back into the schematic and the circuit simulated again. This methodology requires both time and effort. In addition, this methodology has a high risk of error for large circuits.

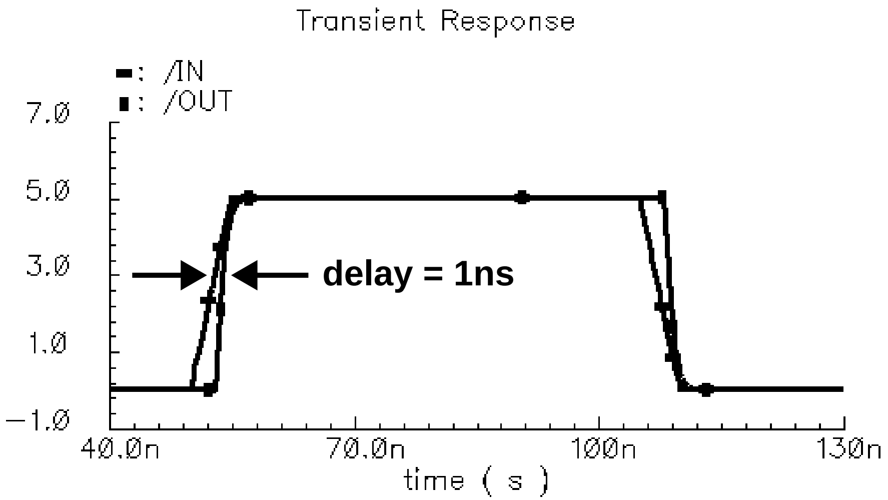
The layout parasitic extraction simulation with extracted circuit simulation is faster, often more accurate, and generally a more reliable method to verify the physical design.

Overview of Parasitic Analysis

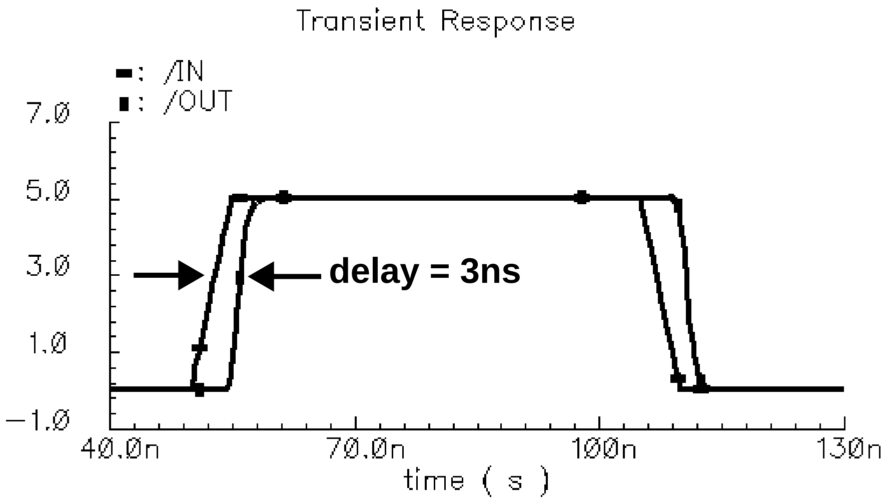
Why Is Parasitic Extraction and Simulation Needed?

Original Circuit Design Submitted to Layout

Fabricated Circuit, Fails to Operate at Specified Frequency!



Simulation without parasitics



Simulation with parasitics

Overview of Parasitic Analysis

A circuit design and simulation often includes the device parasitics such as the junction area and sidewall capacitance. To obtain high-frequency performance in a circuit, you simulate the circuit over corners, and use Monte Carlo Analysis and optimization. However, the circuit was designed without knowing the interconnect parasitics. Since the circuit must be captured in layout, the parasitics are unknown until the layout is completed.

If the circuit is fabricated without simulating with interconnect parasitics, then the circuit may not operate as expected from the design simulations.

In the simulations shown above, a circuit was designed to have a delay less than 1ns. However, the fabricated device failed to operated at the specified maximum frequency. When the fabricated device was simulated using parasitics, the circuit had a significant increase in delay.

At high frequency, the interconnect parasitics become **more dominate** in the overall delay of the circuit.

Supported Layout Software

The Cadence Analog Design Environment 5.0 supports parasitic extraction and simulation for:

- **Diva,** which is documented in the Appendix
- **Assura,** which is documented in Module 17.

Background

Prior to IC 4.4.6, only Diva layout extraction and parasitic simulation was supported.

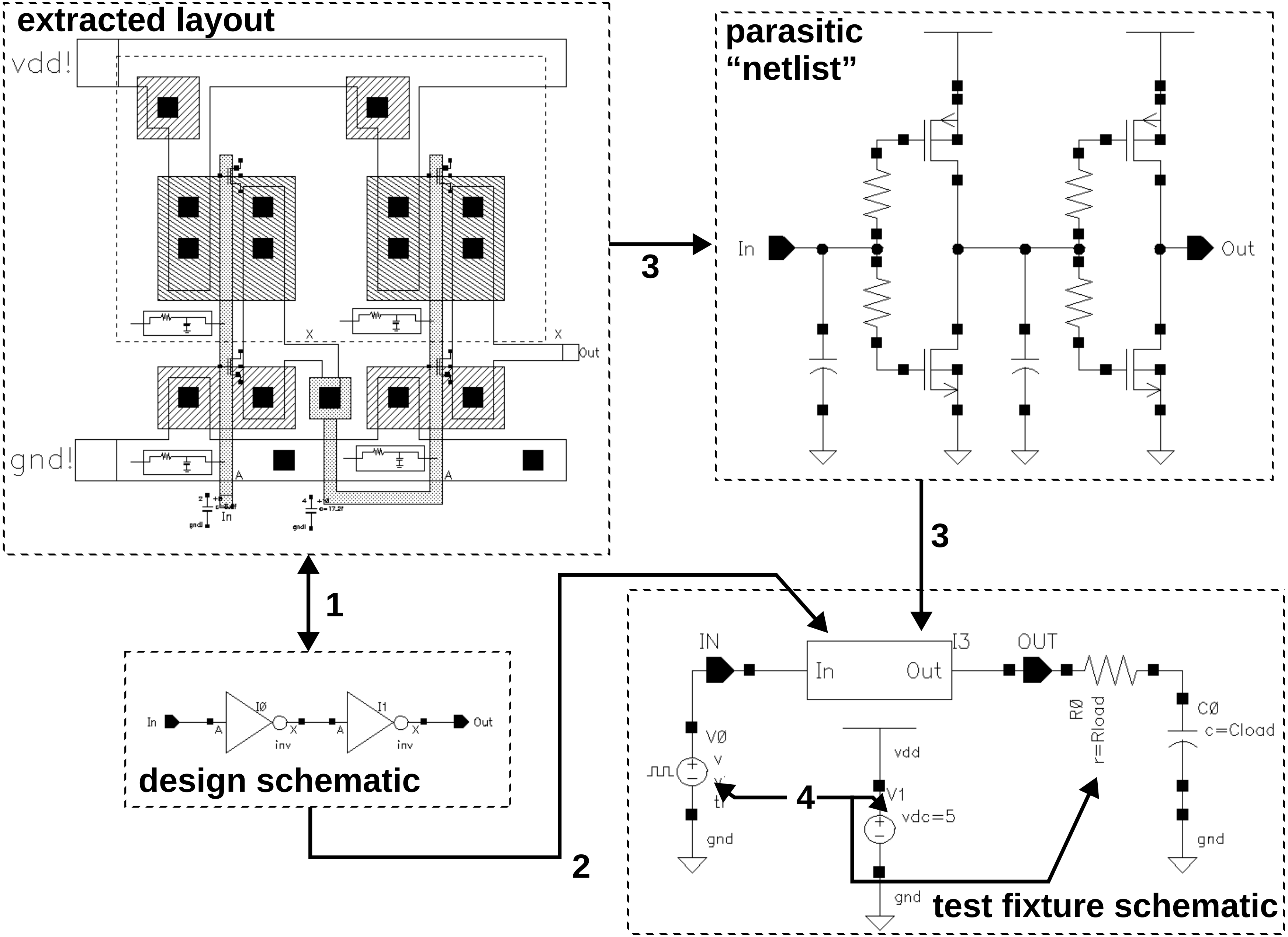
- Diva is now associated for the layout of small circuits.
- Assura is now replacing Diva in more and more design flows for large circuits.
- There is a genuine need for parasitic extraction and simulation tool.

Supported Layout Software

Parasitic extraction and simulation are supported in the Analog Design Environment using both Diva and Assura.

- The Diva flow is covered in Module A of this class
- The Assura flow is covered in Module 16 of this class.
- Only one of the two flows (Diva or Assura) shall be presented in the lecture.
- Only one of the two flows (Diva or Assura) shall be performed in the lab.
- The instructor will inform you which flow is used for the lecture and lab.

What Happens in Parasitic Simulation?



extracted layout

parasitic “netlist”

design schematic

test fixture schematic

1

2

3

3

4

What Happens in Parasitic Simulation?

The numbers in the illustration above correspond to the following:

1. Run LVS to make the comparison between the schematic and extracted representations of the design, verifying that the two are logically equivalent. As a result, the system maps equivalent device terminals (pins) in the extracted and schematic views.

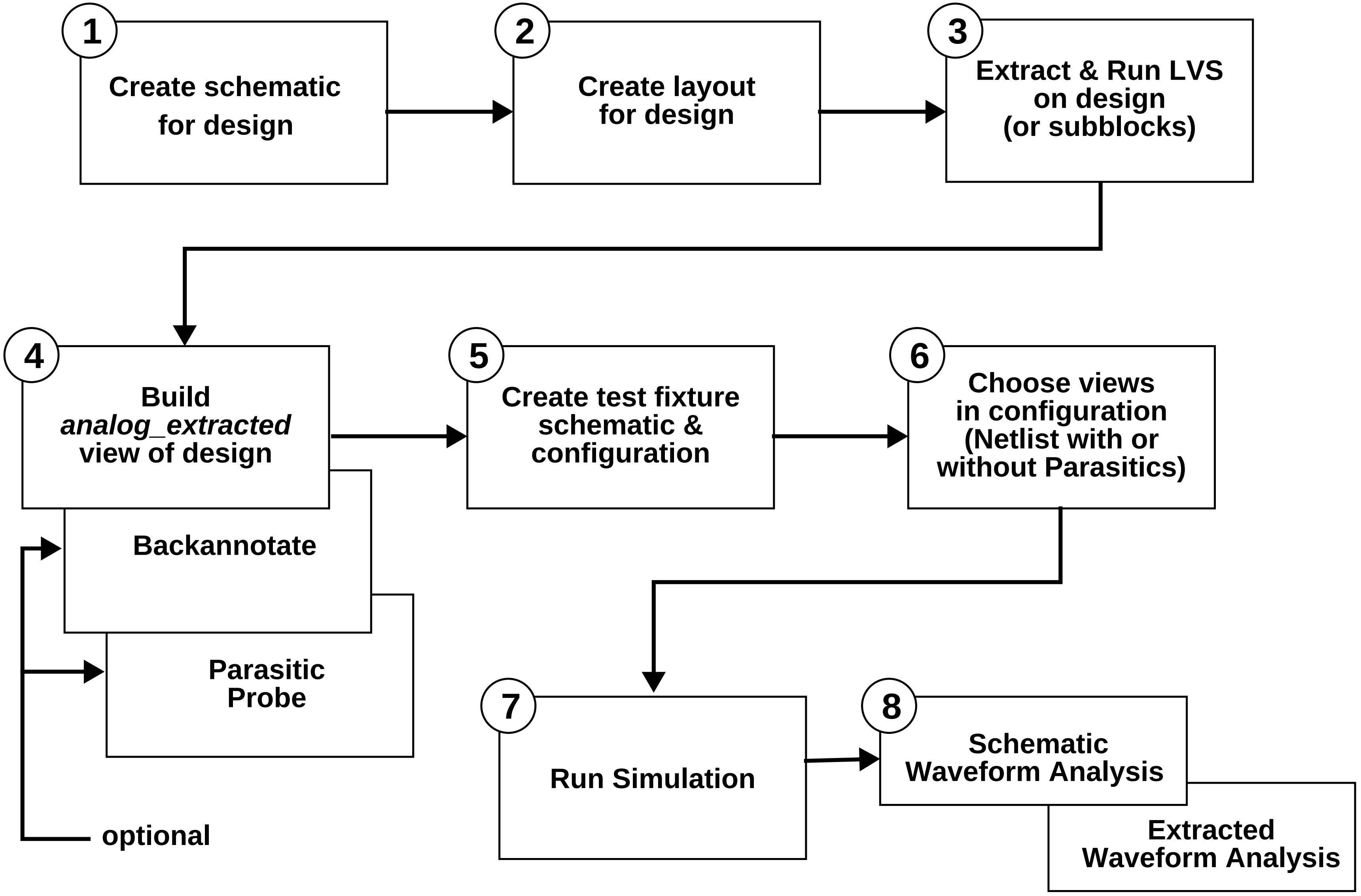
2. Create a simulation test fixture schematic that includes a symbol of the design.

   When running the simulation, the netlist used is actually combined from two sources: a circuit with stimulus and loads, and the parasitic devices.

3. The circuit and parasitic devices that are selected in the schematic come from the extracted netlist.

4. Sources and other components (loads) are used as defined in the top-level test fixture schematic.

An Actual Parasitic Simulation Flow


1
Create schematic for design
2
Create layout for design
3
Extract & Run LVS on design (or subblocks)
4
Build *analog_extracted* view of design
Backannotate
Parasitic Probe
optional
5
Create test fixture schematic & configuration
6
Choose views in configuration (Netlist with or without Parasitics)
7
Run Simulation
8
Schematic Waveform Analysis
Extracted Waveform Analysis

An Actual Parasitic Simulation Flow

1. Start with the schematic for the design. This can be a flat schematic or a hierarchical schematic. The design must be able to run LVS on any portion of the design that will be simulated with parasitics.
2. Create the layout for the design. For any subblock of the design for which the parasitic simulation is run, the hierarchy of the layout must correspond to the hierarchy of the schematic.
3. Extract and run LVS on the portion of the design to simulate with parasitics.
4. Build an *analog_extracted* view of the parasitic portion of the design. At this point additional parasitic information, using **Backannotation** or **Parasitic Probing**, is available for simulation.
5. Create a test fixture that includes a symbol for your design and the appropriate sources and loads to run simulation. Create a *config* view of the design for use with the Hierarchy Editor.
6. Using the Hierarchy Editor, choose which views will be netlisted for simulation. Simulate with parasitics by choosing *analog_extracted* for the design or subblocks of the design. For parasitic simulation, the view must be *analog_extracted*.
7. Run Simulation. The netlist will use parasitic information as specified in your configuration.
8. Any net of the layout or schematic can be selected and then graphically plotted for waveform analysis.

Summary

Topics in this module

- Background of parasitic simulation
- Brief overview
- Design flow
- Supported layout software
- Diva and Assura
- What happens in parasitic simulation
- An actual parasitic simulation flow

Summary

This module presented an overview of parasitic simulation and its value. Both the Diva and Assura layout tools are supported in ADE 5.0. The parasitic flow was presented.

Labs

Lab 15-1 Simulating a Schematic Without Parasitics

Labs

Module 16: Assura Parasitic Simulation Flow

Topics in this module

- Overview and background to Assura parasitic simulation
- Introduction to Assura
- Schematic requirements
- LVS requirements and results
- Parasitic simulation messages and options
- Selective parasitic simulation

Terms and Definitions

CDF	*Component Description Format* is information stored on a cell that defines parameters, simulation rules, and display information.
Selected Parasitic Simulation	Selected parasitic simulation limits the nets that are simulated with parasitic information.
Source/Loads	Components in the schematic that are for simulation purposes only.
LVS	Layout versus schematic checking is used to verify that the layout matches the schematic.
Test Fixture Schematic	A schematic containing a symbol representing the design schematic with all required simulation sources and loads attached to I/O pins of the symbol.
“LVS-able” block	A subblock with a layout and a schematic in the same cell that can be verified with LVS.
MSPS	An Assura user interface for mixed-signal parasitics simulation

Assura Integration into ADE

What is the Assura Integration?

- Cadence’s next-generation physical verification tool
- Very similar in concept to Diva
- Documentation: Chapter 4 of the *Cadence Parasitic Simulation User Guide*

Why is it needed?

- Allows extraction of designs that are too large for Diva to extract in a reasonable amount of time
- Will eventually support AMS Designer and be the primary extraction tool

Assura Integration in ADE

Customers are creating circuit designs that are too large for Diva to extract in a reasonable time. Assura Physical Verification is a next-generation connectivity extraction tool capable of handling the largest customer circuit designs. Assura has a different backend interface than Diva, so the MSPS flow must have an additional interface for Assura added to it.

AMS Designer is Cadence’s next-generation mixed-signal circuit design environment. By supporting AMS Designer, the MSPS flow will give our customers access to our latest toolset, which will help them analyze the large circuits which Assura is capable of extracting.

Assura RCX integration

Integrating Assura into the MSPS flow promises improved capacity, performance, and accuracy over Diva. Assura RCX allows the recognition and extraction of parasitic resistance and capacitance for the largest circuit designs with unprecedented speed and accuracy. The integration of Assura Physical Verification into this flow removes one more obstacle from the path of designers wanting to use MSPS for large circuit designs.

Assura Flow in Analog Design Environment

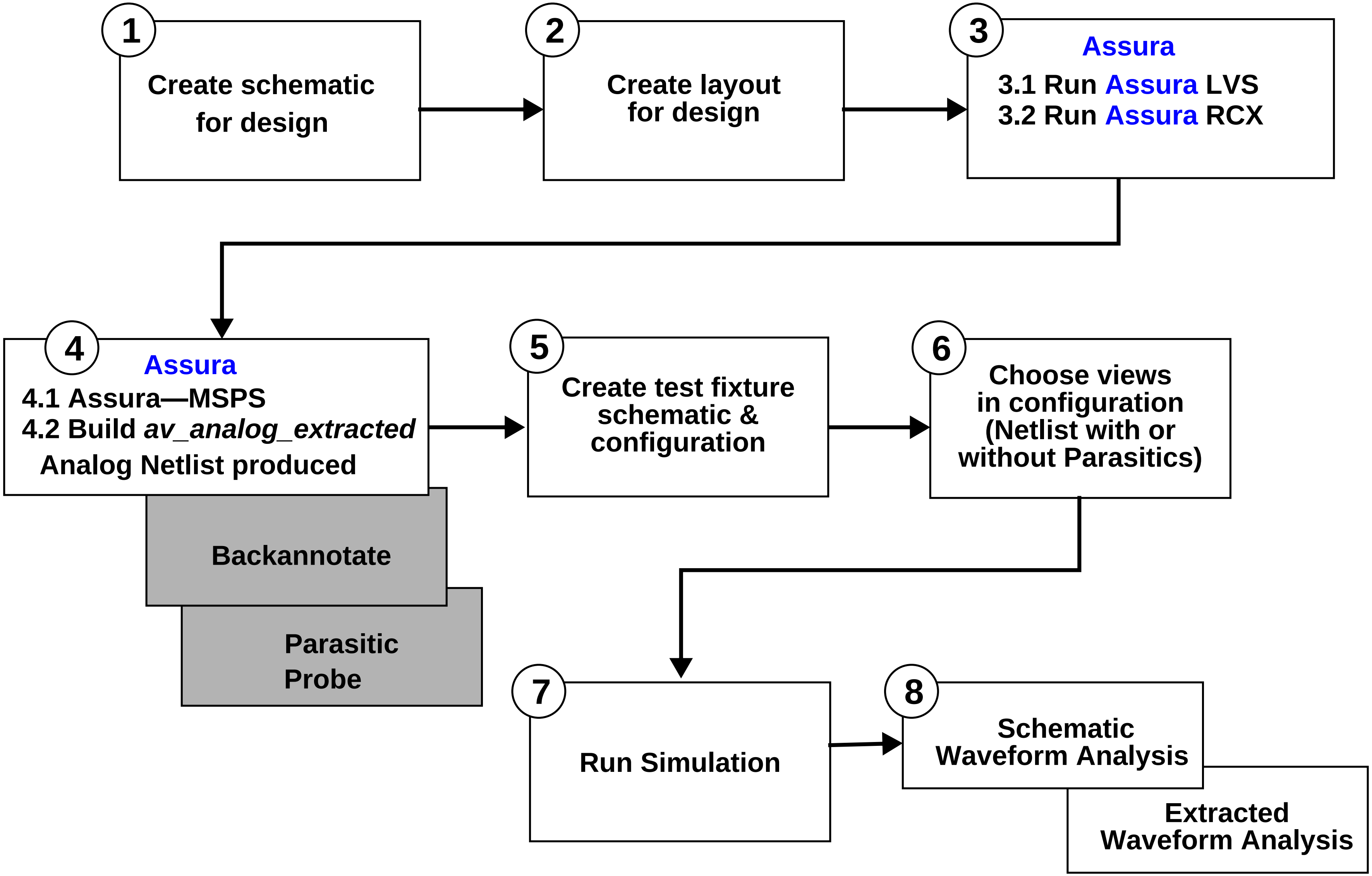
The MSPS flow using Assura is very similar to the current Diva-driven MSPS flow, except that, instead of Diva, Assura is used to perform layout connectivity, parasitic extraction, and extracted view creation.

- Once the extracted view has been created, you can access the Mixed Signal Parasitic Simulation menu.
- This menu allows you to specify the library, cell, and view for both the schematic and extracted views. Once these fields are set, you can perform the same functions found in the Diva flow: Backannotate, Parasitic Probe, Build Analog, and Build Mixed.
- The functionality is the same as the current Diva-based flow; however, the underlying callbacks have been changed to operate on the Assura-generated extracted view.
- In addition, the av_analog_extracted and mixed_extracted views resulting from the Build Analog and Build Mixed choices are different from the views generated by these same choices in the current MSPS flow using Diva.

Aside from the existence of this new property, the primary difference is that the Assura-generated extracted view will use mapped algorithmically hierarchical schematic names for instances and nets.

Assura Flow in Analog Design Environment

Assura Design Flow



1 Create schematic for design

2 Create layout for design

3 Assura
3.1 Run Assura LVS
3.2 Run Assura RCX

4 Assura
4.1 Assura—MSPS
4.2 Build *av_analog_extracted*
Analog Netlist produced

Backannotate

Parasitic Probe

5 Create test fixture schematic & configuration

6 Choose views in configuration (Netlist with or without Parasitics)

7 Run Simulation

8 Schematic Waveform Analysis

Extracted Waveform Analysis

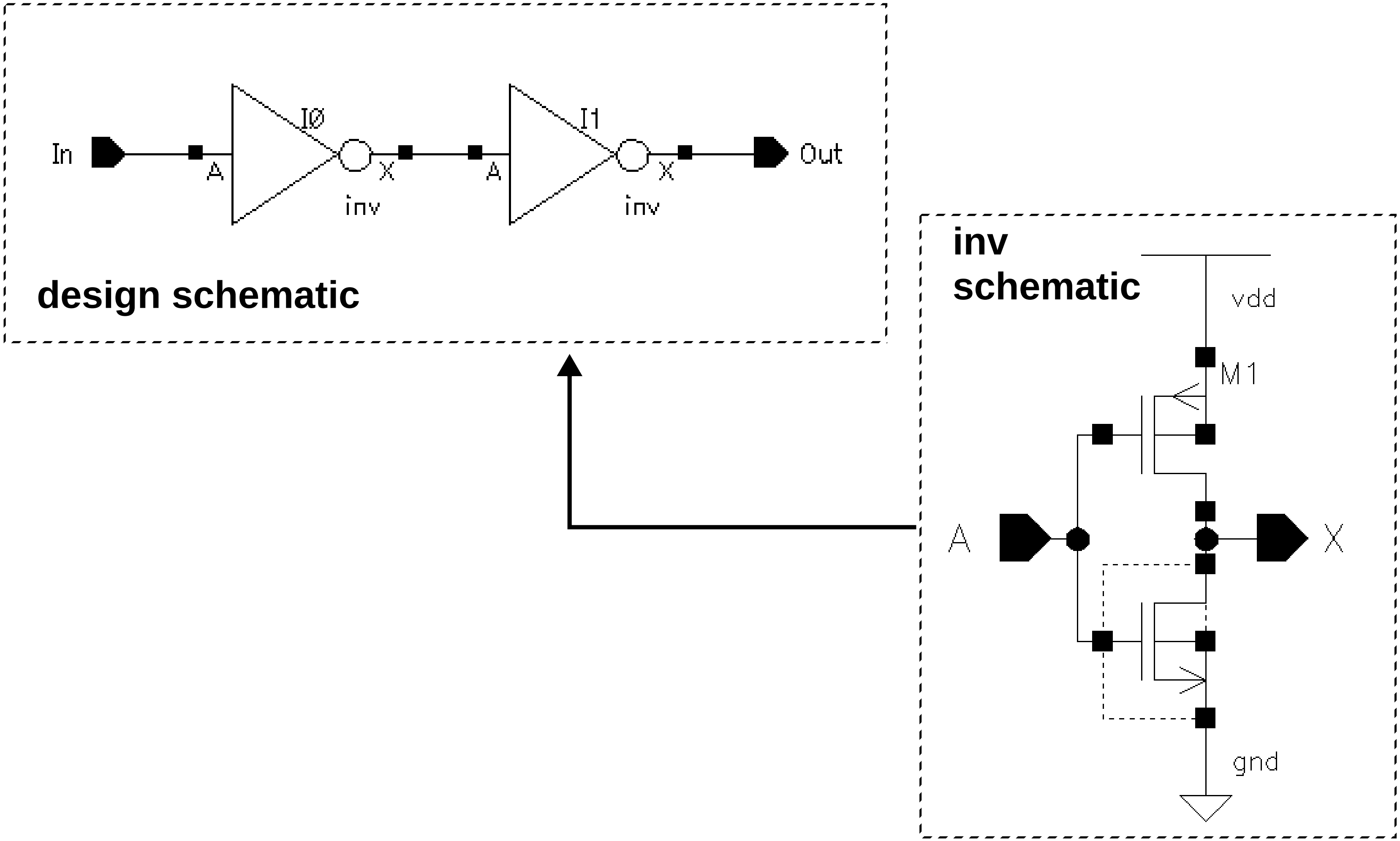
Assura Design Flow

1. Start with the schematic for the design. This can be a flat schematic or a hierarchical schematic. The design must be able to run LVS on any portion of the design that will be simulated with parasitics.
2. From the layout view of the design, select **Tools—Assura**. For any subblock of the design for which the parasitic simulation runs, the hierarchy of the layout must correspond to the hierarchy of the schematic.
3. In Assura, run LVS and RCX on the portion of the design to be simulated with parasitics.
4. In Assura, build an *av_analog_extracted* view of the parasitic portion of the design. At this point, run the OSS Netlister to produce an analog netlist of the design.
5. Create a test fixture that includes a symbol for your design and the appropriate sources and loads to run simulation. Create a *config* view of the design for use with the Hierarchy Editor.
6. Using the Hierarchy Editor, choose which views will be netlisted for simulation. Simulate with parasitics by choosing *av_analog_extracted* for the design or subblocks of the design. For parasitic simulation, the input must be analog netlist from the OSS Netlister.
7. Run the simulation. The netlist will use parasitic information as specified in your configuration.
8. Any net of the layout or schematic can be selected and then graphically plotted for waveform analysis.

Design Schematic

1

Create schematic for design



design schematic

inv schematic

Design Schematic

The design schematic can be flat or hierarchical, and cannot have sources or loads in this schematic. It must contain only those elements that will correspond directly to elements in the design layout.

If the design schematic is hierarchical, parasitic simulation can be run on specific subblocks of the design if corresponding layout exist for each subblock.

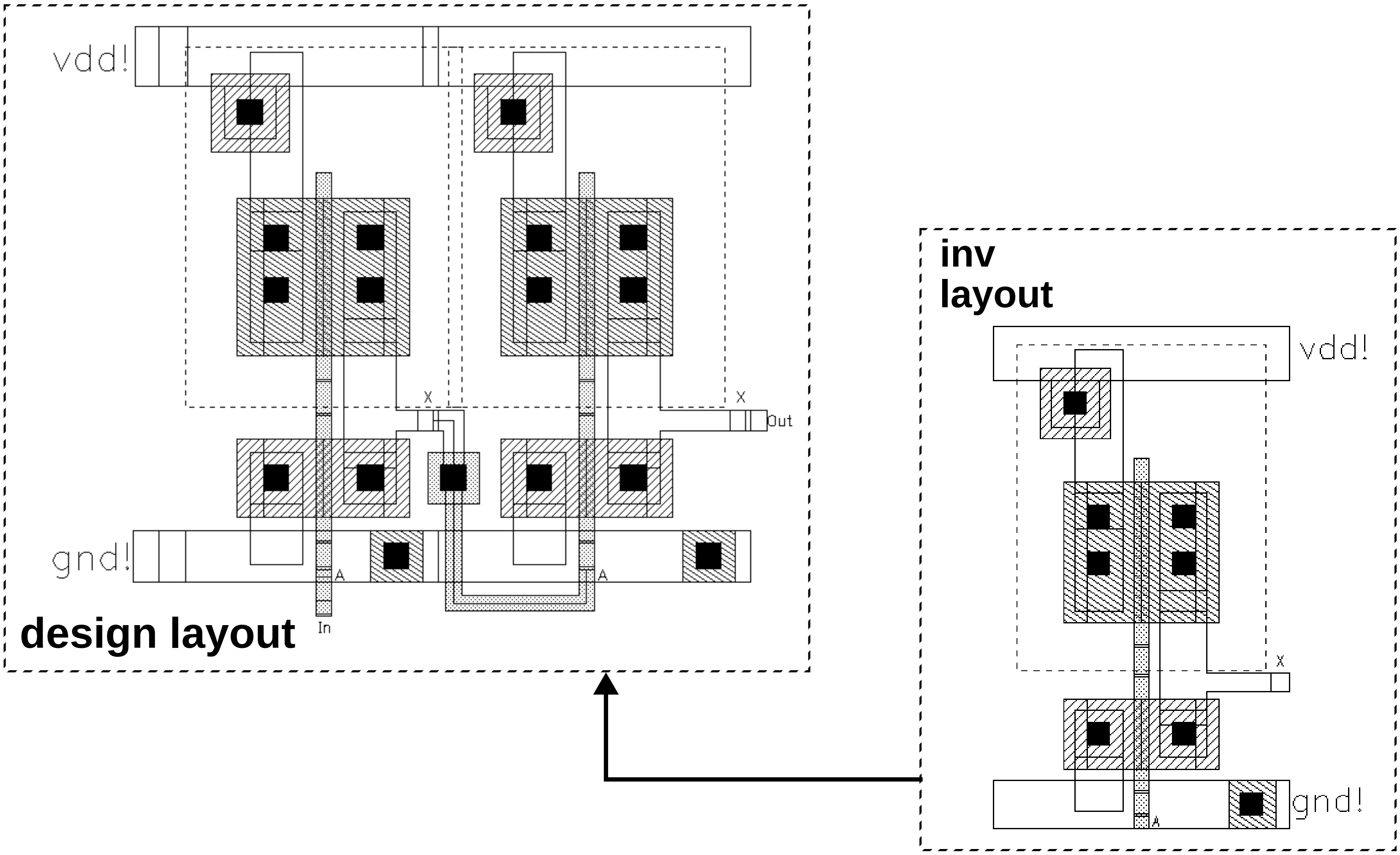
Each analog primitive used in the design needs:

- a *symbol* view.
- a *schematic* view.
- a *layout* view and corresponding extraction rules.
- a simulation model.
- an *auLvs* view.
- *auLvs* simulation information in the cell CDF.

Design Layout

2

Create layout for design



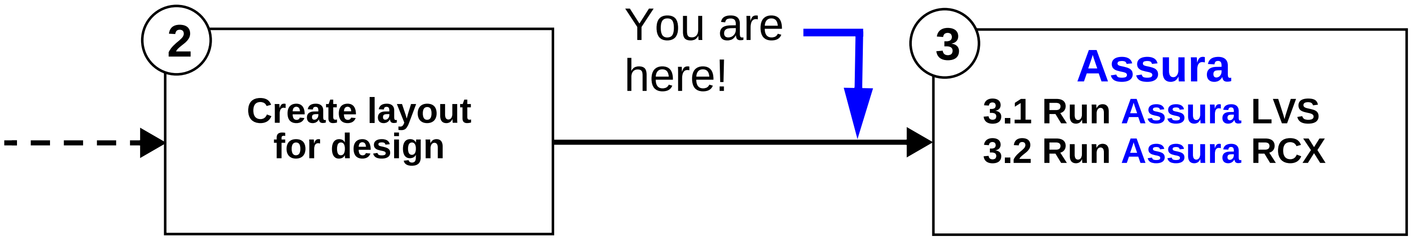
design layout

inv layout

Design Layout

Capture a layout for the design schematic using the Assura Design Tool. The design layout can be flat or hierarchical.

Accessing the Assura Commands



2

Create layout for design

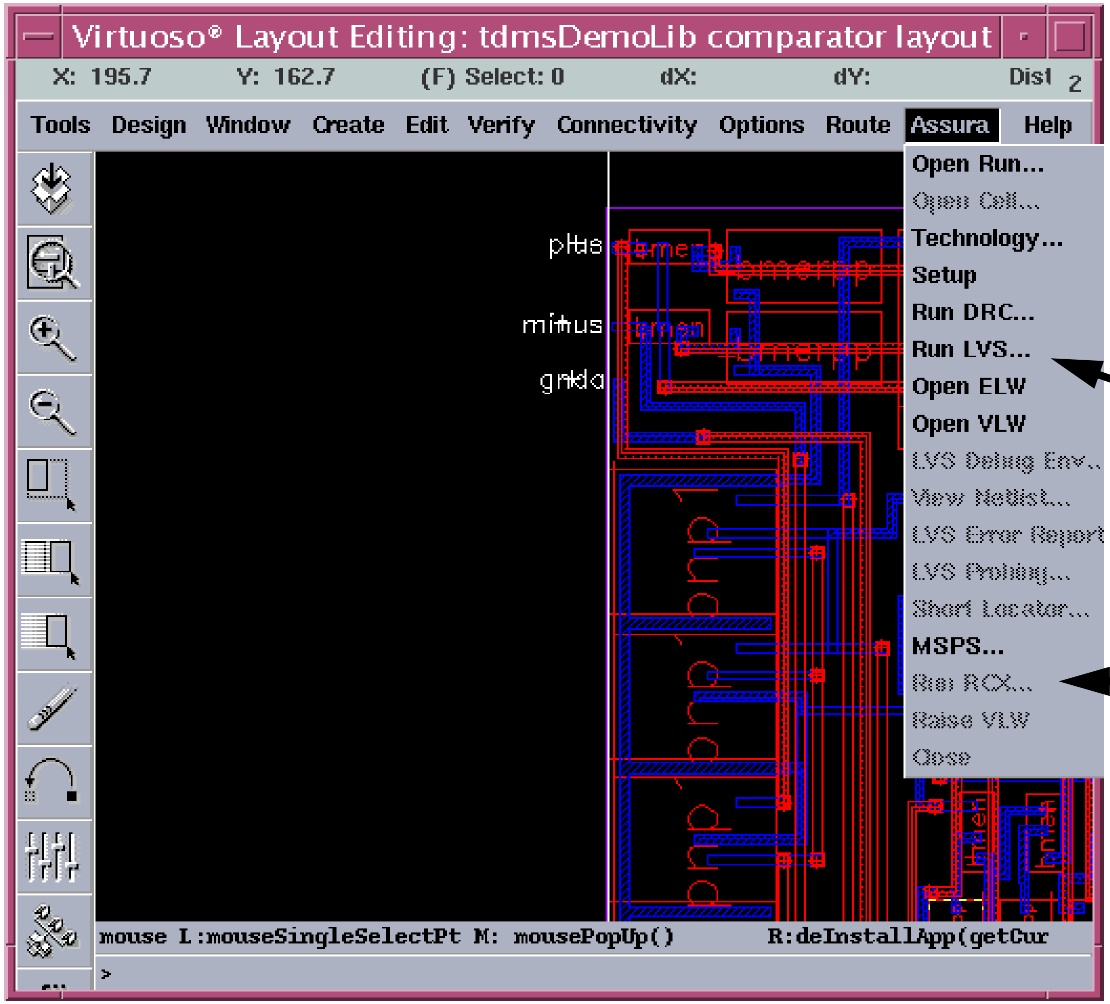
You are here!

3

Assura

3.1 Run Assura LVS

3.2 Run Assura RCX



Virtuoso® Layout Editing: tdmsDemoLib comparator layout

X: 195.7 Y: 162.7 (F) Select: 0 dX: dY: Dist 2

```
mouse L:mouseSingleSelectPt M: mousePopUp()        R:deInstallApp(getCur
>
```

Assura pull-down menu

Run LVS... Starts the Assura Run LVS form

Run RCX... is enabled only after completing LVS

Accessing the Assura Commands

At this point in the Assura Parasitic Simulation flow, the layout for the design has been captured. You are now ready to begin the extraction process, or Block 3 in the flow. At this point, you need to open Composer and activate the Assura pull-down menu to the Composer window.

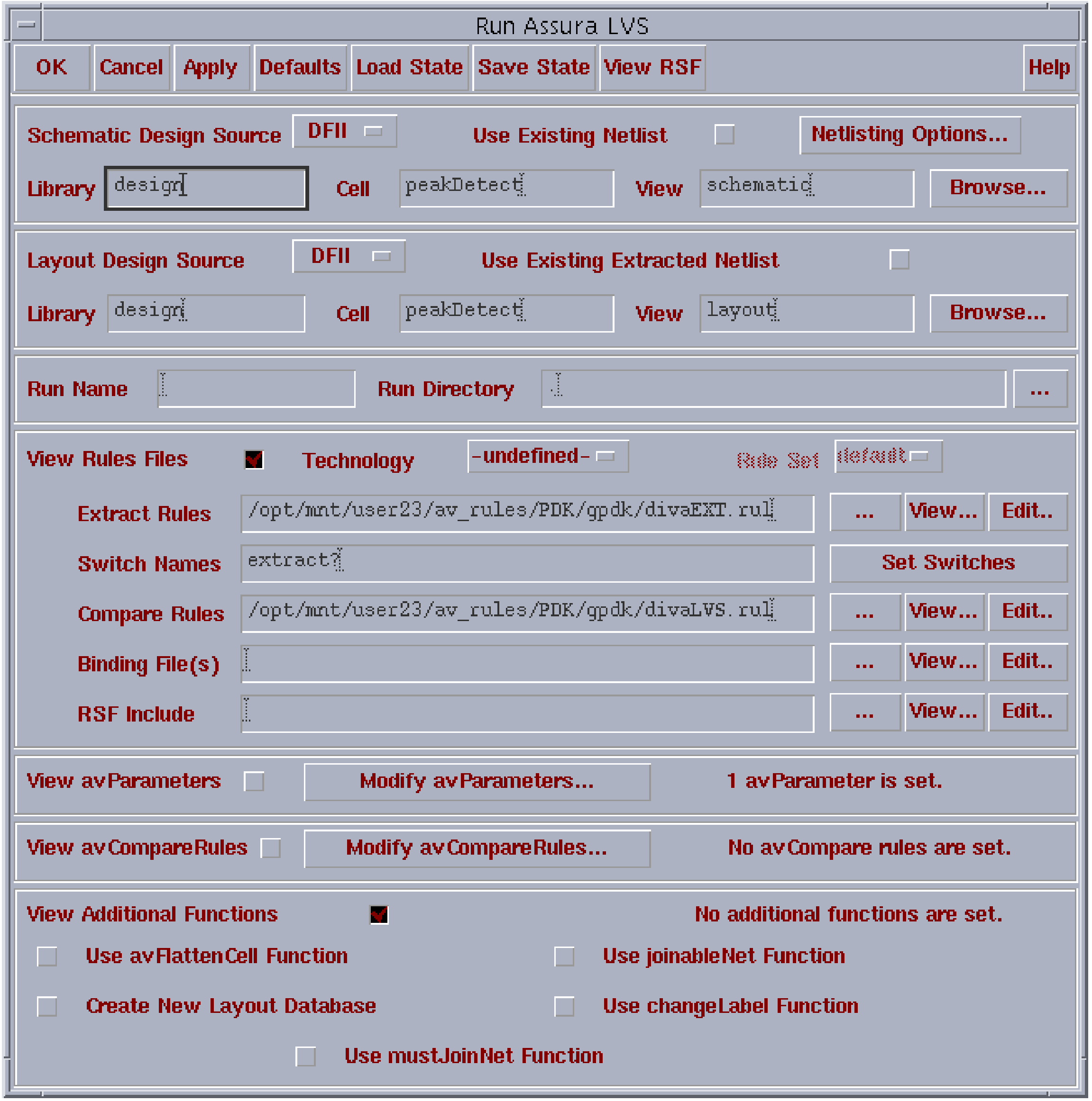
LVS in Assura

3 **Assura**

3.1 Run Assura LVS

3.2 Run Assura RCX

Assura—Run LVS...



Schematic Design Source DFII

Use Existing Netlist □

Library design Cell peakDetect View schematic

Layout Design Source DFII

Use Existing Extracted Netlist □

Library design Cell peakDetect View layout

Run Name

Run Directory

View Rules Files ☒ Technology -undefined- Rule Set default

Extract Rules /opt/mnt/user23/av_rules/PDK/gpdk/divaEXT.rul

Switch Names extract?

Compare Rules /opt/mnt/user23/av_rules/PDK/gpdk/divaLVS.rul

Binding File(s)

RSF Include

View avParameters □ 1 avParameter is set.

View avCompareRules □ No avCompare rules are set.

View Additional Functions ☒ No additional functions are set.

□ Use avFlattenCell Function

□ Use joinableNet Function

□ Create New Layout Database

□ Use changeLabel Function

□ Use mustJoinNet Function

LVS in Assura

Assura requires the extraction files and schematics to perform the Assura LVS.

Select **Run LVS** from the Assura pulldown menu to perform LVS for the target layout and schematic. When LVS is successful, the **Run RCX** field will become enabled.

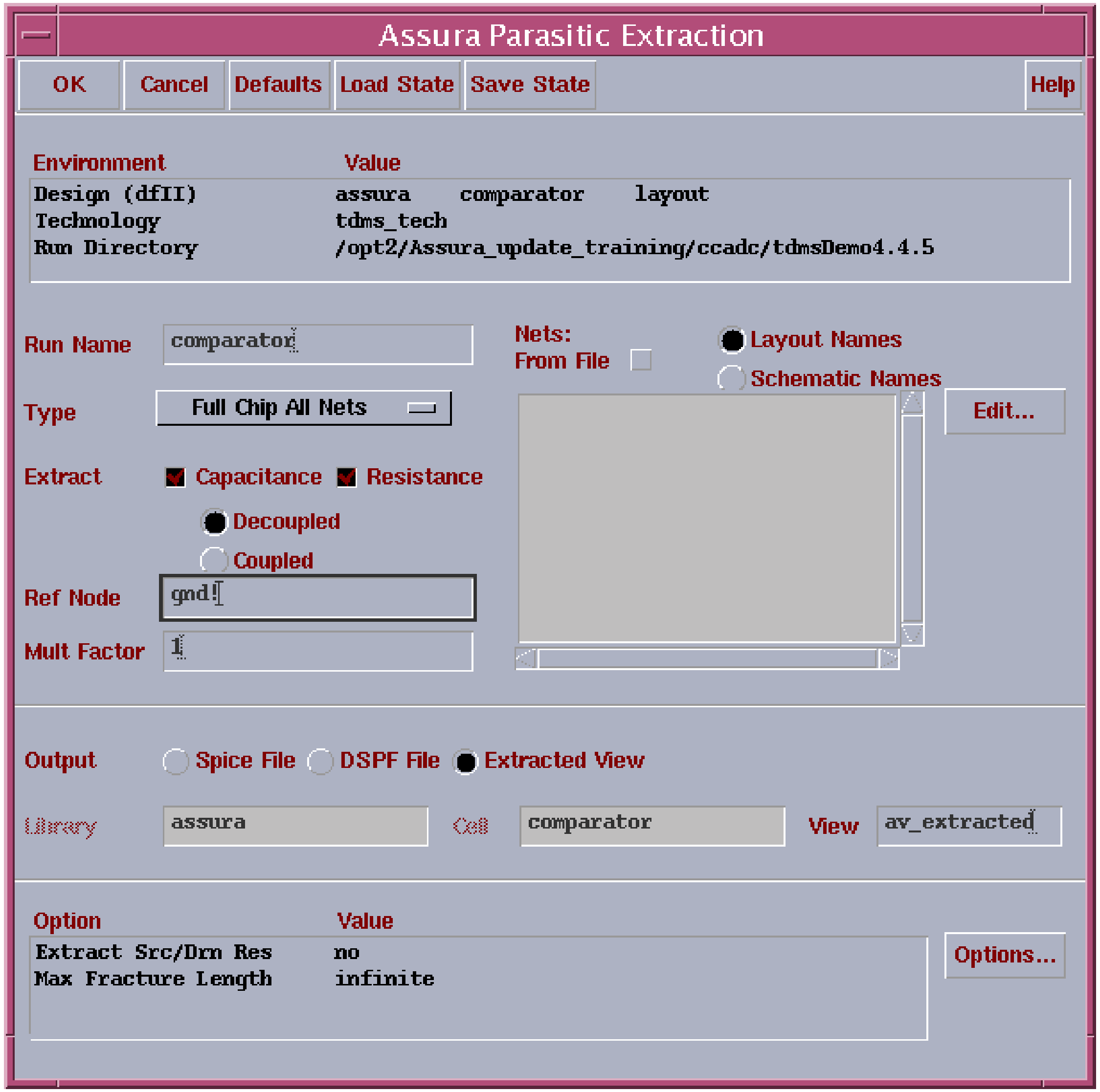
Run RCX in Assura

3

Assura

3.1 Run Assura LVS

3.2 Run Assura RCX



Environment	Value
Design (dfII)	assura comparator layout
Technology	tdms_tech
Run Directory	/opt2/Assura_update_training/ccadc/tdmsDemo4.4.5

Run Name: comparator

Type: Full Chip All Nets

Extract: ☒ Capacitance ☒ Resistance

● Decoupled
○ Coupled

Ref Node: gnd!

Mult Factor: 1

Nets: From File □

● Layout Names
○ Schematic Names

Output: ○ Spice File ○ DSPF File ● Extracted View

Library: assura Cell: comparator View: av_extracted

Option	Value
Extract Src/Drn Res	no
Max Fracture Length	infinite

Run RCX in Assura

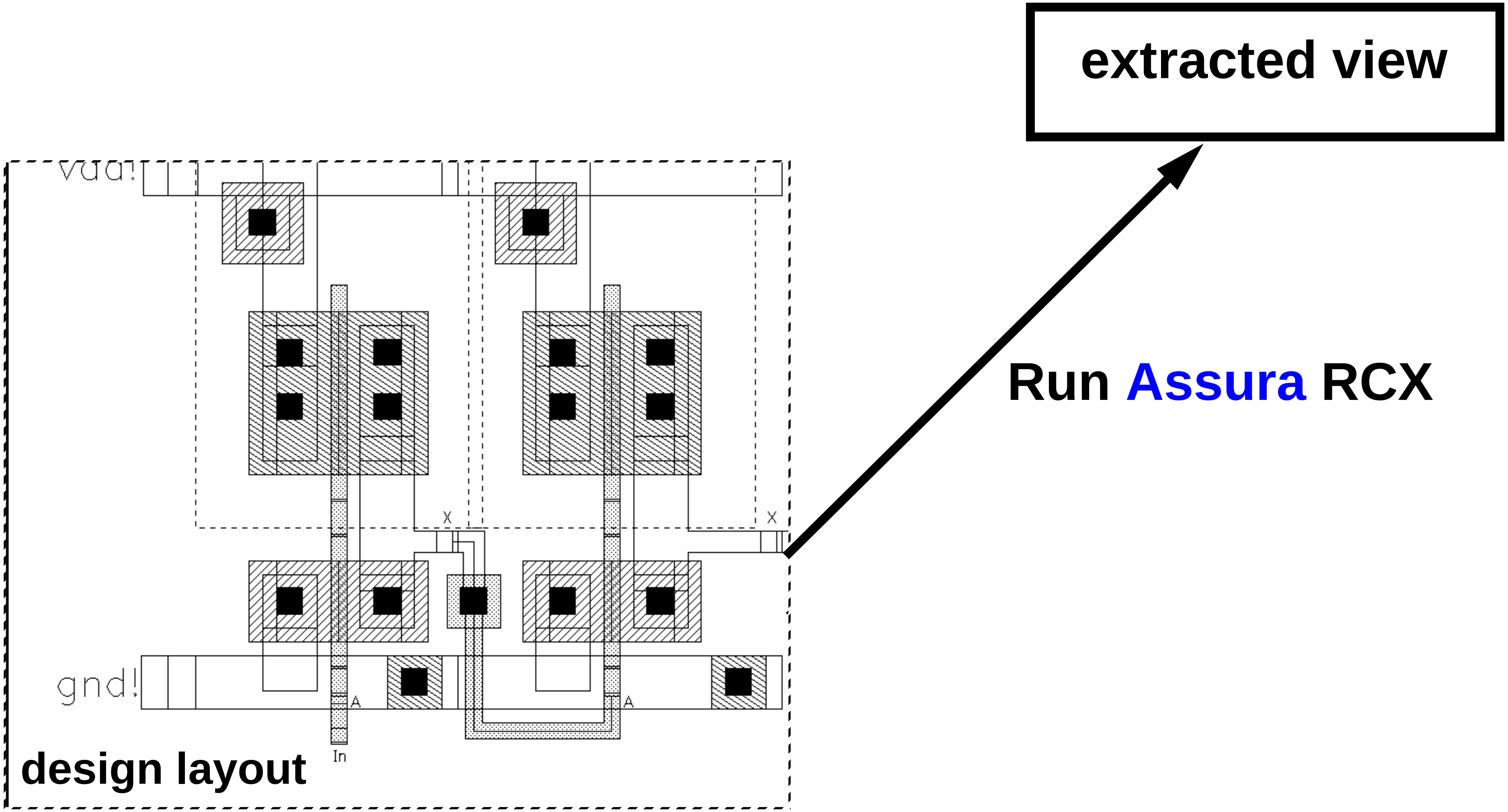
The Assura 2.0 generated extracted view does not have layout polygons present; it uses the CDBA “group” mechanism to tag the parasitic resistor sub-nets, and a new property “extractionCreatedBy” set to “Assura” is now present to distinguish the Assura generated extracted view from the Diva generated extracted view. Most significantly, the Assura extracted view uses algorithmically mapped hierarchical schematic names.

Assura RCX Output

3 **Assura**

3.1 Run Assura LVS

3.2 Run Assura RCX



extracted view

Run Assura RCX

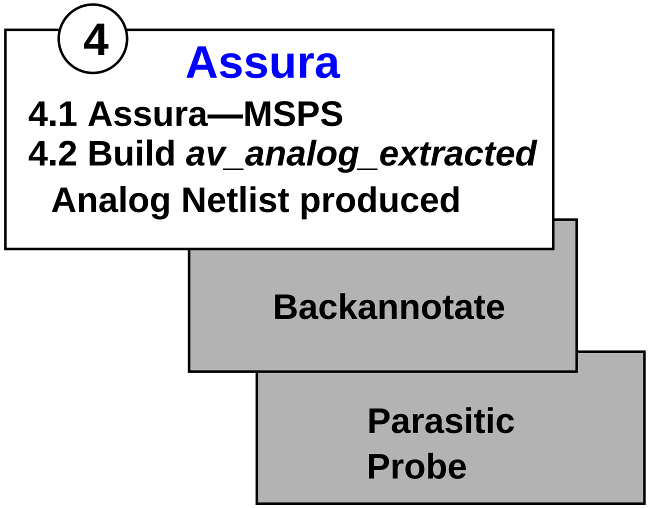
design layout

Assura RCX Output

Running Assura RCX builds an extraction of the layout that has connectivity, designed devices, and parasitic information. This step in the flow requires the layout to be completed, and LVS must be completed successfully.

In addition, Assura extraction rules files must also exist.

Building *av_analog_extracted*



4 **Assura**

4.1 Assura—MSPS

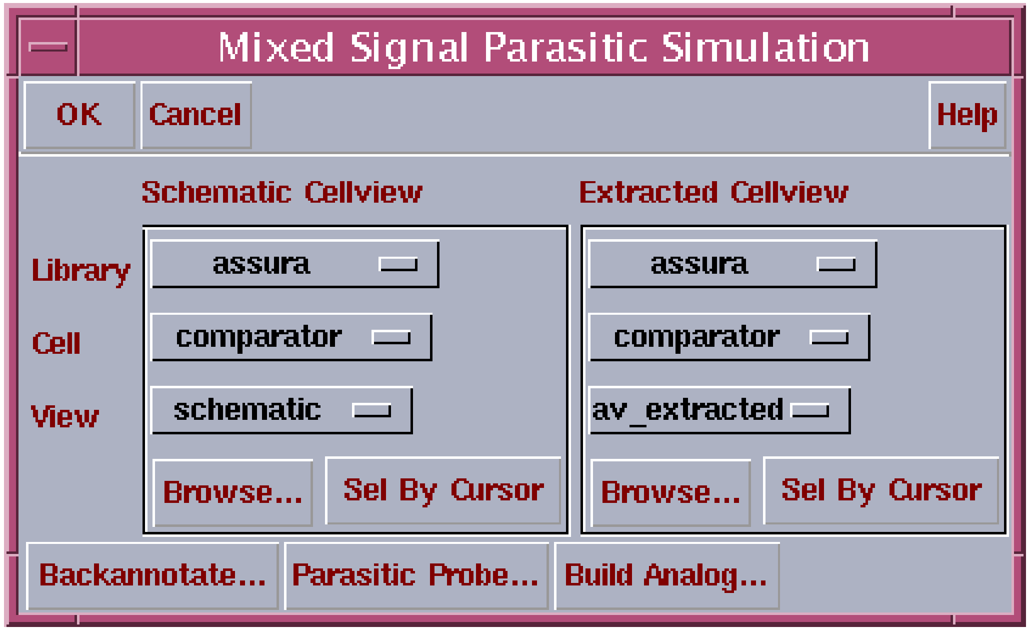
4.2 Build *av_analog_extracted*

Analog Netlist produced

Backannotate

Parasitic Probe

Assura—MSPS...



Mixed Signal Parasitic Simulation

	Schematic Cellview	Extracted Cellview
Library	assura	assura
Cell	comparator	comparator
View	schematic	av_extracted

Building *av_analog_extracted*

Create the Assura *av_analog_extracted* view by selecting **Assura—MSPS** from the Assura pulldown menu.

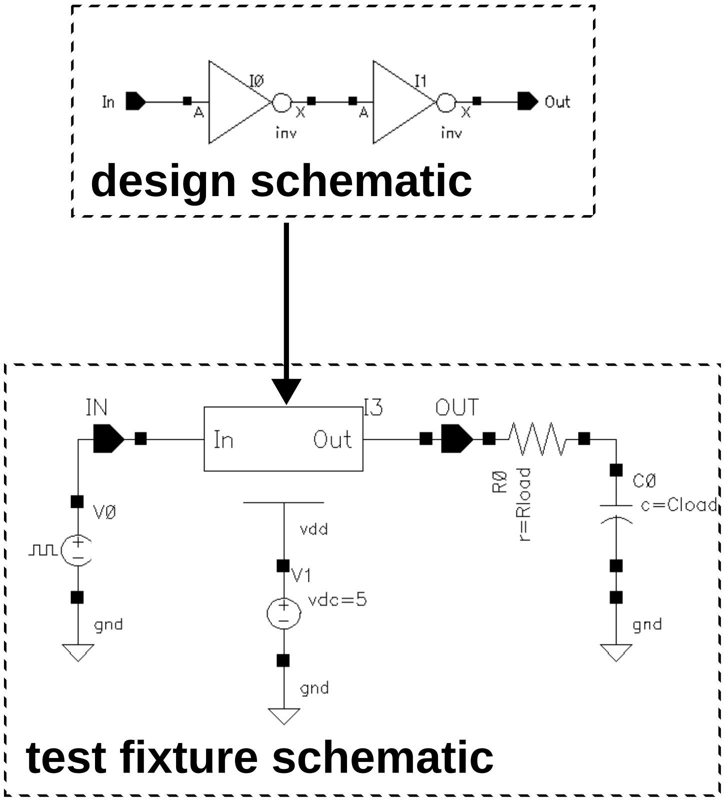
Select the **Build Analog...** button at the bottom of the form.

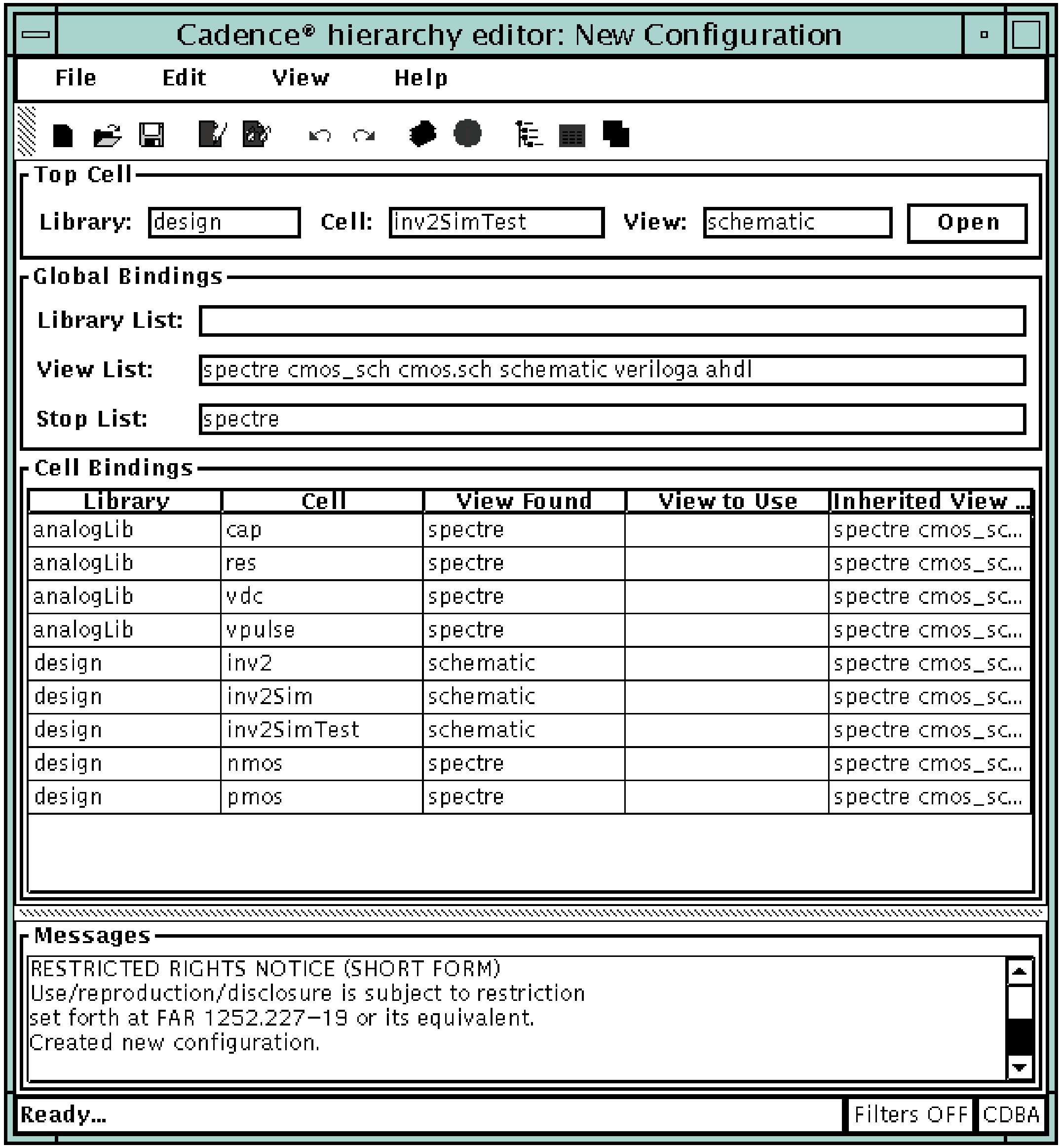
- By default, the analog extracted view is called av_analog_extracted
- The av_analog extracted view is basically a copy of the extracted view
  - Some or all of the parasitics can be removed
  - Nets ending with "!" are recognized as global signals

Note: The buildAnalog procedure is used to build the av_analog_extracted view from the extracted view, which is essentially a copy of the extracted view. Optionally, some or all of the parasitic values can be removed. Also, nets whose names end with the exclamation mark ('!') are recognized and tagged as global signals.

Create Test Fixture Schematic and Configuration

5 Create test fixture schematic & configuration





Top Cell

Library: design Cell: inv2SimTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Cell Bindings

Library	Cell	View Found	View to Use	Inherited View ...
analogLib	cap	spectre		spectre cmos_sc...
analogLib	res	spectre		spectre cmos_sc...
analogLib	vdc	spectre		spectre cmos_sc...
analogLib	vpulse	spectre		spectre cmos_sc...
design	inv2	schematic		spectre cmos_sc...
design	inv2Sim	schematic		spectre cmos_sc...
design	inv2SimTest	schematic		spectre cmos_sc...
design	nmos	spectre		spectre cmos_sc...
design	pmos	spectre		spectre cmos_sc...

Messages

```
RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Created new configuration.
```

Ready... Filters OFF CDBA

Create Test Fixture Schematic and Configuration

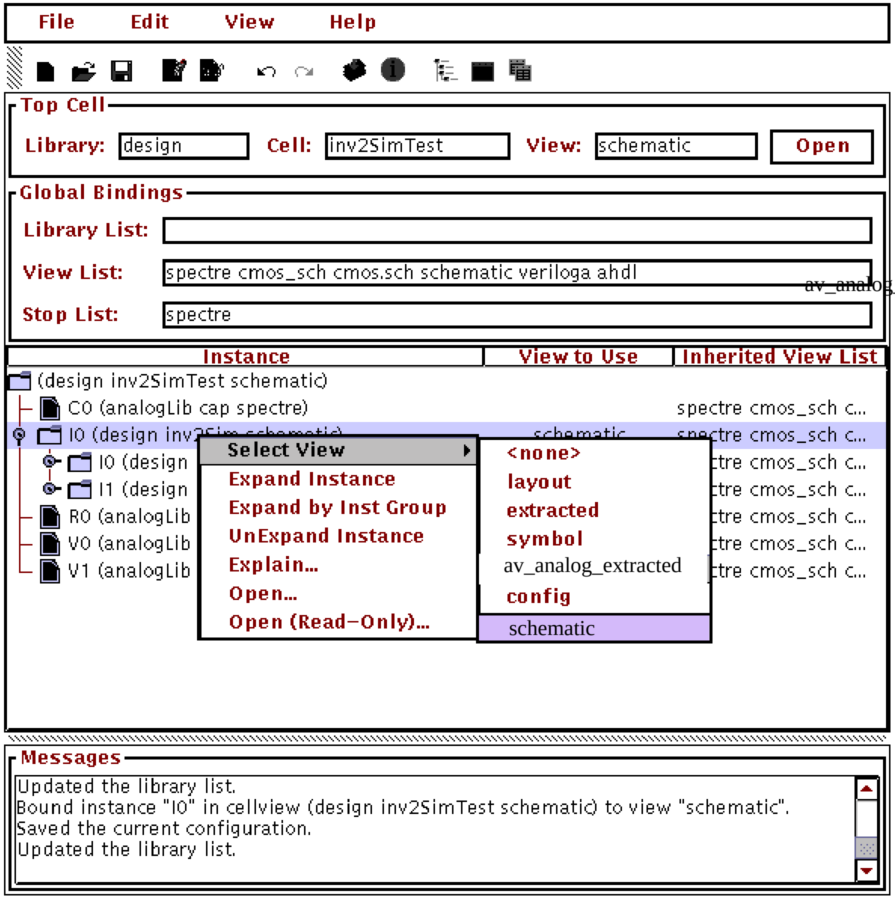
1. Create a *schematic* cellview.
   — Use a symbol for the design and appropriate sources and loads for simulation.
2. Create a *config* cellview.
   — Use the template for the simulator you are using.
   — Use the test fixture schematic as the top cell.

Choose Views in Configuration

6

Choose views in configuration (Netlist with or without Parasitics)

Netlist without parasitics



Top Cell

Library: design Cell: inv2SimTest View: schematic Open

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Instance	View to Use	Inherited View List
(design inv2SimTest schematic)		
C0 (analogLib cap spectre)		spectre cmos_sch c...
I0 (design inv2Sim schematic)	schematic	spectre cmos_sch c...
I0 (design		tre cmos_sch c...
I1 (design		tre cmos_sch c...
R0 (analogLib		tre cmos_sch c...
V0 (analogLib		tre cmos_sch c...
V1 (analogLib		tre cmos_sch c...

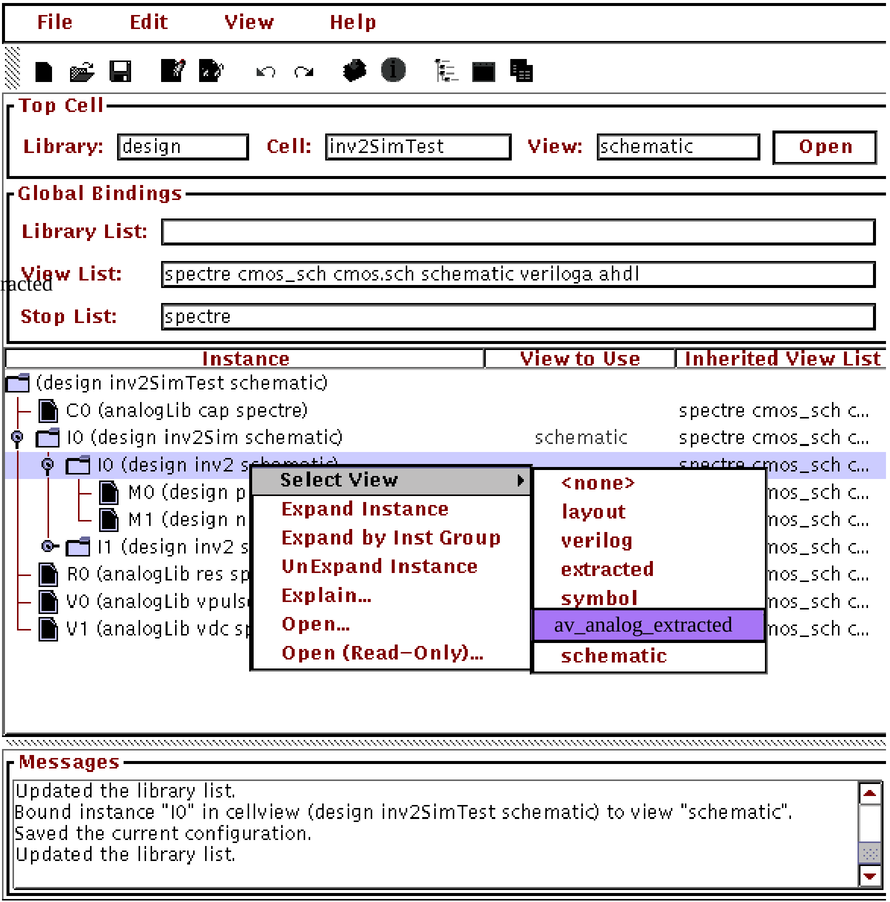
Messages

```
Updated the library list.
Bound instance "I0" in cellview (design inv2SimTest schematic) to view "schematic".
Saved the current configuration.
Updated the library list.
```

Sets View to analog_extracted

av_analog_extracted

Netlist with parasitics



Top Cell

Library: design Cell: inv2SimTest View: schematic Open

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Instance	View to Use	Inherited View List
(design inv2SimTest schematic)		
C0 (analogLib cap spectre)		spectre cmos_sch c...
I0 (design inv2Sim schematic)	schematic	spectre cmos_sch c...
I0 (design inv2 schematic)		spectre cmos_sch c...
M0 (design p		mos_sch c...
M1 (design n		mos_sch c...
I1 (design inv2 s		mos_sch c...
R0 (analogLib res sp		mos_sch c...
V0 (analogLib vpuls		mos_sch c...
V1 (analogLib vdc sp		mos_sch c...

Messages

```
Updated the library list.
Bound instance "I0" in cellview (design inv2SimTest schematic) to view "schematic".
Saved the current configuration.
Updated the library list.
```

Sets View to analog_extracted

Choose Views in Configuration

Choose to use parasitic information for:

- The entire design.
- Specific subblocks at any point in the hierarchy of the design.
- A specific net in a specific subblock. Use the selective simulation components to indicate which net and then use the Hierarchy Editor to choose the subblock view.

To set the views for the design:

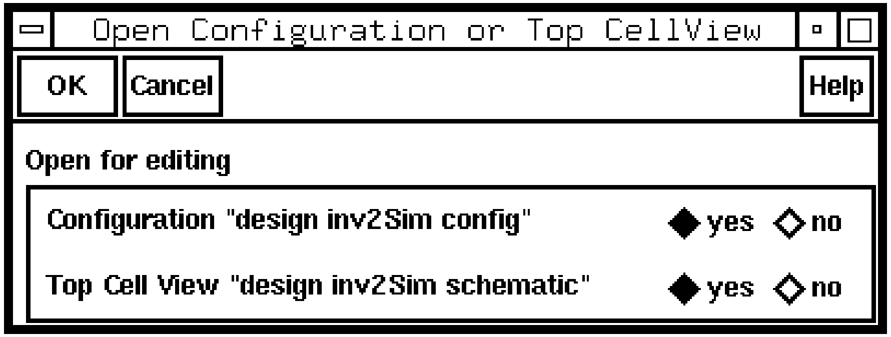
1. In the Hierarchy Editor, use the “tree” icon to expand the hierarchy.
2. For each block that you wish to simulate with parasitics, set the view to *av_analog_extracted*.
3. Click **Update** and then save the changes.

Running Simulation

7

Run Simulation

1. Open the configured schematic from the HED.



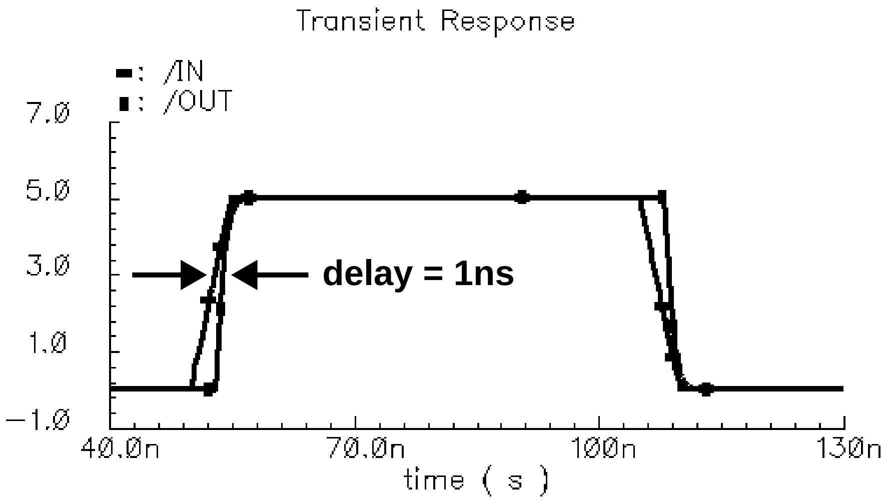
Open Configuration or Top CellView

Open for editing

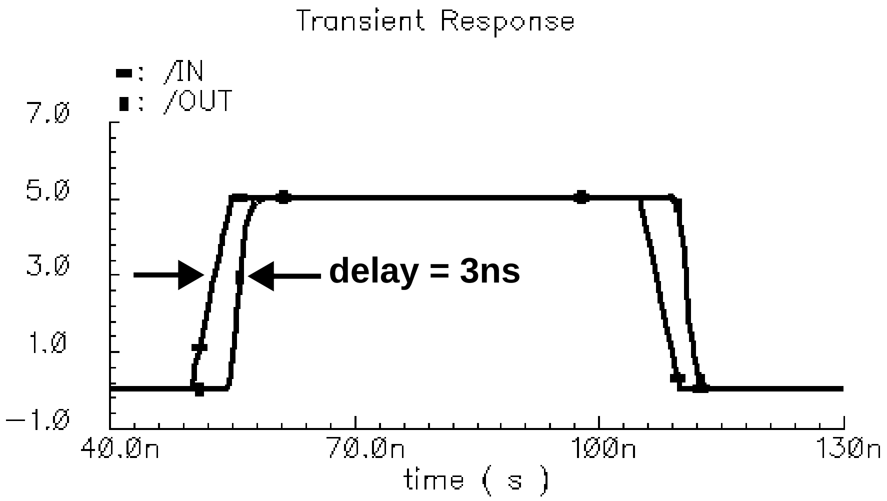
Configuration "design inv2Sim config" ◆ yes ◇ no

Top Cell View "design inv2Sim schematic" ◆ yes ◇ no

2. Start ADE from the configured schematic and run the simulation.



Simulation without parasitics



Simulation with parasitics

Running Simulation

To run simulation using the views defined in the configuration, open the design from the *config* view.

When opening a *config* view, the default is to open the *schematic* view in a window without actually opening the *config* view. However, all netlisting is being controlled by the *config*.

For running parasitic simulation, open the *config* view as well as the *schematic* view to easily change the configuration views as needed.

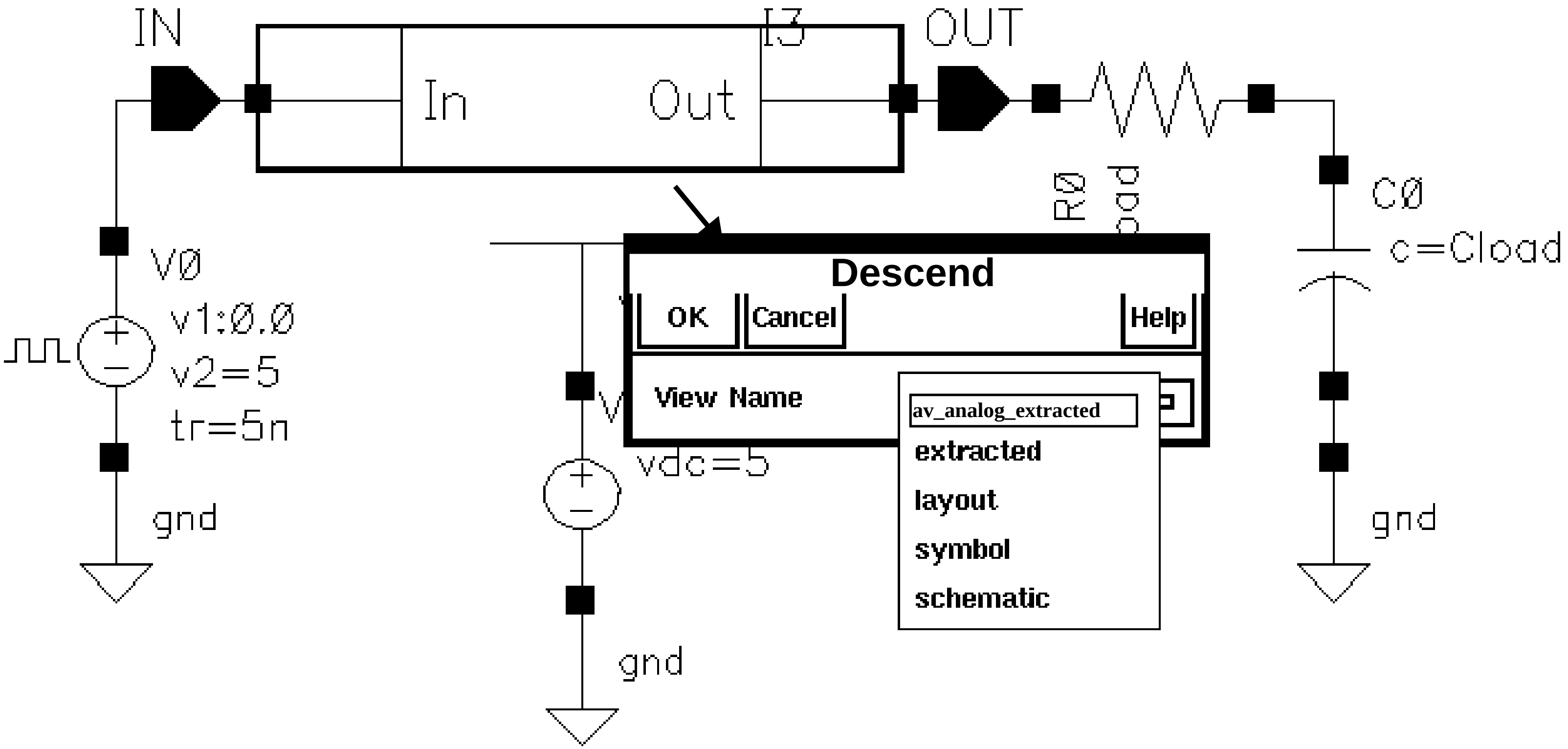
Waveform Analysis

8

Schematic Waveform Analysis

Extracted Waveform Analysis

You can choose waveforms in either the schematic or extracted layout:



View Name av_analog_extracted

Waveform Analysis

In parasitic simulation, the netlist is derived based on the views chosen in the configuration.

When descending into an instance from the test fixture schematic, choose from a list of views to descend into. By default, the first view offered will be the one defined for simulation in the configuration.

Labs

Lab 16-1 Parasitic Simulation Flow

Labs

Appendix A: Diva Parasitic Simulation Flow

Topics in this module:

- Overview
- Design flow
- Schematic requirements
- LVS requirements and results
- Parasitic simulation messages and options
- Selective parasitic simulation

Terms and Definitions

CDF	*Component Description Format* is information stored on a cell that defines parameters, simulation rules, and display information.
Selected Parasitic Simulation	Selected parasitic simulation limits the nets that are simulated with parasitic information.
Source/Loads	Components in the schematic that are for simulation purposes only.
LVS	Layout versus schematic checking is used to verify that the layout matches the schematic.
Test Fixture Schematic	A schematic containing a symbol representing the design schematic with all required simulation sources and loads attached to I/O pins of the symbol.
“LVS-able” block	A subblock with a layout and a schematic in the same cell that can be verified with LVS.

Diva Parasitic Extraction

The Diva flow for parasitic extraction is also supported by the Analog Design Environment. This module is included for the following reasons:

- For those facilities where Diva is used and Assura is not installed.
- For classes at the customer sites where Diva is used.
- For classes at customer sites where Diva is specifically requested.
- For special circumstances, such as when the class is presented by an instructor who is not familiar with the Assura tool.

All versions of the Analog Design Environment as well as the previous versions of the Analog Artist environment support parasitic simulation using Diva.

The Assura parasitic simulation flow has only been available since the release of IC-4.4.6.

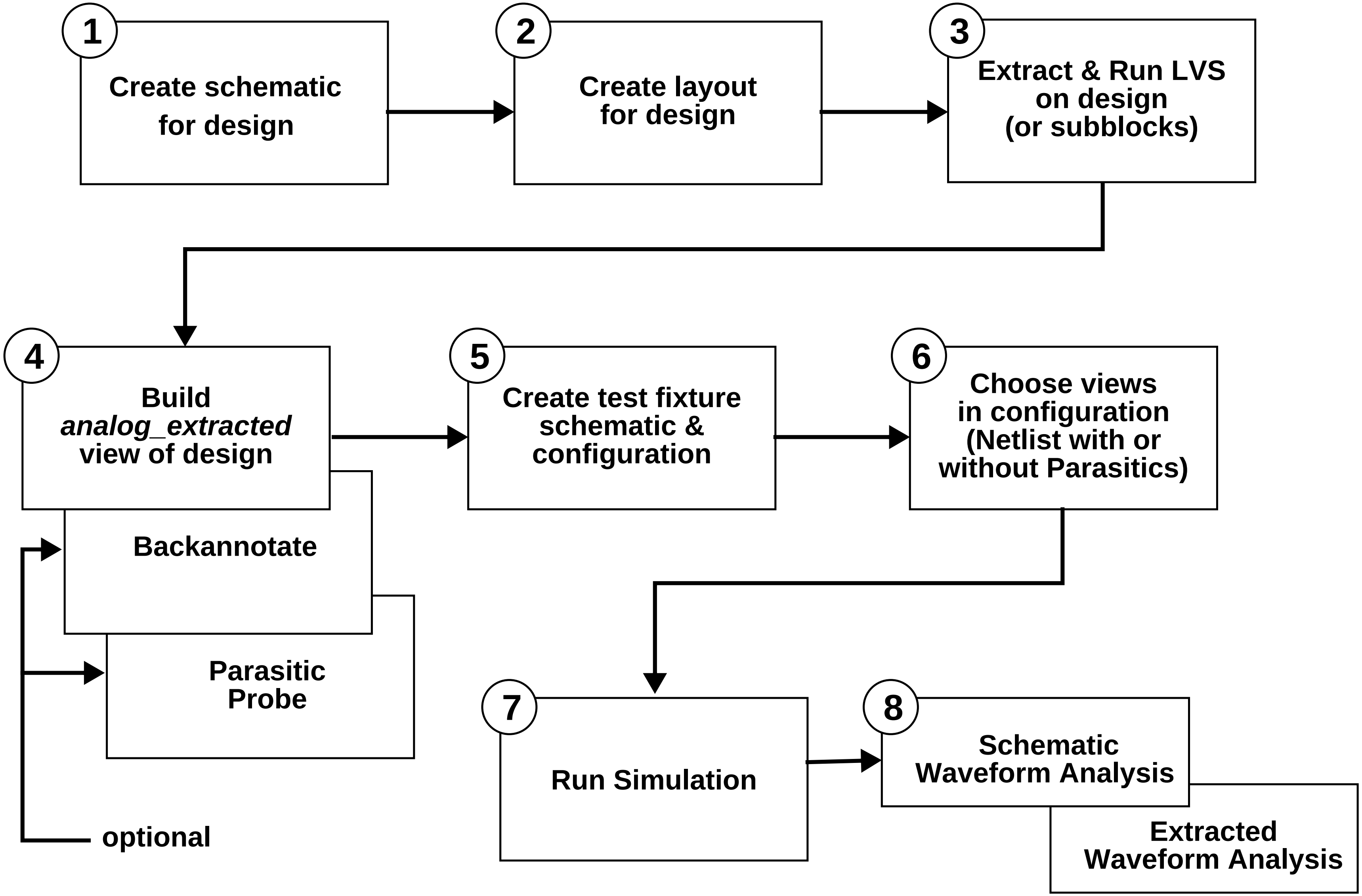
The Diva and Assura parasitic simulation flows have only minor differences.

Diva Parasitic Extraction

Parasitic simulation using the Assura flow has only been available since the release of IC-4.4.6, and in prior releases of ADE and AADS, the Diva flow was used exclusively.

The Diva and Assura flows are similar.

Diva Design Flow with Parasitic Simulation



1 Create schematic for design

2 Create layout for design

3 Extract & Run LVS on design (or subblocks)

4 Build *analog_extracted* view of design

Backannotate

Parasitic Probe

optional

5 Create test fixture schematic & configuration

6 Choose views in configuration (Netlist with or without Parasitics)

7 Run Simulation

8 Schematic Waveform Analysis

Extracted Waveform Analysis

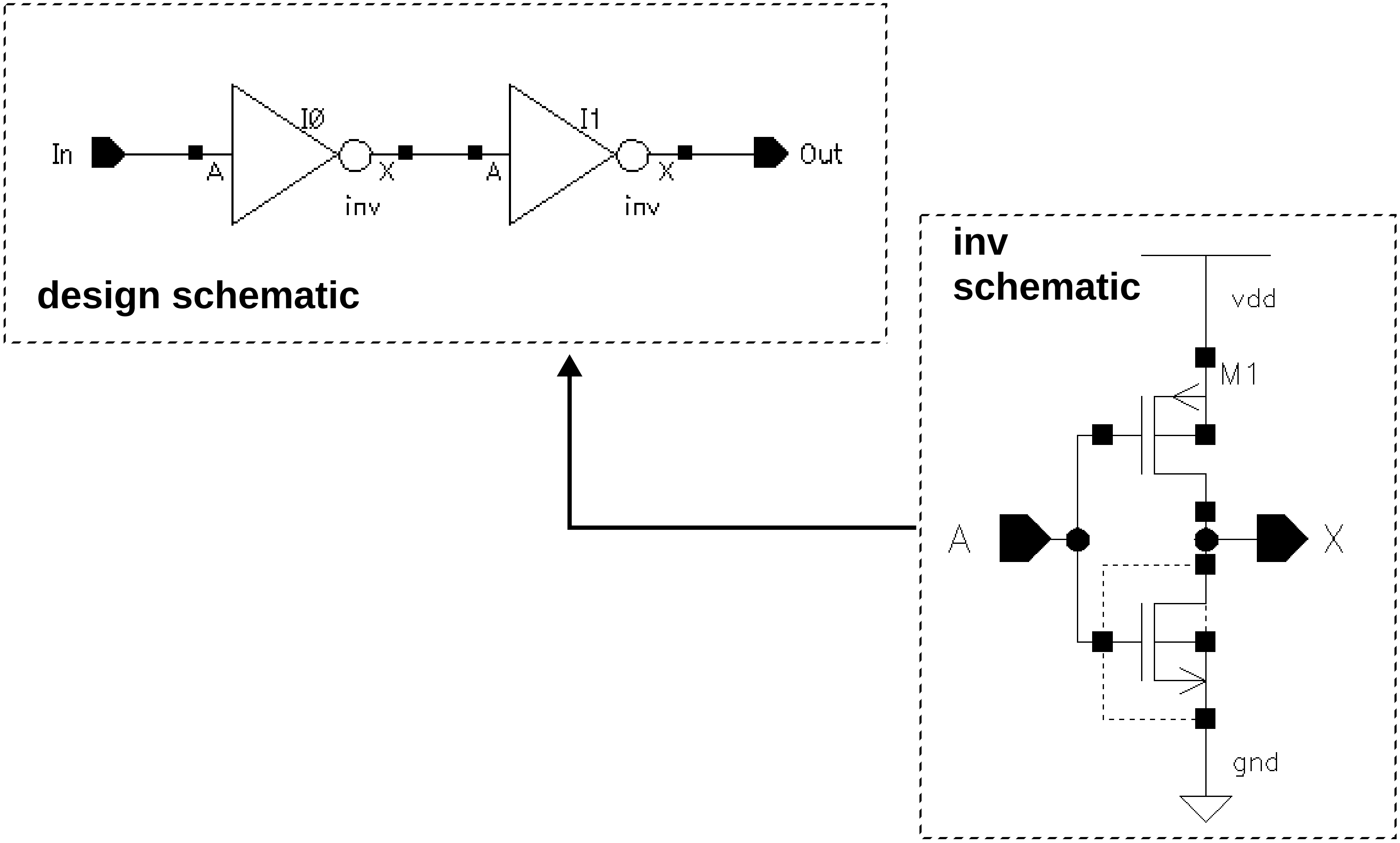
Diva Design Flow with Parasitic Simulation

1. Start with the schematic for the design. This can be a flat schematic or a hierarchical schematic. The design must be able to run LVS on any portion of the design that will be simulated with parasitics.
2. Create the layout for the design. For any subblock of the design for which runs the parasitic simulation, the hierarchy of the layout must correspond to the hierarchy of the schematic.
3. Extract and run LVS on the portion of the design to simulate with parasitics.
4. Build an *analog_extracted* view of the parasitic portion of the design. At this point additional parasitic information, using **Backannotation** or **Parasitic Probing**, is available for simulation.
5. Create a test fixture that includes a symbol for your design and the appropriate sources and loads to run simulation. Create a *config* view of the design for use with the Hierarchy Editor.
6. Using the Hierarchy Editor, choose which views will be netlisted for simulation. Simulate with parasitics by choosing *analog_extracted* for the design or subblocks of the design. For parasitic simulation, the view must be *analog_extracted*.
7. Run Simulation. The netlist will use parasitic information as specified in your configuration.
8. Any net of the layout or schematic can be selected and then graphically plotted for waveform analysis.

Design Schematic

1

Create schematic for design



design schematic

inv schematic

Design Schematic

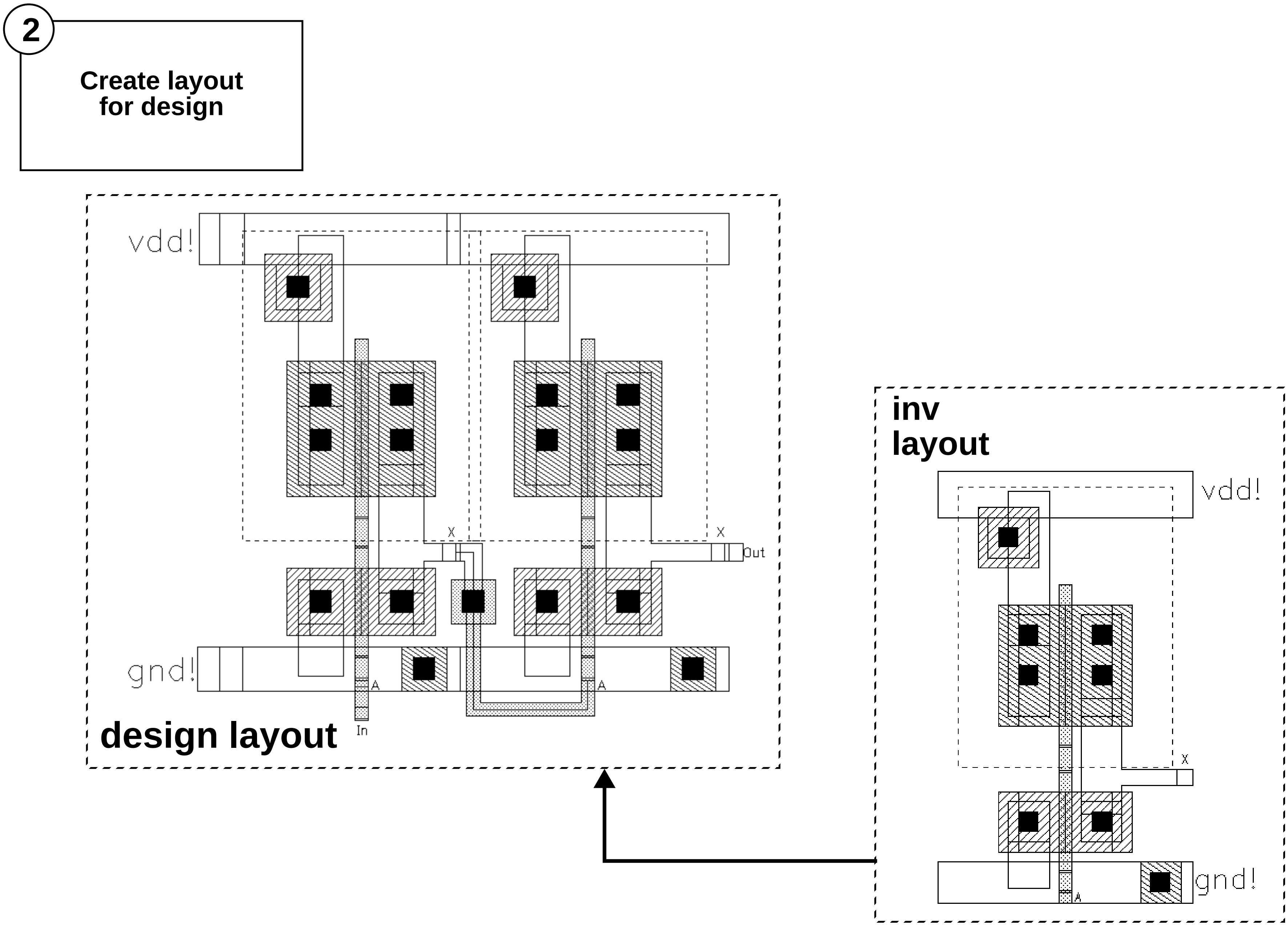
The design schematic can be flat or hierarchical, and cannot have sources or loads in this schematic. It must contain only those elements that will correspond directly to elements in the design layout.

If the design schematic is hierarchical, parasitic simulation can be run on specific subblocks of the design if corresponding layout exist for each subblock.

Each analog primitive used in the design needs:

- a *symbol* view.
- a *schematic* view.
- a *layout* view and corresponding extraction rules.
- a simulation model.
- an *auLvs* view.
- *auLvs* simulation information in the cell CDF.

Design Layout



2

Create layout for design

design layout

inv layout

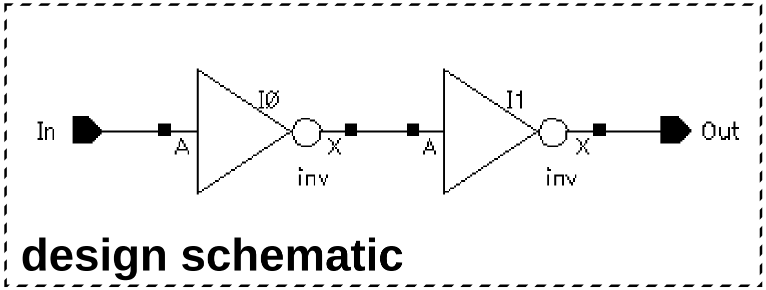
Design Layout

The design layout can be flat or hierarchical.

Extraction and LVS

3

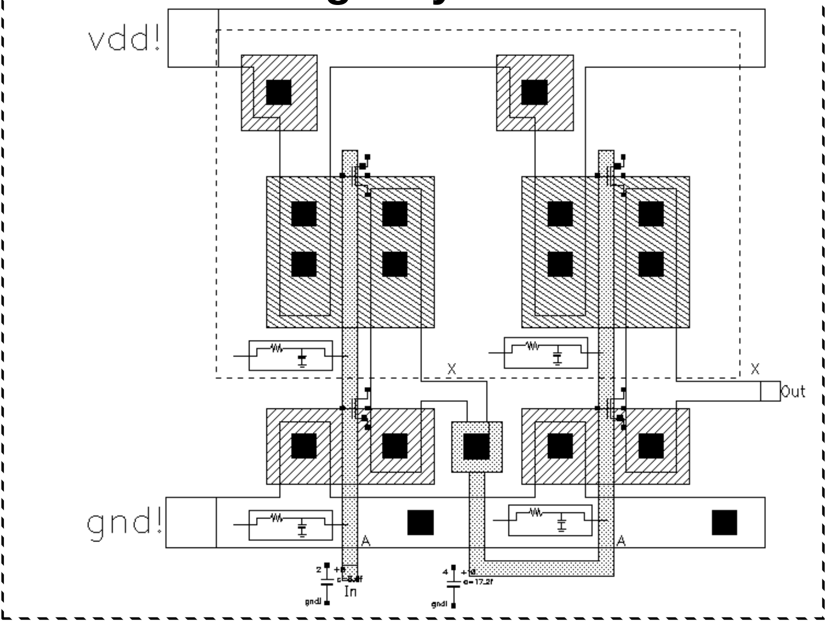
Extract & Run LVS on design (or subblocks)



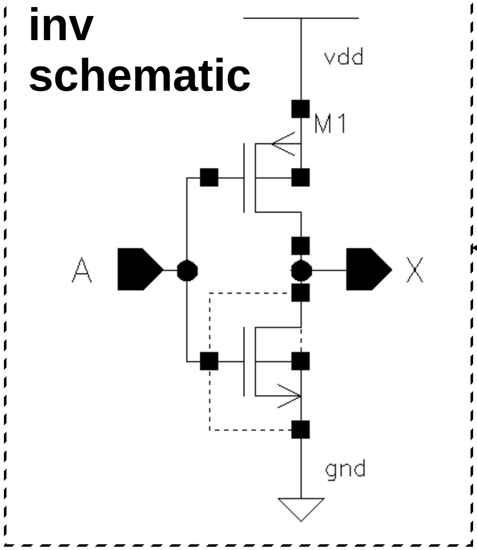
design schematic

LVS

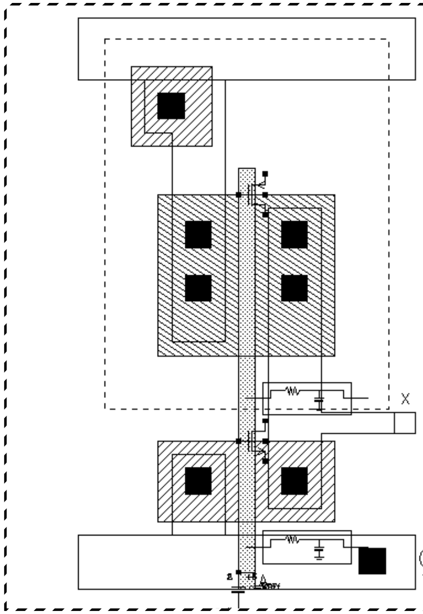
extracted design layout



inv schematic



LVS



extracted inv layout

Extraction and LVS

Use extraction to build a view of the layout that has connectivity, designed devices, and parasitic information for simulation netlisting. The design must pass LVS with the netlists matching using selective parasitics or waveform selection from the schematic.

1. Extract the layout without parasitic devices. Device extraction must include measurement of any device parameters used in simulation.

2. Run LVS between the schematic and extracted layout to verify the design is correct.

3. Extract the layout including parasitic devices. Parasitic cells require:

   — *symbol* and *auLvs* views.

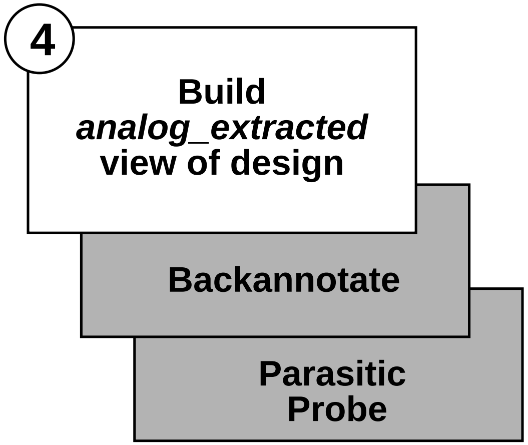
   — CDF component parameters for resistance and capacitance. The parameter names must be **r** and **c** for the parasitic probing and backannotation commands to work.

   — CDF simulation information for auLvs. The symbols can have any name but the *componentName* in the *auLvs* simulation information must be *pcapacitor*, *presistor*, *pinductor*, or *pdiode* for the comparison programs to recognize them as parasitic components. Examples are available in the *analogLib* library found at:

   ```
   <inst_dir>/tools/dfII/etc/cdslib/artist
   ```

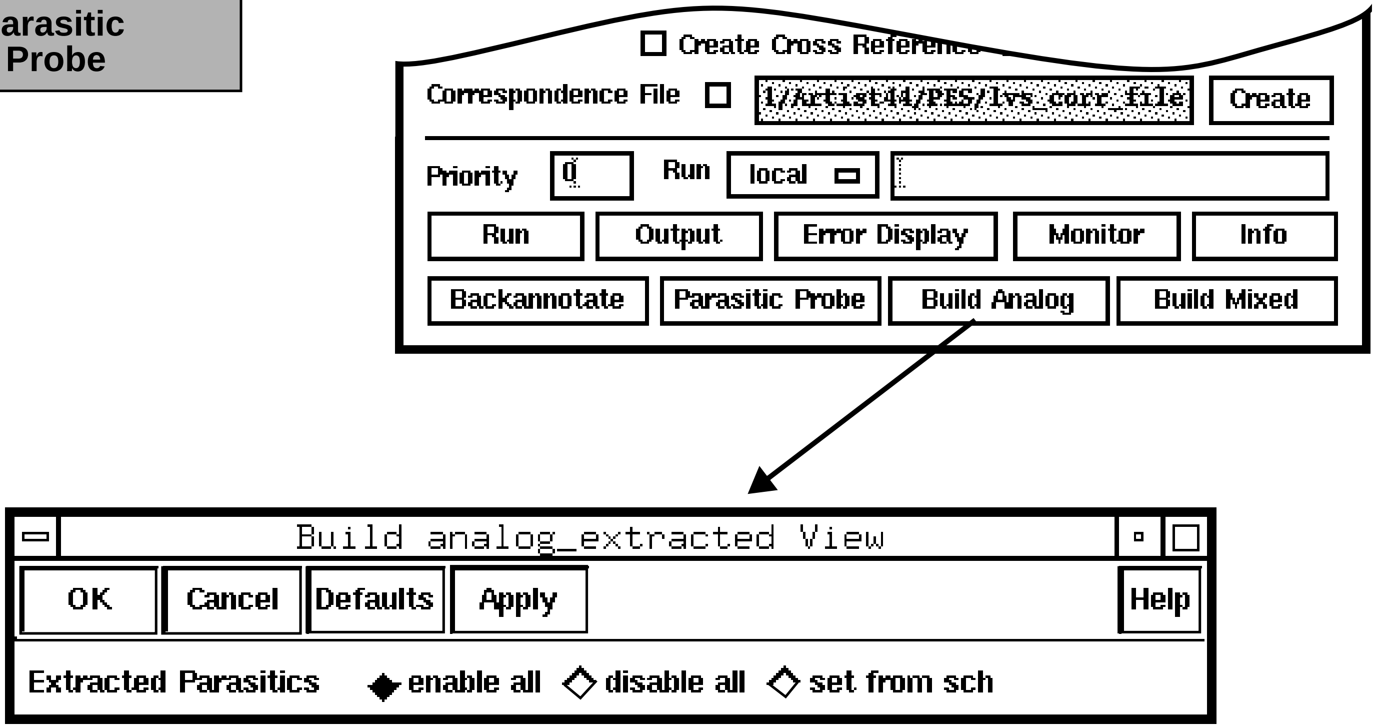
4. Rerun LVS between the schematic and extracted layout with parasitics to establish a mapping between the designs. During LVS with parasitics, information about the parasitics being removed is displayed. This means that the parasitics are removed for the purposes of the LVS netlist comparison. The parasitics are still available for simulation.

Building *analog_extracted*



Do not close the LVS form, but you can iconify it.

Bottom of LVS form:



□ Create Cross Reference

Correspondence File □ 1/Artist44/PES/lvs_corr_file

Priority 0 Run local

Extracted Parasitics ◆ enable all ◇ disable all ◇ set from sch

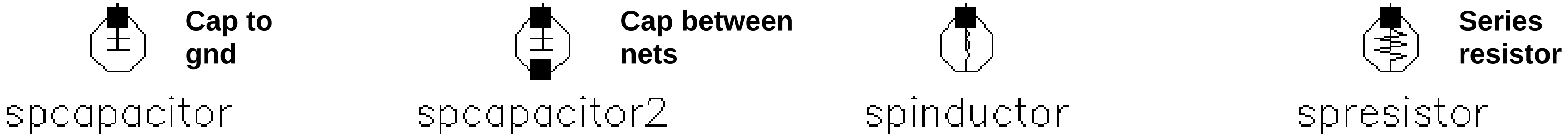
Building *analog_extracted*

Select which parasitics will be used for simulation by defining how the *analog_extracted* view will be built.

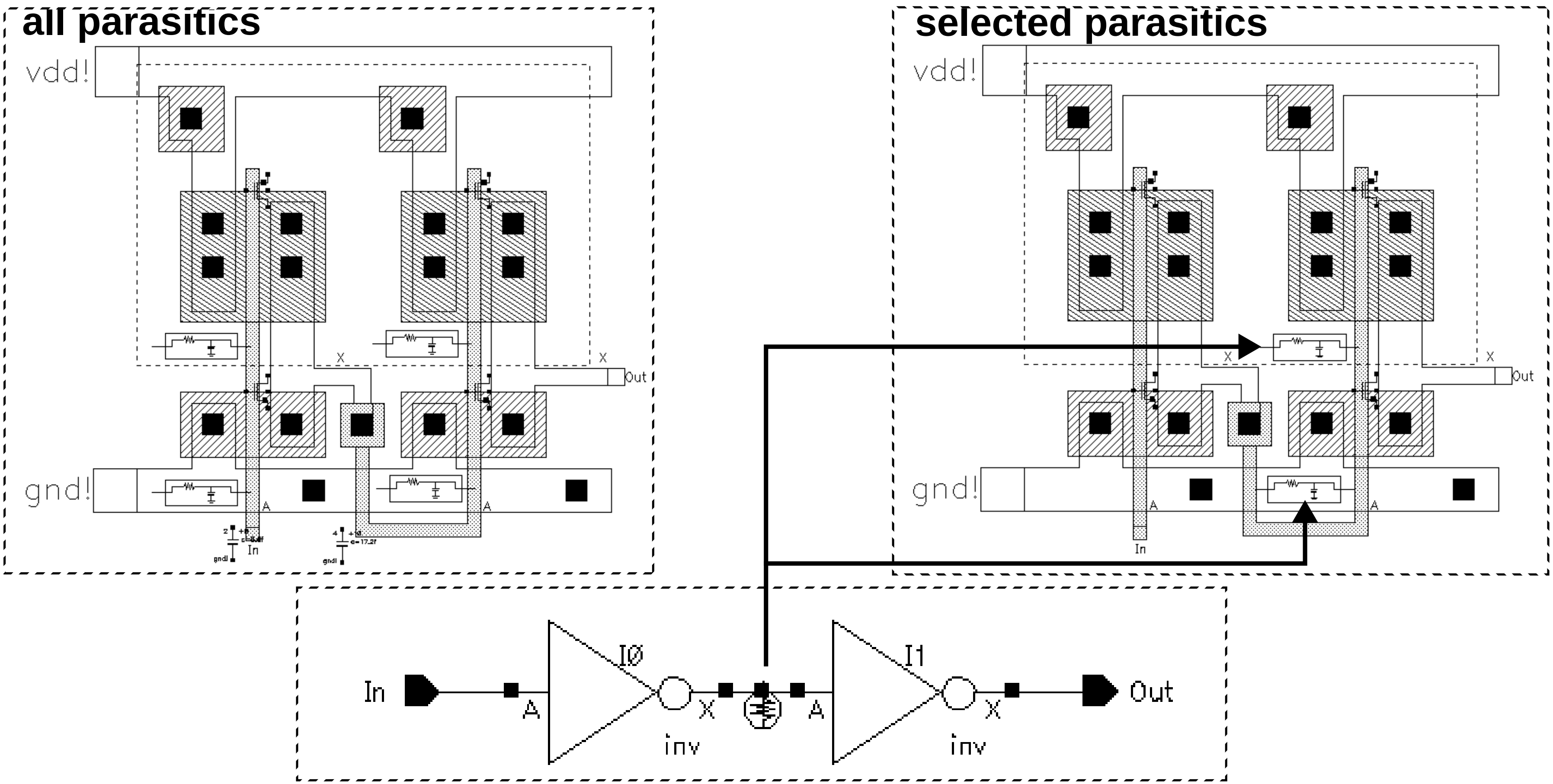
- **enable all** will copy all of the parasitics found during extraction to the *analog_extracted* view. This will result in the greatest accuracy during simulation but also the slowest run time.
- **disable all** will copy only the designed devices found during extraction to the *analog_extracted* view. Choose to use this option to simulate the devices as they were actually built in the layout, using measured device sizes.
- **set from sch** selects which parasitics will be copied to the *analog_extracted* view by placing special symbols on nets in the *schematic* view.

Selective Parasitic Simulation

- Nets are identified in the schematic using *sbaLib* components:



- Enabled by *set from sch* button on Build analog_extracted View form



Selective Parasitic Simulation

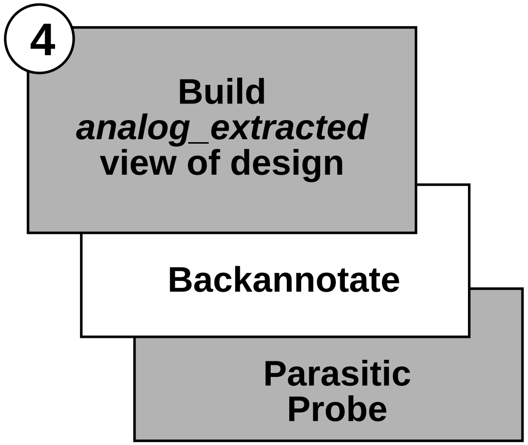
In many cases, only the parasitics on a few critical nets are used during simulation. To use selective parasitics, put special components on specific schematic nets to tell the program to keep the parasitics on these nets. This can significantly reduce the time needed to run a parasitic simulation while keeping the critical information.

The components are available in a library called *sbaLib*, or create your own. A single terminal component with the componentName *spresistor* can be attached to nets to keep parasitic resistors. Note that parasitic resistors are components that have been defined as *presistors* in the CDF. A *presistor* component can include capacitance information as well as resistance information.

The other *sbaLib* components are *spcapacitor*, *spcapacitor2*, and *spinductor*. The components *spcapacitor* and *spinductor* work just like *spresistor*, while *spcapacitor2* is a two-pin component that tells the program to keep all parasitic capacitors between two specific nets. Find *sbaLib* in this directory:

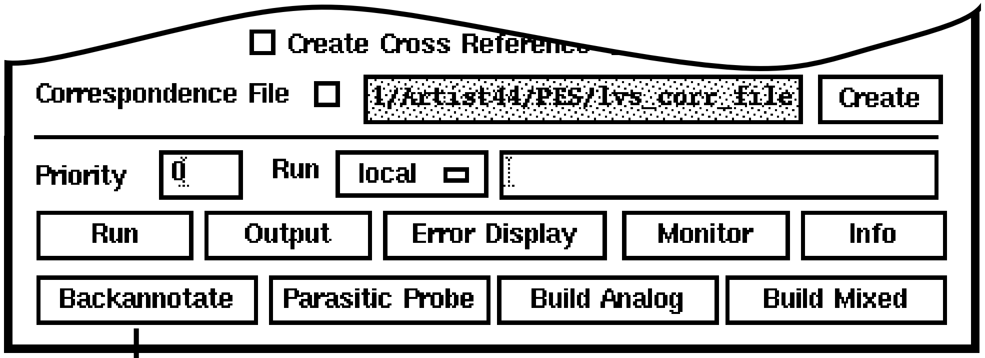
```
        <inst_dir>/tools/dfII/etc/cdslib/artist
```

Backannotation



Bottom of LVS form:

Do not close the LVS form, but you can iconify it.

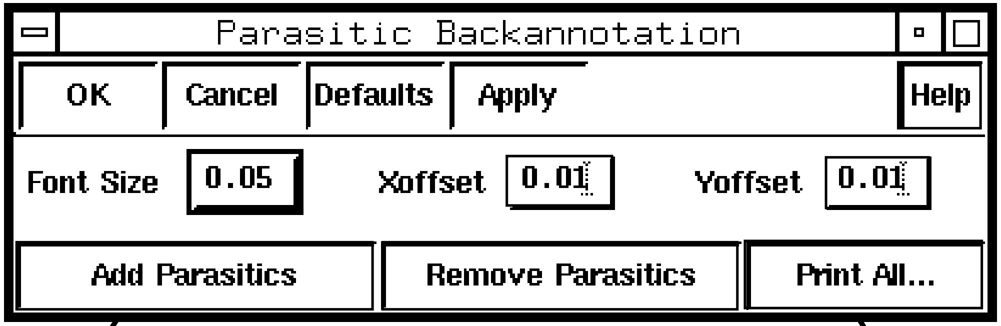


☐ Create Cross Referenc

Correspondence File ☐ 1/Artist44/PES/lvs_corr_file

Priority 0

Run local

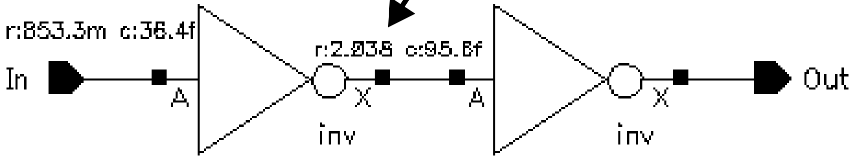


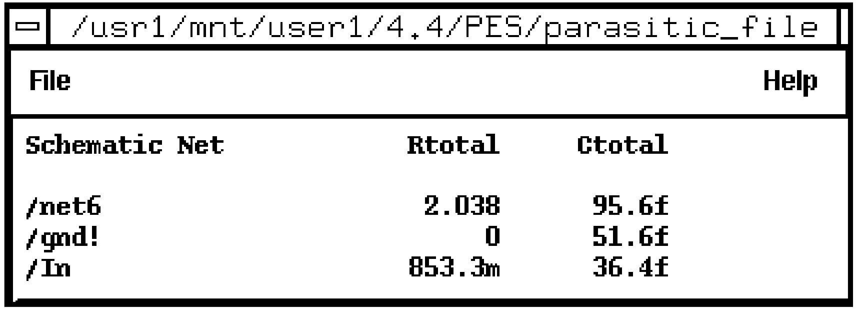
Parasitic Backannotation

Font Size 0.05

Xoffset 0.01

Yoffset 0.01





/usr1/mnt/user1/4.4/PES/parasitic_file

```
Schematic Net                 Rtotal     Ctotal

/net6                          2.038      95.6f
/gnd!                              0      51.6f
/In                          853.3m      36.4f
```

Backannotation

Parasitic backannotation can be invoked anytime after successfully completing *LVS*.

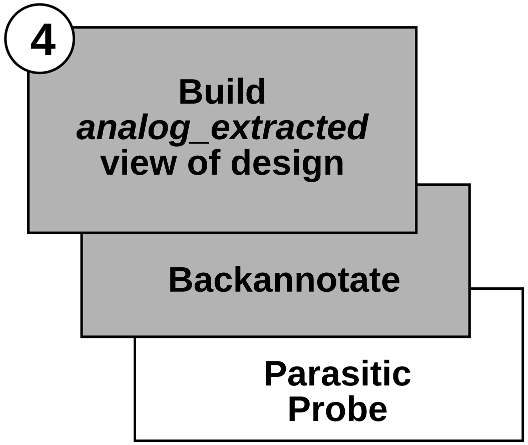
During extraction, the system determines that parasitic devices exist between pins in the layout. During backannotation, the system sums all the parasitic quantities of parasitic networks between pins in the extracted layout view. It then labels the corresponding wires in the schematic window with the total resistance and capacitance between those points.

Backannotation gives instance specific results for hierarchical designs. To see the parasitics for a specific cell, push into it and add the parasitics for that particular instance. Since the information is taken from the extracted view, it is possible for different instances of the same cell to have different parasitics.

Caution

Backannotation is for documentation purposes only. The resistance and capacitance values that are shown above are lumped values for the entire net. This is useful for finding excessive parasitics on critical signals in the schematic design without being familiar with the physical design. To see the effects of the parasitics on circuit performance, use parasitic simulation.

Parasitic Probing



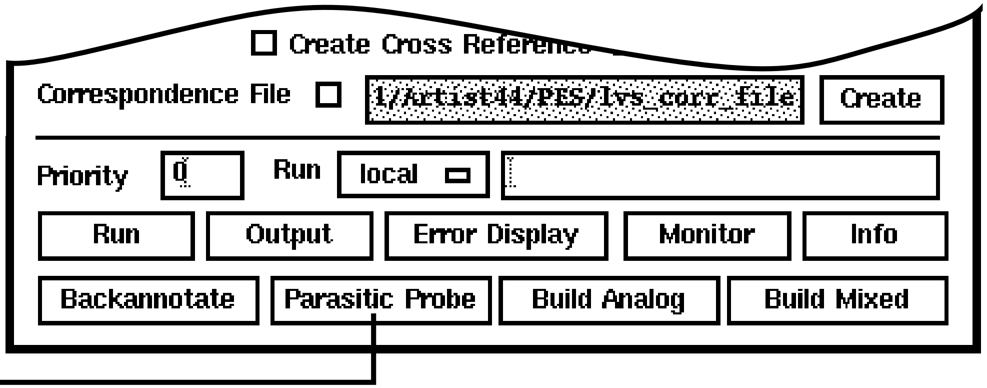
4

Build *analog_extracted* view of design

Backannotate

Parasitic Probe

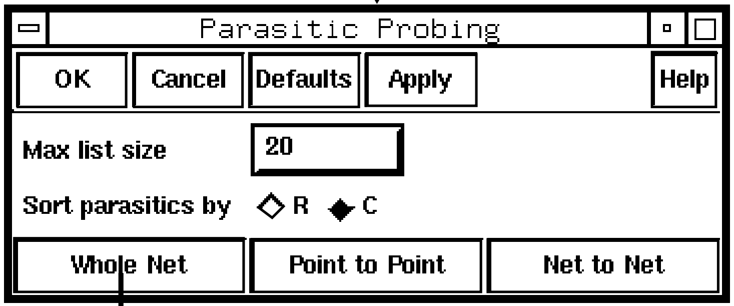
Bottom of LVS form:



□ Create Cross Reference

Correspondence File □ 1/Artist44/PES/lvs_corr_file [Create]

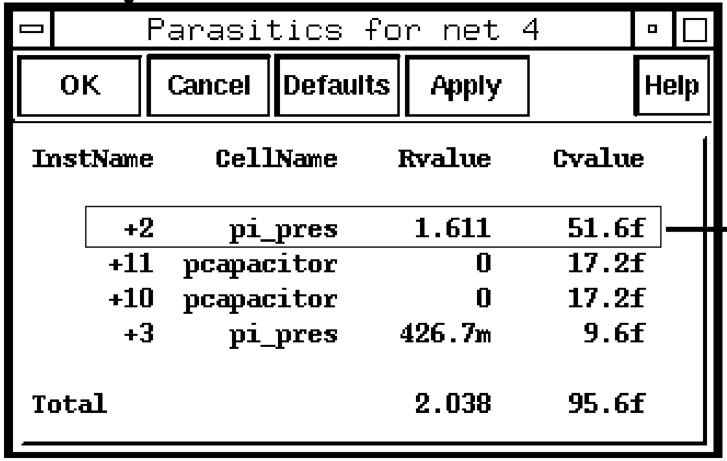
Priority 0 Run local



Parasitic Probing

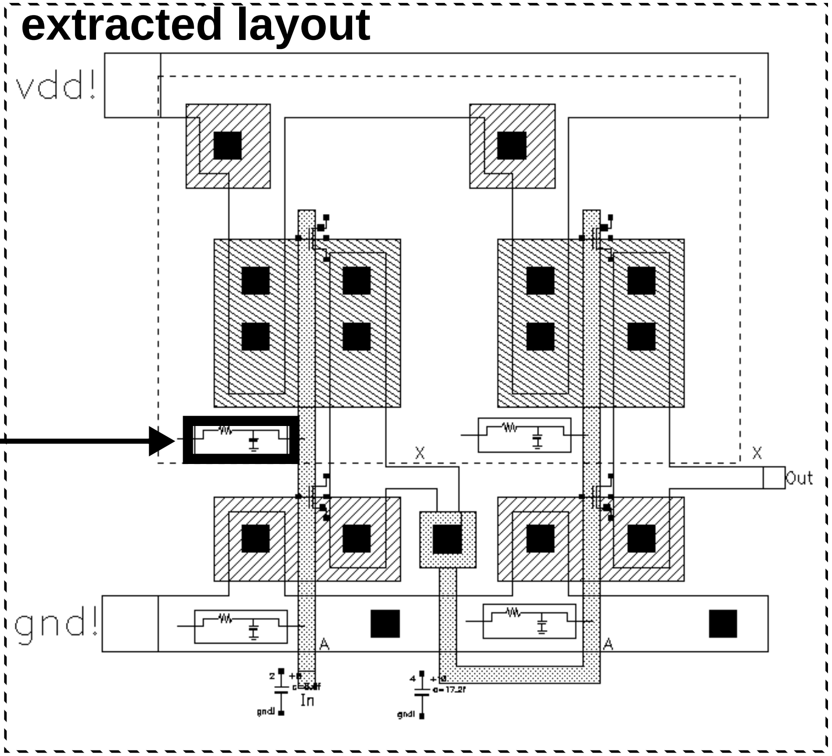
Max list size 20

Sort parasitics by ◇ R ◆ C



Parasitics for net 4

```
InstName     CellName      Rvalue      Cvalue

    +2        pi_pres       1.611       51.6f
   +11     pcapacitor           0       17.2f
   +10     pcapacitor           0       17.2f
    +3        pi_pres     426.7m        9.6f

Total                       2.038       95.6f
```



Parasitic Probing

Use parasitic probing to get detailed information about the parasitics.

- **Whole Net** reports all of the parasitic elements found on a single net.
- **Point to Point** reports all of the parasitic elements found between two points on a net.
- **Net to Net** reports all of the parasitic elements found between two nets (typically capacitors).

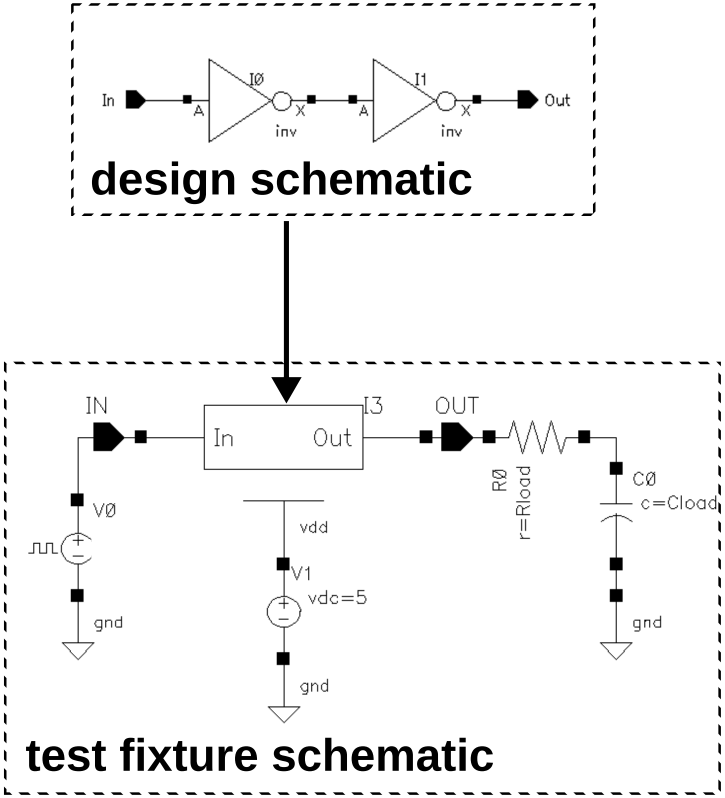
Choose the net in either the schematic or extracted layout, whichever is the current window invoking one of the parasitic probing commands. The net probe is highlighted in that window.

Click on an instance in the parasitic probing report form; it is highlighted in the extracted layout.

To execute another parasitic probe, close the current report window.

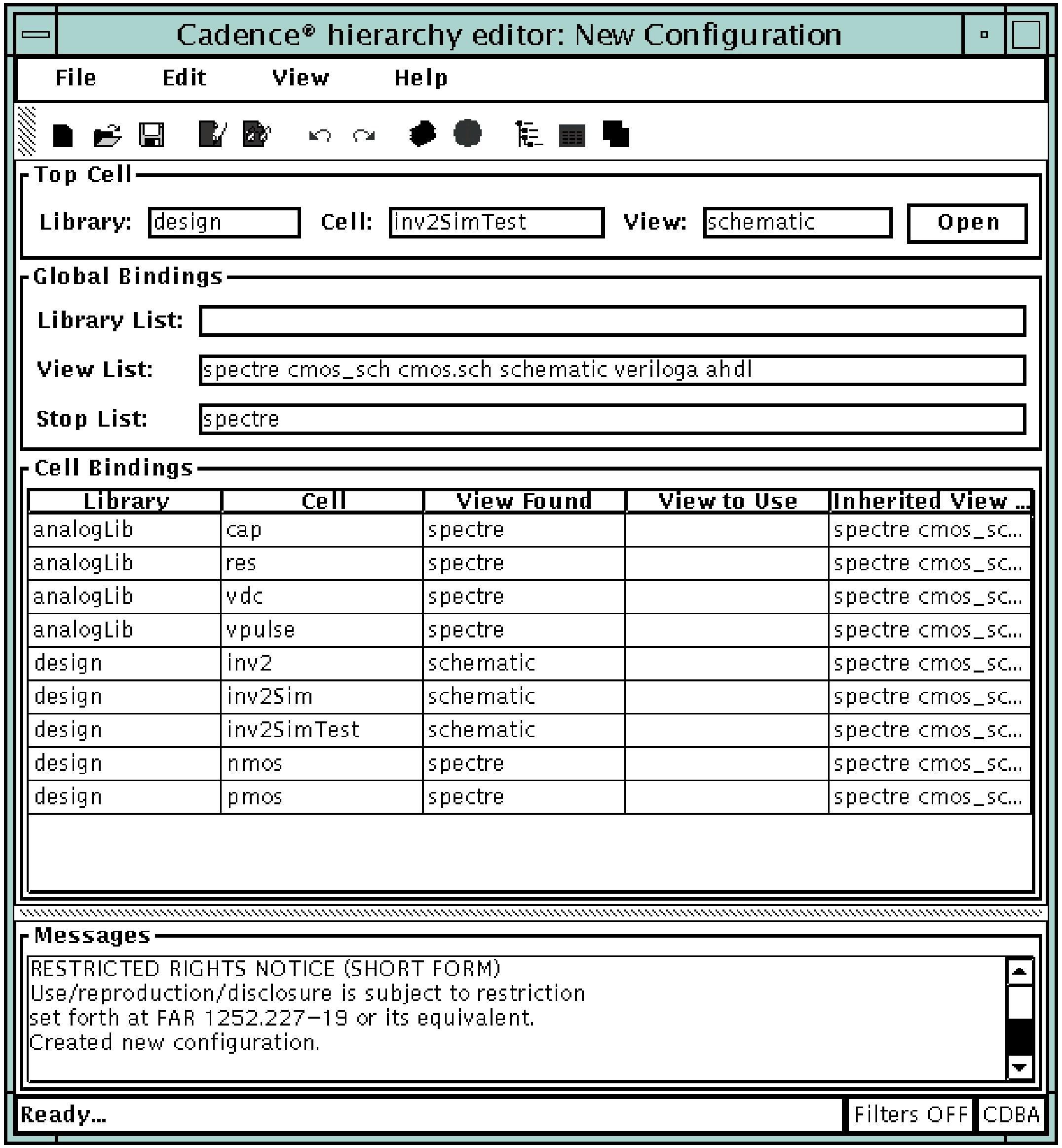
Create Test Fixture Schematic and Configuration

5 Create test fixture schematic & configuration



design schematic

test fixture schematic



Cadence® hierarchy editor: New Configuration

Top Cell

Library: design Cell: inv2SimTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Cell Bindings

Library	Cell	View Found	View to Use	Inherited View ...
analogLib	cap	spectre		spectre cmos_sc...
analogLib	res	spectre		spectre cmos_sc...
analogLib	vdc	spectre		spectre cmos_sc...
analogLib	vpulse	spectre		spectre cmos_sc...
design	inv2	schematic		spectre cmos_sc...
design	inv2Sim	schematic		spectre cmos_sc...
design	inv2SimTest	schematic		spectre cmos_sc...
design	nmos	spectre		spectre cmos_sc...
design	pmos	spectre		spectre cmos_sc...

Messages

```
RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Created new configuration.
```

Ready... Filters OFF CDBA

Create Test Fixture Schematic and Configuration

1. Create a *schematic* cellview, using a symbol for the design and appropriate sources and loads for simulation.

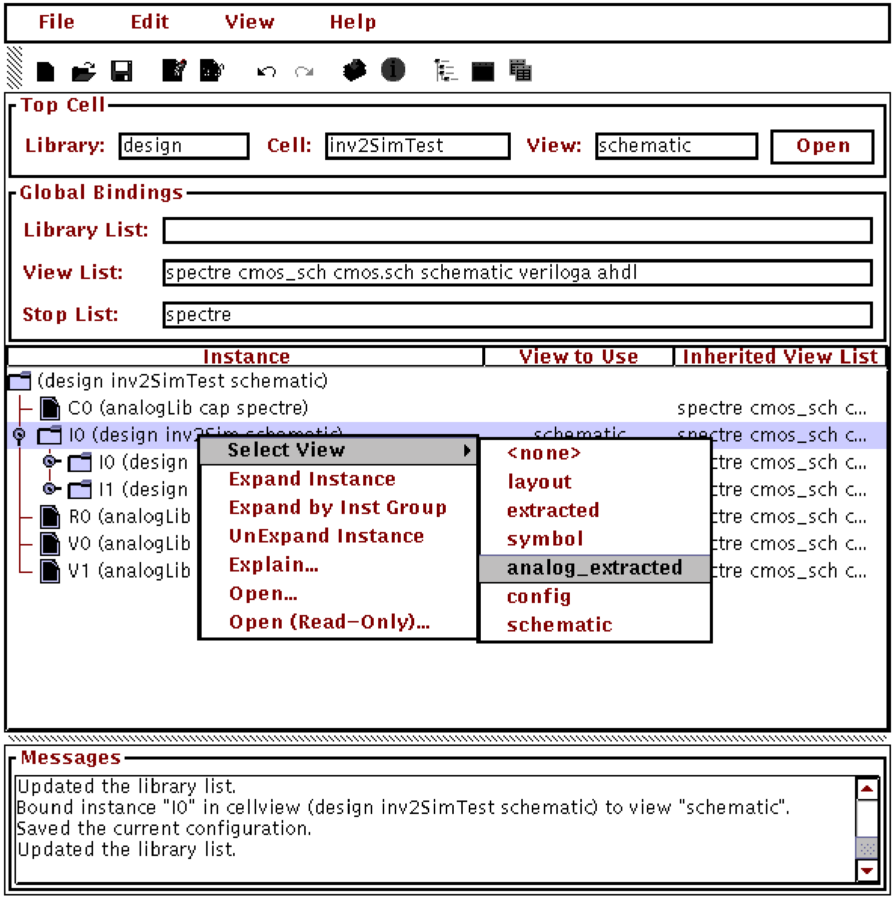
2. Create a *config* cellview.

   — Use the template for the simulator you are using.

   — Use the test fixture schematic as the top cell.

Choose Views in Configuration

6 Choose views in configuration (Netlist with or without Parasitics)



Top Cell

Library: design Cell: inv2SimTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

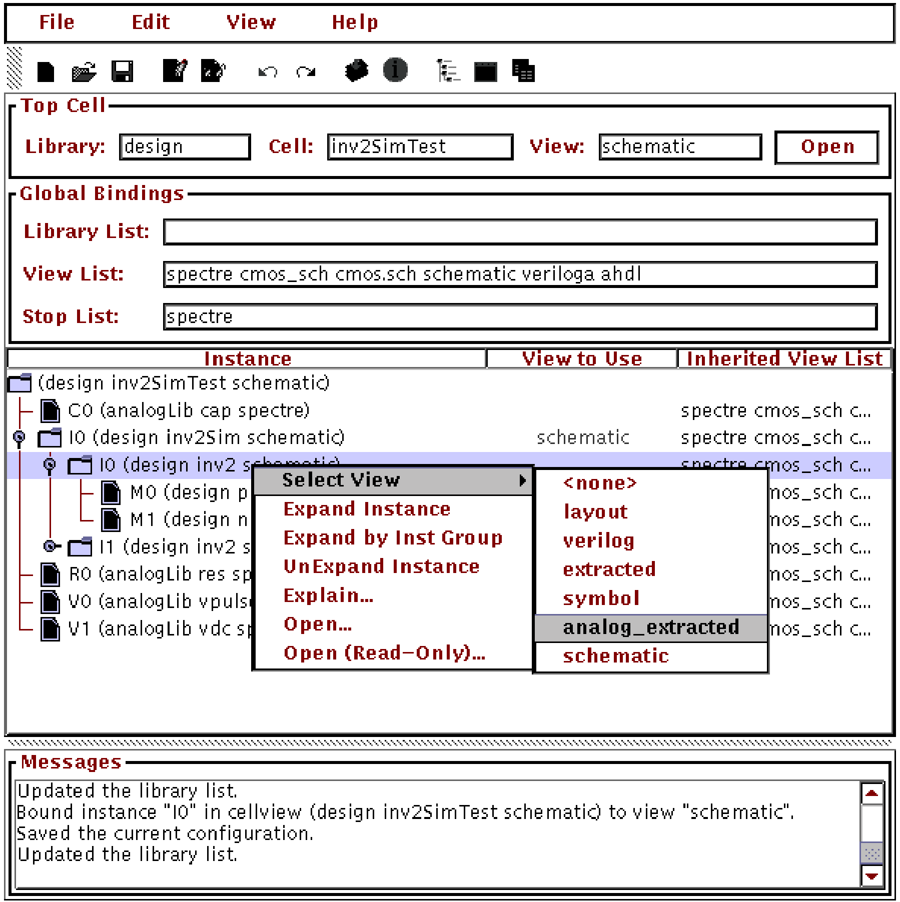
Stop List: spectre

Instance	View to Use	Inherited View List
(design inv2SimTest schematic)		
C0 (analogLib cap spectre)		spectre cmos_sch c...
I0 (design inv2Sim schematic)	schematic	spectre cmos_sch c...
I0 (design		tre cmos_sch c...
I1 (design		tre cmos_sch c...
R0 (analogLib		tre cmos_sch c...
V0 (analogLib		tre cmos_sch c...
V1 (analogLib		tre cmos_sch c...

Messages

```
Updated the library list.
Bound instance "I0" in cellview (design inv2SimTest schematic) to view "schematic".
Saved the current configuration.
Updated the library list.
```

Sets View to analog_extracted



Top Cell

Library: design Cell: inv2SimTest View: schematic

Global Bindings

Library List:

View List: spectre cmos_sch cmos.sch schematic veriloga ahdl

Stop List: spectre

Instance	View to Use	Inherited View List
(design inv2SimTest schematic)		
C0 (analogLib cap spectre)		spectre cmos_sch c...
I0 (design inv2Sim schematic)	schematic	spectre cmos_sch c...
I0 (design inv2 schematic)		spectre cmos_sch c...
M0 (design p		mos_sch c...
M1 (design n		mos_sch c...
I1 (design inv2 s		mos_sch c...
R0 (analogLib res sp		mos_sch c...
V0 (analogLib vpuls		mos_sch c...
V1 (analogLib vdc sp		mos_sch c...

Messages

```
Updated the library list.
Bound instance "I0" in cellview (design inv2SimTest schematic) to view "schematic".
Saved the current configuration.
Updated the library list.
```

Sets View to analog_extracted

Choose Views in Configuration

Choose to use parasitic information for:

- The entire design.
- Specific subblocks at any point in the hierarchy of the design.
- A specific net in a specific subblock. Use the selective simulation components to indicate which net and then use the Hierarchy Editor to choose the subblock view.

To set the views for the design:

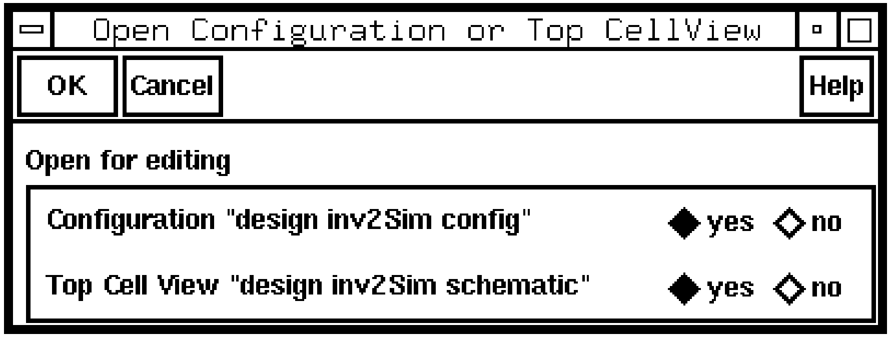
1. In the Hierarchy Editor, use the “tree” icon to expand the hierarchy.
2. For each block that you wish to simulate with parasitics, set the view to *analog_extracted*.
3. Click **Update** and then save the changes.

Running Simulation

7

Run Simulation

1. Open config view of test fixture.



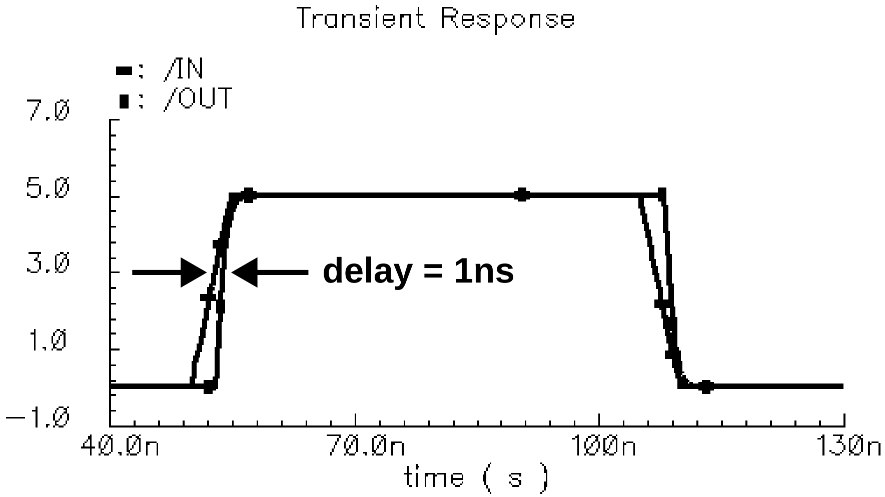
Open Configuration or Top CellView

Open for editing

Configuration "design inv2Sim config" ◆ yes ◇ no

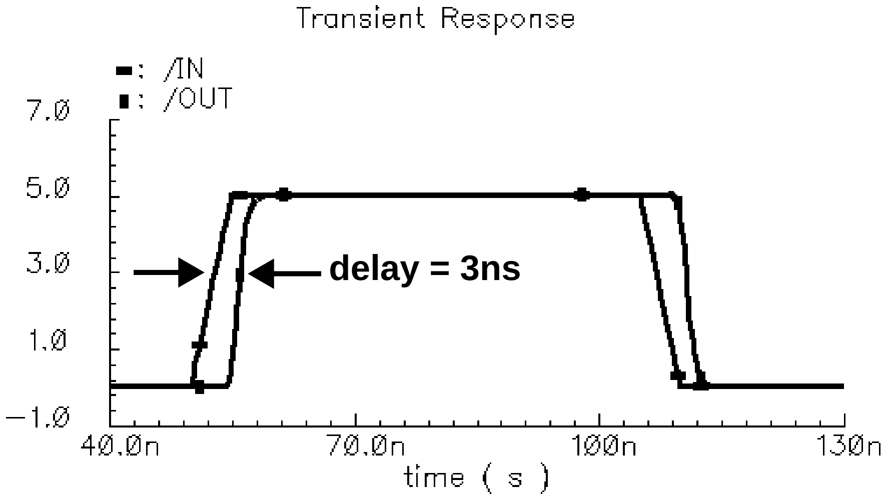
Top Cell View "design inv2Sim schematic" ◆ yes ◇ no

2. Run simulation from the configured schematic view.



delay = 1ns

Simulation without parasitics



delay = 3ns

Simulation with parasitics

Running Simulation

To run a simulation using the views defined in the configuration, open the design from the *config* view.

When opening a *config* view, the default is to open the *schematic* view in a window without actually opening the *config* view. However, all netlisting is being controlled by the *config*.

For running parasitic simulation, open the *config* view as well as the *schematic* view to easily change the configuration views as needed.

Start simulation from the *schematic* window.

Also, start simulation from the CIW, selecting
Tools—Mixed Signal Environment—Simulation.

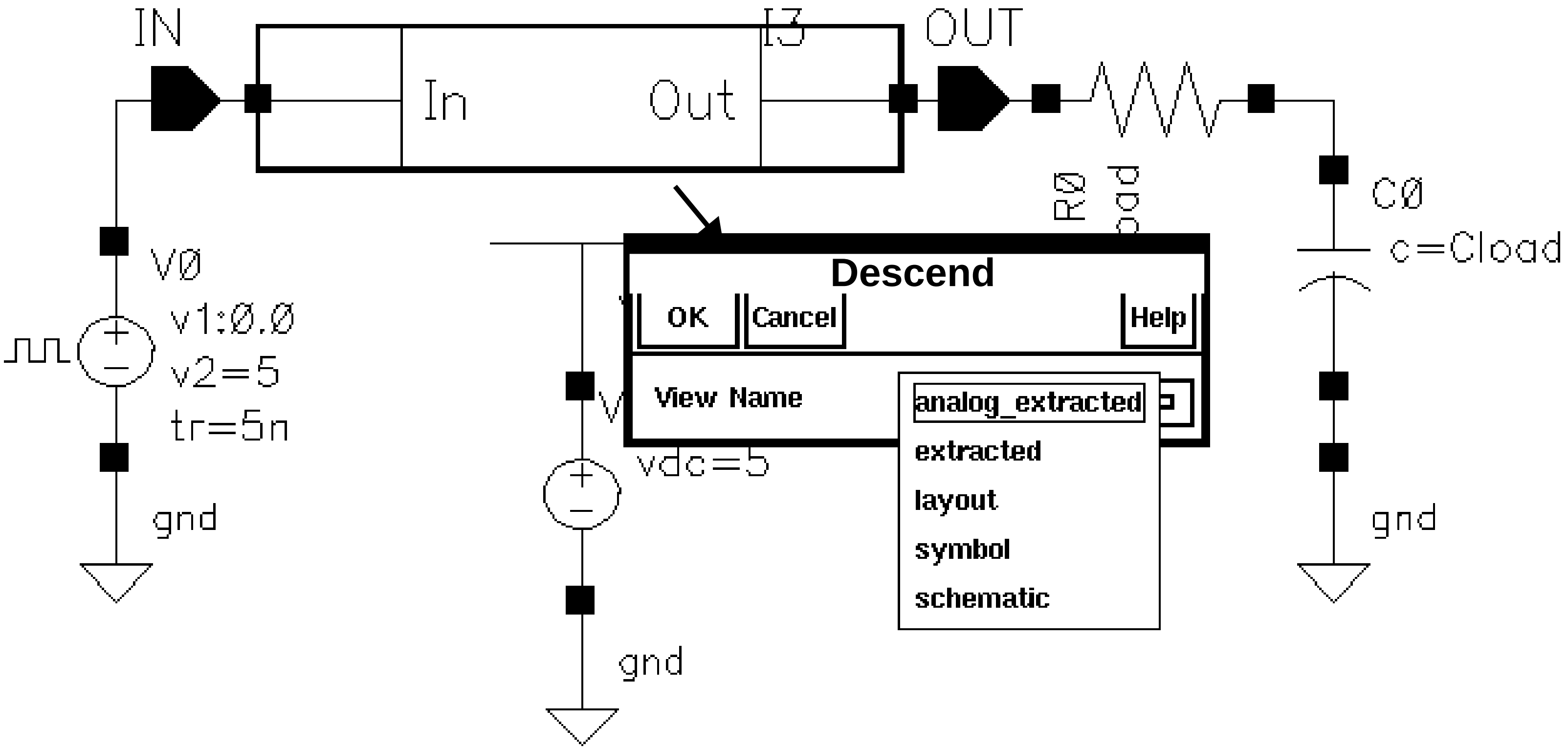
Waveform Analysis

8

Schematic Waveform Analysis

Extracted Waveform Analysis

You can choose waveforms in either the schematic or extracted layout:



View Name analog_extracted

Waveform Analysis

In parasitic simulation, the netlist is derived based on the views chosen in the configuration.

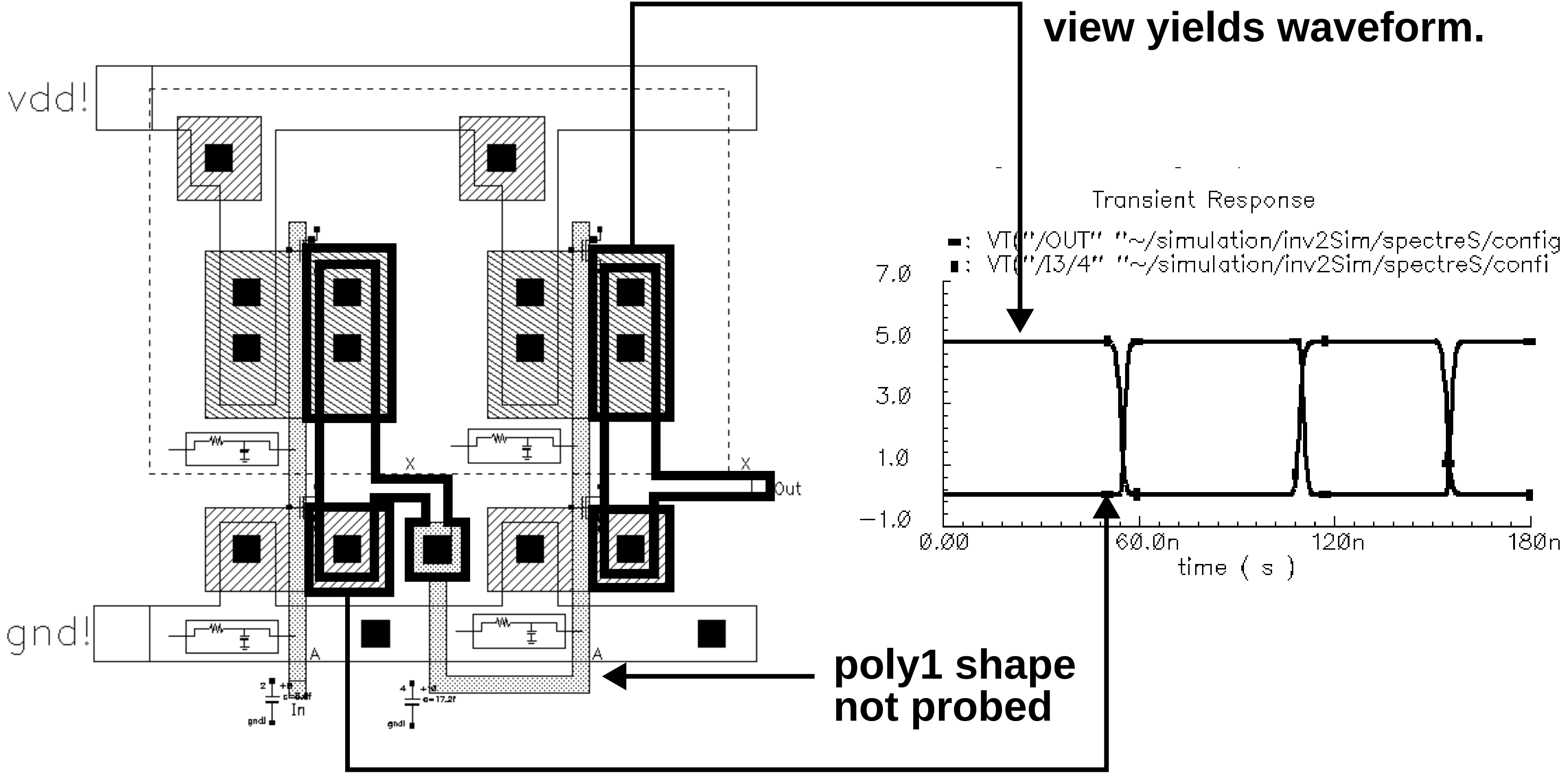
When descending into an instance from the test fixture schematic, choose from a list of views to descend into. By default, the first view offered will be the one defined for simulation in the configuration.

Layout Waveform Analysis

8

Schematic Waveform Anal

Extracted Waveform Analysis


vdd!
gnd!
X
X
Out
A
A
In
Transient Response
VT("/OUT" "~/simulation/inv2Sim/spectreS/config
VT("/I3/4" "~/simulation/inv2Sim/spectreS/confi
7.0
5.0
3.0
1.0
−1.0
0.00
60.0n
120n
180n
time (s)

Probed net in extracted view yields waveform.

poly1 shape not probed

Probe pins or nets in the extracted layout:
Choose pins or layer shapes for which parasitic devices were not extracted.

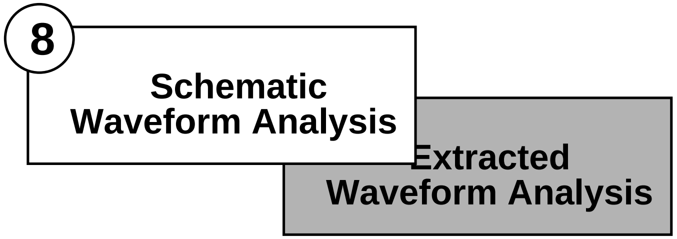
Layout Waveform Analysis

Because the parasitic simulation is usually derived from an *analog_extracted* view, this is typically the first view offered when descending into an instance to probe for waveform information.

This view is often conceptually easier to use for designers familiar with the physical layout. The simulation netlist is mostly generated from the physical layout so points in the extracted layout view map directly to the final simulation netlist. Nodes in the extracted layout represent nodes in the final simulation netlist.

Probe only points that have corresponding net or pin information in the simulation netlist. If extracted resistances exist, the shapes that were converted to resistor networks no longer have any connectivity information associated with them. These shapes will not yield any waveform information. Probe pins or vias attached to these shapes. In this design, resistance was extracted for shapes on layer *poly1*, so those shapes cannot be probed. Probe shapes on layers *metal1*, *ndiff* or *pdiff* because resistance was not extracted for those shapes.

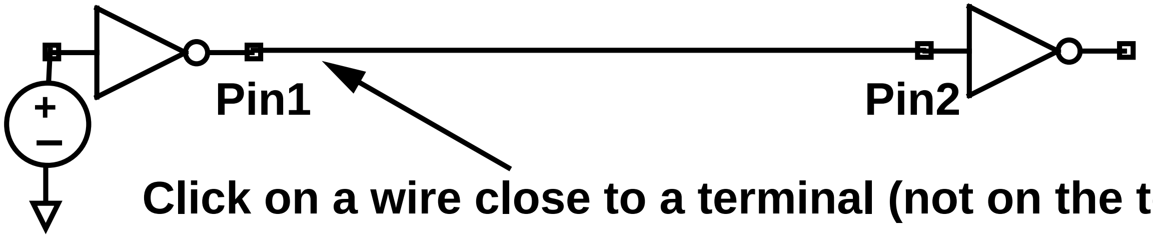
Schematic Waveform Analysis


8

Schematic Waveform Analysis

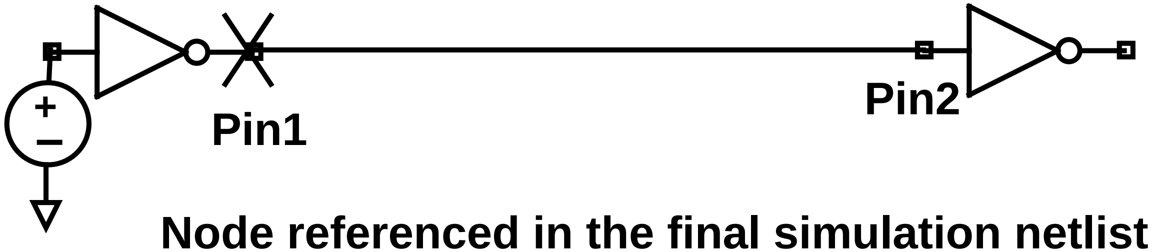
Extracted Waveform Analysis

Use the schematic as a representation of the final netlist:

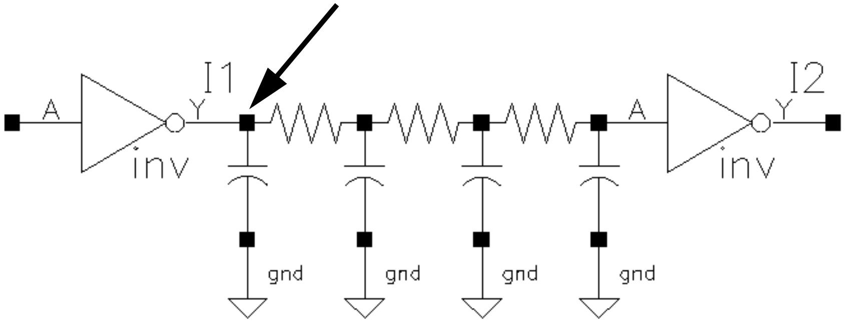


Click on a wire close to a terminal (not on the terminal)

An X in the schematic indicates the corresponding node probed in the final netlist:



Node referenced in the final simulation netlist:



Schematic Waveform Analysis

In parasitic simulation, the netlist is mostly derived from the extracted layout view, but can use the schematic window to probe nodes in the final netlist. For designers not familiar with the physical layout, it is often more relevant to simulate a parasitic netlist and probe in the schematic window, rather than trying to find and probe signals in an unfamiliar layout. In this case, when descending, choose the *schematic* view instead of the *analog_extracted* view.

When probing in a hierarchical design, probe at any level, as long as the probe is very close to a pin that has a corresponding pin in the extracted layout. In the schematic, probe on wires very close to terminals of devices which have corresponding pins in the extracted layout. An *“X”* appears on the terminal of the device attached to that net in the schematic. The net also highlights as it would for analysis without parasitic devices. To observe the effect of parasitic devices on a signal across a net, choose another point on the same net close to a terminal of another device. The new device must also have a corresponding pin in the extracted layout.

To probe a pin that has a corresponding pin in the extracted layout, descend to lower levels of hierarchy.

When plotting the selected signals using commands in the simulation environment window, waveforms are displayed for the corresponding nodes in the final simulation netlist. Any differences in signals probed at either end of a net in the schematic are attributable to parasitic devices inserted in the final simulation netlist.

Labs

Lab A-1 Simulating a Schematic with Parasitics Using the Diva Layout Flow

Labs

Appendix B: WaveScan Display Tools

Topics in this module

- Overview of the WaveScan tools
- Starting a WaveScan session
- Using the WaveScan Results Browser
- Using the WaveScan Display
- Using the WaveScan Calculator
- Data ranging
- Mixed signal waveform plots
- Accelerator Keys

Terms and Definitions

WaveScan	Standalone waveform display tool.
standalone tool	Software tool independent of other operating environments.
.psf	Spectre output file or file containing data
card view	A set of waveform plots selected by rotating a deck.
simulator host	Software tool such as Spectre, cdsSpice, etc. used for simulation.
model library	A text file having model description used by the simulator host.
stimulus template	A user interface used to establish signals used in simulation.
netlist	A textual description of a schematic used by the simulator host.
Waveform Window	A graphical interface used to plot simulation data.
direct plot	User command used for ‘special’ plots to the Waveform Window.
annotating	Process of displaying data back to another window or schematic.

WaveScan Features

An alternate waveform display tool is provided with the IC50 installation of the Analog Design Environment. WaveScan is a new waveform display tool for viewing simulation results.

- Has attractive features preferred by some over the Waveform Window.
- WaveScan is standalone; **it is not integrated** into ADE 5.0
- WaveScan works with the WaveScan Results Browser
- WaveScan works with the WaveScan Calculator
- WaveScan reads and displays **.psf**, which is the Spectre output data.
- Also reads digital wsf, sst2 data
- User interface is Java based; Data access is C++
- Start by typing: **wavescan** in a UNIX window
- WaveScan options executable in unix.

WaveScan Features

WaveScan is a new waveform display tool with attractive features. It features strip, composite, subwindows, and “card” view. It has data ranging, in which data is displayed as specified. WaveScan plots digital signals, including bus data.

WaveScan Options:

These options are arguments to the executable **wavescan** command in unix.

-datadir: Specifies the data directory to be opened on startup.

-graphtemplatefile mygraph: Specifies the graph template file to be used. The default value is none.

-h or -help: Displays information on how to run WaveScan.

-readstate true | false: Specifies whether WaveScan should read the saved state file at startup. Default value is true.

-statefile statefile: Specifies the state file to be used. The default state file is wavescan.xml.

-version: Provides the Cadence release version.

-writestate true | false: Specifies whether WaveScan should write a state file when it exits. Default value is true.

-V: Displays the version number for WaveScan.

-W: Displays the sub-version number for WaveScan

Plotting Features

Features of WaveScan include:

- rectangular, polar, admittance, impedance, real vs. imaginary
- swept data:
- Y vs. Y:
- Difference of two signals
- Strip mode, Composite, Subwindows and “Card” Plot
- Buses, displayed as graphical plus numeric
- Hide/reveal objects
- Zoom: X and Y, Pan
- “Active” window: double-click on anything in window and “Edit Attributes” form for it pops up.
- Cursors
- Labels
- Accelerator Keys (also known as bindkeys)

Plotting Features

The above list shows specific plotting features of WaveScan display environment.

Important

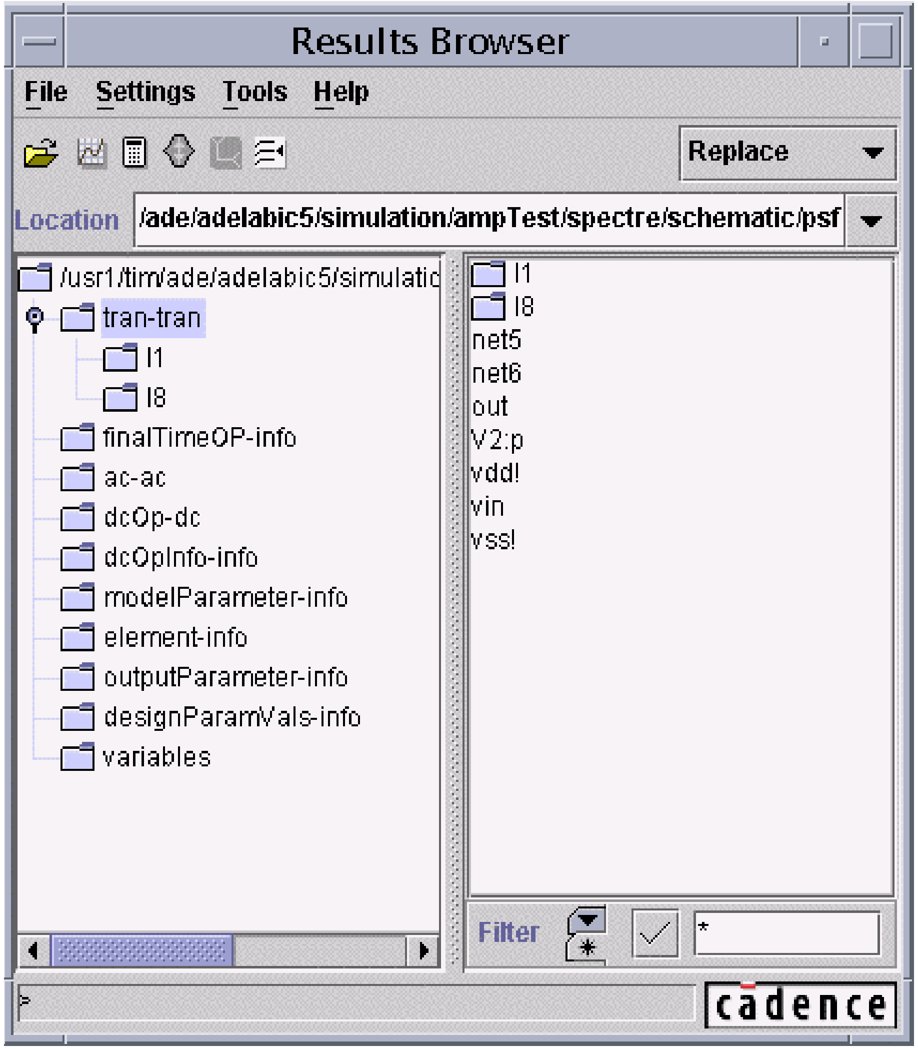
WaveScan is not presently integrated into ADE 5.0. Calculator expressions are not read into the forms by selecting “Get Expression”. WaveScan is not started from the Analog Design Environment. WaveScan is compatible with the .psf data. WaveScan is not opened by selecting a signal with the right mouse button from the Results Browser. Instead, WaveScan has its own Results Browser and Calculator tool.

- The WaveScan graphical user interface is Java based.
- Note that WaveScan is a standalone tool. It can be started at any time from a unix window.
- WaveScan is started by entering the executable wavescan command in a UNIX window.

WaveScan Results Browser

In a UNIX window, enter `wavescan`

The WaveScan Results Browser appears.



Location: /ade/adelabic5/simulation/ampTest/spectre/schematic/psf

- /usr1/tim/ade/adelabic5/simulatic
  - tran-tran
    - I1
    - I8
  - finalTimeOP-info
  - ac-ac
  - dcOp-dc
  - dcOpInfo-info
  - modelParameter-info
  - element-info
  - outputParameter-info
  - designParamVals-info
  - variables

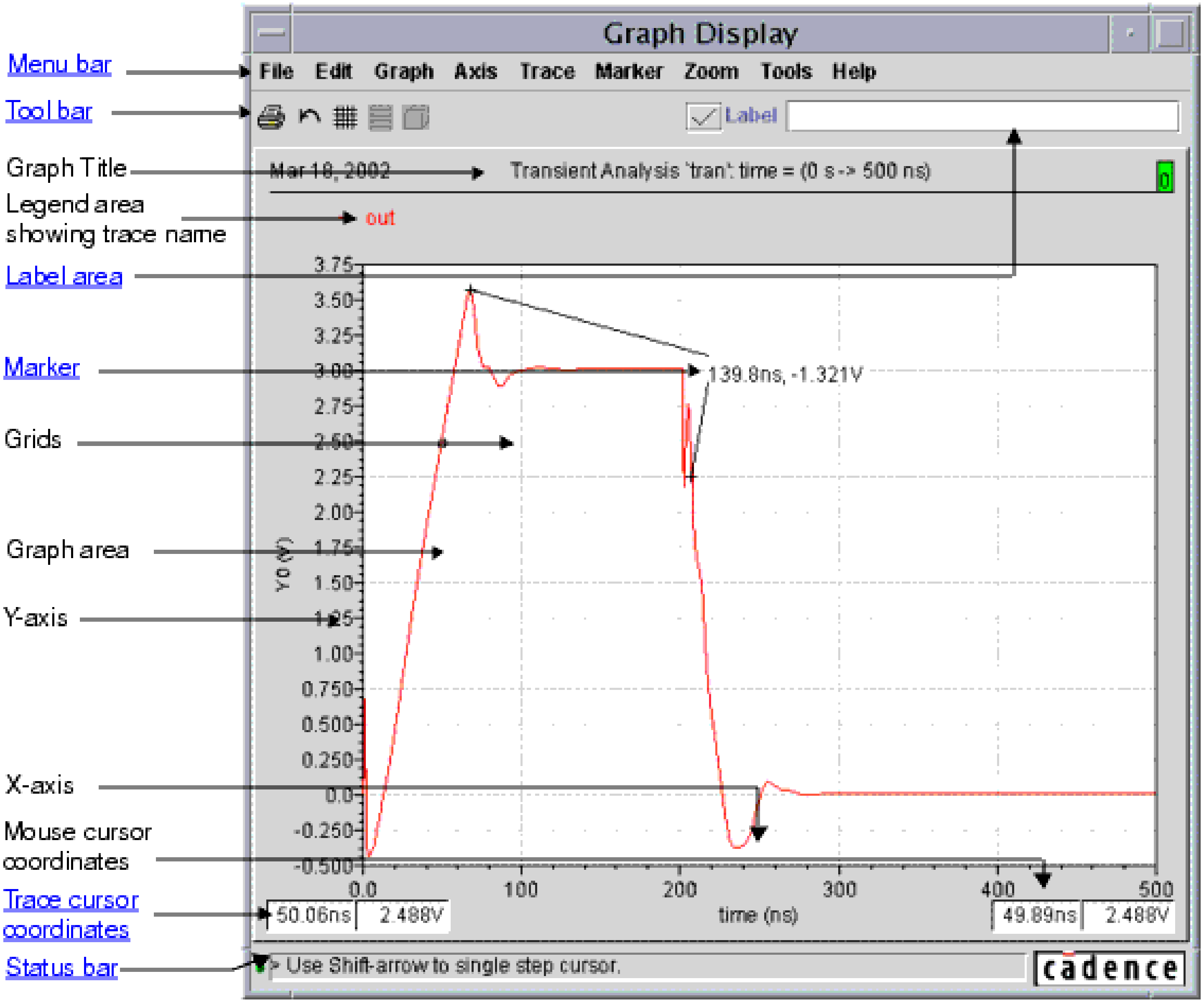
I1
I8
net5
net6
out
V2:p
vdd!
vin
vss!

Filter: *

WaveScan Results Browser

- WaveScan is a standalone environment with its own Results Browser, Waveform Display, and Calculator.
- To start WaveScan, enter `wavescan` in a UNIX window.
- The WaveScan Results Browser appears.
- Click on folders to browse to the results directory with .psf files.
- Click on the node or instance to display data in the WaveScan Waveform Display.

WaveScan Display



Menu bar

Tool bar

Graph Title

Legend area showing trace name

Label area

Marker

Grids

Graph area

Y-axis

X-axis

Mouse cursor coordinates

Trace cursor coordinates

Status bar

Graph Display

Mar 18, 2002 Transient Analysis 'tran': time = (0 s -> 500 ns)

139.8ns, -1.321V

50.06ns 2.488V

49.89ns 2.488V

> Use Shift-arrow to single step cursor.

WaveScan Display

The display shows some of the graphical features of WaveScan. For example, two points are placed on the waveform by the marker to make a differential time and voltage measurement.

The WaveScan display features a mouse cursor and a trace cursor with numerical coordinates. A label area is provided for adding labels without going though an “annotate” menu form.

Important

If you are creating a graph for the first time, it is displayed in a new Graph Display window. If you have a Graph Display window already open, you need to specify the destination for the new graph. You can add a signal to the selected graph (or add a new subwindow). If the trace shares the same unit, it is assigned to the same Y-axis. Otherwise, it is assigned to a new Y-axis. If the selected graph window already has four Y-axes, WaveScan creates the new graph in a new subwindow.

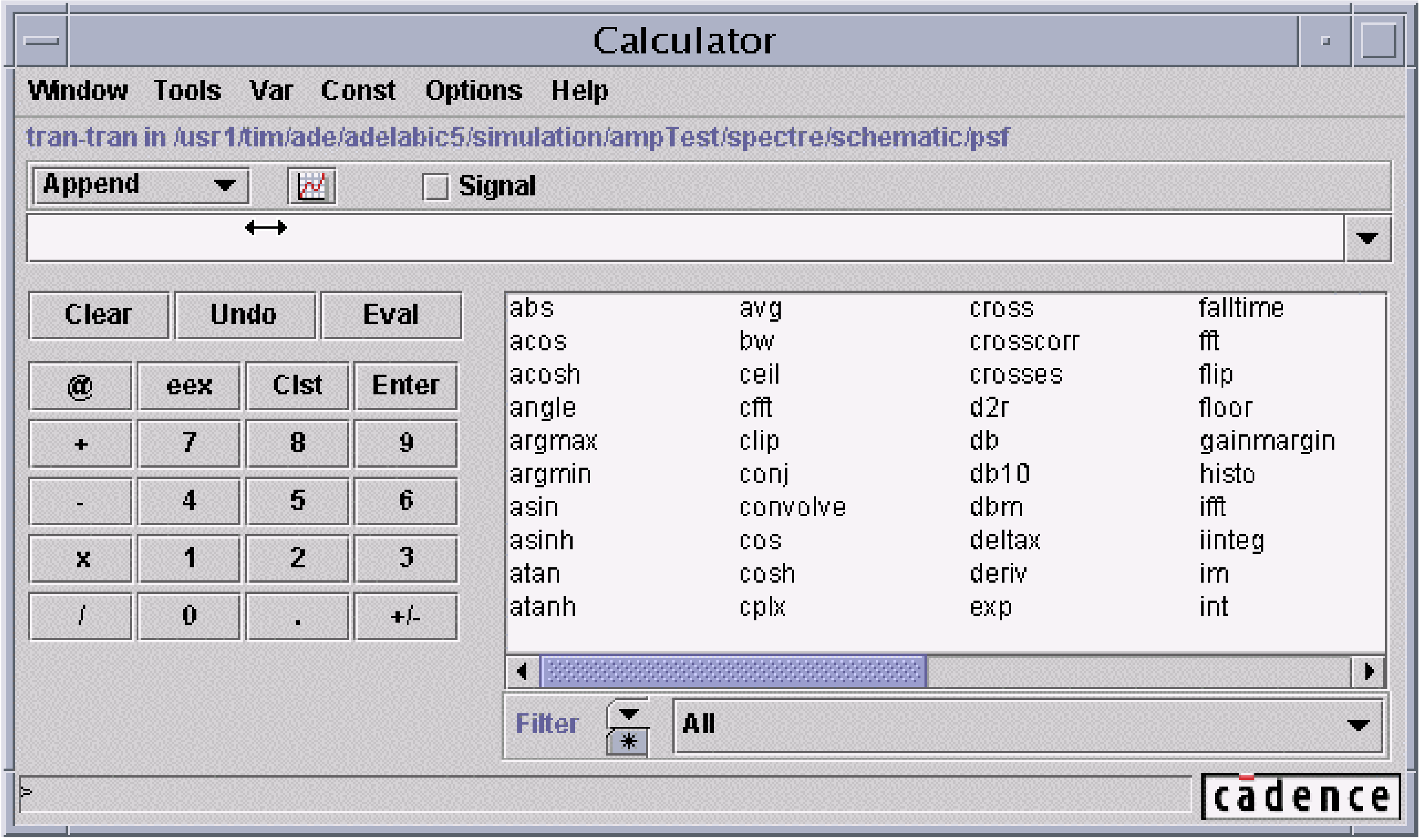
Caution

There is a limitation currently where you cannot plot two signals that have no units on the same axis. This means plots such as gain and phase cannot share the same y-axis.

WaveScan Calculator

In the WaveScan Display, select **Tools—Calculator**

In the WaveScan Results Browser, select **Tools—Calculator**, or select the Calculator icon.



tran-tran in /usr1/tim/ade/adelabic5/simulation/ampTest/spectre/schematic/psf

abs, acos, acosh, angle, argmax, argmin, asin, asinh, atan, atanh

avg, bw, ceil, cfft, clip, conj, convolve, cos, cosh, cplx

cross, crosscorr, crosses, d2r, db, db10, dbm, deltax, deriv, exp

falltime, fft, flip, floor, gainmargin, histo, ifft, iinteg, im, int

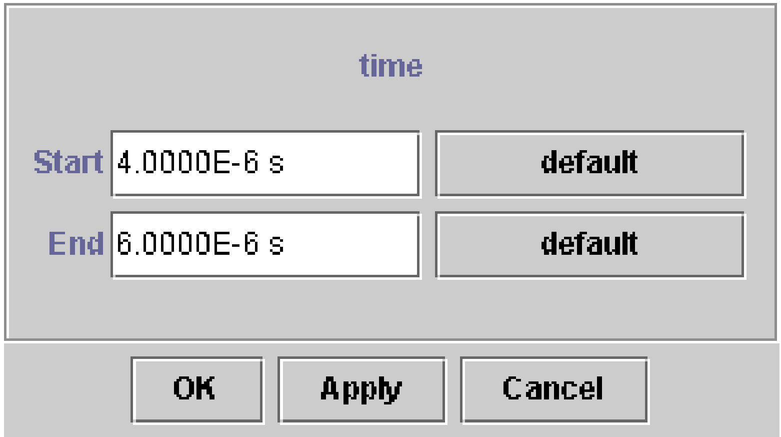
WaveScan Calculator

The WaveScan display environment features a WaveScan Calculator.

The Wavescan Calculator is “somewhat similar” in operation to the Waveform Calculator in the Analog Design Environment.

Data Ranging

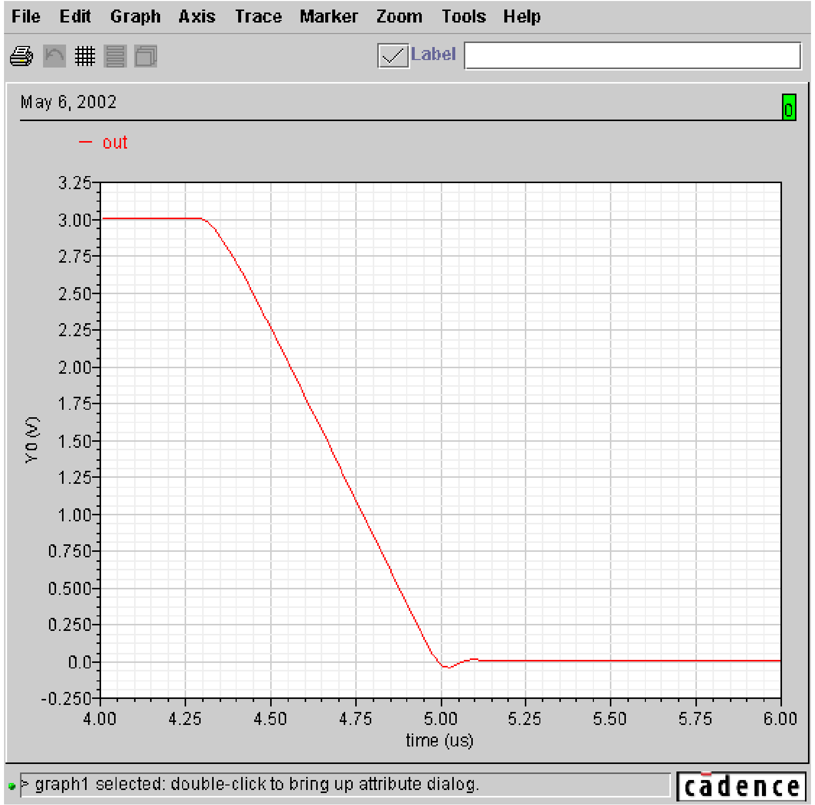
- Can plot a portion of a simulation run, but can only read that portion.
- Cannot scroll outside the portion specified.
- Does not read entire simulation run and only show a portion of it (like AWD). True “range-specific” plotting.
- **Settings—Select Sweep**, or click the Select sweep icon:
- Change time to whatever you want to see



time

Start 4.0000E-6 s

End 6.0000E-6 s



graph1 selected: double-click to bring up attribute dialog.

Data Ranging

The data ranging feature in WaveScan makes it easy to use a very large dataset efficiently by opening just the portion of the dataset that you need. You can specify a particular time range in a transient analysis, and then open the dataset and plot signals over just that range.

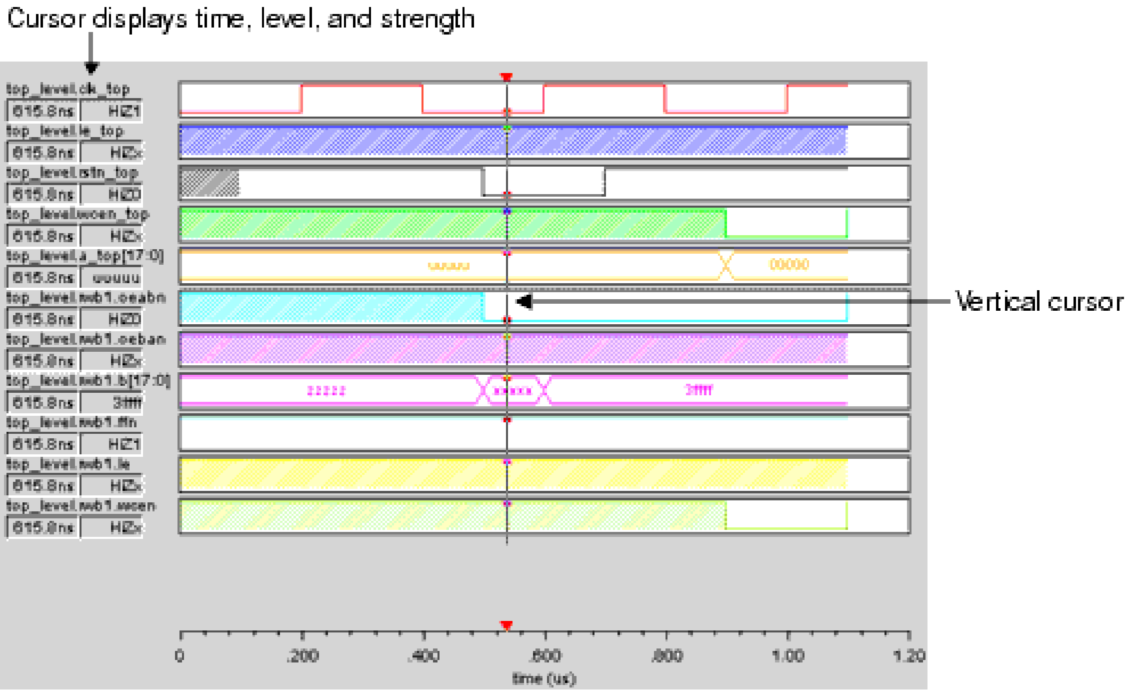
Cannot view anything outside the range you have chosen. You must reset the time to another range before any other data range may be seen.

Caution

Current limitation: you cannot bring ranged data into the calculator for further processing.

Digital and Mixed Signals Plots

- Digital signals are still plotted at top of Waveform Window in strip mode



Cursor displays time, level, and strength

Vertical cursor

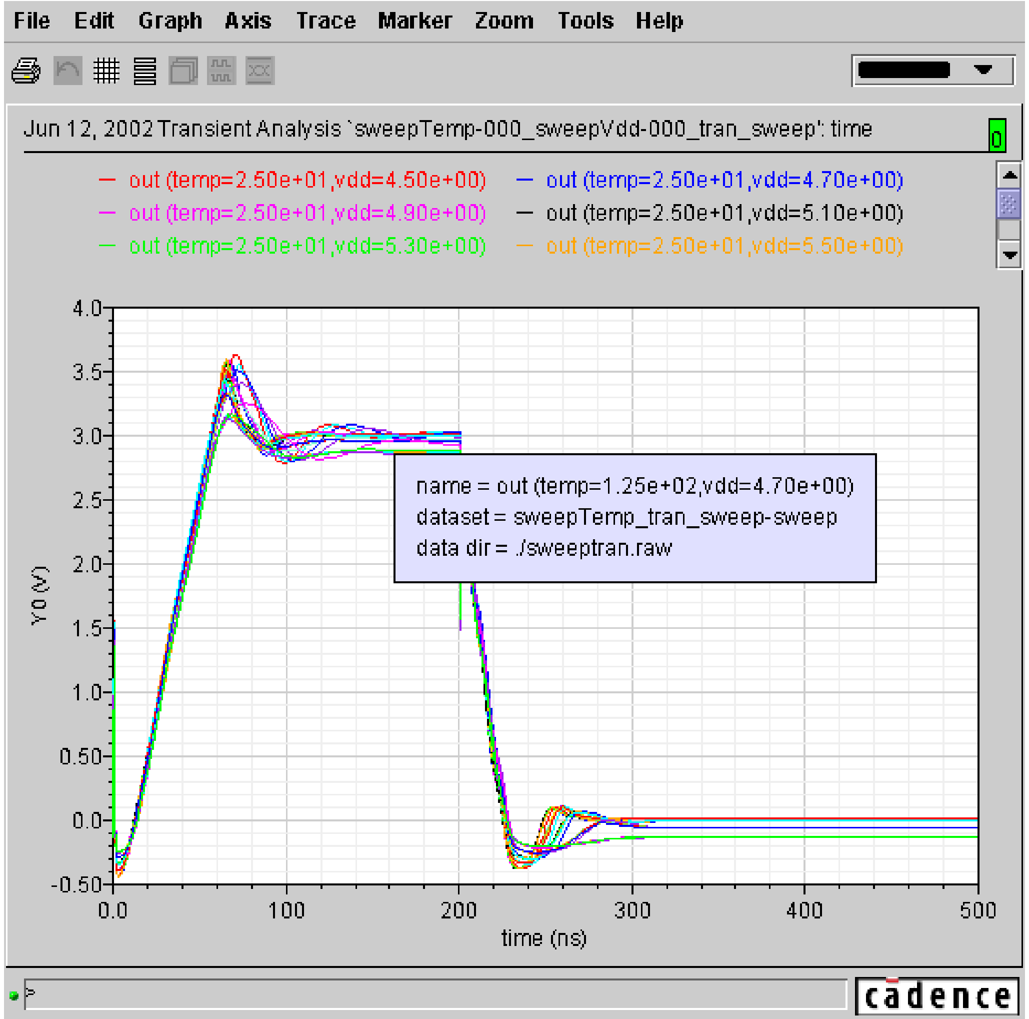
- Create Bus: Select traces then click on **Trace—Bus—Create** (or ☒)
- Expand **Bus—Select Bus** then click on **Trace—Bus—Expand** (or ⎍)
- When performing mixed-signal plotting, analog signals appears below the digital signals.

Digital and Mixed Signals Plots

The window shown above shows how WaveScan displays digital information. The digital information on buses can be expanded. Analog waveforms are displayed under the digital waveforms.

Parametric Plots and the Alt key

1. Select a waveform
2. Press Alt key to get information on that waveform

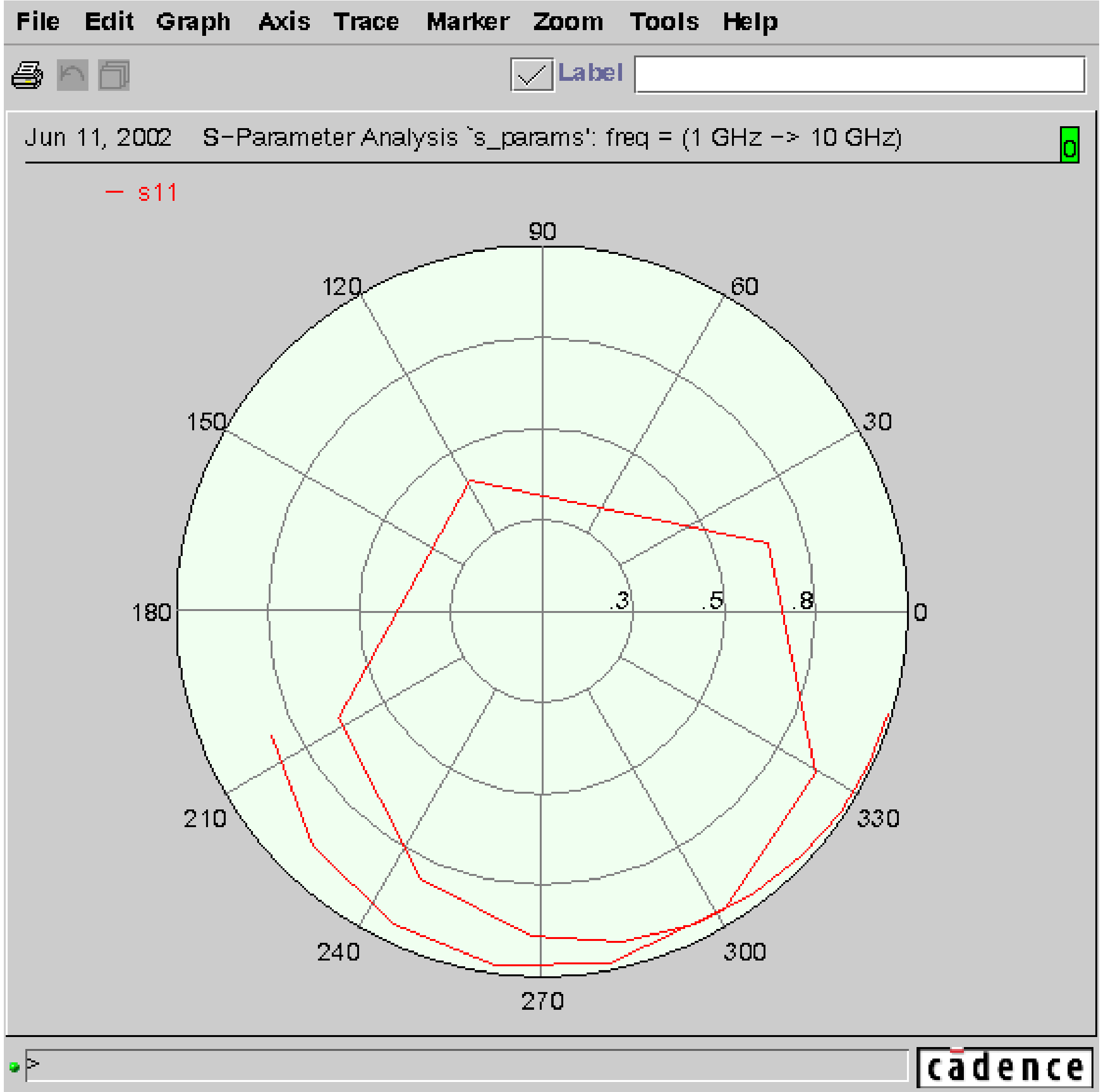

File Edit Graph Axis Trace Marker Zoom Tools Help
Jun 12, 2002 Transient Analysis `sweepTemp-000_sweepVdd-000_tran_sweep': time
0
out (temp=2.50e+01,vdd=4.50e+00)
out (temp=2.50e+01,vdd=4.70e+00)
out (temp=2.50e+01,vdd=4.90e+00)
out (temp=2.50e+01,vdd=5.10e+00)
out (temp=2.50e+01,vdd=5.30e+00)
out (temp=2.50e+01,vdd=5.50e+00)
Y0 (V)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.50
0.0
-0.50
0.0
100
200
300
400
500
time (ns)
cadence

name = out (temp=1.25e+02,vdd=4.70e+00)
dataset = sweepTemp_tran_sweep-sweep
data dir = ./sweeptran.raw

Parametric Displays

Polar Plots

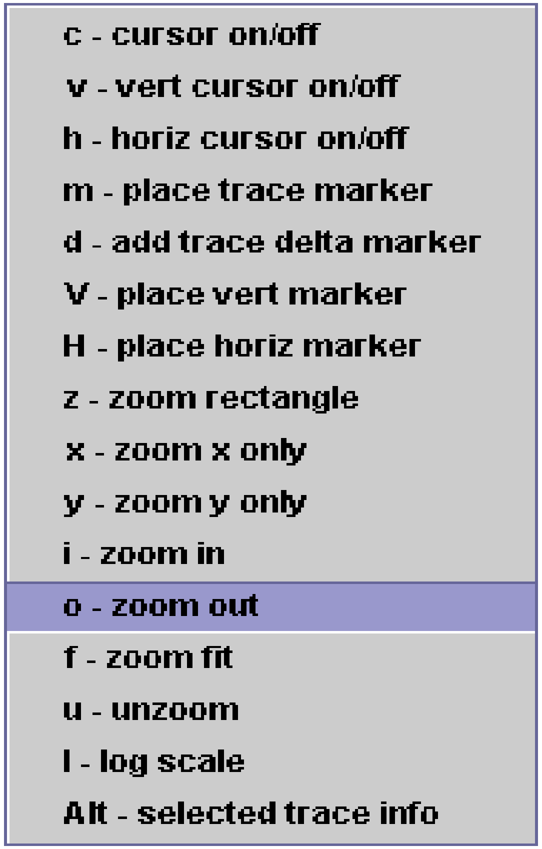
Plotting data using polar coordinates is available in WaveScan. Polar plots include Smith Charts, Impedance, and Admittance diagrams



Jun 11, 2002 S−Parameter Analysis `s_params': freq = (1 GHz −> 10 GHz)

Polar Plots

WaveScan “Accelerator” Keys



c - cursor on/off

v - vert cursor on/off

h - horiz cursor on/off

m - place trace marker

d - add trace delta marker

V - place vert marker

H - place horiz marker

z - zoom rectangle

x - zoom x only

y - zoom y only

i - zoom in

o - zoom out

f - zoom fit

u - unzoom

l - log scale

Alt - selected trace info

WaveScan “Accelerator” keys

In the IC 5.0 version of WaveScan, the Accelerator keys shown above are available. This set of keys cannot be changed or added to in IC 5.0.

Labs

Lab B-1 Using the WaveScan Tool

Labs

Appendix C: Spectre MDL

Topics in this module

- Introduction to Spectre MDL
- Features of Spectre MDL
- Basic language rules and syntax
- Alias measurements
- Built-in functions
- Spectre MDL in the Analog Design Environment
- Limitations

Terms and Definitions

Spectre MDL
Acronym for **Spectre M**easurement **D**escription **L**anguage
Text-based language to run Spectre with measurements.

alias
Commands that control input data to the host simulator.
Commands that run the simulator.

data access commands
Commands that are used to get simulator results.

OCEAN aliases
Abbreviated text that activates specific OCEAN commands.

Introduction to Spectre MDL

- New measurement language in Spectre
- spectremdl is an executable shell script located at
  <install_dir>/tools/mdl/bin
- Can control the Spectre simulator
- Runs in batch mode.
- Provides “.measure” capability
- Has alias measurement feature for easy reuse in other applications.
- To run Spectre MDL type within a UNIX shell: *spectremdl*
  - use the *-design* option to specify a netlist
  - use the *-batch* option to specify an MDL file

Introduction to Spectre MDL

Spectre MDL is a new standalone command language for running Spectre. The language has measurement syntax. As such, a netlist and a MDL file provide a simple method to run and make measurements.

Sample MDL File

```
alias measurement acrun {
       export real dcgain, zerodbcross
       run ac1 // This analysis is defined in the Spectre netlist
       dcgain=db(mag(V(out))/mag(V(net21)))@10
       zerodbcross=cross(sig=db(mag(V(out))/mag(V(net21))),
            dir='fall, n=1, thresh=0, start=0)
     }

alias measurement findoffset{
       run dc1 // This analysis is defined in the Spectre netlist
       export real vio=abs(V(out)-V(inp))
     }

run acrun
run findoffset
```

Sample MDL File

The sample file above shows some of the syntax required for measurements using Spectre MDL.

Important

The “export” statement optional. If it’s just a local variable, you do not need to “export” it. The export is required to save the scalar data to the psf database.

Basic Rules

- Spectre rules still apply!
- Ignores white space
- Comments:
  - // for single line comments
  - /* for block

    comments */
- Identifiers:

real An_Identifier_Name = 15.0

real a_2nd_name = 15.0

real many____underscores = 20.

alias measurement _tran2

alias measurement _tran3_

Basic Rules

Scope Rules Example

- Variables local to alias:

```
alias measurement mytran1 {
               export real out_160n=V(out)@160n
      }

      alias measurement mytran2 {
               export real out_160n=V(out)@160n
      }

      run mytran1

      run mytran2
```

- No conflict exists because out_160n is a local variable inside each alias.

Scope Rules Example

Data Types and Operators

- Spectre MDL supports integer, real data types
- Float and string are not yet available
- Predefined names can be used with single quote mark (i.e., ‘rise)
- Integer constants: ‘yes, ‘no (Boolean true and false)
- Predefined constants: ‘pi, ‘e, ‘inf, ‘nan
- Physical constants: ‘q, ‘c, ‘k, ‘h, ‘eps0, ‘epsrsi, ‘u0, ‘celsius0, ‘micron, ‘angstrom, ‘avogadro, ‘logic0, ‘logic1
- Most operators are available: +, -, !=, *, /, <, <=, ==, >, >=, @, &&, ||

Data Types and Operators

Operator Precedence

Operator	Precedence
+ - (unary)	Highest precedence
@	
* /	
+ - (binary)	
< <= > >=	
== !=	
&&	
||	

Declarations

- There are two ways to make declarations:

```
alias measurement tran1 {
export real out1, out2  // Declaring with a separate statement.
     run tran
     out1=V(out)@1n
     out2=V(out)@2n
  }

alias measurement tran1 {
run tran
  export real out1=V(out)@1n  // Declaring on the line where
                   //variable is used.
  export real out2=V(out)@2n  // Declaring on the line where
                   //variable is used.
      }
```

- Can export scalars only. Cannot export waveforms

Declarations

Using the export qualifier writes the value of the variable to a PSF dataset and then to the .measure file for the simulation. The .measure file name is constructed by adding the .measure extension to the base name of the MDL control file (or you can specify another name). The .measure file is placed in the same directory as the results directory. A -measure command line option exists if you want to write the file somewhere other than the run directory.

Note that the export parameter is optional and if you omit it, the associated parameter is not exported to the database. If you calculate values that are used only in later calculations, you can omit the export qualifier to minimize the number of expression values written to the .measure file.

You must declare variables before you use them, but you can declare them anywhere and initialize them when they are declared.

Probe Functions

- Return values of voltages or currents:

```
V(p,n) // Returns the votage between nodes p and n.
V(Rload:1) // Returns the votage from terminal Rload:1 to ground.
I(Rload:1) // Returns the current through terminal Rload:1.
```

- Illegal to apply the I probe a node or a pair of nodes
- In unambiguous cases, the access functions can be omitted:

```
export real maxout=max(V(out))
```

is the same as:

```
export real maxout=max(out)
```

- If the node has a numerical name, the access function is required:

```
export real maxout=max(V(10)) // Returns a voltage.
export real maxout=max(10) // Returns the value 10.
```

Probe Functions

Use the probe functions to obtain the values of signals.

Spectre MDL provides the V probe function to obtain the potential of a signal and the I probe function to obtain the current.

Alias

- An alias measurement combines a call to an analysis with one or more Spectre MDL expressions:
- Reusable; not netlist specific

```
alias measurement showmaxmin { // alias measurement defined here.
  run tran1(stop=1u)
  export real maxout=max(V(out)) // Could use =max(out) instead.
  export real minout=min(V(out)) // Could use =min(out) instead.
      }
```

- Use the alias with the “run” command:

```
run showmaxmin // This statement runs the measurement.
```

- Can define an alias “on the fly”:

```
  run ac(center=1MHz, span=1kHz) as pb
  run ac(start=1_Hz, stop=10MHz) as sb

  run pb
  run sb
```

Alias

An alias measurement is a Spectre MDL procedure that you can use to run an analysis and extract information about the performance of the circuit. Alias measurements provide a way for you to bind analyses to Spectre MDL expressions, creating procedures that can be called multiple times and parameterized for specific applications.

You can define measurement aliases on the fly by adding the *as* keyword to the run command. The simulator creates the alias before running the analysis. The *as* keyword must follow a measurement and applies only to that measurement. For example:

```
run tran (stop=10u) as tran1
```

This command creates an alias to run tran(stop=10u) as a new analysis called tran1. It also runs the tran analysis.

Propagating Variables

- Can create variables and then use them in your code
- Can pass any tran option card

```
alias measurement trans {
  run tran(stop=5u)
  real rise_edge=cross(sig=V(out), dir=’rise, n=1, thresh=1.5, start=0)
  real fall_edge=cross(sig=V(out), dir=’fall, n=1, thresh=1.5, start=0)
  export real pw=fall_edge-rise_edge

  real iq2c=I(i1.q2:c)
  real iq2b=I(i1.q2:b)
  real iq3b=I(i1.q3:b)
  real iq4b=I(i1.q4:b)
  export real iref = iq2c + iq2b + iq3b + iq4b

      }
run trans
```

Propagating Variables

Variables with calculated values can be used in subsequent Spectre MDL expressions. For example, you might make a complicated expression easier to read by using other expressions to calculate preliminary values.

Using *foreach* Statements

- foreach statement supported for sweeping

```
alias measurement findavg{
               run tran(stop=5u)
               export real myavg=avg(V(out))
      }
foreach load_cap from swp(start=0, stop=8p, step=2p) {
       foreach temp from {0.0, 20.0, 50.0, 100.0} {
            run findavg
       }
}
```

Caution

FCS issue: nested sweep must have **“real”** values in it.

— In the above example, (0, 20, 50, 100) would fail

Using *foreach* Statements

Output of foreach

```
Swept Measurements :
Measurement Name    :  findavg
Analysis Type :  tran
myavg                    load_cap @ 0          temp @ 0    =  1.55681
myavg                    load_cap @ 0          temp @ 20    =  1.56033
myavg                    load_cap @ 0          temp @ 50    =  1.56027
myavg                    load_cap @ 0          temp @ 100    =  1.56229
myavg                    load_cap @ 2e-12          temp @ 0    =  1.5592
myavg                    load_cap @ 2e-12          temp @ 20    =  1.557
myavg                    load_cap @ 2e-12          temp @ 50    =  1.56025
myavg                    load_cap @ 2e-12          temp @ 100    =  1.5623
myavg                    load_cap @ 4e-12          temp @ 0    =  1.5566
myavg                    load_cap @ 4e-12          temp @ 20    =  1.55774
myavg                    load_cap @ 4e-12          temp @ 50    =  1.56035
myavg                    load_cap @ 4e-12          temp @ 100    =  1.5623
myavg                    load_cap @ 6e-12          temp @ 0    =  1.55723
myavg                    load_cap @ 6e-12          temp @ 20    =  1.55699
myavg                    load_cap @ 6e-12          temp @ 50    =  1.56036
myavg                    load_cap @ 6e-12          temp @ 100    =  1.56265
myavg                    load_cap @ 8e-12          temp @ 0    =  1.55906
myavg                    load_cap @ 8e-12          temp @ 20    =  1.55768
myavg                    load_cap @ 8e-12          temp @ 50    =  1.56234
myavg                    load_cap @ 8e-12          temp @ 100    =  1.56325
```

Search Statements

- Runs multiple simulations to find the value of a design parameter that corresponds to the circuit meeting or failing a specific performance criterion
- Varies design variables values automatically

```
alias measurement setup {
  export real vddelay, outcross, Tsetup, vcdelay, setdelay, maxq
  run tran(stop=40n)
  vddelay=cross(sig=V(data), thresh=2.5, dir=’rise, n=1)
  vcdelay=cross(sig=V(clock), thresh=2.5, dir=’rise, n=1)
  outcross=cross(V(q),thresh=2.5)
  maxq=max(q)
  setdelay=vdata:delay
   Tsetup=vcdelay-vddelay
}
  search vdata:delay from binary(start=2n, stop=10n, tol=1p) {
      run setup
  } until ( setup -> maxq < 2.5)
```

- “vdata:delay”: delay parameter on vsource named “vdata” is varied until criteria met on “maxq”

Search Statements

The search statement provides a way to find the value of a design parameter that corresponds to the circuit meeting or failing a specific performance criterion. The statement operates by running the simulation repeatedly, varying the values of interest each time, until a specified condition is met. This capability is typically used to determine values such as setup time and maximum load.

The above search statement varies the value of vdata:delay, using a binary search pattern, from 2n to 10n while monitoring the value of maxq. The maxq expression describes the successful transition of a flip-flop output. When the value of maxq is less than 2.5, the setup time is too short to cause the transition. When the value of maxq is equal to or more than 2.5, the setup time is long enough to cause the transition. The search continues until the search statement resolves, within the tolerance of 1ps, the time at which the transition fails to occur.

The search statement runs the simulation repeatedly, varying the value of vdata:delay each time. The vdata:delay value is set to 2.0ns for the first simulation and to 10.0ns for the second. These are the values specified by the start and stop parameters of the search statement. For example, let’s say you run this simulation with a flip-flop. For the 2.0ns value, the flip-flop does transition; for the 10.0ns value, the flip-flop does not transition. This different result on the end points is important because it indicates to Spectre MDL that the value of interest is somewhere between the two. If the results on the end point were the same, it would be impossible for the search statement to locate a point where the value transitions.

After simulating the end points, Spectre MDL simulates the midpoint (call it point M). If the simulation of M succeeds, Spectre MDL then simulates the point halfway between M and the endpoint that failed. If the simulation of M fails, Spectre MDL instead simulates the point halfway between M and the endpoint that succeeded. The binary search continues in this manner, gradually converging on the time where the flip-flop no longer transitions.

Autostop

- Stops simulation when data is available for all calculations in the MDL file
- Save simulation time
- Available in transient analysis only
- Only functions that determine specific events, such as delay and event measurements, can cause an automatic stop.
  - — If statements contain non-event functions (i.e., max), they are evaluated over the simulation period defined by the event functions
- Example autostop statement in ADE netlist output.

```
tran1 tran stop=6u method=gear2only autostop=yes
```

- MDL file:

```
alias measurement trans {
      run tran1
           export real out_1u=V(out)@1u
           export real prop_delay_fall=deltax(sig1=V(inp), sig2=V(out), dir1=’fall,
              n1=1, start1=0, thresh1=1.5, dir2=’fall, n2=1, start2=0, thresh2=1.5)
      }
         run trans
         run tran(stop=6u, autostop=’yes) as AutoRun_6u
```

Autostop

The autostop command saves time and memory resources by terminating the simulation when all event measurements have been evaluated. Event-type measurements use threshold, event or delay type functions.

- Example of autostop in Spectre MDL file:

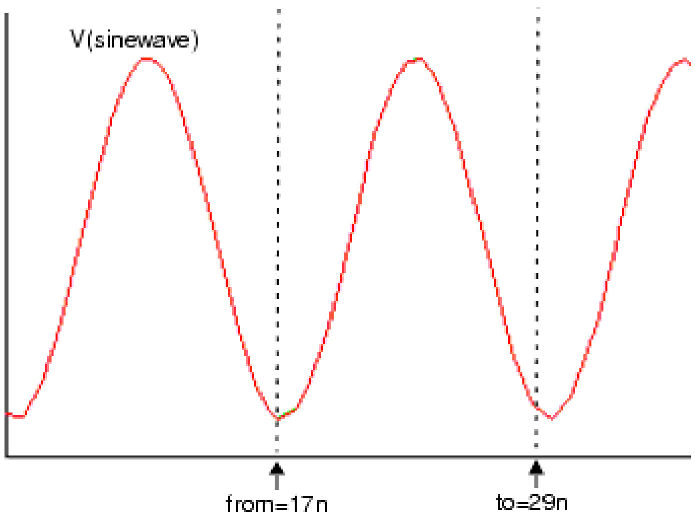
```
alias measurement trans {
    run tran1
   export real out_1u=V(out)@1u
   export real prop_delay_fall=deltax(sig1=V(inp), sig2=V(out),
   dir1=’fall,
   n1=1, start1=0, thresh1=1.5, dir2=’fall, n2=1, start2=0,
   thresh2=1.5)
         }
   run trans
   run tran(stop=6u, autostop=’yes) as AutoRun_6u
```

Spectre MDL Built-in Functions

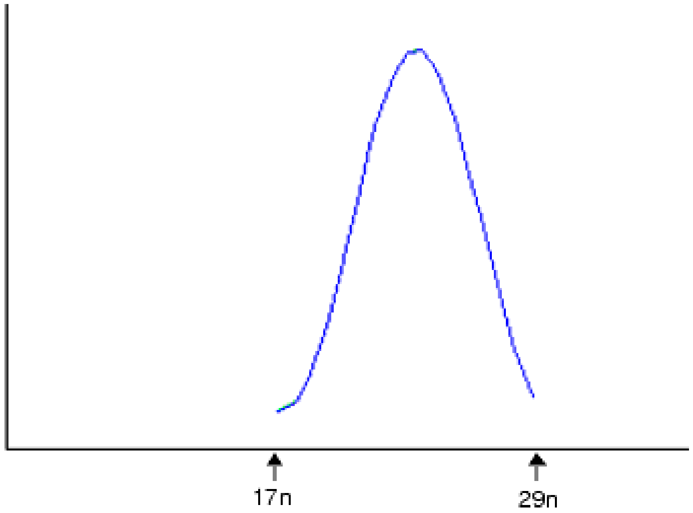
- Spectre MDL has the following built-in functions:

abs, acos, acosh, angle, argmax, argmin, asin, asinh, atan, atanh, avg, bw, ceil, cfft, clip, conj, convolve, cos, cosh, cplx, cross, crosscorr, crosses, d2r, db, db10, dbm, deltax, deriv, exp, falltime, fft, flip, floor, gainMargin, histo, I, ifft, iinteg, im, int, integ, ln, log10, mag, max, min, mod, overshoot, ph, phaseMargin, pow, pp, pzbode, pzfilter,r2d, re, real, risetime, rms, round, sign, sin, sinh, srate, sqrt, tan, tanh, trim, V, xval, yval

- Waveform Calculator, Special Functions
- The **trim** function work in both **Spectre MDL** and **WaveScan**
- Example of **trim** function in **WaveScan**, the following signal



becomes this:



Spectre MDL Built-in Functions

Spectre MDL has built-in functions, which includes most of the Special Functions from the Waveform Calculator.

Example of the trim built-in function above.

trim: Returns the portion of a signal between two points along the abscissa.

Syntax: trim(sig [, from[, to]]) OR trim(sig=sig [, from=from] [, to=to])

Arguments:

sig: The signal. Data types: real for scalar

from: The starting abscissa. Data types: real for scalar

to: The ending abscissa. Data types: real for scalar

The following example works in a Spectre MDL control file.

```
export real trimOut = max ( trim( sig=V(sinewave), from=17n, to=29n
))
```

In WaveScan: trim (sig=V(sinewave), from=17n, to=29n)

Running Spectre MDL from UNIX

- To run Spectre MDL from the UNIX operating system, enter:

  ```
  unix> spectremdl -batch myfile.mdl -design input.scs
  ```

- If the netlist has the same prefix as the design, then specify:

  ```
  unix> spectremdl -batch input.mdl
  ```

  or:

  ```
  unix> spectremdl input.mdl
  ```

- spectremdl -h returns Spectre Help. You have a new friend for MDL help:

  ```
  unix> spectremdl -usage
  unix> spectremdl -h <function>
  ```

  gives help for syntax for the specified function.

- When run, results are saved in file <batch_name>.measure.

  — To save results under a different name, use the -measure option:

  ```
  unix> spectremdl -batch input.mdl -measure myoutput.measure
  ```

  or

  ```
  unix> spectremdl -measure myoutput.measure input.mdl
  ```

Running Spectre MDL from UNIX

Syntax Specification:

```
spectremdl -batch <file.mdl> [-design <file.scs>] [-measure
<out.meas>]
               -design <file.scs> [-batch <file.mdl>] [-measure
<out.meas>]
               -usage
```

-batch <file.mdl>: The pathname or filename of the Spectre MDL file to be executed. If this argument is omitted, it will default to a filename consisting of the basename of the design file, with an extension of .mdl appended. This file must exist in the current directory. Both -design and -batch arguments must be supplied if the basenames are different.

-design <file.scs>: The pathname or filename of the netlist to be loaded by Spectre. If this argument is omitted, a file with the basename of the batch argument, with an extension of .scs or .ckt is searched for in the current directory, and used as the default.

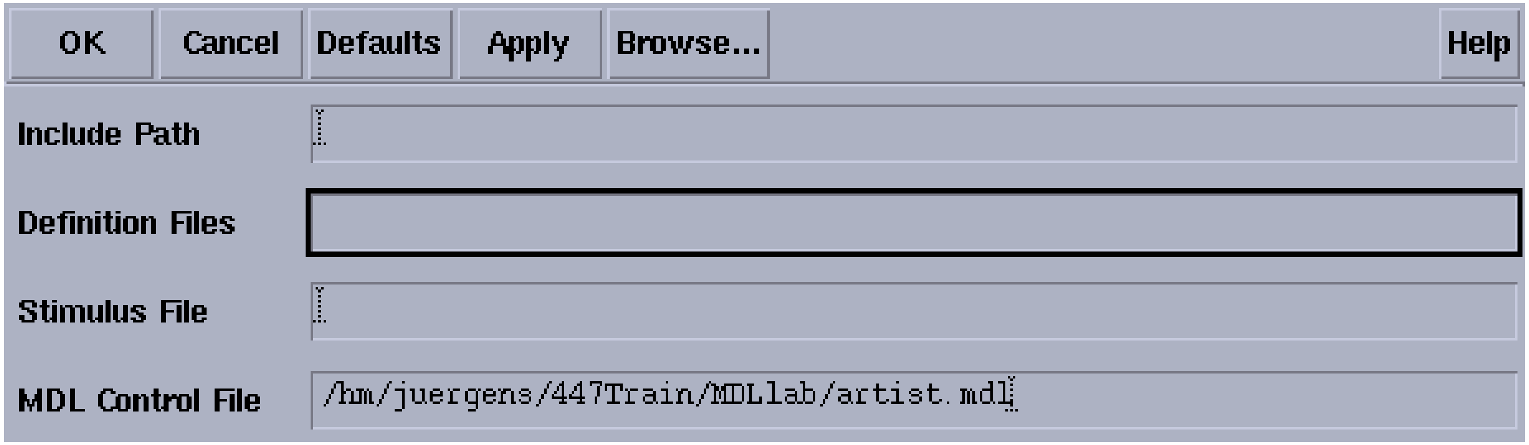
-measure <file.measure>: The default output filename is taken from the basename of the design argument, and appended with the .measure extension. This option creates the output file with the specified name. If a pathname is not specified, the output file is created in the directory of the design argument.

-usage: Display this help text.

All other command line arguments are passed directly to Spectre. If the Spectre MDL file does not contain any export statements, or if Spectre exits with a return code, the output file is not created.

Spectre MDL in the Analog Design Environment

- To use your Spectre MDL file within Analog Design Environment, select in the simulation window: **Setup - Simulation Files ...**
- In the “Simulation Files Setup” form, add the path to the MDL Control File text field.



Include Path

Definition Files

Stimulus File

MDL Control File /hm/juergens/447Train/MDLlab/artist.mdl

- The “run” name in the MDL file must be like the name in Spectre Direct. For example ‘run ac’ works, but ‘run ac1’ fails.
- Note that only the analyses in the MDL file will run; all “info” statements are ignored.
- Data saved in psf directory: acrun-meas_ac, trans-meas_tran
- To view the measurements in ADE, select: **Results—View MDL .measure file**

Spectre MDL in the Analog Design Environment

The Analog Design Environment supports the use of Spectre MDL. The Spectre MDL input file is invoked by using the Setup menu in the simulation window.

Select **Setup—Simulation Files...**

The **Simulation Files Setup** form appears. Enter the path for the Spectre MDL input file into the **MDL Control File** text field.

When the simulation run is complete, the measurement file is viewed in the Analog Design Environment by selecting **Results—View MDL .measure file**.

Spectre MDL in OCEAN

Spectre MDL can run from OCEAN.

- However, there is a requirement to use v( ) or i( ) data access command to print measurement data.
- Example:

```
ocean> selectResults(’"acrun-meas_ac")
ocean> outputs
ocean> getData( "bandw" )
ocean> ocnPrint(v("bandw"))
          1.41217e+07
ocean> ocnPrint( i("bandw"))
       1.41217e+07
```

Spectre MDL in OCEAN

Spectre MDL can be used within the OCEAN operating environment.

The are some special restrictions. With this release of software, measurements using Spectre MDL require the voltage or current access function. In other words, in the example shown above, the **bandwidth** measurement **“bandw”** must be preceded with v or i access functions.

Labs

Lab C-1 Using Spectre MDL

Labs

Appendix D: Match Analysis, *dcmatch*

Topics in this module

- Special modeling requirements
- Introduction to device mismatch
- Effects of device mismatch on analog circuits
- Design considerations
- Layout matching
- *dcmatch* analysis flow

Terms and Definitions

device mismatch A physical limitation, no two devices have identical electrical properties.

offset error Mismatch error that requires additional voltage or current to correct.

input offset Mismatch error measured at the inputs of opamps and comparators.

common centroid A geometric layout technique to reduce errors from process gradients.

implant shadowing In fabrication, mask layer forms a shadow to an angled ion beam.

DNL Differential non-linearity, error between steps in ADCs and DACs.

INL Integral non-linearity. Total error across all codes in ADCs and DACs

ADC Analog-to-Digital converter, processes an analog input to digital codes.

DAC Digital-to-Analog converter, processes digital codes to an analog output.

dcmatch A type of analysis to evaluate the effects of using matched devices.

Overview of Device Mismatch

- Two devices “replicated” on the same chip are **never** identical. To some degree, differences in the electrical properties of the two devices **always** occurs.
- In a large array of devices replicated on the same chip, no device is identical to any of the other device. Again, the electrical properties will be different.
- Mismatches are produced by random processes, and produce the following effects:
  - — Undesired effects in analog circuits.
  - — Offset errors in comparators and amplifiers
  - — Voltage distributions in band-gap reference
  - — DNL and INL in DACs and ADCs.
- In simulation, devices are often modeled using identical numerical values and expressions for the electrical behavior.
- Simulations often fail to include the effects of mismatch.
- Precise values of mismatch cannot be predicted.
- Statistical methods are used to analyze mismatch.

Overview of Device Mismatch

Random effects during fabrication produce distributions in the physical structures and electrical properties of electronic devices. Some of these random processes include lattice defects, surface defects, optical errors, mechanical vibrations, edge effects, and localized impurities. As such, the device parameters or electrical properties of replicated components are not identical.

Mismatched devices produce numerous undesirable effects in analog circuits. Mismatch contributes to input offset on operational amplifiers and comparators. The mismatch also reduces CMRR. In many band-gap circuits, the mismatch produces a large distribution in output voltage. In DACs and ADCs, the mismatch produces DNL and INL errors.

Circuit simulators use device models to describe electrical behavior. The models use equations and often have identical numerical values for replicated devices. Often the simulation results do not show the effects of device mismatches because the mismatch was never modeled.

Design Considerations

Numerous factors either increase or reduce mismatch errors.

These include:

- Layout dimensions (width and length)
- Matched physical layout
- Orientation (rotation, mirror image, misalignment, etc.)
- Spatial separation (distance between replicated devices)
- Proximity effects (how close are other unrelated devices)
- Common centroid techniques (1 and 2 dimensional arrays)
- Dummy devices
- Contact resistance
- Package stress and “edge effects”

Important

These effects shall also influence the model extraction for the mismatch analysis.

Design Considerations

Many design considerations influence device mismatch. Many of the considerations involve the device geometries. Capacitors, resistors and transistors are produced by a series of geometric shapes. Any inconsistencies in duplicating these shapes contributes to device mismatches.

The width and length of capacitors, resistors, and transistors are altered by these replication errors. A geometric replication error has a mean and variance. A small design length is proportionally more sensitive to such an error than a larger design length, (Lerror/L). By increasing the width and length, the error on W/L or L/W ratios become proportionally smaller and the W/L ratios become more accurate.

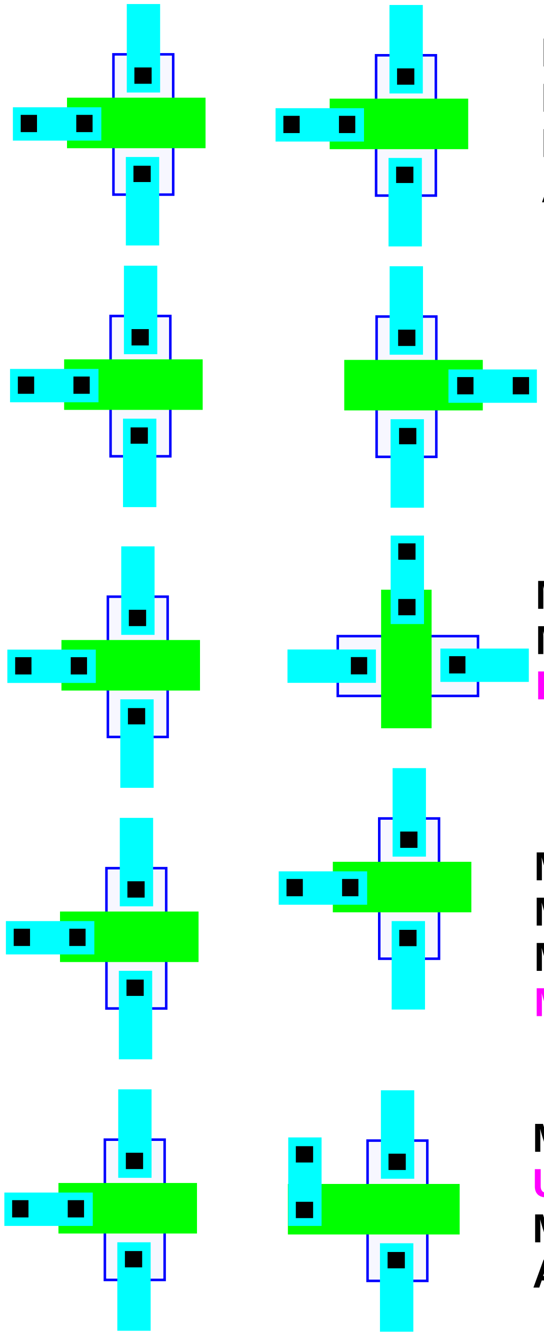
Where device matching is required, it is important to actually match the device geometries. Such devices should not be rotated or misaligned. The use of a “mirror layout” between two devices will sometimes improve the matching. However, a mirror layout is sensitive to processes with “implant shadowing.” If you are not certain if implant shadowing occurs in the process, avoid mirror devices where matching is required. The devices should also see the same geometric conditions in all directions. For this reason dummy devices are often where critical matching is needed.

Fabrication often produces process gradients across the chip. A process gradient is a condition where model parameters tend to increase or decrease in a specific direction. To reduce the device mismatches due to process gradients, the spatial distances between matched devices should be small. Another design technique used to reduce the effects of gradients is called **common centroid**. Common centroid is a geometric arrangement where each of the “matched devices” comprises an array of components. The array is then used so that the gradients have equal and opposite effects on the components. Common centroid reduces mismatch for **first order gradients**. For second order gradients the common centroid design shall increase the mismatch.

Important

These factors influence the model extraction for mismatch analysis. All of these factors should be properly modeled if used where device matching is needed.

Layout Matching



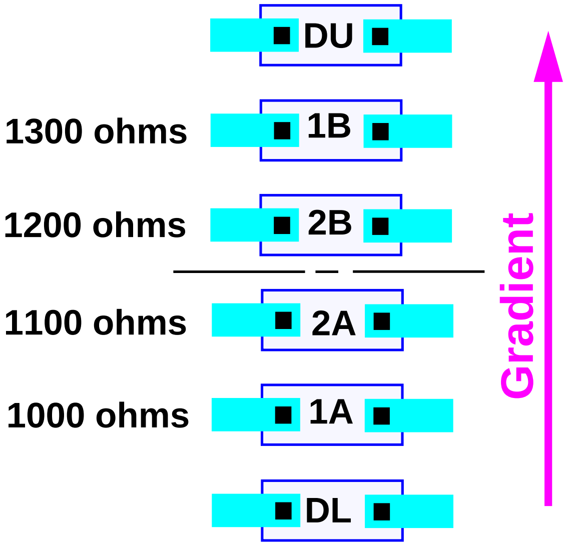
Matched W and L
Matched layout
Matched orientation
Aligned in Y

Matched W and L
Matched layout
Mirror orientation!
Aligned in Y

Matched W and L
Matched layout
Rotated orientation!

Matched W and L
Matched layout
Matched orientation
Misalignment X and Y!

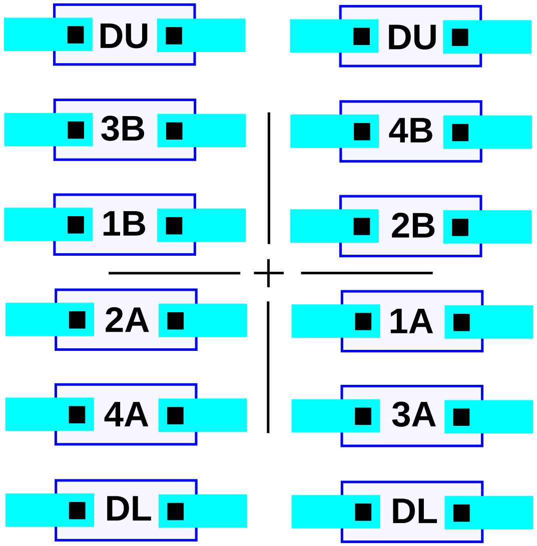
Matched W and L
Unmatched layout!
Matched orientation
Aligned in Y



The 1-Dimensional Array

Common Centroid Design

The 2-Dimensional Array



Layout Matching

The diagram shows both the matched and the unmatched layout examples. For best modeling of device mismatch, each example should have a corresponding mismatch model.

The 1-dimensional common centroid shows values for the resistor elements and the proper sequence for adding the elements. In this example, a vertical process gradient occurs in the sheet resistance. In this example:

R1 = R1A + R1B = 1000 ohms + 1300 ohms = 2300 ohms

R2 = R2A + R2B = 1100 ohms + 1200 ohms = 2300 ohms

The 2-dimensional common centroid has process gradients of sheet resistance in both x and y directions. Each resistor element in the increasing x direction is 100 ohms smaller. Each resistor element in the increasing y direction is 100 ohms larger.

R1 = R1A + R1B = 1.0K + 1.2K = 2.2K

R2 = R2A + R2B = 1.1K + 1.1K = 2.2K

R3 = R3A + R3B = 0.9K + 1.3K = 2.2K

R4 = R4A + R4B = 1.0K + 1.2K = 2.2K

Important

Common Centroid is used in DACs and ADCs to reduce INL or integral non-linearity error. In the case of current sources comprising MOS transistors, the 2-dimensional sequence shown is also used.

Special Modeling Requirements for dcmatch

The **dcmatch** analysis has been added to the **Choosing Analysis** form. It is important to know that this new analysis has special model requirements. The dcmatch analysis works with bsim3v3, bsim4, res, vbic.

The lab activity for this module uses bsim3v3. You will be instructed to view the model file. The mismatch model parameters shown have been extracted for a specified process. The model extraction for these parameters is unique to that process only.

The dcmatch parameters are unique to a specific process. To work properly, the dcmatch model parameters must be properly extracted for the process used. When this manual was written very few fabrication facilities supported extraction of these model parameters.

In addition to the statistical properties of the fabrication facility, the geometric conditions discussed must be included with the model extraction. Using different geometric conditions shall alter NMOS and PMOS mismatch model parameters.

Special Modeling Requirements for dcmatch

Special consideration must be used when dcmatch analysis:

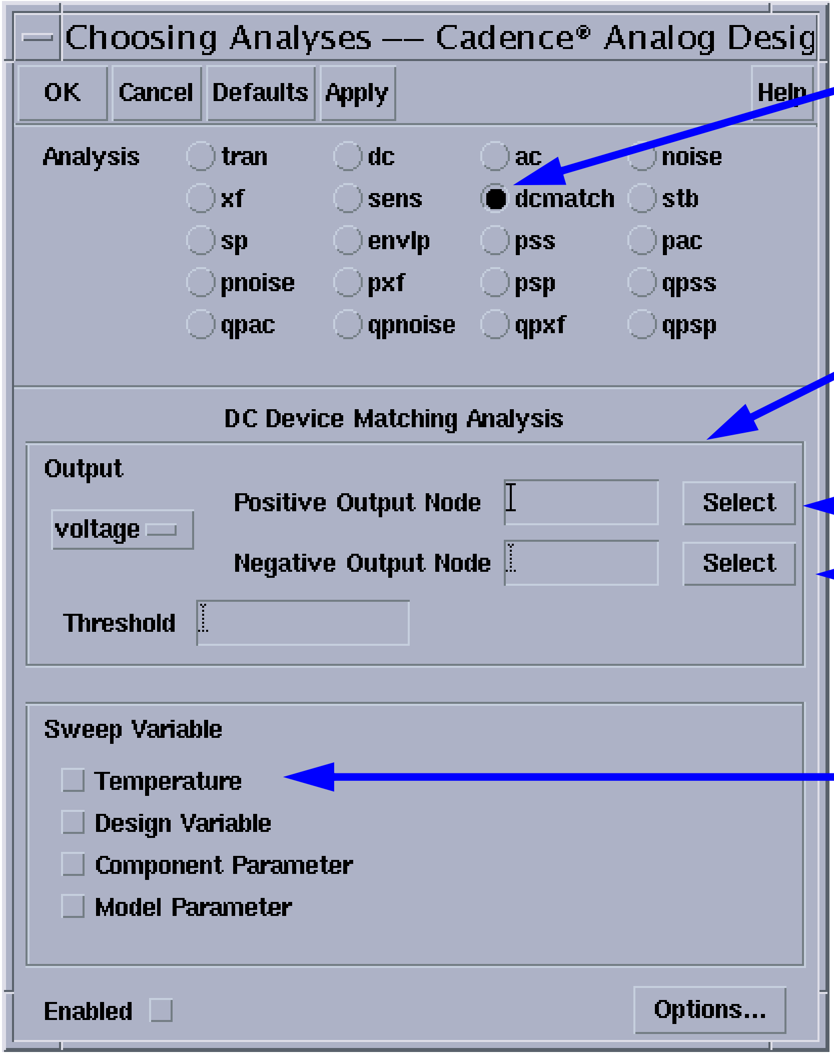
- The use of dcmatch analysis requires special models.
- The model parameters must be properly extracted.
- The model parameters are unique to a specific foundry.
- The model parameters are geometry dependent.

Starting the dcmatch Analysis

From the simulation window, select: **Analyses—Choose...**

The Choosing Analyses form appears.

In the form select dcmatch.



Choosing Analyses — Cadence® Analog Desig

OK | Cancel | Defaults | Apply | Help

Analysis				
	○ tran	○ dc	○ ac	○ noise
	○ xf	○ sens	● dcmatch	○ stb
	○ sp	○ envlp	○ pss	○ pac
	○ pnoise	○ pxf	○ psp	○ qpss
	○ qpac	○ qpnoise	○ qpxf	○ qpsp

DC Device Matching Analysis

Output

voltage

Positive Output Node [ ] Select

Negative Output Node [ ] Select

Threshold [ ]

Sweep Variable

□ Temperature

□ Design Variable

□ Component Parameter

□ Model Parameter

Enabled □ Options...

1. Select this button!

2. The form changes to allow input of dcmatch information.

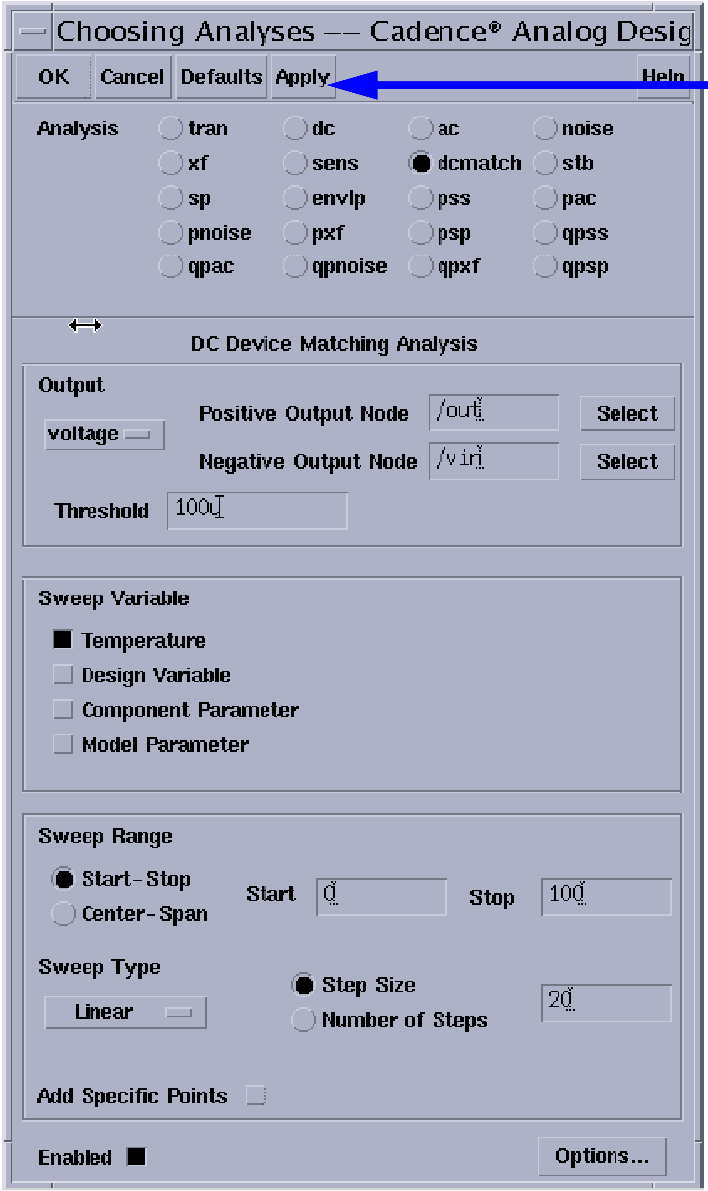
3. Use these buttons to select nodes on the schematic.

4. For a **dc sweep** select one of these buttons.

Starting the dcmatch Analysis

dcmatch Analysis Selection

Select **OK** or **Apply** in the Choosing Analysis form.



Analysis: ○ tran ○ dc ○ ac ○ noise ○ xf ○ sens ● dcmatch ○ stb ○ sp ○ envlp ○ pss ○ pac ○ pnoise ○ pxf ○ psp ○ qpss ○ qpac ○ qpnoise ○ qpxf ○ qpsp

DC Device Matching Analysis

Output: voltage

Positive Output Node: /out (Select)

Negative Output Node: /vin (Select)

Threshold: 100u

Sweep Variable

- ■ Temperature
- □ Design Variable
- □ Component Parameter
- □ Model Parameter

Sweep Range

- ● Start-Stop — Start: 0 — Stop: 100
- ○ Center-Span

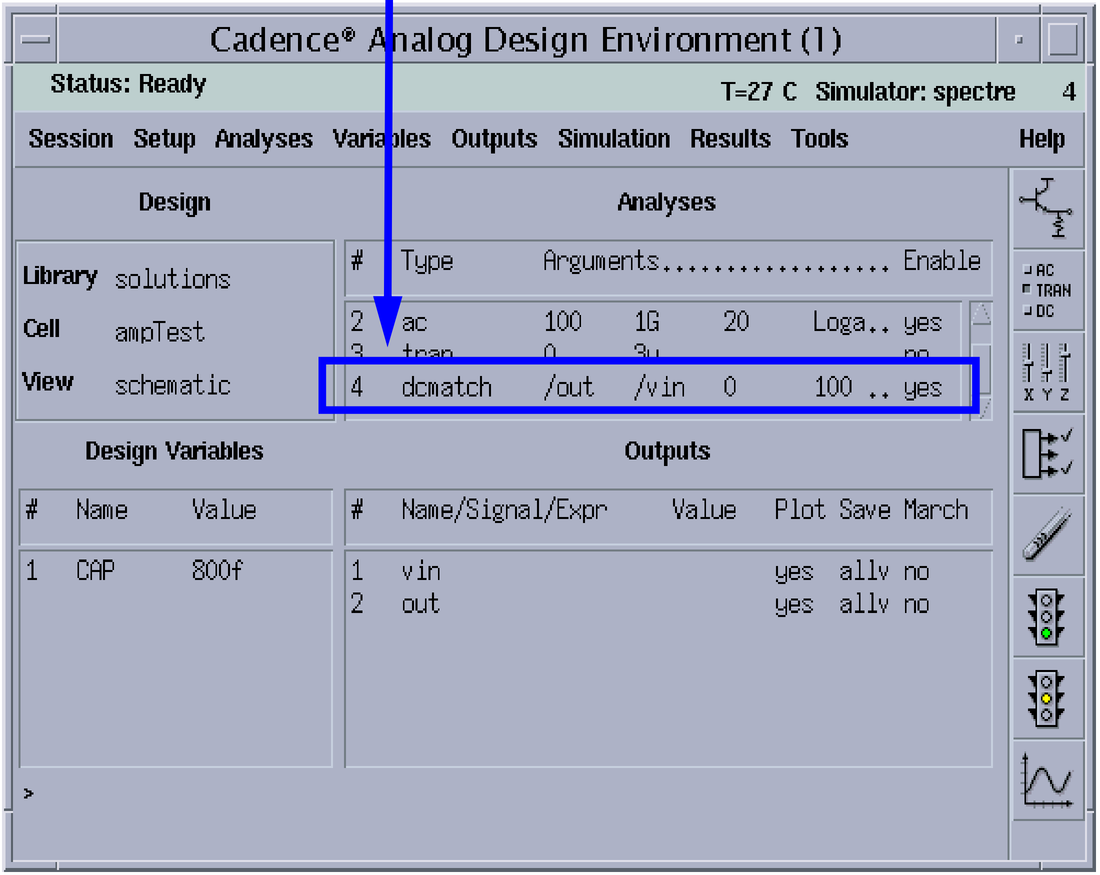
Sweep Type: Linear

- ● Step Size: 20
- ○ Number of Steps

Add Specific Points □

Enabled ■

Then verify the **dcmatch** analysis appears in the Analyses field of the simulation window.



Status: Ready T=27 C Simulator: spectre 4

Design

Library	solutions
Cell	ampTest
View	schematic

Analyses

#	Type	Arguments.................					Enable
2	ac	100	1G	20	Loga..		yes
3	tran	0	3u				no
4	dcmatch	/out	/vin	0	100	..	yes

Design Variables

#	Name	Value
1	CAP	800f

Outputs

#	Name/Signal/Expr	Value	Plot	Save	March
1	vin		yes	allv	no
2	out		yes	allv	no

dcmatch Analysis Selection

Labs

Lab D-1 dcmatch

Labs

Appendix E: Advanced Topics in ADE

Topics in this module

- Overview of advanced topics
- Applications of inherited connections
- Setting net expressions and using the netSet property
- Netlist and Run with inherited connections
- Introduction to Verilog-A
- Introduction to the Mixed Signal Design Environment

Terms and Definitions

Verilog-A	An HDL language used for describing analog behavior or structure.
veriloga	View name for a netlistable module written Verilog-A.
Modelwriter	A Cadence tool for generating Verilog-A code for selected modules.
MSDE	Mixed Signal Design Environment.
IPC	Inter Process Communication, a special program used in MSDE.

Overview of Advanced Topics

This optional module provides instruction on the advanced features of the Cadence Analog Design Environment. The covered topics include:

- Introduction to Verilog-A
- Introduction to Mixed Signal Design Environment

Note that the Analog Design Environment is a gateway to using Verilog-A and the Mixed Signal Design Environment.

- Classes are provided by Cadence Education Services.

Overview of Advanced Topics

The Analog Design Environment is a gateway to using other Cadence design tools. These other tools and advanced features of the Analog Design Environment are discussed in this special module. Completion of this module is not required.

Introduction to Verilog-A

The Verilog®-A Language

- Is an extension of the Verilog language (with many HDL constructs from the Verilog language) used to describe analog/mixed signal models, and is an Open Verilog International (OVI) standard.
- Is a multidiscipline language that models electrical, mechanical, fluid dynamic, and thermodynamic systems.
- Is used with Spectre Circuit Simulator in the Analog Design Environment, in the Analog Workbench (AWB), AMS Designer, or in a standalone mode.
- Supported simulators in the Analog Design Environment:
  - Spectre—known as Spectre Direct simulator
  - Spectre Verilog—mixed Spectre Direct and Verilog-XL simulators
  - Spectre S—socketed Spectre simulator
  - Spectre S Verilog—socketed Spectre and Verilog-XL simulators
- Supports top-down design
  - Easy to learn and use
  - Easy to transition from abstract to detailed models

Introduction to Verilog-A

The Verilog-A software is a high-level language that uses modules to describe the structure and behavior of analog systems and their components.

Verilog-A is extremely useful in:

- Defining a system architecture.
- Verifying the feasibility of a system design.
- Developing interface specifications, and completing chip level simulations.

In addition, Verilog-A simplifies the design process in a front to back design flow.

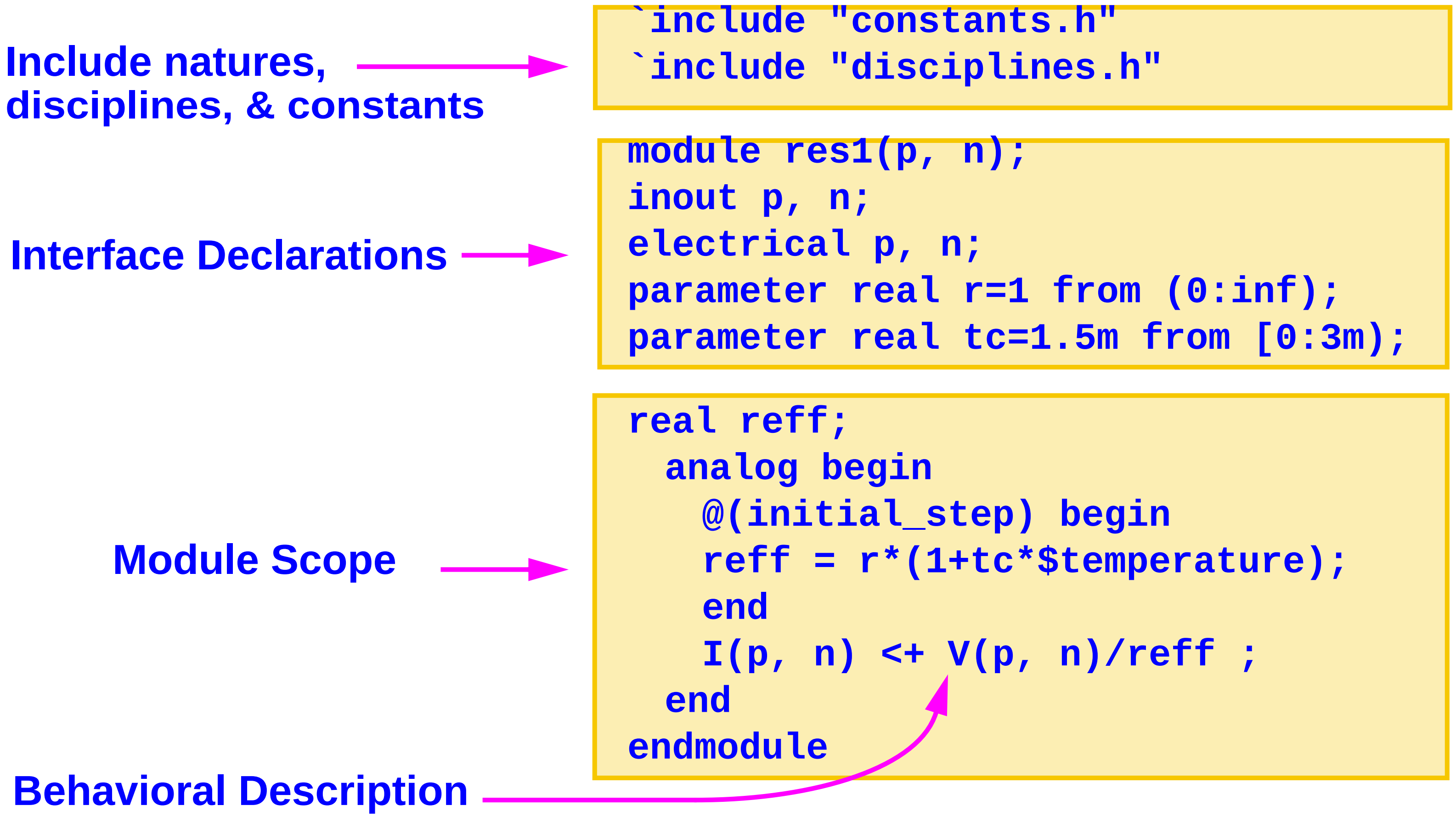
Verilog-A can describe a wide range of conservative systems and signal-flow systems, such as electrical, mechanical, fluid dynamic, and thermodynamic systems. Cadence provides numerous models in all these disciplines that can be used as a starting point for specific model applications.

For consistency, Verilog-A uses many constructs found in the Verilog and C programming languages. These languages use modules, which are the fundamental user-defined primitives.

To describe a system, specify both the structure of the system and the behavior of its components at different levels. At the highest level, use Verilog-A to define the overall system as a structural model. At lower levels, use Verilog-A to define the internal modules as structural or behavioral models, thus defining the interconnections among submodules.

The Verilog-A Module

A code example of a behavioral Verilog-A module



Include natures, disciplines, & constants

```
`include "constants.h"
`include "disciplines.h"
```

Interface Declarations

```
module res1(p, n);
inout p, n;
electrical p, n;
parameter real r=1 from (0:inf);
parameter real tc=1.5m from [0:3m);
```

Module Scope

```
real reff;
  analog begin
    @(initial_step) begin
    reff = r*(1+tc*$temperature);
    end
    I(p, n) <+ V(p, n)/reff ;
  end
endmodule
```

Behavioral Description

The Verilog-A Module

A Verilog-A module is a text file that is accessible as a *veriloga* view of a cell from the Library Manager tool. The text file comprises lines of code using the Verilog-A language syntax. This text file has *`include* statements that access the *constants.h* and the *disciplines.h* files, which are required for all behavioral modules. In addition this text file includes **interface declarations** and a **module scope**.

To use the *veriloga* view for a simulation, just create a corresponding *symbol* view and and instantiate that *symbol* view into a schematic.

Advantages of Verilog-A

- Verilog-A is a behavioral language that simulates faster than structural.
- Can be used to simulate almost anything in Spectre, including:
  - Mechanical structures
  - Fluid dynamic systems
  - Thermodynamic
  - Magnetic and electromagnetic structures
- Verilog-A is a modeling language that is easy to learn. Cadence Educational Services provides both classroom and internet training.
- Verilog-A modules are very easy to use.
- Extensive Library support

  `<install_dir>/tools/dfII/samples/artist/ahdlLib`

  `<install_dir>/tools/dfII/samples/artist/rfLib`

  `<install_dir>/tools/dfII/samples/artist/spectreHDL/Verilog-A`

  The libraries contain **hundreds** of samples of predesigned Verilog-A modules for common circuit structures.
- Development tools include ***Modelwriter*** and ***AHDL Debugger***

Advantages of Verilog-A

Verilog-A simulates must faster than circuit structures because behavioral descriptions are used. Behavior reduces the size of the solution matrix and the number of iterations to solve the solution matrix.

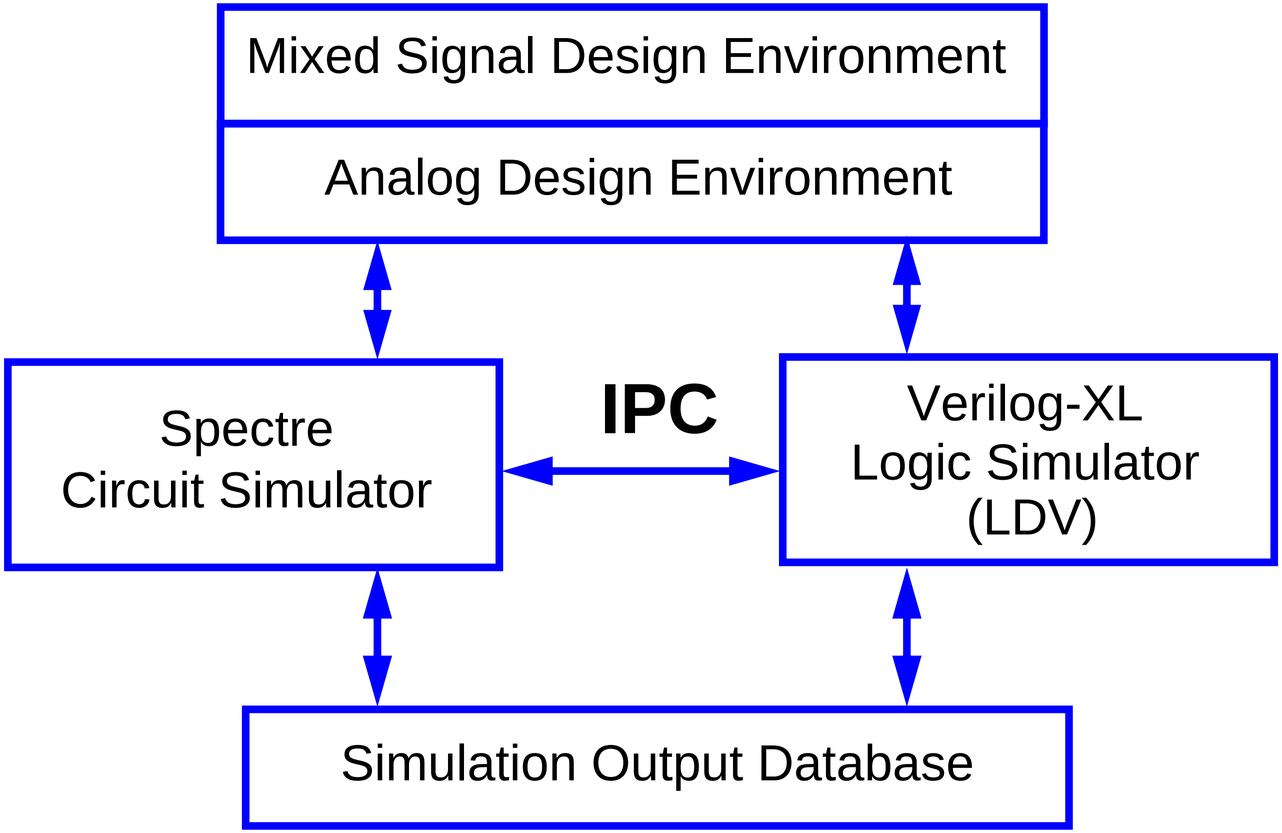
Verilog-A is a multidiscipline language. It can be used to solve electrical circuits, as well as mechanical, thermodynamic, magnetic and fluid dynamic problems. Almost anything can be simulated using Verilog-A and the Spectre Circuit Simulator.

Verilog-A is very easy to use. The Modelwriter tool will write models for preselected circuits. These Verilog-A models are automatically written with assigned ports and with user-specified parameters.

Cadence Educational Services provides both public and private classroom instruction in Analog Modeling with Verilog-A. The classroom course is taught in two just days. Topics include writing Verilog-A modules, capturing a system hierarchy, system modeling, and running simulations. The class also covers language syntax, model disciplines, formatting output, and filtering signals. A lab activity is included in each module.

The course is also available using the Cadence Internet Learning Series (iLS). This class covers the same topics presented in the classroom. There are also downloadable lab activities. The iLS class offers the advantage of “Anytime - Anywhere” learning. Once you obtain an iLS account, you can use the internet and browser of your choice to access the class material.

Introduction to Mixed Signal Design Environment



Mixed Signal Design Environment
Analog Design Environment
Spectre
Circuit Simulator
IPC
Verilog-XL
Logic Simulator
(LDV)
Simulation Output Database

The above diagram shows a conceptual overview of the Mixed Signal Design Environment. MSDE is actually a software extension of the Analog Design Environment. When LDV is installed along with a valid LDV license file, then the MSDE extension of Analog Design Environment is enabled.

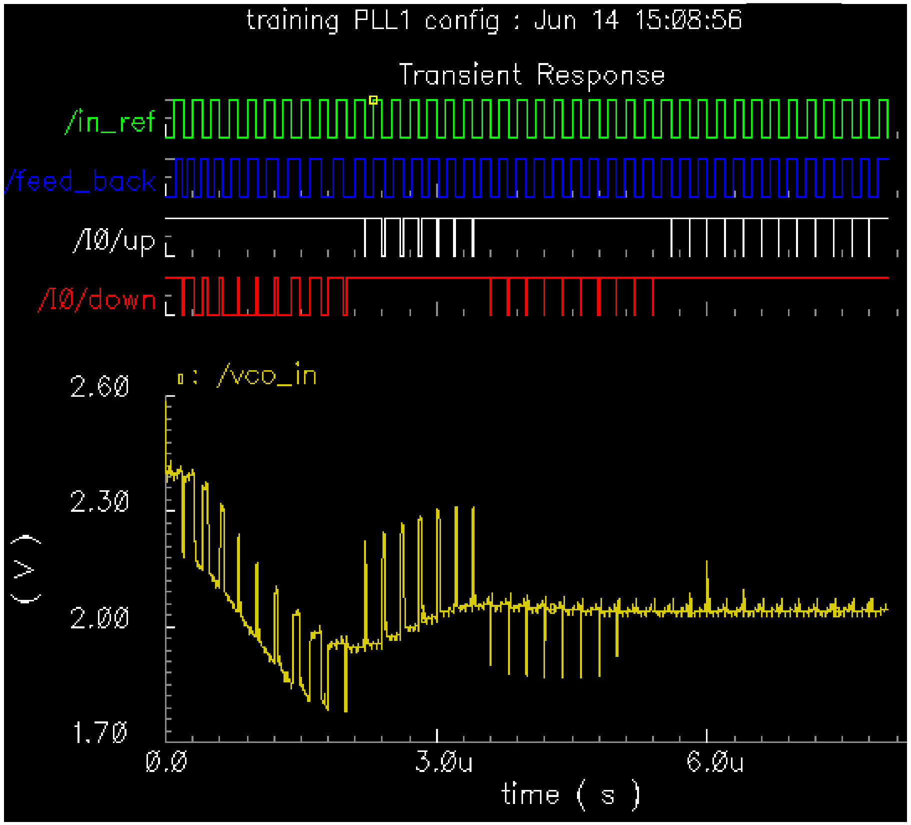
Introduction to Mixed Signal Design Environment

The Mixed Signal Design Environment combines the Spectre Circuit Simulator with the Verilog-XL simulator from LDV. The mixed signal environment also provides an Inter Process Control to interrupt the Spectre Circuit Simulator whenever digital events need to be processed in the Verilog-XL simulation.

MSDE is a true mixed-signal simulation environment. The two simulators operate together passing the needed analog and digital data back and forth. Each simulator operates within its specific domain. When signals connect from analog to the digital domain, or from digital to analog domain, then interfaces element do the signal conversion between the domains.

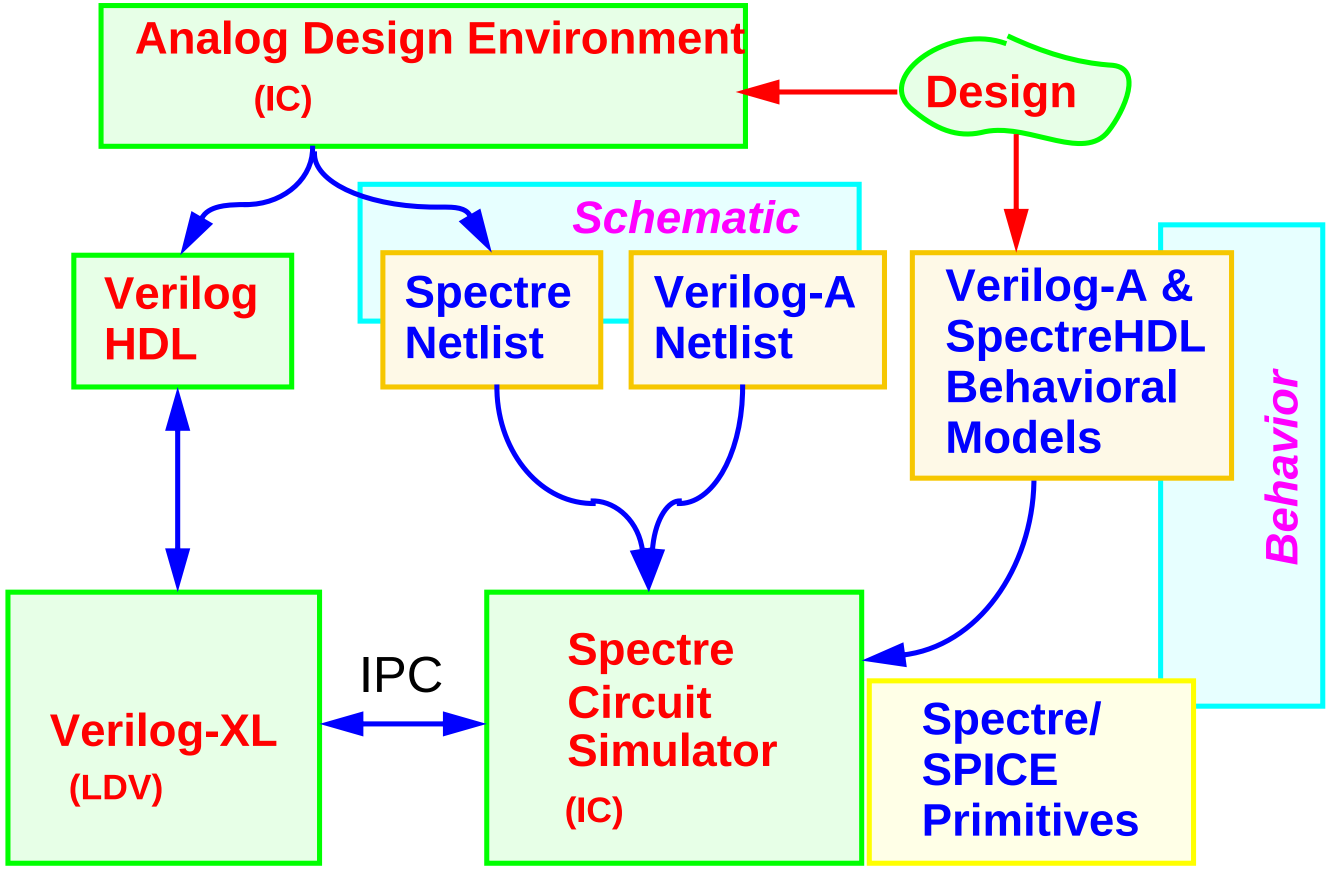
Advantages of Mixed Signal Design Environment

1. True mixed signal simulation and waveforms.
2. Mixed Signal Parasitic Simulation.
3. Backannotation.
4. Works with Verilog-A.



Advantages of Mixed Signal Design Environment

Mixed Signal Design Environment and Verilog-A



Analog Design Environment
(IC)
Design
Schematic
Verilog HDL
Spectre Netlist
Verilog-A Netlist
Verilog-A & SpectreHDL Behavioral Models
Behavior
IPC
Verilog-XL
(LDV)
Spectre Circuit Simulator
(IC)
Spectre/ SPICE Primitives

Mixed Signal Design Environment and Verilog-A

The Mixed Signal Design Environment is compatible with Verilog-A. The diagram above shows a conceptual overview of how netlisting is realized. The above diagram is also used to conceptualize a design flow using the Analog Design Environment, the Mixed Signal Design Environment, and Verilog-A.

Labs

Lab E-1 Verilog-A Overview

Labs