Virtuoso® Platform Update Training: Analog Design

Version 6.1.0

Lecture Manual **December 14, 2006**

cadence

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Cadence Trademarks

Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address above or call 800.862.4522.

Allegro®
Accelerating Mixed Signal Design®
Assura®
BuildGates®
Cadence® (brand and logo)
CeltIC®
Conformal®
Connections®
Diva®
Dracula®
ElectronStorm®
Encounter®
EU CAD®
Fire & Ice®
First Encounter®
HDL-ICE®
Incisive®
InstallScape™
IP Gallery™
NanoRoute®
NC-Verilog®
NeoCell®
NeoCircuit®
OpenBook® online documentation library
OrCAD®
Palladium®
Pearl®
PowerSuite®
PSpice®
SignalStorm®
Silicon Design Chain™
Silicon Ensemble®
Silicon Express™
SKILL®
SoC Encounter™
SourceLink® online customer support
Specman®
Spectre®
Speed Bridge®
UltraSim®
Verifault-XL®
Verification Advisor®
Verilog®
Virtuoso®
VoltageStorm®
Xtreme®

Other Trademarks

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

All other trademarks are the property of their respective holders.

Confidentiality Notice

Table of Contents
Update Training: Analog Design Environment

Module 1 Introduction to the Analog Design Update Training

Module 2 Design Framework II Environment

Module 3 Schematic Editor L

Module 4 Schematic Editor XL

Module 5 Front-end Translators

Module 6 Introduction to Constraints

Module 7 Using ADE L

Module 10 Generating Specifications

Module 11 Data Assistant

Module 12 ADE XL and GXL Customization

Module 13 Corners and Sweeps

Module 14 Virtuoso Design Characterization and Modeling

Module 15 ADE GXL Parameterization and Optimization

Module 16 Using Virtuoso Parasitic Estimation and Analysis with ADE GXL

Introduction to the Analog Design Update Training

Module 1

December 14, 2006

Course Objectives

This course provides an overview of the new features and use models for analog design and simulation in the IC 6.1.0 release for those familiar with IC 5.1.41.

After completing this course, you will be able to

- Generate new or open existing analog designs with Virtuoso® Analog Design Environment (ADE) in the classic and XL modes
- Run analog simulations with ADE in the classic and XL modes
- Use the new features of optimization and parasitic emulation in the GXL mode to improve the accuracy of your designs

This course is one of three update courses for the IC 6.1.0 software release. The other two are:

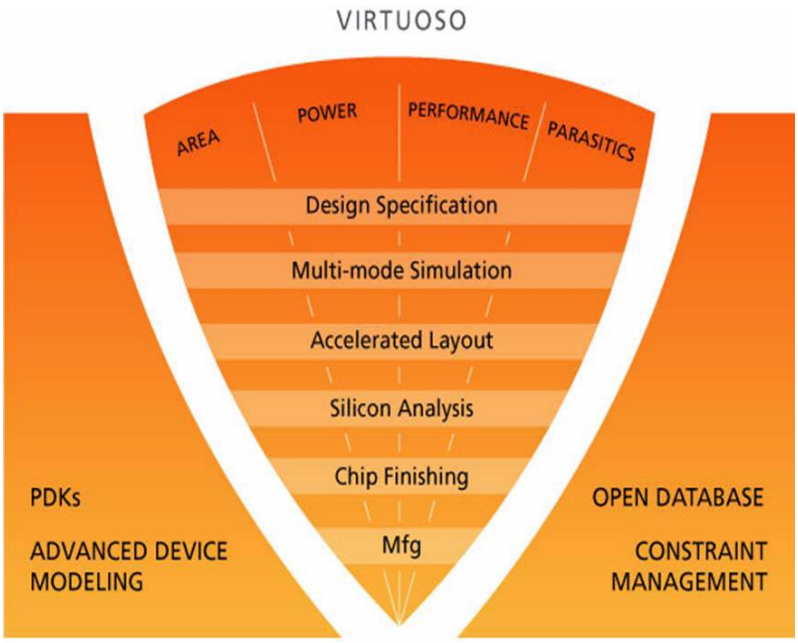
- *Virtuoso Platform Update Training: Infrastructure*

  The infrastructure course covers the changes to the SKILL® programming language, to the Cadence Design Framework II environment, and the upgrade from the Cadence CDB database to the OpenAccess database.

- *Virtuoso Platform Update Training: Physical Design*

  The physical design course covers changes to the physical design environment.

New Integrated Virtuoso Platform



Virtuoso Product Segmentation

The right tool for the right job!

- L — Basic Level

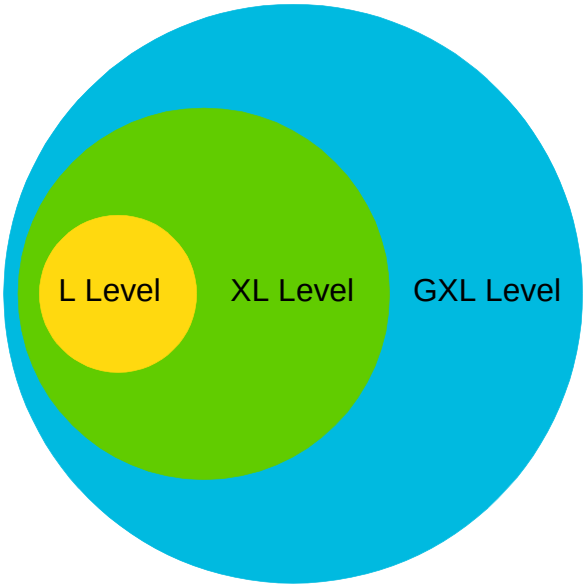
  Basic design creation and implementation updated with productivity enhancements.

- XL — Advanced Level

  Tightens communication between design and implementation engineers, speeding creation, implementation and repair.

- GXL — Automated Level

  Advanced tools suited for tackling difficult yield, parasitic and chip finishing tasks.



Day 1 Schedule

- Module 1, Introduction to the Analog Design Update Training
- Module 2, Design Framework II Environment
  - Lab 2-1 Trying the New Environment Features
  - Lab 2-2 Examining the gpdk090 Technology File
  - Lab 2-3 Modifying a Toolbar Menu
- Module 3, Schematic Editor L
  - Lab 3-1 Consolidated Banner Menus
  - Lab 3-2 Using Tabs
  - Lab 3-3 Introduction to Toolbars

Day 1 Schedule (continued)

- Module 4, Schematic Editor XL
  - Lab 4-1 Using the Go Toolbar
  - Lab 4-2 Workspaces
  - Lab 4-3 Bookmarks
  - Lab 4-4 Navigator Assistant
  - Lab 4-5 Search Toolbar and Assistant
  - Lab 4-6 Property Editor Assistant
  - Lab 4-7 World View Assistant
- Module 5, Front-end Translators
  - Lab 5-1 Running SpiceIn from Design Framework II
  - Lab 5-2 Running SpiceIn from the Command Prompt

Day 2 Schedule

- Module 6, Introduction to Constraints
  - Lab 6-2 Creating Constraints Using the Circuit Prospector
  - Lab 6-2 Creating Constraints Using the Circuit Prospector
  - Lab 6-3 Constraints Propagation
- Module 7, Using ADE L
  - Lab 7-1 Using ADE L
  - Lab 7-2 ADE L with ampTest
- Module 8, Multiple Tests in ADE XL
- Module 9, Using Design Variables
  - Lab 9-1 Setting Up Tests in ADE XL
  - Lab 9-2 Adding a Second Test

Day 3 Schedule

- Module 10, Generating Specifications
  - Lab 10-1 Generating Specifications
- Module 11, Data Assistant
  - Lab 11-1 Using the Data Assistant
  - Lab 11-2 Restoring a Checkpoint
- Module 12, ADE XL and GXL Customization
- Module 13, Corners and Sweeps
  - Lab 13-1 Corners and Sweeps in ADE XL
  - Lab 13-2 Corners and Sweep in ADE XL with ampTest

Day 4 Schedule

Module 14, Virtuoso Design Characterization and Modeling

- Lab 14-1 Top-Down Modeling
- Lab 14-2 Bottom-up Modeling
- Lab 14-3 Design Verification

Module 15, ADE GXL Parameterization and Optimization

- Lab 15-1 Optimizing a Design in ADE GXL

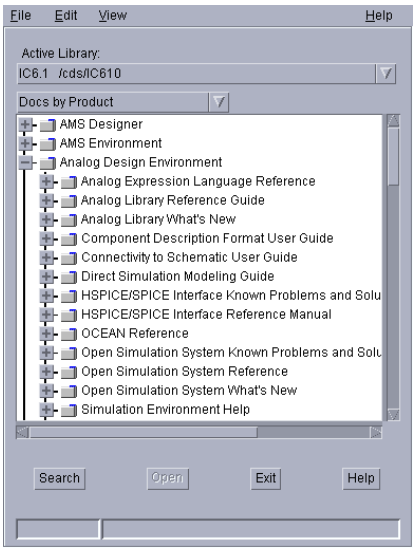
Module 16, Using Virtuoso Parasitic Estimation and Analysis with ADE GXL

- Lab 16-1 Parasitic Estimates Setup and Simulation in ADE GXL
- Lab 16-2 Refining Extracted Views
- Lab 16-3 Comparing Estimated Parasitics with Extracted Parasitics

Getting Help: CDSDoc

Enter *cdsdoc* on the UNIX or Linux command line to access the installed Cadence online manuals. The following documents apply specifically to the Virtuoso ADE L, XL, and GXL products:

Virtuoso ADE L User Guide
Virtuoso ADE XL User Guide
Virtuoso ADE GXL User Guide



- In Linux, open Mozilla first before starting CDSDoc.
- To add document libraries to the CDSDoc “Docs by Product” cyclic field, open the *cdsdoc.ini* file in the *.cdsdoc* subdirectory of your login directory, and add the paths to the install path for each tool set that you want.

```
//cdsdoc.ini file example
DocDir1=<install_path_for_MMSIM6.1>
DocDir0=<install_path_for_IC6.1.0>
```

- Other useful manuals include:

  Virtuoso Unified Custom Constraints User Guide

  Virtuoso Schematic Editor L User Guide

  Virtuoso UltraSim Simulation User Guide

  Spectre Circuit Simulator User Guide

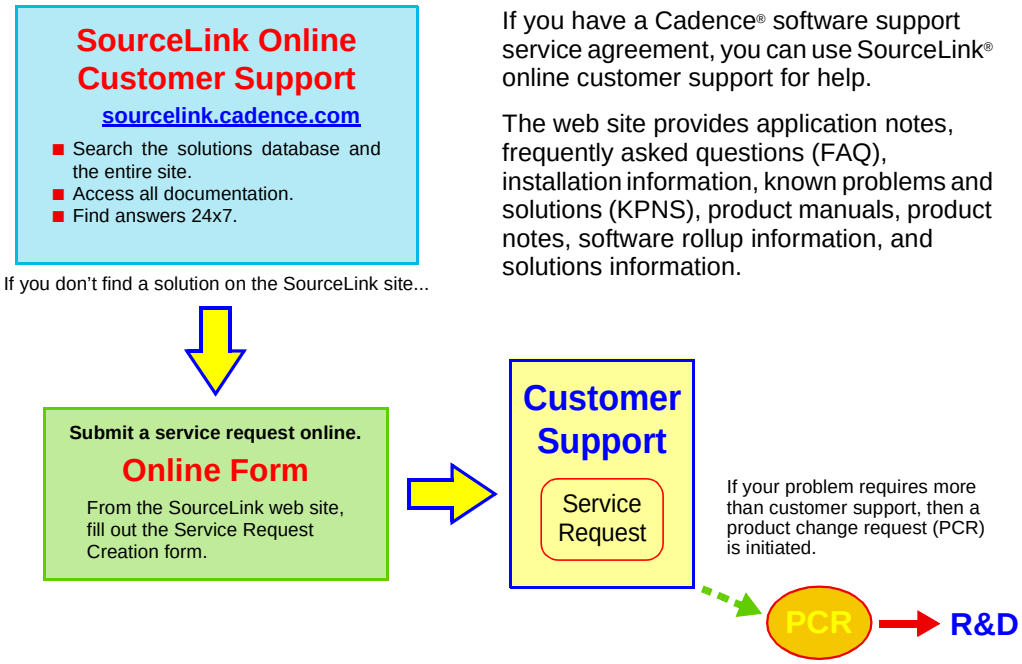
  WaveScan User Guide

Getting Help: Help Menus

Each Virtuoso window has a Help menu item that will open the appropriate online documentation reference in a browser window.

In Linux, remember to open Mozilla first.

Getting Help: SourceLink Online Support



If you have a Cadence® software support service agreement, you can use SourceLink® online customer support for help.

The web site provides application notes, frequently asked questions (FAQ), installation information, known problems and solutions (KPNS), product manuals, product notes, software rollup information, and solutions information.

To view information in SourceLink® online customer support:

1. Point your web browser to sourcelink.cadence.com.
2. Log in.
3. Enter search criteria.

   You can search by product, release, document type, or keyword. You can also browse by product, release, or document type.

   For example, you can select Analog Design Environment as the *Product* and then refine the search to any particular topic.

Course Data and Intended Use

- The Ethernet-Phy data presented in the labs is based on the Cadence AMS Methodology Kit's segment representative design.
- The materials and data are solely intended to provide the foundation for learning the new features and changes in IC 6.1.
- It is intended for the student's personal use only.
- It is intended to only be used with Cadence IC 6.1 software.
- For more information about terms of use, see the README file in the course database.

Lab

There is no lab for this module.

Design Framework II Environment

Module 2

December 14, 2006

Module Objectives

In this module, you will

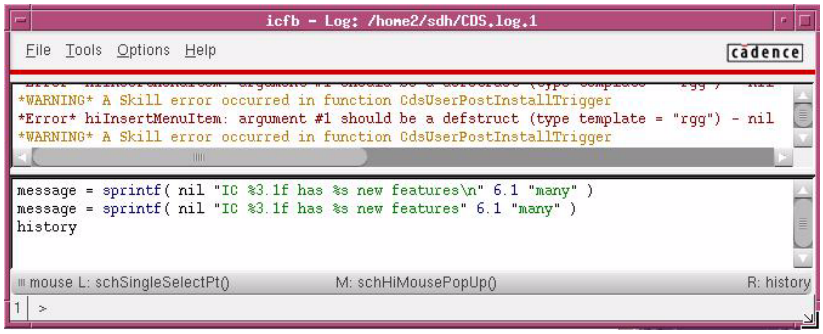
- Try the new features for the Command Interpreter Window (CIW)
- Use the improved cdsFinder and CDSDoc search utilities
- Understand toolbar customization
- Review planned changes to starting the Virtuoso® workbench (*.exe* files)
- Examine local customization files (*.cadence* directory)
- Understand the unified technology file implementation
- Preview the incremental technology database (ITDB) model and impact
- See the Virtuoso Custom Design Cockpit
- Examine the token licensing model
- Examine the database features for Cadence® Design Framework II on OpenAccess (OA)
- Examine platform requirements and the *checkSysConf* utility

Terms and Definitions

OA	OpenAccess database that the IC 6.1.0 environment is based on
DFII	Cadence Design Framework II, the base environment for Virtuoso applications
CIW	Command Interpreter Window
Assistant	Frequently used interface that can be docked around a drawing canvas
Toolbar	Horizontal banner of icon buttons on application windows to provide access to common functions
Bookmark	Saved cellview reference for access through a bookmark menu
ITDB	Incremental Technology Data Base, which provides incremental technology file loading and access
Virtuoso Layout Suite	A collection of tools for layout design
Token	License option to allow flexible access to a suite of tools

CIW Improvements

- Expandable, multi-line input area
- Auto-highlighting of SKILL® keywords, strings, and constructs
- CDSDoc search facility is available on document pages from the Help menu
- Recently opened design windows are listed in the File menu
- Bookmarks of cellviews are listed in the File menu
- Warnings and error messages are color-coded
- Command history can be displayed and commands can be re-invoked
- CIW is automatically raised on an error or warning message



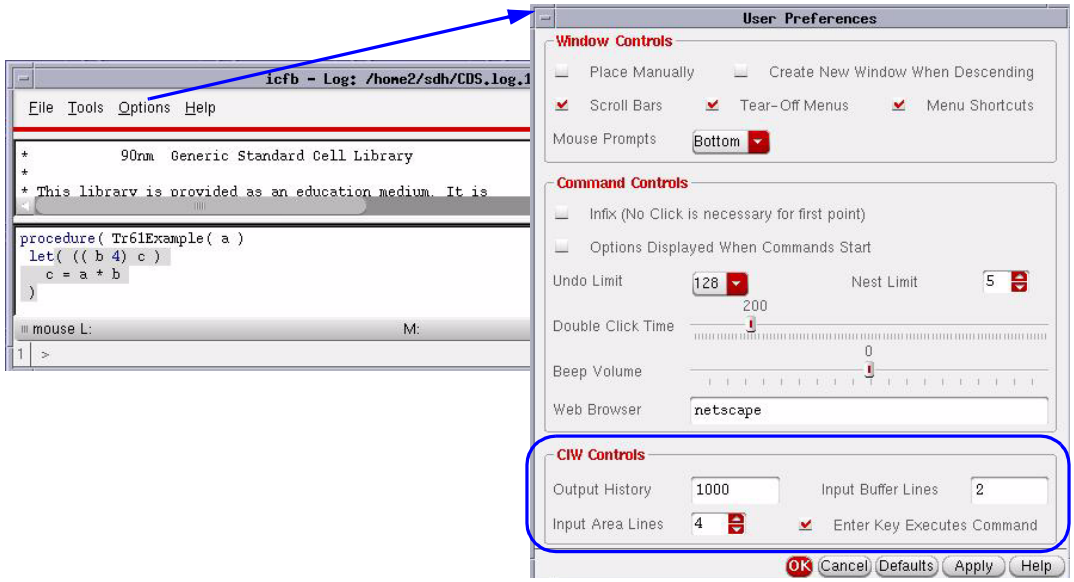
New environment variables (*.cdsenv*) to raise the CIW on an error or warning message:

```
ui raiseCIWonError boolean t
ui raiseCIWonWarning t
```

The default value is *nil*, which means the CIW will not be raised on a error or warning message.

Expandable, Multi-line Input Area

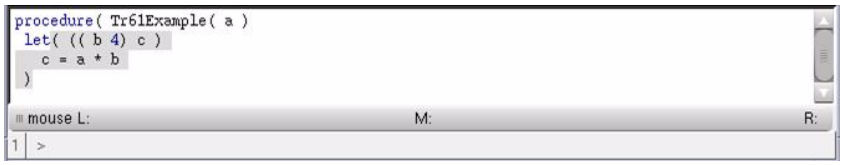
- Set the number of input area lines in the User Preferences form -OR- graphically stretch the input window using the mouse
- Set the number of input buffer lines in the User Preferences form. This specifies the number of previous commands that appear in the input area that may be edited.



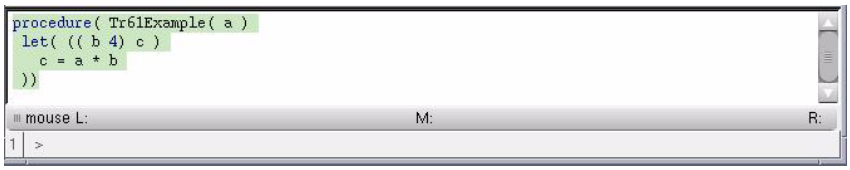
Auto-highlight of Keywords, Strings and Constructs

- SKILL keywords are highlighted in blue and strings in green.

- SKILL code groups between parens are highlighted in gray as you type or by clicking on a paren.



- Complete SKILL procedures are highlighted in light green as you type or by clicking on a paren.



Design Framework II Environment 2-9

To turn OFF auto-highlighting add the following line to your *.cdsenv* file:

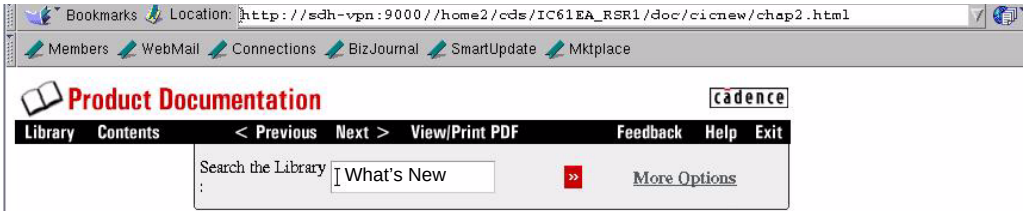
```
ui ciwSyntaxHighlighting boolean nil
```

To control the highlight colors there are *.cdsenv* settings under the *ui* category as follows:

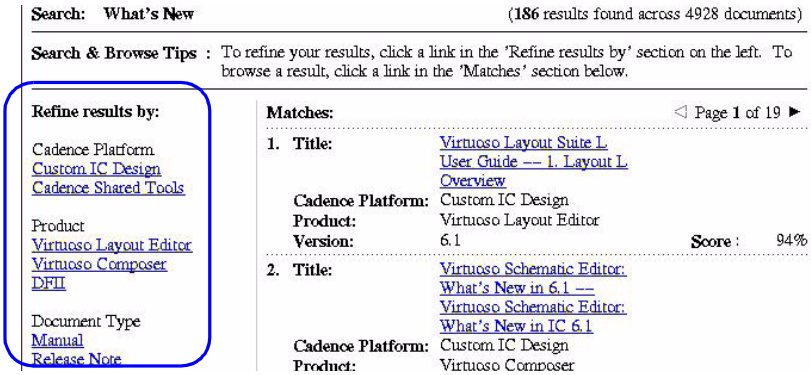
```
uiciwWarnColor string "#b87b00"
uiciwErrorColor string "dark red"
uiciwMatchParenColorstring "#dcdcdc"
uiciwMismatchParenColorstring "red"
uiciwMatchCmdColor string "#cce8c3"
```

CDSDoc Search Facility on Document Pages

- The search bar appears on the top of the document pages. You can type the word you want to search in the Search the Library field and then click to search the Cadence documents.

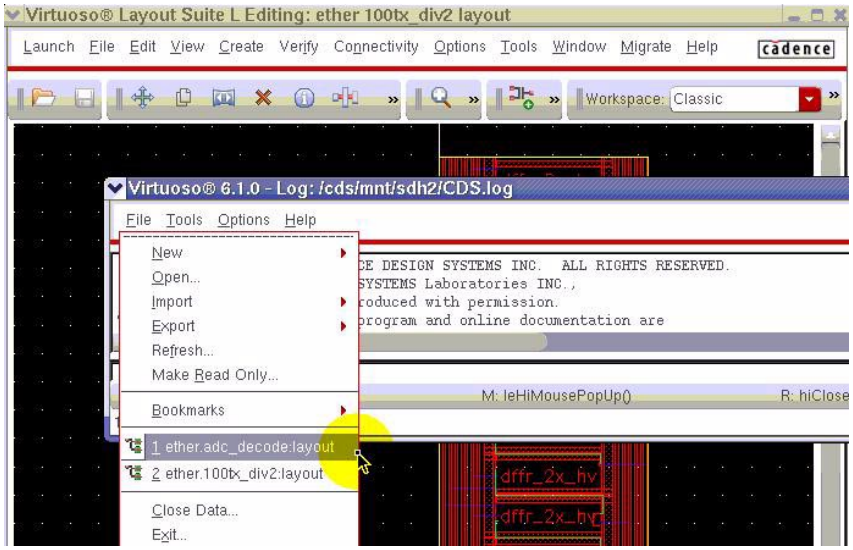


- You can also refine your search by selecting a specific Cadence platform, product, or document type under the *Refine results by* column.



Recently Opened Windows in File Menu

- The list of most recently opened cellviews is located in a history file and appears in the File menu.
- The oldest item falls off the menu when the maximum length is reached.



You can control the number of files in the recently-used-menu list from the *Recently used file list* form (**Options—File Preferences**) by setting the number of entries to the desired value.

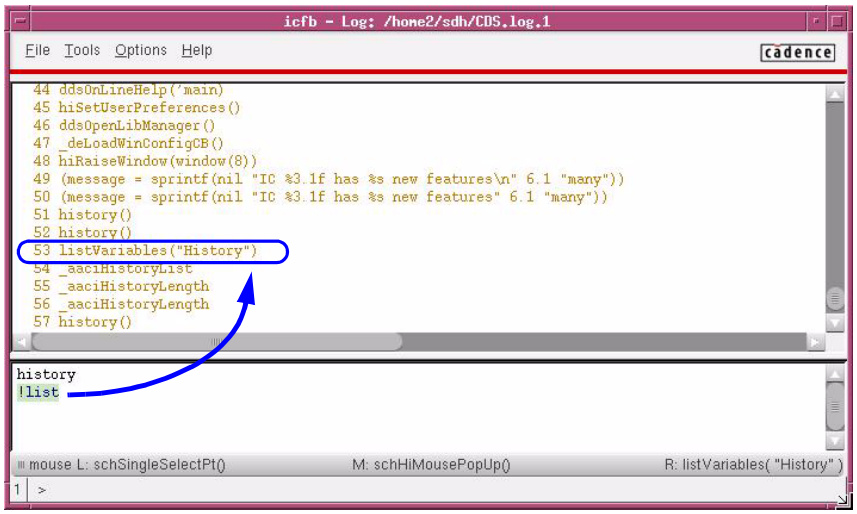
Also, you can place an entry in the *.cdsenv* file, for example:

```
ddserv fileHistoryLimit int 2
```

History Command

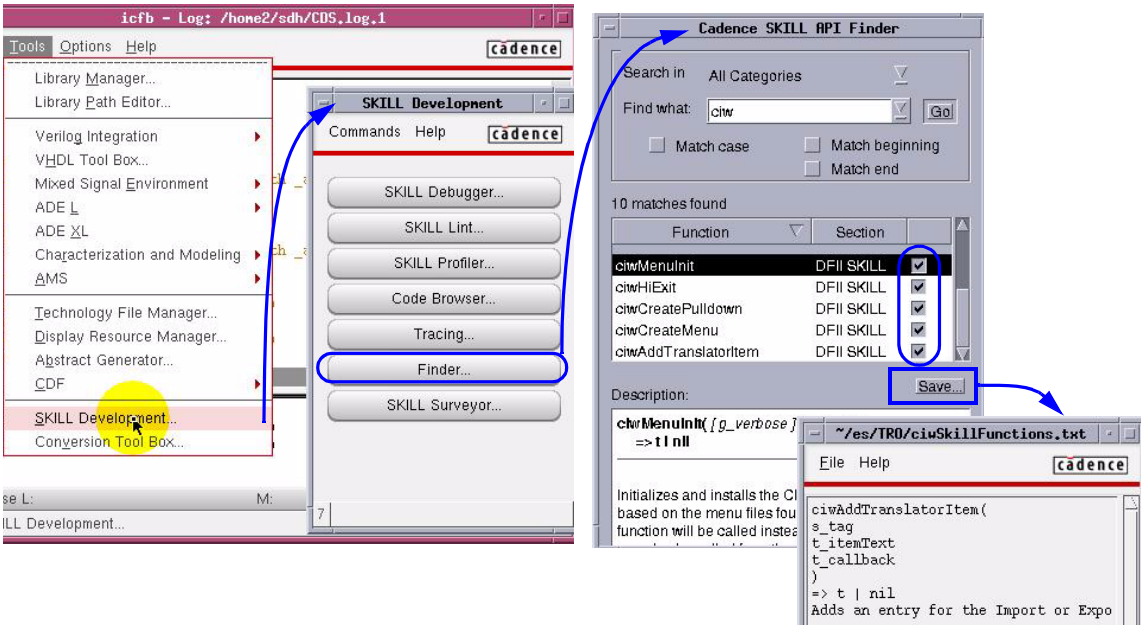
- Enter *history* in the CIW to see the most recent commands in the buffer.
- To re-invoke a command enter *!* and a string of unique command characters

  If the previous command is *listVariables("History")*, re-invoke with *!list*.
- To change the number of commands stored in the history buffer, enter (for example) *_aaciHistoryLength = 100*.



SKILL Finder Search and Save

- Utility to look up *SKILL Quick Reference* guide definitions.
- Search for SKILL functions and sort results by name or section.
- Select specific function definitions to save to a file.



SKILL Finder comments:

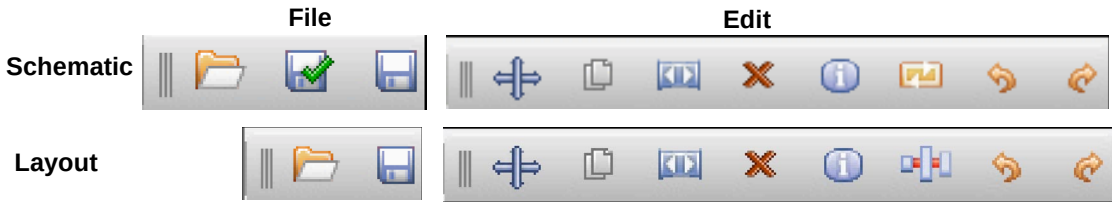
- You can use regular expressions in addition to the buttons on the form.
- Use ^ for start
- Use *$* for end
- Use .* for any number of characters
- For example, to locate all functions that start with *le* (Layout Editor functions) and have the word *Pin* in them use:

  ```
  ^le.*Pin
  ```

- The **Save** button will not append to a file, but overwrite it. You are prompted before the overwrite can occur.

Toolbar Customization

Toolbars provide consistent icon menus across windows and applications. They are available for all applications. Some examples are:



You can customize the toolbars by modifying a definition file. The default toolbar definition file is located at:

your_install_dir/share/cdssetup/dfII/toolbars/byApplication/*appName*.toolbars

Here is an example from the *Layout.toolbars* file:

```
( ( nil
  name leFileToolbar
  text "File"
  items (
       ( *** item1 definition***)
       ( *** item2 definition***)
)) ...
```

Toolbar Customization Example

The toolbar name (such as *leFileToolbar*) must be unique across all applications. The text the program uses for the toolbar title and tooltip text (such as *File*) need not be unique.

```
(( nil
name leFileToolbar
text "File"
  items (
  (
      nil
      type action
      name leFileToolbarOpen
      text "Open"
      iconFile "file-open.png"
      callback "deFileOpen()"
      disabled t
      )
      (
      nil
      type action
      name leFileToolbarSave
      text "Save"
      iconFile "file-save.png"
      callback "geSave()"
      enableCondition modified
))...
```

Toolbar Customization File Definition

Main Toolbar

Mandatory fields: name, text, **items**

Optional Fields: Invisible, toolButtonStyle *(textOnly, iconOnly, textBesideIcon, textUnderIcon)*

Item Type action

Mandatory Fields: type (action), name, text, callback, iconFile *(.png format)*

Optional Fields: subAction, disabled, checkable, checked

Item Type comboBox

Mandatory Fields: type (comboBox), name, toolTip, items, callback, width

Optional Fields: disabled

Item Type typein

Mandatory Fields: type (typein), name, toolTip, callback, value, width

Optional Fields: disabled

Item Type separator

Mandatory Fields: type (separator), name

Item Type inheritToolbarsFrom

Mandatory Fields: inheritToolbarsFrom (application)

Important

The iconFile is looked up in the standard setup.loc locations prefixed by the path icons/24x24. The standard locations include <installdir>/share/cdssetup, ./.cadence and ~/.cadence.

A complete explanation of each field can be found in the *Virtuoso Design Environment User Guide*. See the “Customizing Toolbars” section of the “Customizing Your Design Environment.”

Planned Changes to Starting the Virtuoso Workbench

Approach

- Reduce the number of workbenches from seven to one new workbench, *virtuoso*, containing the cleaned-up content from *icfb*. Planned in phases, *icfb* replaced first.
- Concentrate testing, profiling and tuning on one workbench

Benefits

- Less confusion for all
- More effective, focused testing and tuning

What will change in IC 6.1.0?

- You will invoke the Virtuoso software with a single command —*virtuoso*.
- For the next year *virtuoso* will be linked to the *icfb* executable files.

In the IC 6.1.0 release, the *icfb* workbench is gone, and it is replaced by a link to the new *virtuoso* workbench.

This link will stay in place for a year, so tests and scripts will still work, but should be changed as soon as time permits.

Current workbenches

- Layout only
  - layout (43MB)
- Front end only
  - icde (36MB
  - icds (39MB)
  - icms (67MB)
- Layout and front end combined
  - layoutPlus (71MB)
  - msfb (104MB)
  - icfb (104MB)

Customization Using the .cadence Hierarchy

The *.cadence* hierarchy contains local customization for files that Cadence provides in the *your_install_dir/share/cdssetup/dfII* hierarchy. For each location specified in the *setup.loc* file, the program first searches for *.cadence* directories.

Your local *.cadence/dfII* directory can contain the following files and subdirectories:

File or Directory	Description	Notes
history/*userName*.history	File: History tree for Data assistant for Virtuoso Analog Design Environment XL	Do not edit
Navigator	Directory: contains *Options.xml* file for Navigator	Do not edit
Toolbars	Directory: custom toolbar definition files	
Workspaces	Directory: Created and updated when you save a custom workspace. It may contain several application specific subdirectories with several *.workspace* files.	*.workspace* binary files cannot be edited
workspace.default	File: the default workspace you specified	
jobpolicy	Directory: job policy definitions	
layerSets/*technology*	Directory: LSW layer sets you save per technology	

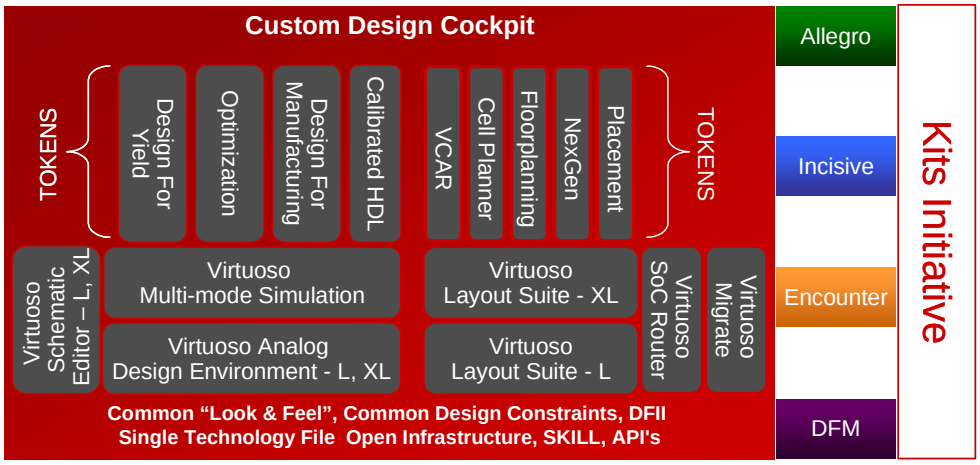
Important

Each *.cadence* directory contains a *cadence.signature.xml* file which you must not edit or remove.

Files and directories in the *your_install_dir/share/cdssetup/dfII* hierarchy include:

File or Directory	Content
Virtuoso Layout Suite GXL	*Modgen.patterns* file for Module Generator
cds.lib	Default library definitions
ci	*config.xml* file defining constraint information
default.drf	Default display resource settings
workspaces	Application subdirectories containing Cadence-provided *.workspace* definition files and *workspace.default* file
toolbars	Application subdirectory of Cadence-provided *.toolbars* files.

Virtuoso Custom Design Cockpit



Three tiers of products that provide functionality matched to your tasks.

L = Good: Core functionality needed for primary design tasks

XL = Better: All L functions PLUS productivity enhancements

GXL = Best: All XL functions PLUS tokenized access to advanced tool features

A token is a license feature that allows flexible access to a suite of tools.

Why Tokens?

- Provide easier access to the breadth of technology Cadence provides.
- Successfully piloted with MMSIM (suite of Cadence simulators).
- You can access more advanced capability which requires more tokens by temporarily reducing usage of other features.
- You have control over when and how often to use the advanced features.
- When advanced features are in production use or more users need the GXL features, then you can purchase additional tokens.
- It is easy to try out new features without needing temporary keys.

Token Licenses

The table shows token licenses for Virtuoso Analog Design Environment GXL and Virtuoso Layout Suite GXL.

GXL Feature	Tokens	GXL Feature	Tokens
GXL Cockpit	4	Chip Editor	4
Digital Placer	2	Analog Placer	8
Modgens	2	Layout Optimize	8
Cell Planning	4	Chip Assembly Router	8
Floor Planning	4		

- Each primary feature requires a predefined number of tokens.
- Each cockpit is a prerequisite for the other features.
- Virtuoso Layout Suite example: Digital Placer requires six tokens (four for the Cockpit plus two for the Placer).
- A planned Token Manager utility will allow you to manage the tokens.
- Token quantities subject to change, token amounts are only an example.

The Token Manager utility will help answer questions like:

- How can I tell how many tokens are available?
- How can I tell how many tokens a tool will use?
- How can I tell how many tokens I am currently using?
- How can I restrict a certain number of tokens to a specific user?

If you want to free up tokens you can close the utility or feature you are using. The task manager in Virtuoso Layout Suite (LS) can help with this.

If you want to switch back to Virtuoso LS XL from Virtuoso LS GXL and free up tokens you will need to use the License Manager to free up the Virtuoso LS GXL cockpit license.

Single Technology File

- Consolidation of all separate tool-specific technology files into a single composite technology file.
- Support for OA objects
  - Vias and viaDefs
  - Sites
  - Tracks
- Includes sections for constraints (convert rules from CDB techfile)
  - Group technology rules hierarchically
  - Tools can use constraint groups
  - Table-based “nanometer” rules
- Will include Incremental Technology Database (ITDB) file management


```
; Technology File gpdk090
;*** CONTROLS ***
   controls( techParams(...) viewTypeUnits(...) mfgGridResolution(...) )

;*** LAYER DEFINITION ***
   layerDefinitions( techPurposes(...) techLayers(...) techLayerPurposePriorities(...)
   techDisplays(...) techLayerProperties(...) techDerivedLayers(...) )

;*** LAYER RULES ***
   layerRules( equivalentLayers(...) functions(...) routingDirections(...) labelLayers(...)
   stampLabelLayers(...) currentDensity(...) currentDensityTables(...) )

;*** VIADEFS ***
viaDefs( standardViaDefs(...) customViaDefs(...) )

;*** CONSTRAINT GROUPS ***
   constraintGroups(
   ("LEFDefaultRouteSpec"... interconnect(...) routingGrids(...) viaStackingLimits(...)
   spacings(...) )
   ("skip_a_track" ... spacings(...) interconnect( (validLayers (...) validVias( (...) )
   ("wide" ... spacings( ...) interconnect(...) )
   ("virtuosoDefaultSetup" ... interconnect(...) spacings(...) )
   ("foundry" ... spacings(...) spacingTables( (minSpacing ...)) viaStackingLimits(...)
   antennaModels(...) electrical(...) )

;*** DEVICES ***
tcCreateCDSDeviceClass() ruleContactDevice(...) tfcDefineDeviceProp)(...)
multipartPathTemplates(...) )

;*** LE RULES ***
leRules( leLswLayers(...) )

;*** siteDefs ***
siteDefs(...)

;*** VIASPECS ***
viaSpecs(...)
```

ITDB Overview

- Supported by OA specification
- Will impact many parts of the environment and applications
- No longer a single monolithic text file (IC 5.1.41 CDB and prior releases)
- Now multiple text files can be configured and loaded to produce an “effective” techfile in virtual memory.
- Tool features that reference technology data will need to understand which file to reference (from those that make up the effective techfile).
- Tool GUIs and techfile access will need updates to assure a comprehensive solution.

ITDB Capability

ITDB is a native OA 2.2 capability, developed in cooperation between Si2 and Cadence to enable:

- Managing complex process options
- Layer options (masks)
- Optional devices
- Giving the appropriate person (team) the rights to manage only the specific section of technology information that is relevant
- Technology information to be stored outside of the base process information
  - Tool set-up
  - *siteDef* data for standard cells
  - Place-and-route vias

ITDB Implementation

- There is still a one-to-one correspondence between a technology library and a technology database.
- **However**, a technology library can inherit, through an ordered set of references, information from other technology libraries.

ITDB Conflict Management and SKILL Access

Conflict Management

- Each type of object has a predefined set of conflict detection rules associated with it. For example:
  - Layers CANNOT be redefined in a technology graph (same name or number)
  - The VIADEF name must be unique in a graph
- DFII provides support from SKILL

```
techId~>refLibs (set/get)
techId~>refLibNames (get, when the graph is bound)
techId~>hasConflict
```

ITDB Setup

- No setup or PDK modification is needed if you do not want to take advantage of ITDB
- CAD group
  - The basic technology file must refer to the other libraries via a *refLib* construct in the *control* section

```
controls(
refLibs("stdCell" "metalization1" "base)
)
```

  - This can be a nested process (for example, one of the libraries in the *refLibs* section could have a *refLibs* section pointing to other libraries)
  - The libraries referenced must be listed in the *cds.lib* file (explicitly via a DEFINE statement or via an INCLUDE statement)
  - There is not a single best way to do that. It depends on the requirements of each company. Flexibility is one of the major strengths of the ITDB implementation.

ITDB Use Model — Attachment Versus Inheritance

When end users create a new library they can either specify to “attach” to or to “inherit” from an already existing technology library.

- **Attachment**: the fact that the technology database is incremental is totally transparent to the final users. No addition or modification can be made to the technology database.
- **Inheritance**: end users inherit all the definitions resulting from the union of the various technology databases. Moreover, because a local *TechDB* is created, they will be able to add new definitions to the technology database.

The GUI of the **Create Library** command in DFII has been enhanced as has now **Inherit from an Existing Technology File** as one of the possible choices.

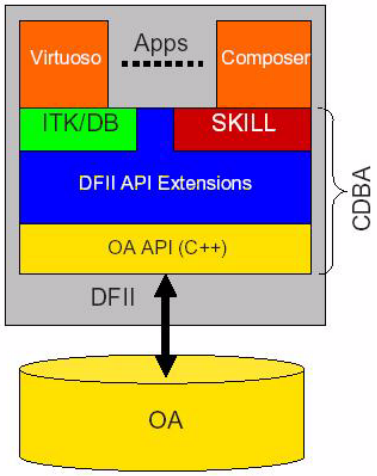
New Database Model: DFII on OpenAccess 2.2

What is DFII on OA 2.2?

- Applications, public interfaces, CDBA, OA 2.2 API + extensions

- DFII API extensions
  - Private extensions to the OA API
  - Retain CDBA features
    - LPP definitions for proper display
    - SKILL interface
    - ITK/DB C interface
  - Support new DFII/CDBA features (such as region query)

- DFII is implemented natively on OpenAccess with the use of custom extensions to retain or add support of unique features of DFII-based tools.



Virtuoso and OA Objects

- OA objects enable interoperability between tools

  Example:

  - In Custom IC, Create Wire uses pathSeg and via objects for routing connectivity.
  - In IC Digital, the Encounter tools interpret the OA objects as routing elements.
  - So, no LEF or DEF translations are required.

- Virtuoso tools on CDB (IC 5.1.41 and earlier releases)
  - All layout elements were shapes on layers of different types/purposes.
  - Any shape could be created on any layer and assigned any purpose.
- Virtuoso tools on OA 2.2 (IC 6.1.0 and future releases)
  - Most layout elements are first class objects handled directly by the OA API.
  - Layer-purpose pairs (LPPs) are now for DFII display capability only.

Platform Requirements and checkSysConf File

Platform	OS Version	Memory/Swap	Display	Comment
SunOS/Sparc	Sol 5.8/5.9/5.10	128/200 MB	24 TrueColor	
SunOS/i386	Sol 5.10	128/200 MB	24 TrueColor	PC platform
Linux/i686	RH 3.0/4.0 WS	256/512 MB	24 TrueColor	
Linux/x86_64	RH 3.0/4.0 WS	256/512 MB	24 TrueColor	
IBM	AIX 5.3.0.0	128/200 MB	24 TrueColor	

The above information is extracted directly from the *checkSysConf* data files located in the Cadence install path at:

```
<installpath>/share/patchData/<platform>
```

checkSysConf and associated documentation can be obtained at:

```
sourcelink.cadence.com/docs/files/releases/sys_conf_check/welcome.html
```


X86_64 denotes 64-bit processors whose instruction sets are compatible with the X86 standard such as Intel Xeon-EM64T and AMD Opteron.

Labs

Lab 2-1 Trying the New Environment Features

Lab 2-2 Examining the gpdk090 Technology File

Lab 2-3 Modifying a Toolbar Menu

Schematic Editor L

Module 3

December 14, 2006

Module Objectives

This module describes the new tiered structure for the Virtuoso® Schematic Editor tool. It also covers the new functionality and user interface changes associated with the first tier, Virtuoso Schematic Editor L.

Module 4, Schematic Editor XL, describes the functionality and user interface changes for the second tier.

The third tier, Virtuoso Schematic Editor GXL, is not yet available.

In this module you will:

- Learn about the good-better-best tiers in the Schematic Editor
- Examine the new features in Schematic Editor L:
  - Navigate the banner menus
  - Try the tear-off menus
  - Use tabs within a session window
  - Manipulate toolbars within a session window

Terms and Definitions

DFII	Cadence® Design Framework II is the base environment for Virtuoso applications
CIW	Command Interpreter Window
Assistant	Frequently used interfaces that can be docked around a drawing canvas
Toolbar	Horizontal banner of icon buttons on application windows to provide access to common functions
Bookmark	Saved cellview reference for access through a bookmark menu

Tiers in Schematic Editor

- In IC 6.1.0, Virtuoso Schematic Editor is available in two tiers:
  - The basic tier called Virtuoso Schematic Editor L
  - The advanced tier called Virtuoso Schematic Editor XL
- The two tiers are separately licensed.
- The default tier level can be set for Virtuoso Schematic Editor and Virtuoso Symbol Editor as follows:
  - From the GUI, use **File—Set Default Application** from the Schematic Editor or Symbol Editor. A dialog box comes up for selection:

  - From the *.cdsenv* file, use the following entry:

```
graphic schematicDefaultTier string "XL"
```

Schematic Editor L 3-5

License feature details:

- XL: The license feature is called Virtuoso_Schematic_Editor_XL. The feature number is 95115.
- L: The license feature is called Virtuoso_Schematic_Editor_L. The feature number is 95100.

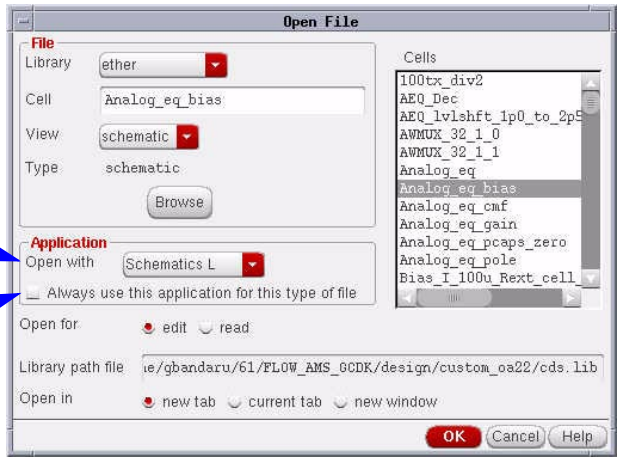
IC610 licenses will be backward compatible with the IC5141 stream starting with IC5141 USR4. Users running IC5141USR4 or later can use IC610 licenses to run their applications.

Tiers in Virtuoso Schematic Editor (continued)

- The default tier level can also be set based on the view type.

Use this form field to choose an application — say Schematic XL — when opening the currently chosen cellview

Enable this button if you want to default to the chosen application for any cellview of the same viewType as the one currently chosen



- Alternatively, use the following entries in your *.cdsenv* file. Examples:

```
graphicschematicDefaultAppstring "Schematics XL"
graphicschematicSymbolDefaultAppstring "Symbol XL"
graphicmaskLayoutDefaultAppstring "Layout XL"
```

Schematic Editor L 3-7

This method of setting default tier level based on view type will work when

- Opening a cellview using **File—Open** from a session window
- Opening a cellview using **File—Open** from the CIW
- Opening a cellview using **File—Open With** from the Library Manager

Consolidated Pull-down Menus

- Menus and menu items have been renamed to resemble those found on the familiar software applications available on personal computers.
- Virtuoso Schematic Editor and Virtuoso Layout Suite now share a common menu structure as much as possible.
- As a result of this consolidation effort, there is some rearrangement of menus and menu items. Examples:
  - The **Tools** and **Design** menus are removed.
  - New menus called **Launch** and **File** are added.
  - The **Design—Create Cellview—From CellView** command is now **Create—Cellview—From CellView**.
  - **Window—Close** is now **File—Close**.
  - The **Sheet** menu is removed and its menu items are distributed among the **Create** and **Edit** menus.

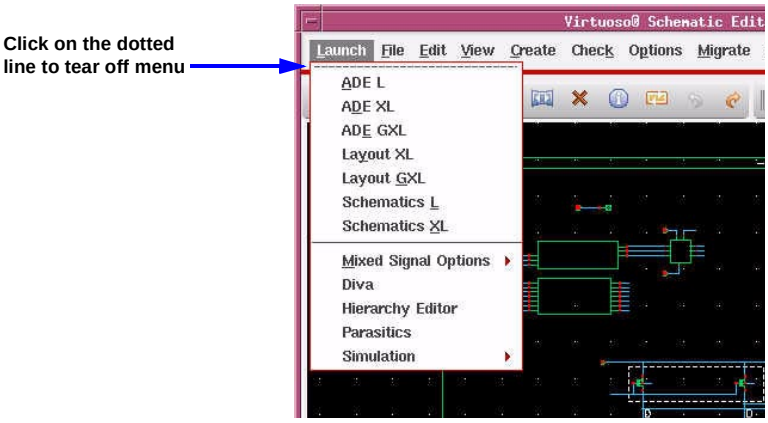
Schematic Editor L 3-9

Refer to *Virtuoso Schematic Editor: What's New in 6.1* for details on *New Menu Structure* in Virtuoso Schematic Editor L and Virtuoso Schematic Editor XL.

Tear-off Menus

In IC 6.1.0, a menu can be torn off from its parent window.

- This feature is useful for displaying a menu that you wish to use repeatedly.
- Both banner menus and slider menus can be torn off.
- A torn-off menu retains its name



Tear-off Menus (continued)

- The tear-off capability is enabled by default. It can be toggled as follows:
  - With the *.cdsenv* variable:

```
ui        tearOffMenus     boolean t
```

  - From the User Preferences form (**CIW: Options—User Preferences**):

- Tear-off menus works on banner menus and slider menus of any given session window.

Schematic Editor L 3-13

Updates done to the Tear-Off Menus option do not effect the session windows already open when the update is done.

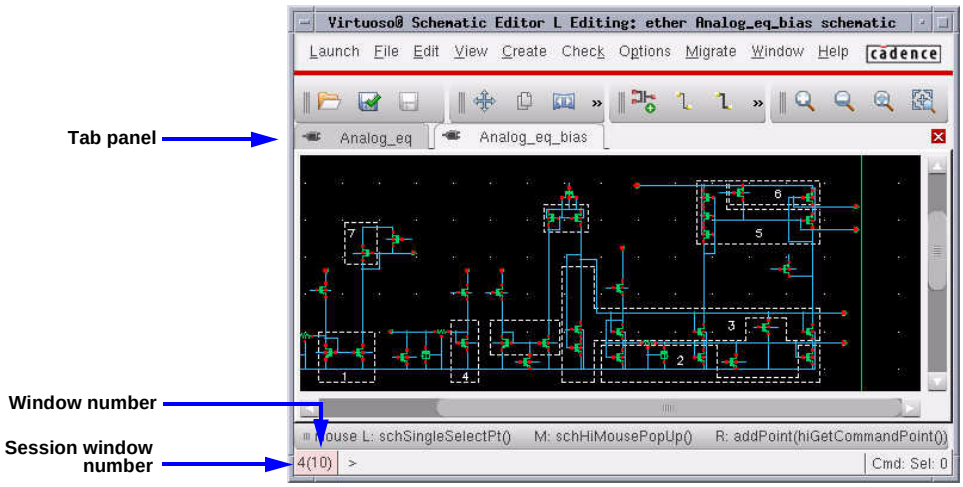
Tabs

- IC 6.1.0 allows multiple design editor windows to be displayed in separate tabs in a given *session window.*
- Tabs are one or more windows displayed within a given session window.
- What is a session window?
  - IC 6.1.0 introduces a new type of window called session window
  - A session window is a *container* window that contains design editor windows as well as their assisting sub-applications (dock assistants)
  - A session window, referred to as an *swindow*, consists of
    - A tabbed main canvas area that contains one or more windows displayed in separate tabs. These windows can be of the following types: graphics, text, hypertext, HTML, form, or encap.
    - Dock areas along its four edges, each of which can contain multiple dockable windows
    - Toolbar docking areas along the four edges, each of which can contain multiple toolbars
    - A menu bar. The menu bar cannot be docked or floated.

Tabs (continued)

- Windows in session windows appear in tabs and are also referred to as *tab windows*.
- A session window has an identifier, *swindowID*. Its SKILL® output is: *swindow:#*. Its number is displayed in the GUI as *swindowNumber(windowNumber)*.

  Example: *4(10)*
  where *4* is the session window number is *10* is the window number



Schematic Editor L 3-17

- In SKILL, create a session window with *hiCreateWindow()* and display it with *hiDisplayWindow()*.

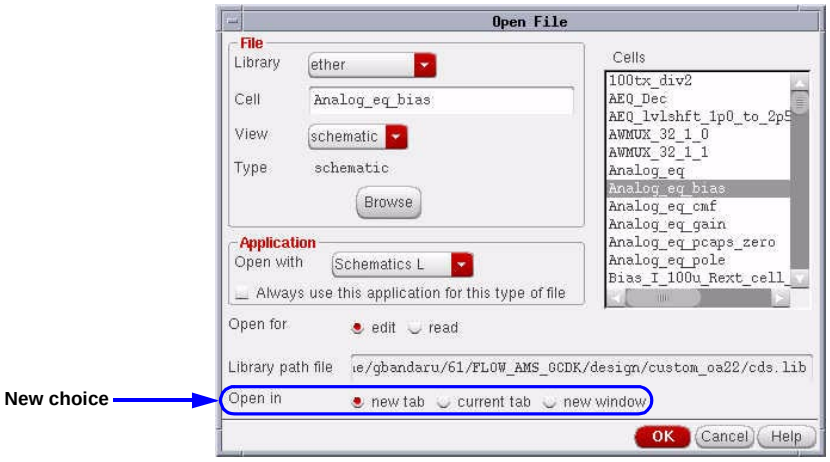
  An empty session window cannot be displayed. At least one window must be associated with any given session window.
- You can get a handle to the session window having your design window using the new SKILL function *hiGetSessionWindow()*.

```
hiGetSessionWindow(w_windowID)
=> w_sessionWindowID | nil
```

Tabs (continued)

When opening a new cellview (**File—Open** from a session window), you have a choice to open it

- In a new tab
- In the current tab
- In a new session window



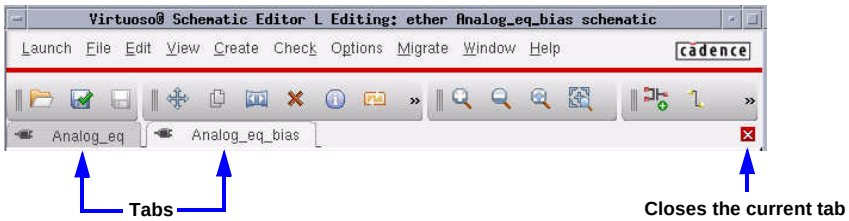
This choice for opening windows is not available when opening a cellview from the CIW **(File—Open**) or from Library Manager. These actions always open a new session window.

Tabs (continued)

A tab can be closed by

- Making the desired tab window current and choosing **File—Close** from its container session window
- Right-clicking on the tab and choosing **Close Tab**.
- Making the desired tab window current and clicking the **x** at the right end of the tab banner

The tab panel becomes available only when there is more than one design displayed in a given session window.



Toolbars: General Information

- Toolbars in IC 6.1.0 replace the fixed menus in the earlier releases.
- Toolbars are container objects that can contain the following:
  - Combo boxes
  - Type-in text boxes
  - Buttons
    - Icon or text buttons
    - Toggle buttons
    - Buttons with drop-down lists
  - Separators
- Toolbars can be added to all types of windows, including session windows and dockable windows.
- Session windows have four toolbar dock areas—one along each side of the window. You can drag a toolbar by its handle and dock it in any of these areas.
- A toolbar cannot be floated. It has to be docked to a window.

SKILL API for toolbars:

- There is a list of SKILL functions available to define your own toolbars, attach your toolbar to a given window, and do relates tasks.
- Enter *listFunctions("Toolbar")* in the CIW to get the list of functions. These functions are documented in *Cadence User Interface SKILL Functions Reference Manual*.

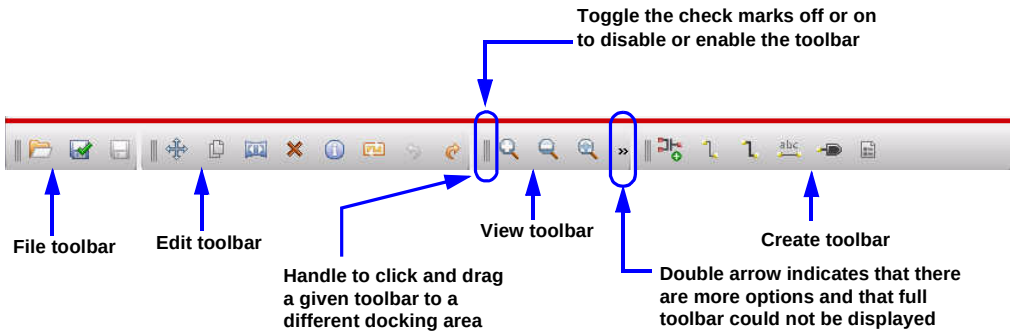
  Some of the toolbar SKILL functions are listed below. All of the toolbar SKILL functions are described in the Cadence Documentation.

```
hiCreateToolbar()              hiDeleteToolbar()
hiShowToolbar()                hiHideToolbar()
hiInsertToolbar()              hiGetWindowToolbars()
hiPlaceToolbar()               hiAddToolbarItem()
hiAddToolbarItems()            hiCreateToolbarTypein()
hiCreateToolbarComboBox()      hiInsertToolbarItem()
hiInsertToolbarItems()
```

- You can also create toolbars with toolbar files. See the *Virtuoso Design Environment User Guide* for information about toolbar files.

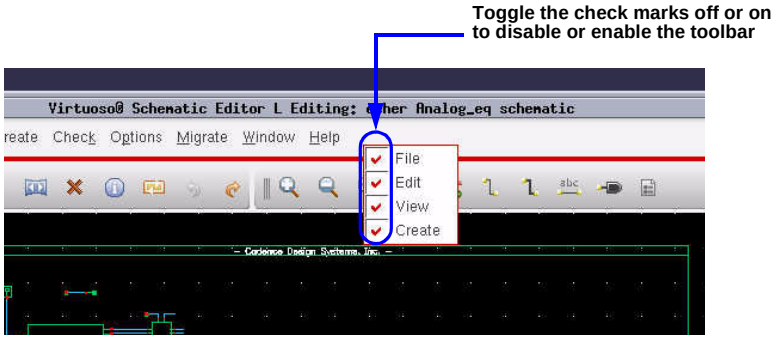
Toolbars in Schematic Editor L

- Four toolbars are provided in the Virtuoso Schematic Editor L environment. All are enabled by default. They are **File**, **Edit**, **View** and **Create**.
- These toolbars correspond with the banner pull-down menus with same names
- You can drag a toolbar by its handle and dock it to any of the four toolbar dock areas of a given window
- When the mouse pointer is on a toolbar handle, the cursor changes from an arrow to a *move* cursor. 



Toolbars in Virtuoso Schematic Editor L (continued)

- When you right-click on a toolbar, a pop-up menu appears that lists all the toolbars that are currently installed in any of the four toolbar dock areas of the window.
- If a toolbar is hidden, it is unselected in the menu. Selecting it makes it visible and places it in its former position in the toolbar area.
- Bubble text on toolbars:
  - Placing the mouse cursor on a toolbar handle pops up that toolbar's name.
  - Placing the mouse cursor on an icon within a toolbar pops up the command name that the icon is associated with.



Schematic Editor L 3-27

A right-click anywhere on the banner menu in a session window brings up the same pop-up menu that comes up with a right-click on any toolbar.

Labs

Lab 3-1 Consolidated Banner Menus

Lab 3-2 Using Tabs

Lab 3-3 Introduction to Toolbars

Schematic Editor XL

Module 4

December 14, 2006

Module Objectives

This module covers the functionality and user interface changes associated with Virtuoso® Schematic Editor XL, the second tier of the Virtuoso Schematic Editor tool.

Module 3, Schematic Editor L, covers the first tier.

In this module you will:

- Examine new Schematic Editor XL features that build on top of the Schematic Editor L tool:
  - Navigate cellview hierarchy using the Go toolbar
  - Define and arrange workspaces
  - Keep track of your design windows using bookmarks
  - Try the variety of new assistants
    - Locate hierarchical design information with Navigator assistant
    - Find design data and attributes with the Search assistant and toolbar
    - Locate and edit object properties with Property Editor assistant
    - Use the dockable World View assistant
    - Edit, add and delete constraints using the Constraint Manager assistant
    - Accelerate constraint entry using Circuit Prospector assistant

Terms and Definitions

DFII	Cadence® Design Framework II is the base environment for the Virtuoso applications
CIW	Command Interpreter Window
Assistant	Frequently used interface that can be docked around a drawing canvas
Toolbar	Horizontal banner of icon buttons on application windows to provide access to common functions
Bookmark	Saved cellview reference for access through a bookmark menu

Schematic Editor XL

Virtuoso Schematic Editor XL provides the following additional features to what is available through the Virtuoso Schematic Editor L

- Go toolbar
- Workspaces (with toolbar interface)
- Bookmarks (with toolbar interface)
- The Navigator assistant, which provides an alternative means of viewing schematic objects
- An interactive search facility using the Search assistant and toolbar
- The Property Editor assistant, which can be used to display, organize and edit single or multiple object properties
- The World View assistant, which allows you to quickly navigate around the design canvas area
- The Constraint Management assistant, which allows for the addition, modification, and deletion of constraints
- The Circuit Prospector assistant, which helps accelerate the constraint entry process

Schematic Editor XL 4-5

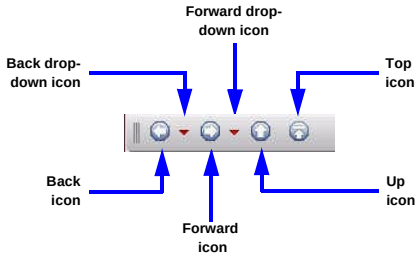
Virtuoso Schematic Editor GXL has not yet been released.

Go Toolbar

- Use the Go toolbar to do the following:
  - Sequentially or non-sequentially navigate through a cellview hierarchy
  - Navigate between cells and views in various designs
- Use of the Go toolbar for navigation applies only to the current session

Important

When working in a design-managed environment, using the Go toolbar to revisit design views may result in additional checkout and checkin operations.



- Accessing the Go toolbar
  - Select **Window—Toolbars—Go**.
  - Right-click the toolbar area and select **Go** from the pop-up menu.

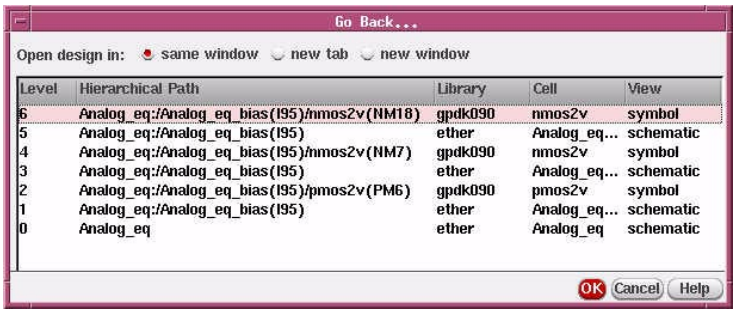
Go Toolbar (continued)

Icons and menus in the Go toolbar:

- **Back** icon: Moves one level back to the previously displayed cellview.
- **Back** drop-down menu: Displays a list of cellviews that you can choose to go back to.
- **Forward** icon: Moves one level forward to the cellview that was displayed prior to the use of the last Back command.
- **Forward** drop-down menu: Displays a list of cellviews that you can choose to go forward to.
- **Up** icon:
  - Activated only if you have descended into a hierarchy.
  - Moves you up one level in the hierarchy.
- **Top** icon:
  - Activated if you have descended into a hierarchy.
  - Moves you up to the cellview that represents the top of the hierarchy.

Go Toolbar (continued)

- The maximum number of cellviews displayed at a time in the **Back** and **Forward** drop-down lists is five.
- When the number of possible cellviews exceeds five, an additional **more** menu item appears on the list.
- Click **more** on the Back list to open the Go Back form and select a view to go back to. Click more on the Forward list to select a view to go forward to.



Workspaces

- A workspace is a layout of configurable user interface components that can be customized to assist your work in a particular application or when performing a particular task.
  - You can use a workspace to define the existence, size, and position of a set of associated user interface components within an application.
  - You can choose a particular arrangement of assistant panes (dockable workspace components that can be resized and repositioned in a session window) and toolbars available to the current application.
- You can define the following properties of your window layout in a workspace:
  - Which assistant panes are docked, floating, and hidden
  - Which toolbars are docked, floating, and hidden
  - Positions and geometries of the assistant panes and toolbars

Workspaces (continued)

Workspaces you create in Virtuoso Schematic Editor XL can be accessed in Virtuoso Schematic Editor L.

- Workspace functionality can be accessed from **Window—Workspaces** menu from a session window displaying a design in Virtuoso Schematic Editor L.
- However, the Workspace toolbar is only available in Virtuoso Schematic Editor XL.

You can save many custom workspaces as per your needs.

- Cadence provides a default workspace for Virtuoso Schematic Editor L called *Classic*.

  Toolbars that appear as a part of *Classic* workspace are: File, View, Create, Edit and Workspaces
- Schematic Editor L do not have any assistant panes.
- The *Classic* workspace for Schematic Editor L is stored at: *<inst_dir>/share/cdssetup/dfII/workspaces/Schematics/Classic.workspace*
  - The *Classic.workspace* file is a binary data file.
  - You will need write permission to the Cadence hierarchy in order to modify a Cadence-provided workspace.
- In the Schematic Editor L environment, you may
  - Enable the toolbars you need
  - Rearrange the size and location of these enabled toolbars using **Banner** pull-down menu **Window—Workspaces** from the session window
  - Save it as your own custom workspace
  - Set your custom workspace as the default workspace for Schematic Editor L, as needed

Workspaces (continued)

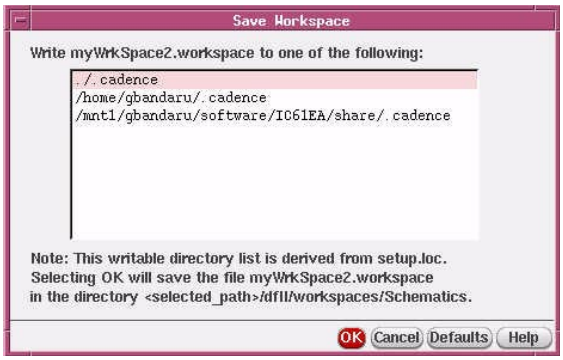
Saving your own custom workspace:

1. Modify the toolbar size, location, and visibility according to your needs.
2. Modify the assistant size, location, and visibility according to your needs.
3. Enable the Workspace toolbar if it is not already enabled.
4. On the Workspace toolbar, in the Workspace field, highlight the existing workspace name and type a new name for your workspace, such as *myWrkSpace2.*
5. Click on the Workspace toolbar icon called **Save Workspace**.
6. Choose the location to save the workspace in the dialog box that comes up.

In Virtuoso Schematic Editor XL, assistant panes can be added to your custom workspace.

Workspaces (continued)

Save Workspace form



As mentioned in the form, the choices are built based on your write permissions to directories listed in the *setup.loc/CSF* file.

Workspaces (continued)

Deleting a Custom Workspace

1. Select **Window—Workspaces—Delete**.
2. Choose the workspace name to delete in the form that comes up.
3. Click **OK**.

Loading a Custom Workspace

- Choose **Window—Workspaces—Load** and select your workspace.
- In the Workspace cyclic field, select your workspace.

Note: Your workspace might change if you change from one cellview to another by moving up or down the current hierarchy, even though you are not changing tabs in your session window.

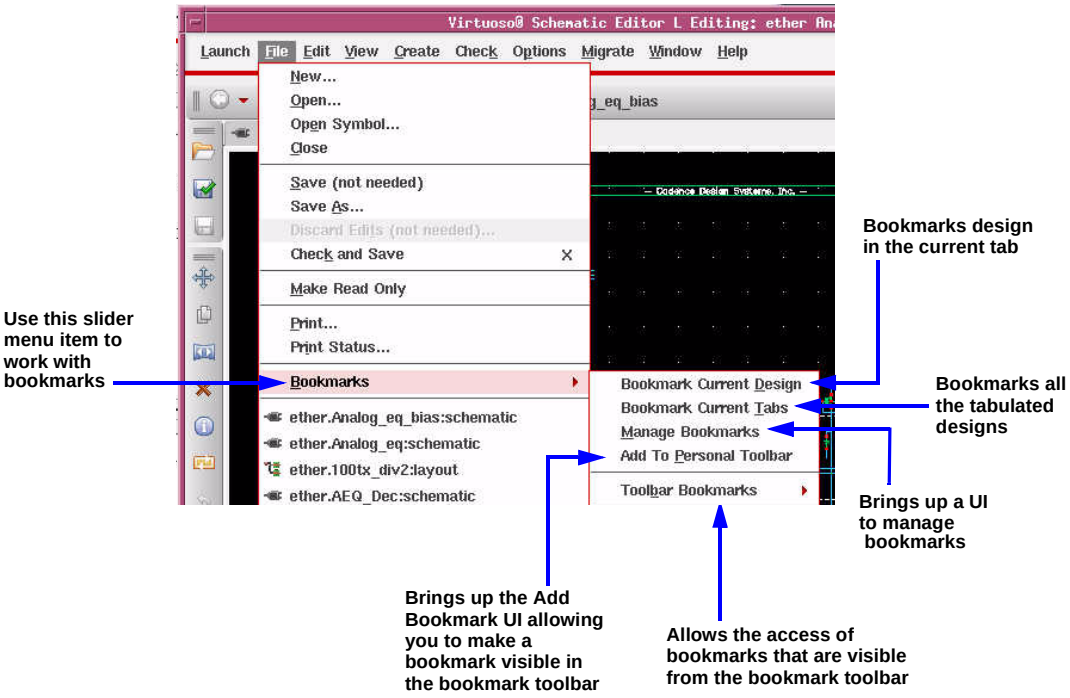
Bookmarks

- You can bookmark design views and return to them during the current or future sessions.
- Bookmarks are persistent across Cadence® sessions.
- You can create a bookmark of a single view or a composite bookmark of all tabs in your session window.
- You can select bookmarks from the menu or from the Bookmarks toolbar.

Note: Bookmarks are disabled in Virtuoso Schematic Editor L. The slider menu **File—Bookmarks** is grayed out and the Bookmarks toolbar is not available.

Bookmarks (continued)

Bookmarks menu

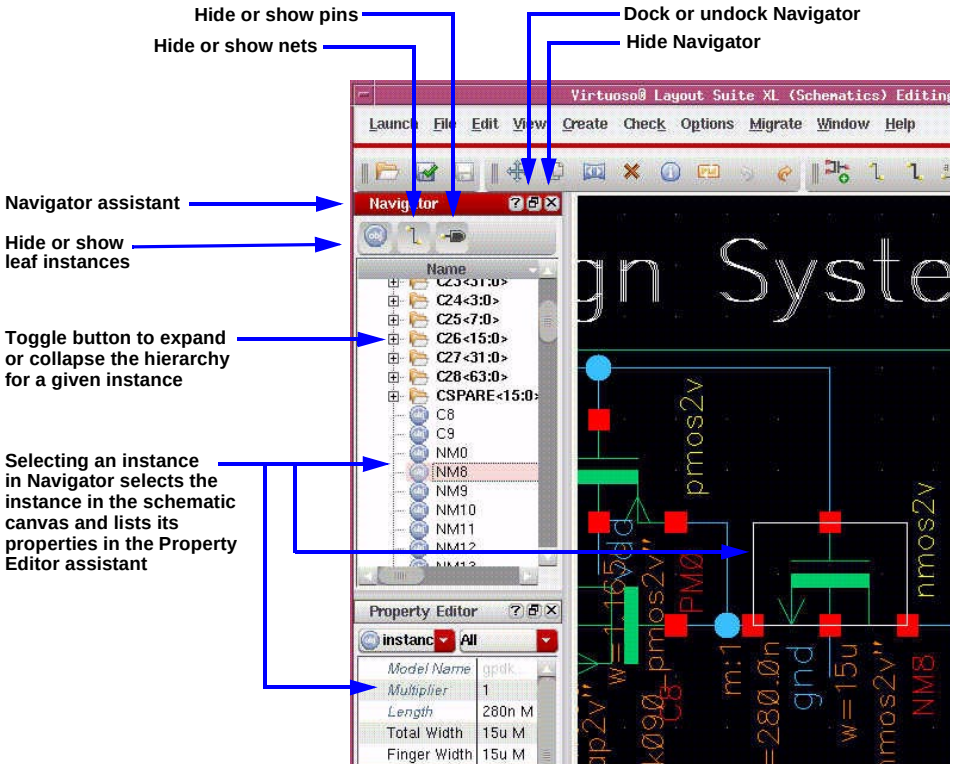


- The Bookmarks menu item is also available from **CIW: File—Bookmarks**.
- It is context-sensitive menu: The options available from CIW vary from what is available from a session window.

Navigator Assistant

- The Navigator is an assistant that provides facilities to view schematic objects in a tree view utilizing the Navigator tree.
- The Navigator tree is an hierarchical, expandible and collapsible navigation system, used to display large datasets, while utilizing icons to confer status information.
- The Navigator also provides options to only view instances, pins, and/or nets, and allows you to sort this information by columns, for example, by cell name or by view name.
- Cross-selection is also available from and to the Navigator.
  - Selecting an instance the Navigator causes the same instance to be selected in the active cellview tab and in other assistants such as the Property Editor assistant and Constraint Manager assistant, if in use.
  - If the Virtuoso Schematic Editor XL session window is up along with its associated Virtuoso Schematic Editor XL window, then the cross-selection will extend to the corresponding instances in the Virtuoso Schematic Editor XL window too.

Navigator Assistant (continued)



Navigator Assistant (continued)

Main Purposes for Using Navigator

- Enable the display of a fully unfolded hierarchy (occurrence tree) that serves as a design navigator
- Provide a convenient method of selecting instances, nets and pins across a schematic hierarchy

Accessing Navigator

- Navigator is available by default in the Basic and Constraints workspaces in Schematic Editor XL.
- Choose **Windows—Assistants—Navigator** in a Schematic Editor XL window.
- Right-click in the main Schematics XL menu or toolbar area and select **Navigator**.

By default, Navigator is docked at upper left area of the session window. However, just like toolbars, any assistant can be dragged (by clicking and holding on the assistant title bar) and docked to any of the four sides of the session window.

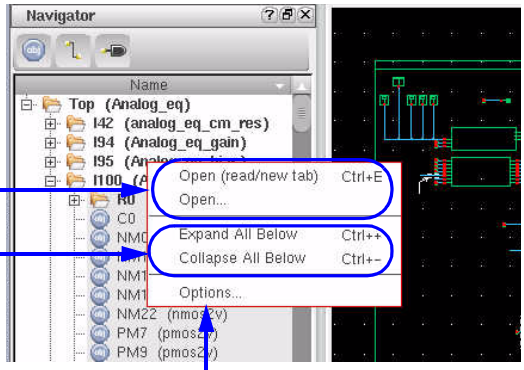
You can use the following keyboard shortcuts to aid your interaction with the Navigator:

Key Combination	Action
Up arrow	Move up tree by one row or object
Down arrow	Move down tree by one row or object
Left arrow	Collapse current object and contents
Right arrow	Expand current object and contents
Home/Page Up	Move to top of tree
End/Page Down	Move to bottom of tree
Ctrl ++	Expands all below current object
Ctrl +-	Collapses all below current object

The last two shortcuts in the table are also available as commands that can be accessed through the Navigator context menu.

Navigator Assistant (continued)

Context-sensitive pop-up menu in Navigator:



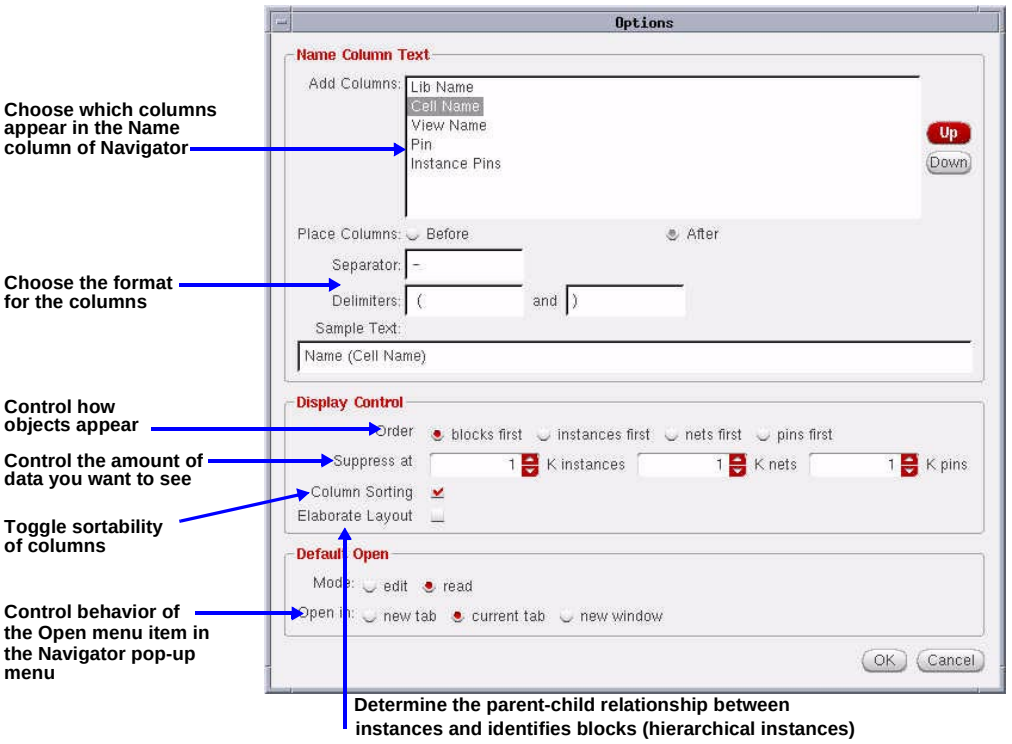
Available when object is a cellview instance

Available when object has hierarchy beneath it

Available on all objects within Navigator. Clicking on it brings up the Navigator Options form.

Navigator Assistant (continued)

Options form



Navigator Assistant (continued)

Zooming In to a Selected Object

Navigator maintains the current level of zoom in the design canvas when centering around the selected objects.

To zoom in to an object selected in the Navigator:

1. Set a zoom level in your design canvas
2. Then select an object in the Navigator assistant. The design canvas re-adjusts to show the selected object at the zoom level you set.

Search Assistant

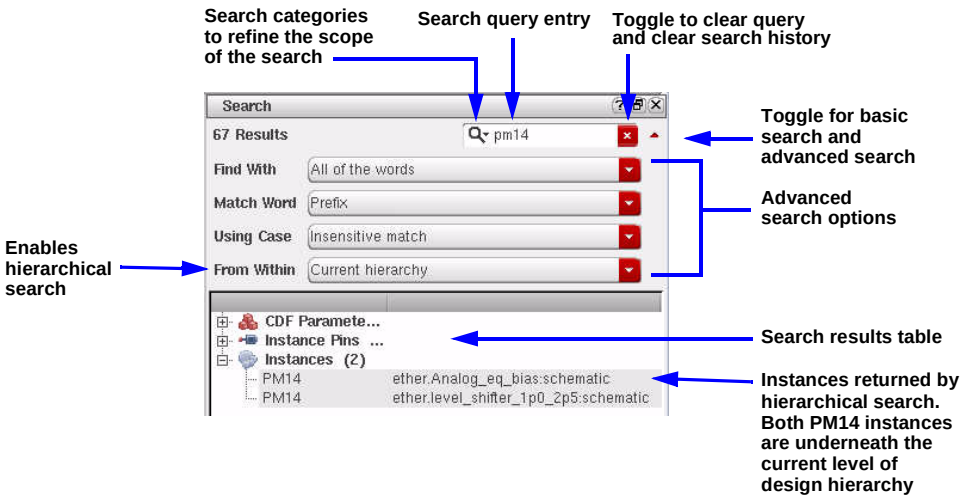
- The Search assistant provide a wide range of design search facilities like:
  - Dynamic context search categories
  - Fast keyword-directed searching with immediate feedback
  - A user interface (UI) that makes design data searching both simple and productive
- Key features provided by Search include the ability to:
  - Use search categories and groups
  - Perform dynamic, context-aware searching
  - Use keyword searching with immediate feedback
  - Use an intuitive, simple to use UI
  - Refine search queries on initial search results
- Precise searches within the results from a search are possible.
- This search mechanism can work hierarchically (via an advanced option).

The **Edit—Find** and **Edit—Replace** functionality is still available.

Search Assistant (continued)

To access the Search assistant:

- Select **Window—Assistants**, and then check the **Search** option.
- Right-click in the main Schematics XL toolbar area and select **Search** from the upper section of the context-menu displayed

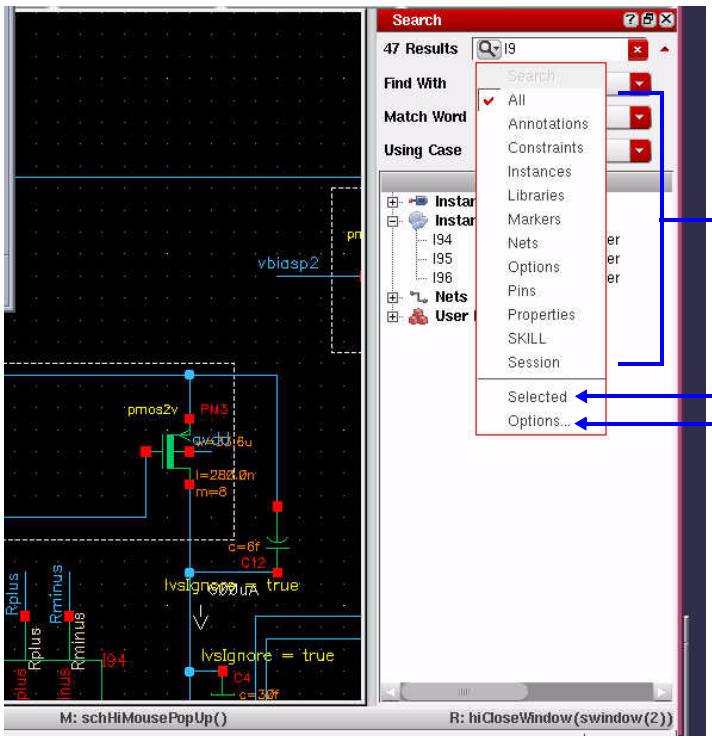


Note: Search query history is not maintained between sessions.

Terms

- Category: A search option that matches a specific type of content
- Context Aware: An application specific collection of search categories
- History: Displays the most recent search criteria entered
- Hit: A target object that matches a given query
- Query: An expression that contains words, keywords, and operators
- Result: A collection of hit groups that are generated by the search engine
- Target: A category object that is matched using a search query
- Keyword: A special query word that matches a category or a category group
- Word: A regular query token that is matched with the content search data

Search Assistant (continued)



Available search categories (defaults to all)

Confines the search to the objects currently selected in the design canvas.

Brings up Search Options form, which controls the categories that are displayed in the Search Categories menu.

Search keyboard shortcuts

Key Combination	Action	Key Combination	Action
Backspace	Deletes character to left	Ctrl+A	Moves cursor to
Ctrl+B	Moves cursor one	Ctrl+Backspace	Deletes word to left of
Ctrl+C	Copies selected text to	Ctrl+D	Deletes character to right
Ctrl+Delete	Deletes word to right of	Ctrl+E	Moves cursor to end of
Ctrl+=	Expand all tree branches	Ctrl+F	Moves cursor one
Ctrl+H	Deletes character to left	Ctrl+Insert	Copies selected text to
Ctrl+K	Deletes to end of line	Ctrl+Left	Collapse currently
Ctrl+Minus	Collapse all branches	Ctrl+Plus	Expand all branches
Ctrl+Right	Expand currently selected	Ctrl+V	Pastes clipboard text into
Ctrl+X	Deletes selected text and	Ctrl+Y	Redoes the last undone
Ctrl+Z	Undoes the last operation	Delete	Deletes character to right
End	Moves cursor to end of	Home	Moves cursor to
Left Arrow	Moves cursor 1 character	Right Arrow	Moves cursor 1 character
Shift+Delete	Deletes selected text and	Shift+Insert	Pastes clipboard text into
Shift+Left	Moves and selects text 1	Shift+Right	Moves and selects text 1

Search Assistant (continued)

Advanced Search Options

- The options in the Find With section are used to specify the default matching operators AND (All Of The Words), OR (Any Of The Words), EXACTLY (The Exact Phrase), and NOT (None Of The Words).
- The options under the Match Word section let you specify a regular query token that must be matched in the current content search data to display a successful result. The options available here are Prefix, Substring, Exactly, and Suffix.
- The options under Using Case let you determine whether search results must be case-sensitive (Sensitive Match) or whether any text case is acceptable (Insensitive Match).

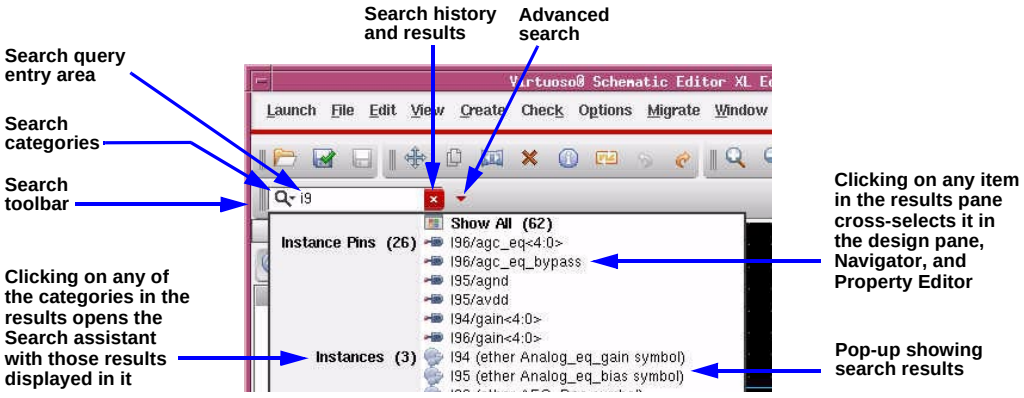
Search Assistant (continued)

Search Results Presentation

- The initial categories displayed are ranked according to the following rules:
  - Categories with the largest number of hits
  - Categories that are associated with the current session window
  - Categories with objects visible in the current view
  - Categories that have been recently selected
- Within a category hits are then ranked using:
  - The Levenshtein Distance (a measure of the similarity between two strings, referred to as the source string and target string)
  - The number of matching target items
  - The match order

Search Toolbar

- To access the Search tool:
  - Select **Window—Toolbars**, then check the **Search** option.
  - Right-click in the main Schematics XL toolbar area and select **Search** from the lower section of the context menu that appears.



- The Search tool displays results in a pull-down menu, unlike the Search assistant, which displays them in a results table.

If the search pattern you entered is taking a considerable amount of time, the tool brings up a dialog box mentioning the percentage completed of the search task. It provides a **Stop** button in case you want to stop the search operation.

Property Editor Assistant

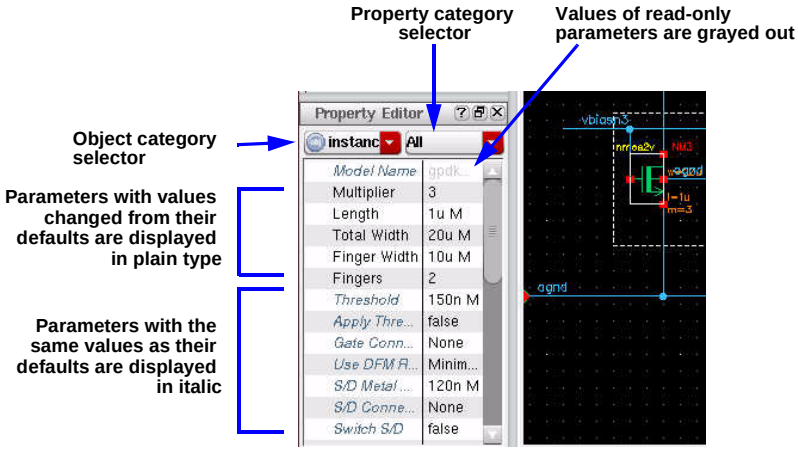
- The Property Editor is a dockable assistant that provides an intuitive and compact method of viewing and editing single or multiple object property values
- Some of the key capabilities provided by Property Editor are:
  - Filtering by object or property categories
  - Visualizing the current status of an object property, for example if the property value is non-default or read-only
  - Comparing values across multiple selected objects
  - Directly editing individual property values across single and multiple objects
  - Displaying and editing compound properties and their values

Property Editor Assistant (continued)

To access the Property Editor assistant:

- Select **Window—Assistants—Property Editor**.
- Right-click in the main Schematics XL menu or toolbar area and select Property Editor

The Property Editor is also available in the Basic and Constraints default window workspaces.



Property Editor keyboard shortcuts:

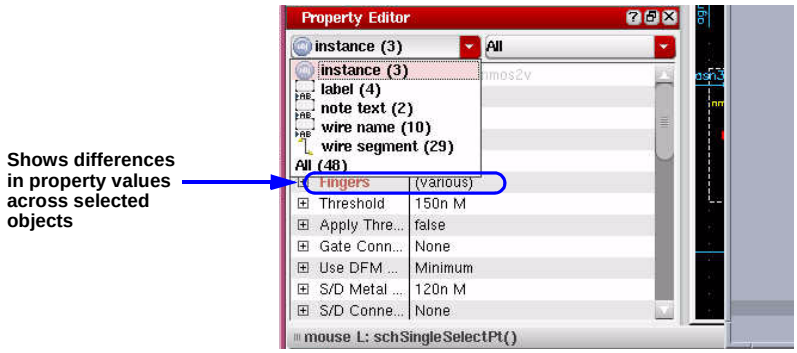
Key Combination	Action
Ctrl and P	Switches to View by Property mode
Ctrl and O	Switches to View by Object mode
Ctrl and +	Expands all collapsed properties
Ctrl and -	Collapses all expanded properties
Ctrl and down arrow	Advances to the next top-level property and expands it while collapsing the previously selected property
Ctrl and up arrow	Moves to the previous top-level node and expands it while collapsing the previously selected property

Note: Use **Control** and the **up** or **down** arrow keys to quickly expand or collapse table properties or objects.

Property Editor Assistant (continued)

Object Category Selector menu

- After making object selections in the current session window, you can click on the Object Category Selector drop-down menu in the Property Editor to view the number of objects in each object category currently selected
- Object category icons are also placed to the left of each object category to help distinguish object categories.



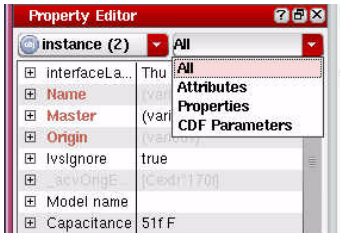
- Selecting one of the object categories narrows down the contents in the Property Editor to that category.

The following object categories can be listed in the Object Category Selector drop-down list if selected in the current session window: instance, instance pin, pins, pin names, wire names, net expressions, property labels, instance labels, wire segments, note shapes, note text, symbol pins, symbol labels, symbol shapes, and selection box.

Property Editor Assistant (continued)

Property Category Selector

- The Property Category Selector is a drop-down list that displays the property categories associated with the current object category selection in the Object Category Selector.



- Selecting a property category from the Property Category Selector drop-down list filters the property categories displayed in the Property Name column.

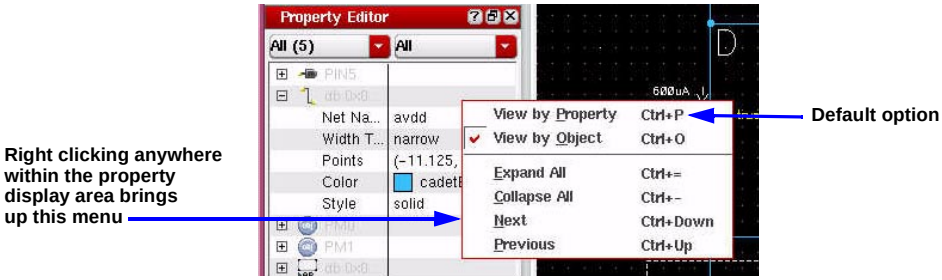
Property Editor Assistant (continued)

Editing Property Values

1. Click on the property value field that you want to update.
2. Update the value and press Enter to post the update.

   You can edit property values for a single object or for multiple objects.

Displaying Information by Property or Object



Property Editor Assistant (continued)

The Property Editor and Component Description Format (CDF) support:

- Property Editor is a dockable assistant, as opposed to a standard schematic editor form (such as the Edit Object Properties form), hence there are some differences in behavior
- The *cdfgForm* parameter is set to *nil* when using Property Editor to display CDF, because Property Editor is not a form but a dockable assistant.
- In case your callbacks use *cdfgForm*, you should ensure that they can handle a case where *cdfgForm* is *nil*.
- As there is no form, the *cdfgForm -cdfModified* flag is not supported in the Property Editor.

You can use the Property Editor to emulate *cdfgForm*. This can be useful in cases where it is not possible to edit the CDF definition (and access to *cdfgForm* is limited to certain properties).

Setting the SKILL® variable *oiEmulateCdfgForm* to *t* will cause the Property Editor to set *cdfgForm* to be effectively the same as cdfgData.

This means that all of the properties associated with *cdfgData* are accessible through *cdfgForm*.

With emulation enabled, the Property Editor will also display a warning any time a callback tries to access *cdfgForm*. This is done to alert you to the fact that the callback operation may not be working as intended.

the *cdfgData* warning message can be disabled by setting the SKILL variable *oiSuppressCdfgFormMessages* to *t*.

Important

You need to verify that your callbacks are giving correct results using this emulation

World View Assistant

The World View assistant pane a simple yet powerful navigation tool.

World View is especially useful in larger designs where it can display the complete schematic design, and its relationship to the main window, even if the main window has been greatly zoomed in.

To enable World View:

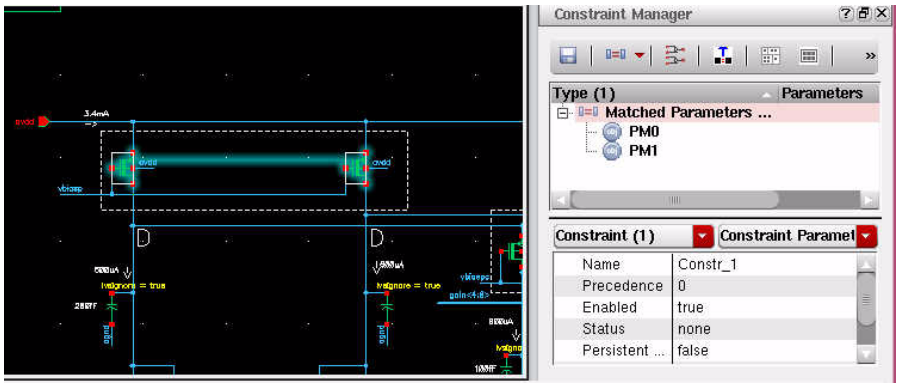
- Select **Windows—Assistants—World View** OR
- Right-click in Schematics XL menu or toolbar area and select World View

Since World View is a dockable assistant, it can be docked on any side of the design canvas. It can also be resized and also floated if so desired

World View is not included in any of the default workspace configurations.

Constraint Manager Assistant

- The Constraint Manager dockable assistant allows for the addition, modification, and deletion of constraints from and to the Cadence constraint storage system.
- The Constraint Manager user interface is comprised of two main component parts: the Constraint Manager table and Constraint Editor.

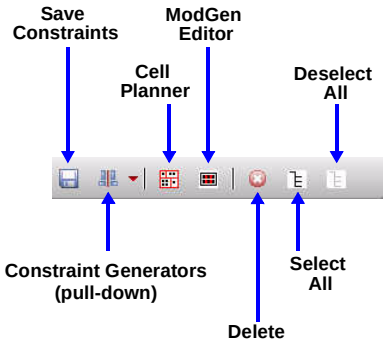


Constraint Manager Assistant (continued)

- The Constraint Manager displays a full set of constraints for a design wherever you are in the hierarchy with the current cellview, and wherever the constraints were created from in that hierarchy
- Examples of constraints include matching, symmetry, orientation, relative orientation, IR drop, parasitic filtering, parasitic estimation, clustering, alignment, distance, boundaryDef, power structure/guard rings (layout only), custom module generators (modgens) and cell plans.
- To access the Constraint Manager:
  - Select **Window—Assistants—Constraint Manager** OR
  - Select **Constraint Manager** from the Toolbars context-sensitive menu.
- The Constraint Manager is also automatically loaded when you select the *Constraints* or *Constraints-Helper* workspaces from the Workspace Configuration toolbar pull-down menu.

Constraint Manager Assistant (continued)

Constraint Manager toolbar:



Constraint Manager Assistant (continued)

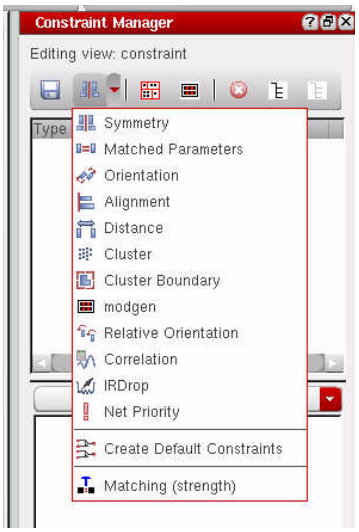
Save Constraints

- Saves the constraints listed in the Constraint Manager.
- A consistency check is performed to ensure that the listed constraints all contain the correct members.
- The **File—Save/Save As** options in the Schematic Editor update to *Save (Constraint)* when a constraint update has been made but not yet saved. Consequently the schematic **File** menu save options can also be used to save constraint status.

Constraint Manager Assistant (continued)

Constraint Generators (pull-down)

- Creates the selected constraint comprising of the currently selected object group members in the design canvas or the Navigator assistant
- Particular constraint creation conditions will have to be met for a specific constraint type to be created.



Create Default Constraints is used to set the associated constraints to the result groups selected in the Circuit Prospector.

Matching (strength) is an example to set a template of constraints to a selected pair of instances to be unmatched with a variable - user selected - strength.

Constraint Manager Assistant (continued)

Cell Planner

- Invokes the Layout Editor in cell planner mode
- The Cell Planner is a constraint-aware interface that allows you to plan a cell by grouping devices into different partitions, or rooms, within a cell

Module Generator

- Invokes the layout editor and displays the Modgen Editor.
- The Module Generator provides a simple and fast method of generating multiple Pcell/QCell instances into a complex, highly matched array based on defined Pcell/Qcell leaf structures

Delete

- Deletes the currently selected constraint (or object group) in the Constraint Manager table

Select All and Deselect All

- Self-explanatory

Constraint Manager Assistant (continued)

Interactive Constraint Searching

- You can use interactive search functionality (Search assistant and toolbar) to help locate design components.
- This can be particularly useful when trying to populate constraints or templates with nets and instances that are located in large, complex designs.
- The Search assistant and toolbar functionality can search through all the constraint members stored in the current cellview.
- Selecting one of these hits will cross select the constraint member on the design canvas, if it exists. Cross-selection works only with instances, instance terminals, nets and pins.

Circuit Prospector Assistant

Circuit Prospector is designed to greatly accelerate the constraints entry process.

- It provides a flexible framework for capturing the information that a designer is looking for.

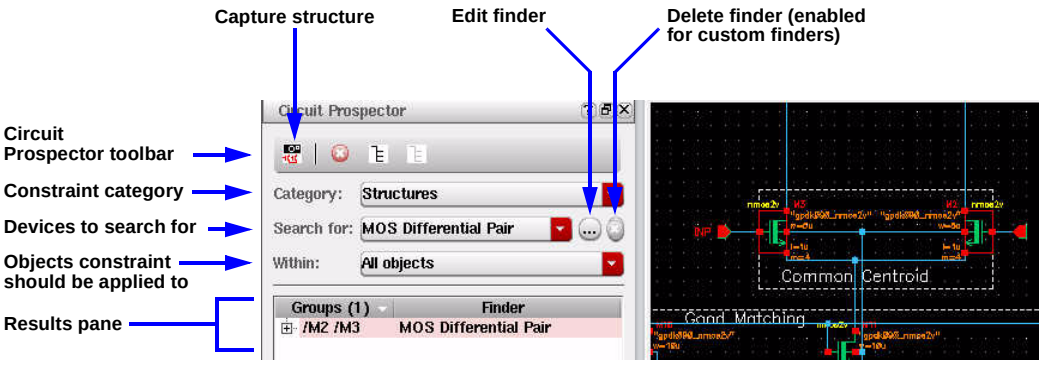
  This information could include common circuit structures, related devices (same size, area, or type), devices with particular characteristics or connectivity, or symmetrical devices, nets, pins with particular properties.

- It helps in locating such occurrences structures and devices, and applying appropriate constraints to them.

To access Circuit Prospector:

- Select **Window—Assistants—Circuit Prospector** OR
- Select **Circuit Prospector** from the Toolbars context-sensitive menu OR
- Select the Constraint-Helper default workspace from the Workspace Configuration pull-down menu on the toolbar.

Circuit Prospector Assistant: User Interface



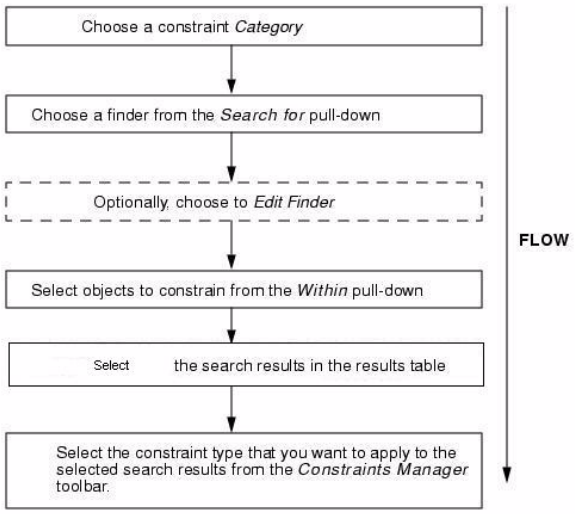
- Constraint categories available are: Active Devices, Passive Devices, Structures, Nets and Pins.
- Each constraint category contains a collection of related finders (search algorithms).
- Once a constraint category has been selected, a list of filtered finders is displayed in the Search form pull-down.
- Using the Within option, you can choose to apply the selected finder to all objects, constrained objects, or unconstrained objects.

Terms and definitions

- Finder: Search algorithms that search for devices, pins, and nets that meet specific, customizable, search criteria.
- Structure: These can be Active Devices, Passive Devices, Nets, or Pins.
- Iterator: SKILL functions that analyze cellview devices, nets, and pins in a variety of ways, for example then can iterate all selected objects, all instances, or all symmetrical devices.

  Finders are wrappers around iterators that provide a SKILL expression for an iterator to apply so that it can return a list of objects that will satisfy that expression. In this way, iterators can be used by a multitude of finders
- Constraint generator: Used to generate constraints for those groups of nets, instances, or pins that are selected in the results section of the Circuit Prospector assistant. A constraint generator will generate one or more constraints with either default or specific parameter values.

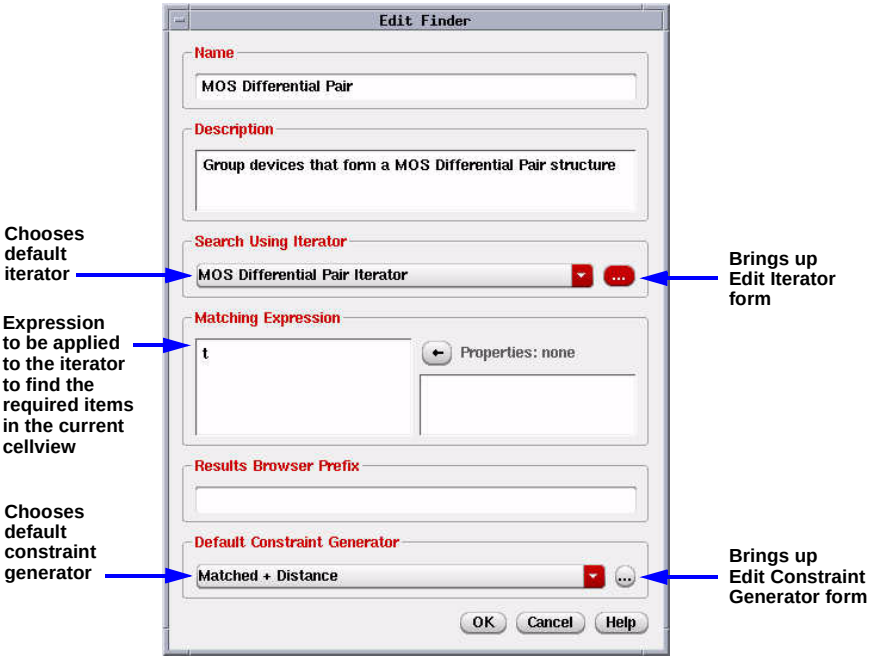
Circuit Prospector Constraint Creation Flow



Edit Finder brings up a form that allows you to update the name and description of the finder, which iterator to use for search, and which default Constraint Generator to used. It also provides access to Edit Iterator and Edit Constraint Generator forms.

Circuit Prospector Assistant (continued)

Edit Finder form



You will typically use the Constraint Prospector and Constraint Generators in tandem. In the Constraint Generators pull-down click on “Create Default Constraints” to set the associated constraints to the groups selected with the Circuit Prospector.

Labs

Lab 4-1 Using the Go Toolbar

Lab 4-2 Workspaces

Lab 4-3 Bookmarks

Lab 4-4 Navigator Assistant

Lab 4-5 Search Toolbar and Assistant

Lab 4-6 Property Editor Assistant

Lab 4-7 World View Assistant

Front-end Translators

Module 5

December 14, 2006

Module Objectives

- Understand the SpiceIn enhancements
  - Convert a Spectre® netlist to a schematic
  - Convert an HSpice netlist to a schematic
  - Examine the new SpiceIn parameter mapping feature
  - Examine the license requirements for running SpiceIn
  - Run SpiceIn from within Cadence® Design Framework II
  - Run SpiceIn from the command prompt
- Learn what's new in VerilogIn
- Learn what's new in VHDLIn

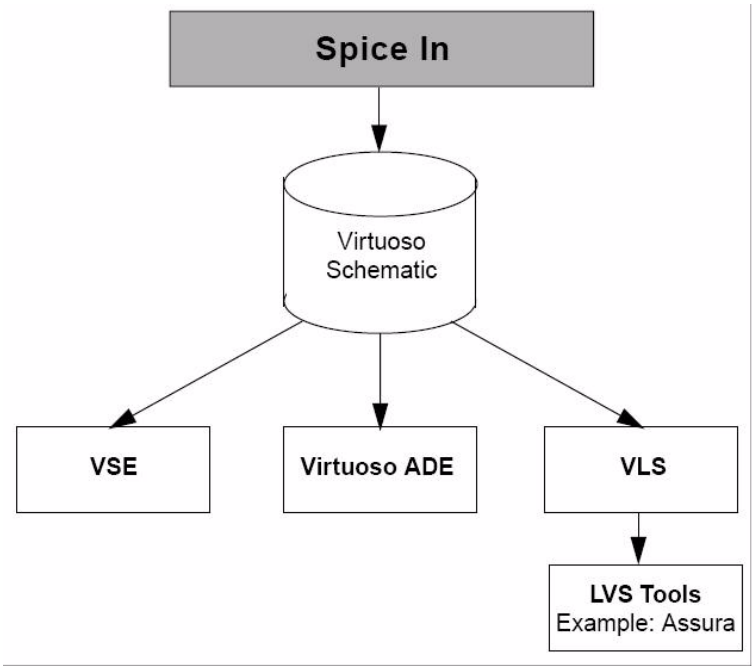
SpiceIn

What Is SpiceIn?

- SpiceIn is a Cadence product that imports HSpice, Spectre, SPICE, and CDL (textual) netlists into the Virtuoso® environment and creates either Virtuoso schematic or Virtuoso netlist views.
- SpiceIn enables import of designs represented as textual SPICE, HSpice, Spectre or CDL netlists into Virtuoso Schematic and thus plug in to flows that use
  - Virtuoso Analog Design Environment
  - Virtuoso Layout Suite (including XL) — subsequently running layout-versus-schematic (LVS) checks using LVS tools, such as Assura® Verification

Note: You have the option to modify or update the imported schematic representations before passing them on to other tools in the Virtuoso platform.

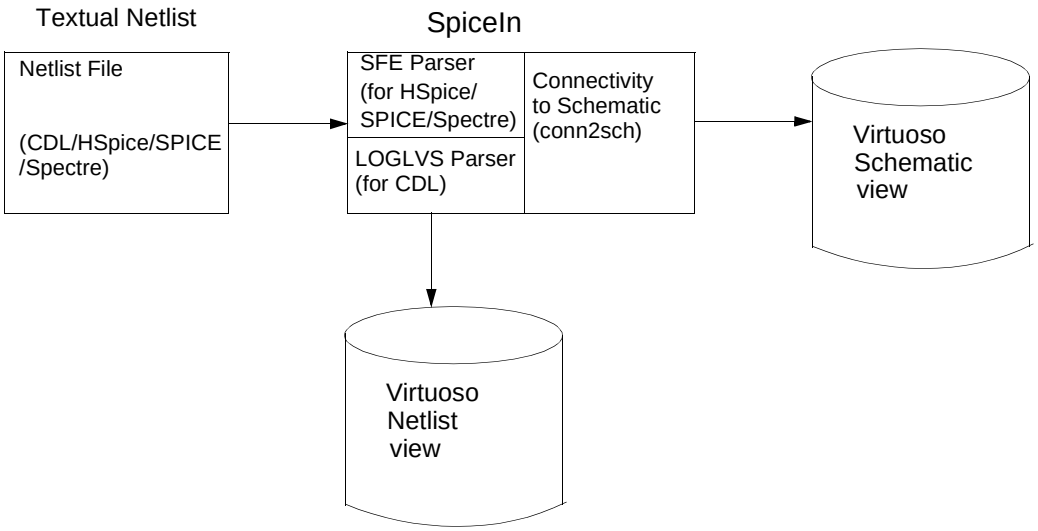
The following figure shows the flows that can use SpiceIn:



How Does SpiceIn Work?

- In IC 5.0.33, Cadence introduced a revamped CDLin which essentially
  - Ran the Dracula® LOGLVS-based parser on the input CDL netlist and created a Design Framework II netlist view
  - Then ran the Connectivity to Schematic tool (*conn2sch*) on the netlist view to create a DFII schematic view
- In IC 6.1, SpiceIn essentially added a Spectre Front End (SFE) parser in addition to the Dracula LOGLVS-based parser for CDL netlists.
  - The SFE allows for importing SPICE, HSpice or Spectre netlists into the DFII netlist view.
  - Connectivity to Schematic (*conn2sch*) is then run on the netlist view to create a DFII schematic view.

How Does SpiceIn Work? (continued)



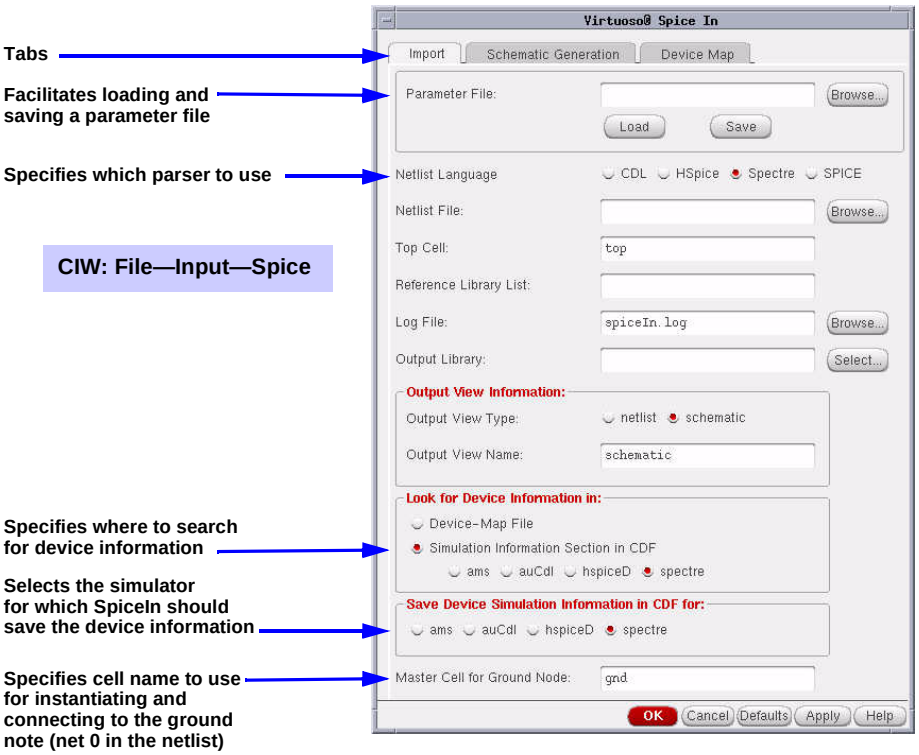
SpiceIn lets you create either a netlist view or a schematic view for your CDL, SPIC, HSpice, or Spectre netlist

SpiceIn License Requirements

SpiceIn is a licensed product. It searches for and checks out the first available license feature as per the following list (in that order):

- One license of Virtuoso Schematic Editor XL
- One license of Virtuoso Layout Suite XL
- Four tokens of Virtuoso Layout Suite GXL

Starting SpiceIn from DFII



Importing a CDL netlist in IC 6.1.0 is essentially the same as in IC 5.1.41. The process has the same limitations because the netlist parser is still the Dracula LOGLVS parser.

How to Use the Spice In Form

1. Choose a netlist language based on the netlist type being imported.
2. Give a top cell name, especially in case of importing a Spectre or an HSpice netlist into DFII.
3. Populate the reference library list.

   Keep in mind that the cell name specified in the Master Cell for Ground Node option also uses this list.
4. Specify the output library and output viewType information.
5. For the Look for Device Information in field:
   - Choose **Device-Map file** in case you are using the Digital PDK reference library.
   - Choose **Simulation Information section in CDF** when using Analog PDK as your reference library. In this case, also choose a simulator.
6. For the Save Device Simulation Information in CDF for field, choose a simulator
7. Use the Parameter File field at the top to load or save a parameter file.

If you are importing SPICE data into an existing library, none of the existing views are overridden by default. There is no GUI field to control this. A parameter file can be loaded with the option *overwriteCells* set to *all* in order to overwrite all existing cells when importing.

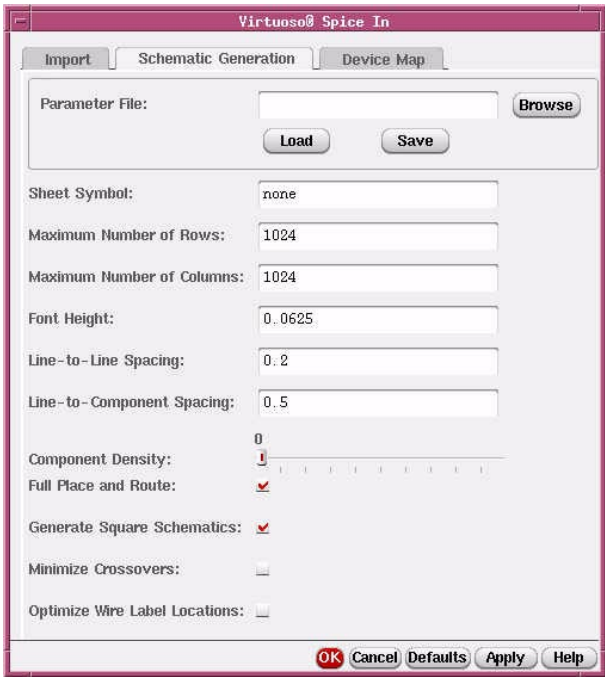
Controlling the Defaults in the Spice In Form

Various options in the Spice In form can be controlled by using the following entries in the *.cdsenv* file.

```
spiceIn topCell string "top"
spiceIn outputViewName string "schematic"
spiceIn outputViewType string "schematic"
spiceIn logFile string "spiceIn.log"
spiceIn overwriteExistingCells string "none"
spiceIn language string "Spectre"
spiceIn simName string "spectre"
spiceIn outputSimName string "spectre"
spiceIn expandMOSMultiplicityFactor boolean nil
spiceIn masterCellForGnd string "gnd"
spiceIn devInfo string "Simulation Information Section in CDF"
```

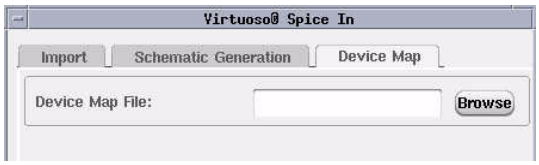
SpiceIn: Schematic Generation Options

There is no change to this form from previous releases.



Specifying Device Mapping

The Spice In form lets you specify a device mapping file.



Key constructs in the device mapping file:

```
devSelect := <primitive_device_name> <mapped_device_name>
devSelect := nfet nmos2v
devSelect := trpmos pmos2v
propMatch := <parameter_name> <parameter_value>
propMatch := w 10u
```

propMatch is an optional construct and is used to qualify the *devSelect* construct that immediately precedes it

The *devSelect* construct was called *devMap* in IC 5.1.41.

```
termOrder := <list of terminals seperated by whitespace character>
```


Device mapping enables you to map device instances in the netlist to components from your PDKs.

SpiceIn: Additional Features

- When importing an HSpice netlist into a schematic:
  - The global parameters specified with a *.param* command are imported as design variables for the top cell that is imported.
  - Any parameter specified using a *.param* command within a *.subckt* definition for cell is imported as a CDF parameter for that cell.
- When importing a Spectre netlist into a schematic:
  - The global parameters specified with a *parameters* command are imported as design variables for the top cell that is imported.
  - Any parameters specified using a *parameters* command within a *subckt* definition for cell is imported as a CDF parameter for that cell.
- Uses terminal order (*termOrder*) of the specified simulator for creation of connections between the nets of an instance and the terminals of the master cell of the instance.
- Saves the terminals of a subcircuit as *termOrder* in the CDF simulation information of the specified simulator.

Starting SpiceIn from the Command Prompt

- The binary to run from the command prompt is called *spiceIn* and is located in the *<IC61_install>/tools/dfII/bin* directory

  spiceIn syntax:

```
spiceIn -param <paramFile>
unix> spiceIn -help
/mnt1/IC61EA/tools/dfII/bin/32bit/spiceIn.exe Usage:
[-help]                 Displays this message.
[-V]                    Provides Cadence release version.
[-version]              Provides Cadence release version.
[-W]                    Provides Cadence release subversion.
[-param paramFile]      Takes SKILL parameter file as input.
```

SpiceIn Parameter File

What It Is

- The SpiceIn parameter file contains the required and optional parameters for SpiceIn.
- Syntactically, it is a SKILL® disembodied property list containing the name-value pairs of various parameters.

How to Create It

- Populate the Spice In form fields (fields in all three tabs) with the desired values.
  - In the **Parameter File** pane in the **Import** tab, specify a parameter file name you want to save and click on the **Save** button in that pane.
  - Schematic generation options are saved in a temporary file in the */tmp* folder
- Create a text file with all the options and their values.

  You can copy the sample parameter file *param.il* from the *<IC61>/tools/dfII/samples/spiceIn* directory and modify it.

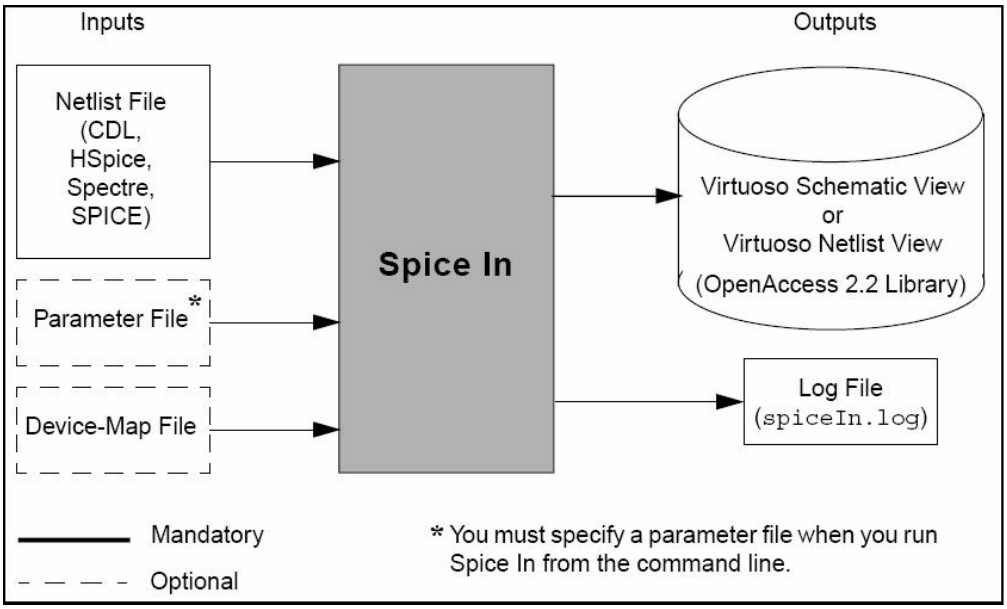
Sample SpiceIn Parameter File

```
spiceInParams = list(nil
      'simName "spectre"
      'outputSimName "spectre"
      'netlistFile ""
       'outputViewName "schematic"
       'outputViewType "schematic"
       'language "spectre"
       'devMapFile ""
       'outputLib ""
       'topCell "top"
       'logFile "spiceIn.log"
      'refLibList ""
       'overwriteCells "none"
       'overwriteCellsList ""
      `masterCellForGnd "gnd"
)
```


The parameter to specify a schematic generation options file is

```
'conn2schParamFile"<path to schParamFile>"
```

SpiceIn: Recap on Inputs and Outputs



What's New in VerilogIn?

- Changes to the GUI:
  - The Verilog In form is now tab-based.
  - All the VerilogIn options appear on the main tab.
  - Schematic generation options appear on a second tab.
  - Load and save features have been enhanced to make use of the new file browser.
  - The Verilog In form can now be resized horizontally so that you can see longer string values entered into the form fields.
- VerilogIn provides 64-bit bus support.

What's New in VerilogIn? (continued)

VerilogIn uses the *ncvlog* parser and supports verilog2001 constructs by default.

- *ihdl.exe* (the binary that is called when VerilogIn is run) has a version of *ncvlog* compiled into it.
- If *ncvlog* (shipped in IUS installations) is available in the current path then it is used, otherwise the built-in *ncvlog* is used by VerilogIn.
- You can set *ihdl_use_van=t* in the Command Interpreter Window or in the *.cdsinit* file to revert back to using VAN if so desired.

  This can also be set from on the command line as:

```
setenv IHDL_USE_VAN ON
```


Support for *ncvlog* as the default parser for VerilogIn is available from the IC5141USR3 release onwards.

VAN is a netlist generation utility used prior to this update.

What's New in VHDLIn?

Changes to GUI:

- The VHDL In form is now tab-based.
- All the VHDLIn options appear on the main tab.
- Schematic generation options appear on a second tab.
- The VHDL In form can be resized horizontally so that you can see longer string values entered into the form fields.

Labs

Lab 5-1 Running SpiceIn from Design Framework II

Lab 5-2 Running SpiceIn from the Command Prompt

Introduction to Constraints

Module 6

December 14, 2006

Module Objectives

- Understand the definition of a constraint
- See examples of constraints
- Examine a table showing the tools that support the various constraints
- Edit, save and delete constraints using the Constraint Manager
- Create constraints using the Circuit Prospector
- Customize the Circuit Prospector to locate unique circuit patterns
- Create Constraint Generators to apply to selected circuits
- Customize the Constraint Manager to use new editors and custom displays
- Examine additional Constraint Manager assistant features
- Understand the constraint SKILL® API

What Is a Constraint?

Constraints are rules that need to be followed during design implementation. Consider a constraint as a factor that bounds the performance of a system, in this context a design object, with respect to its goal.

A constraint is passed and used from one tool to another. Examples include:

- Design constraints
  - To meet specific design goals
  - User-defined, typically specified by circuit designer
- Layout-specific rules
  - To meet specific physical design goals
- Process rules defined in the technology file
  - Technology specific
  - Foundry specific

Examples of process rules:

- Minimum metal width
- Minimum metal spacing

Examples of design constraints:

- Matching differential pair for analog circuit performance
- Spacing and shielding for signal integrity requirements

Example of layout-specific rules:

- Fixing pin locations for ECO flow

Benefits of Using Constraints

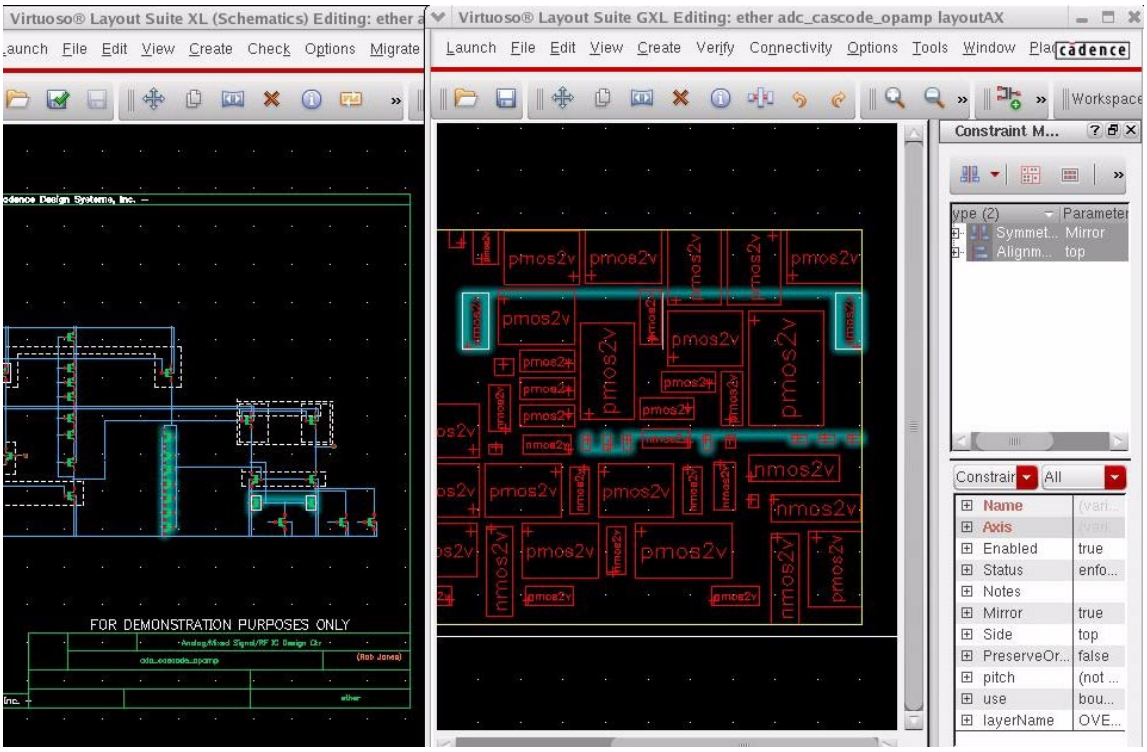
To improve productivity and shorten design cycles

- Constraint-driven automation
- Constraint-driven interactive editing
- Support unified constraints database front to back
- Support unified constraints database across tools
- Compliance and verification of constraints in layout design
- Constraint-driven design iterations

Note: Constraints apply to all design types but are more relevant to analog designs because of stringent matching requirements, for example, symmetry, parameter matching and interdigitation patterns.

Constraint Example

An example of unified custom constraints



Which Tools Support Constraints?

1. From Virtuoso® Schematic Editor to other tools in the flow:

Constraint	Description	Applied To	Used By
Alignment	Align two or more objects	Instances, pins, boundary	Top-level floorplanning, Constraint Assisted editing, Analog and Custom Digital Auto Placement, layout optimization
Symmetry	Make objects symmetric about an axis	Instances, pins, nets	Top-level floorplanning, constraint assisted editing, analog and custom digital auto placement, router (Virtuoso Chip Assembly Router), layout optimization
Matched Parameters	Match specified properties of two or more objects	Instances	Constraint-assisted editing and analog auto placement
Orientation	Define allowed orientation of one or more objects	Instances	Top-level floorplanning, constraint assisted editing, analog and custom digital auto placement
Relative Orientation	Define orientation of one or more objects with respect to a master object	Instances	Constraint-assisted editing, analog and custom digital auto placement

For detailed descriptions of constraints, refer to the “Default Constraint Types” appendix of the *Virtuoso Unified Custom Constraints User Guide*.

Which Tools Support Constraints? (continued)

1. From Virtuoso Schematic Editor to other tools in the flow (continued):

)

Constraint	Description	Applied To	Used By
Layout Structure	Complex physical structures such as modgens	Instances	Constraint-assisted editing (groups) and analog auto placement (modgens and cell plans)
Distance	Distance range between edge, center or origin of two objects	Instances, pins	Top-level floorplanning, custom digital auto placement, layout optimization
Cluster	Grouping to indicate proximity	Instances	Top-level floorplanning, custom digital auto placement
IR Drop	Allowable voltage drop	Nets, terminals	Virtuoso Chip Assembly Router
Parasitic Estimate	R/C parasitic estimates to be met by the layout software	Instance terminals, nets	Virtuoso parasitic estimation and analysis (PEA)
Parasitic Filter	Reduction of parasitic data in extracted views by filtering out parasitic instances	Instances, nets, cellviews	PEA
Correlation	Coefficient of correlation between two devices for Monte Carlo simulation	Instances	Virtuoso Analog Design Environment

Which Tools Support Constraints? (continued)

2. Layout-specific constraints:

Constraint	Description	Applied To	Used By
Area Utilization	Specify percentage of boundary area occupied by components, to account for routing	Boundary (top level/ soft block)	Top-level floorplanning.
Boundary Area	Allowable width and height ranges for the boundary	Boundary (top level/ soft block)	Top-level floorplanning, constraint-assisted editing, analog auto placement
Power Structure	Associating a guard ring with a layout structure	Layout structure (modgen)	Modgen creation
Fixed	Objects can not be disturbed by automated tools but can be moved interactively	Instances, pins, interconnects	Top-level floorplanning, constraint assisted editing, analog and custom digital auto placement, router (Virtuoso Chip Assembly Router)
Locked	Objects can not be disturbed either by automated tools or interactively	Instances, pins, interconnects	Top-level floorplanning, constraint-assisted editing, analog and custom digital auto placement, router
Net Priority	Define relative priority of nets in terms of criticality	Nets	Top-level floorplanning, router
Process Rule Over-ride	Non-default rule sets to over-ride minimum process rules, typically for DFM/ DFY or for design specific requirements such as performance	Shapes, routes, pins, nets, net classes, area, design (only nets through UI)	Interactive routing (wire editing), router

Constraint Manager Overview

- How to access the Constraint Manager

  1. Launch XL or GXL.

  2. Select **Windows—Assistants—Constraint Manager**.

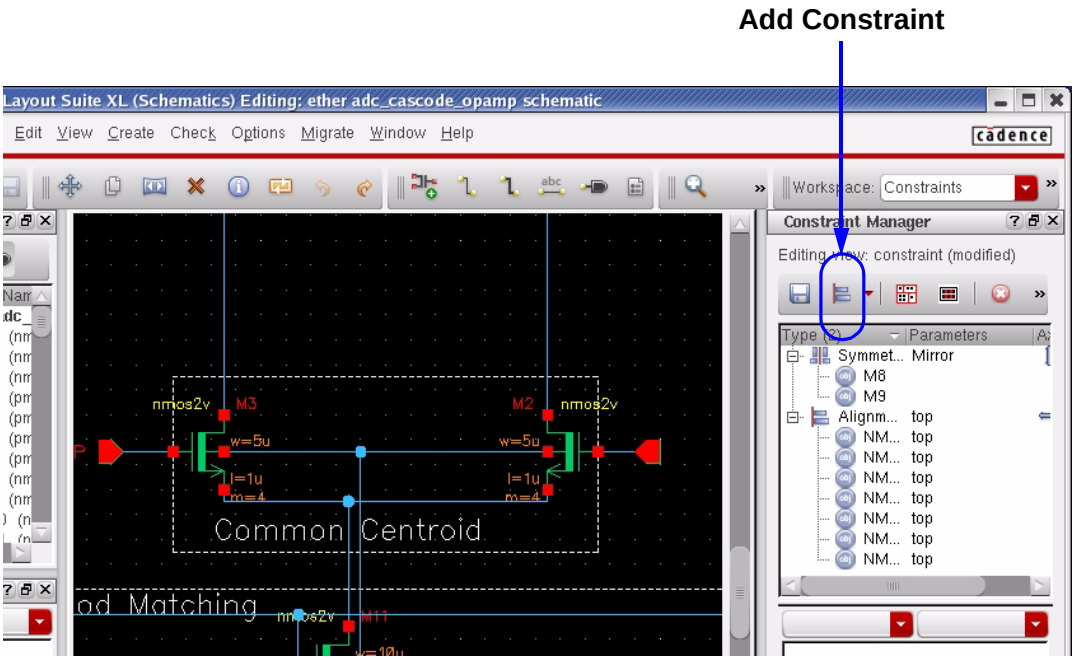
  3. Select **Constraint Manager** from the Toolbars context-sensitive menu.

  4. Select **Constraints** from the **Workspace Configuration** toolbar pull-down.

- Constraints between schematic and layout
  - Automatic propagation from logical to physical when Layout XL is initialized.
  - Support one-to-many or many-to-many correspondence.
  - Constraints transferred from schematics can be enabled or disabled or given lower precedence.

- A transferred constraint is considered to be out-of-context (indicated by orange text color) if any of its members is not present in the layout.
- Constraint conflict is caused by a constraint failing its consistency check. For example, an instance cannot be a member of more than one symmetry constraint. However, there is no verification check to ensure that a constraint in a schematic is achievable in a layout. Resolving this is the responsibility of individual applications.

Creating Constraints Using Constraint Manager



Introduction to Constraints 6-19

Steps to create constraints using Constraint Manager:

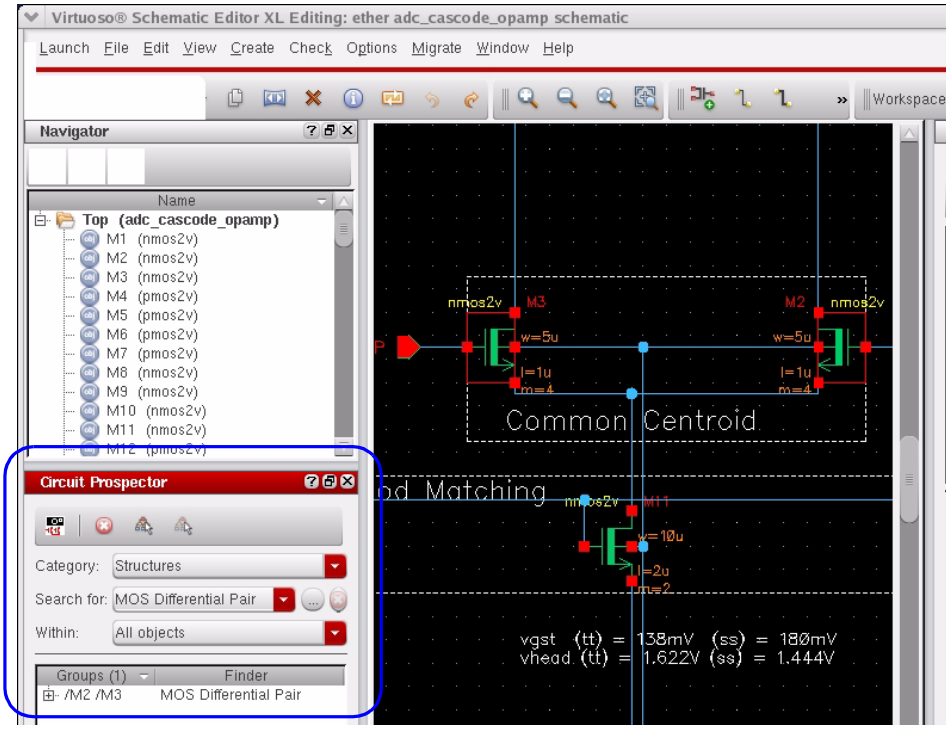
- Select those objects that you want to apply constraints to.
- Choose the constraint type to be created by either
  - Selecting the constraint type from the pull-down on Constraint Manager

  or
  - Right-clicking over the Constraint Manager table and selecting **Add Constraint**.

Circuit Prospector Overview

- Circuit Prospector
  - Identifies patterns based on a predefined set such as current mirror, diff pair, etc.
  - Identifies patterns based on a custom set captured/created by user.
- Benefits of using Circuit Prospector
  - Apply constraints to common circuit structures identified by Circuit Prospector.
  - Accelerate the constraints entry process.
- How to access **Circuit—Assistants—Circuit Prospector**.
  - Select **Circuit Prospector** from the Toolbars context-sensitive menu.
  - Select **Constraints-Helper** from the Workspace Configuration toolbar pull-down.

Creating Constraints Using Circuit Prospector



Steps to create constraints using Circuit Prospector:

1. Select a Circuit Prospector category.
2. Search for structure.
3. Select the search result which the constraint is to be applied to.
4. Create the constraint.

Circuit Prospector Terms

- *Categories* are groups of related finders (search algorithms) that are used to store structural definitions.
- *Finders* are search algorithms that search for devices, pins, and nets that meet specific, customizable, search criteria.

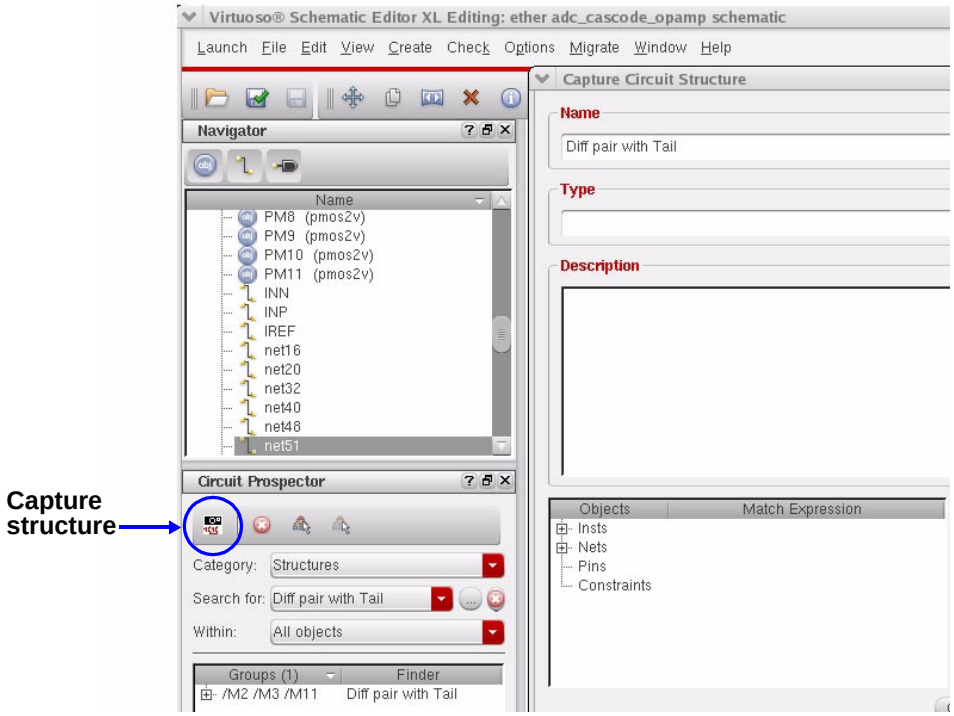
Categories	Finders (Search For)
Active Device Example: FETs, MOSFETs, BJTs, HBTs, SOIFETs, HEMTs	Active Same Cell Name Active Same Cell Name and Size Active Common Gate Active Same Well Active X or Y Symmetric Pairs All
Passive Device Example: resistors, capacitors, inductors, transmission lines, diodes	Passive Same Cell Name Passive Same Cell Name and Size All

Circuit Prospector Terms (continued)

Categories	Finders (Search For)
Structures Example: Active Device, Passive Device, Nets, Pins	MOS Current Mirror MOS Cascoded Current Mirror MOS Transmission Gate MOS Differential Pair MOS Common Gate MOS Parallel MOS Active Load MOS Inverter MOS Cross Coupled Quad All
Net	Supply Nets Non-Supply Nets All
Pins	Pins All

Circuit Prospector: Capturing New Structures

In addition to the existing predefined sets, you can capture new structures.



- GUI description
  - **Update From Selected** — Choose to add further instances, nets, or pins to the structure, by selecting them on the canvas, before selecting the **Update From Selected** button.
  - **Recapture Constraints** — Lets you add further constraints, created in the Constraints Manager, to the captured structure.
  - **Delete** — Deletes any constraints, instances, net, or pins that are currently selected in the Object/Match Expression table from the structure.
  - **Match Expression** — Captures the important properties for that instance within the captured structure. Examples: libname, cellname, etc.
- Steps to capture new structure
  - Select the instances, nets or pins on the design canvas that you want capture.
  - Select the **Capture Structure** button on the Circuit Prospector assistant toolbar.
  - Fill out the form.
  - Click on **OK**. The new structure shows up in the Search for field.

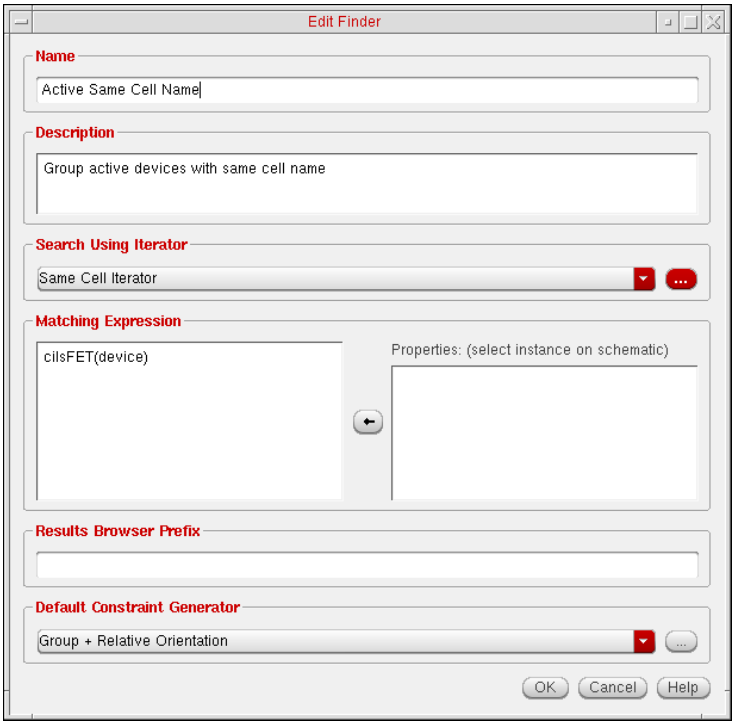
Note: Before a structure is saved to disk, a connectivity check is performed to ensure that each instance in the structure is connected to at least one other instance in the structure. This is an essential requirement so that the structure recognition algorithm can find the structure. If there are any problems with connectivity, then an error pop-up window is displayed and the connectivity of the structure will need to be changed before it can be saved.

Customizing Circuit Prospector: New Finder

- A finder consists of three elements:
  - An iterator — a pre-filter that retrieves the element found in the schematic on a per-type basis (devices, nets or pins), or by symmetry or structure.
  - A matching expression — is used to operate a second level of filtering. If the matching expression is evaluated as true, then the result of the iterator is listed in the table.
  - A default constraint generator — defines the set of constraints to apply to the found members of the corresponding searched pattern.
- The iterator and constraint generator listed in the *ciRegisterFinder* function must be correctly registered. The custom finder can be registered by loading the SKILL file into the application or by storing the file in a *.cadence/dfII/ci/finders* directory.
  - The SKILL file must have the naming format of *<finder>.il*.
    where *<finder>* must be the name used to register the finder with an "_" (underscore) character replacing each blank.
  - The *.cadence* directory must be on the file search path.

Creating a New Finder: The Edit Finder Form

Access the Edit Finder form by clicking on the **...** button adjacent to the Search for field in Circuit Prospector.



- GUI description
  - Search Using Iterator — Specifies which iterator the selected finder should use.
  - Matching Expression — An expression to be applied to the iterator to find the required items in the current cellview.
  - Default Constraint Generator — Specifies which constraint generator is to be associated with the currently selected finder.
  - Results Browser Prefix — Currently ignored.
- The iterator SKILL functions for all the predefined iterators are documented in *Virtuoso Unified Custom Constraints Configuration Guide*.
- The iterator functions take *CellView* and *matchExpression* as arguments, where *CellView* is the current cellview and *matchExpression* is the matching expression. The functions return object lists that satisfy *matchExpression*, while iterating through the passed *CellView*.

Creating a New Finder: SKILL API

The *ciRegisterFinder* SKILL command can be used to register a finder within Circuit Prospector.

Note: Although not recommended, you can modify predefined finders using the existing finder name in the *ciRegisterFinder* function. However, by doing so you may miss updates developed in future releases. It is preferred that you create a new finder by adding its name to the list of finders in one or more of your pre-defined or new categories.

Example:

To catch all nets that are named with either one of the substrings *clk* or *clock* in your schematic, you can define a new finder in a *Clocks_Nets.il* file. This file will be stored in the *.cadence/dfII/ci/finders* directory located in the file search path (see the Cadence® setup search file *setup.loc*). The content of the *Clocks_Nets.il* file may look like:

```
ciRegisterFinder(
   list(nil
      'name "Clock Nets"
      'description "Find all clock nets by scanning their names"
      'iterator "Net Iterator"
      'expression "if(index(net->name \"clk\") || index(net->name \"clock\") then net->name else
nil)"
      'defaultCGen "Custom Clock Net Priority"
      'legal CGens nil
   )
)
```

Customizing Circuit Prospector: New Categories

The *ciRegisterAssistant* SKILL command can be used to register a category within Circuit Prospector.

- Each finder listed in the *ciRegisterAssistant* function must be correctly registered for the corresponding category to be successfully registered.
- The custom category can be registered by loading the associated SKILL file directly into the application or by storing the file in a *.cadence/dfII/ci/categories* directory.
  - The SKILL file must also be in the format <*categoryName*>*.il*.
    where <*categoryName*> must be the name used to register the category with an "_" (underscore) character to replacing each blank.
  - The *.cadence* directory must be on the file search path.

Note: Although not recommended you can modify predefined categories using the existing category name in the *ciRegisterAssistant* function. If you do this however you may miss updates to the category that could be made in future releases. It is therefore recommended instead that you create a new category for your custom finders and include as many pre-defined finders as you require in that custom category.

Example:

To set constraints versus net types, you should first of all create a list of finders for the Circuit Prospector to look for relevant candidates. Then, you can register a custom category for these new finders with, for example, a *Various_Nets.il* file containing the following:

```
ciRegisterAssistant(
  list(nil
    'name "Various Nets"
    'description "Net finders"
    'finderNames list (
    "Supply Nets"
    "Ground Nets"
    "Clock Nets"
    "Sample and Hold Nets"
    "Differential Nets"
    "All Nets"
    )
  )
)
```

Creating Custom Constraint Generators

The *ciRegisterConstraintGenerator* SKILL command can be used to register a customized constraint generator.

- Available in the Constraint Manager's constraint creation toolbar drop-down list
- Available in the Generate Constraints list in the context menu
- The custom constraint generator can be registered by loading the SKILL file into the application or by storing the file in a *.cadence/dfII/ci/generators* directory.
  - The SKILL file must have a *.il* extension.
  - The *.cadence* directory must be on the file search path.

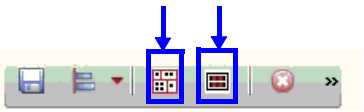
Example:

By default, a Matching (strength) constraint generation option is available on both the Constraint Manager toolbar and the Generate Constraints context-menu option. The Matching (strength) constraint generator is registered as follows:

```
ciRegisterConstraintGenerator(
   list(nil
      'name "Matching (strength)"
      'description "Generate various levels of Matching constraints"
      'expression "_ciRunMatchingConstraintsGenerator(args instsNetsPins cache)"
;;; expression to generate constraints"
      'addToToolbar t ;;; whether you want to add a button to the toolbar for this generator
      'iconName "templateMatched"
;;; icon to use on the toolbar. templateMatched.png must exist in the icon search path
      'args list( "strength" 'enum "low" "medium" "high")
      )
)
```

Customizing the Constraint Manager

- The current external editors available from the Constraint Manager are:
  - Module Generator
  - Cell Planner



- You can also plug other third-party editors into Constraint Manager.

  The custom constraint editor can be registered by loading the SKILL file (which must have an *.il* extension) into the application or by storing the file in a *.cadence/dfII/ci/editors* directory.

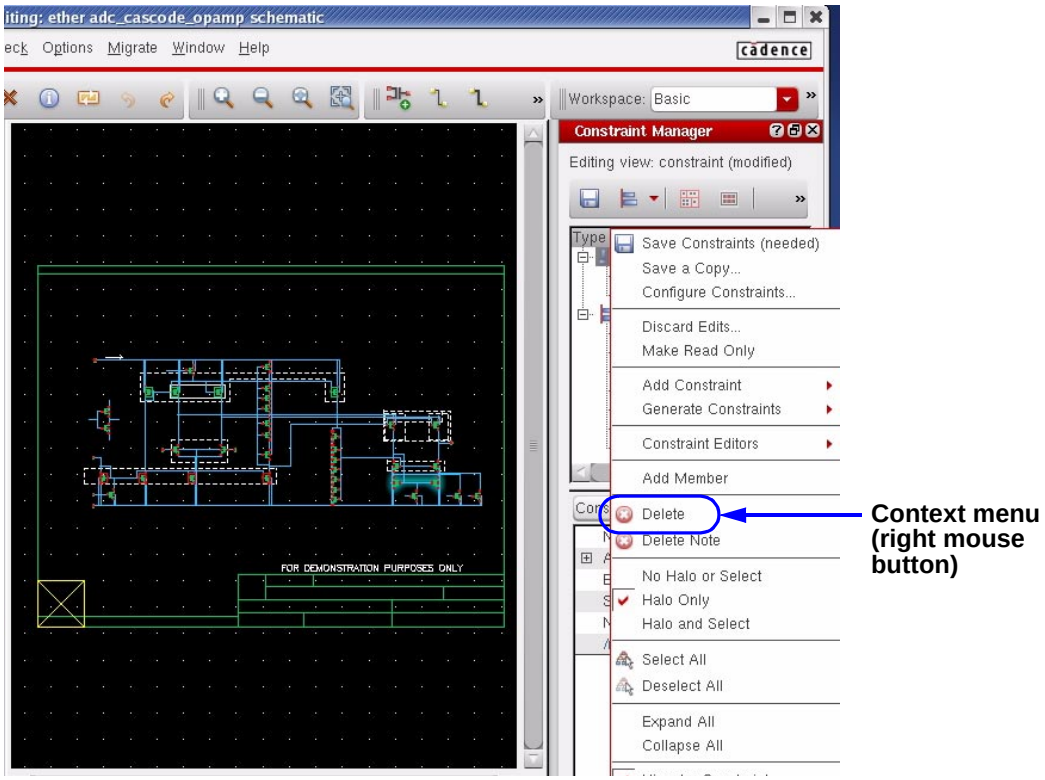
- You can control what constraint type and parameter names are displayed in the Constraint Manager using the default UI configuration file *config.xml.*

  The original *config.xml* file can be found in *$CDSHOME/share/cdssetup/dfII/ci/config.xml*. You can override this by creating a *config.xml* file in a *.cadence/dfII/ci* directory located on the Cadence file search path.

- An example of plugging in Module Generator to the Constraint Manager user interface:

```
ciRegisterConstraintEditor(
  list(nil 'name "Module Generator"
  'description "Modgen - Layout Structure Editor"
  'constraintType "modgen"
  'constraintParams list( "type=module")
  'editorAvailableExpression "isCallable('mgCreateOrEdit)"
  'startEditorExpression mgCreateOrEdit(geGetEditCellView ()
  constraint nil)"
  'iconName "nexGenEd"
  'addToToolbar t
  )
)
```

Deleting Constraints

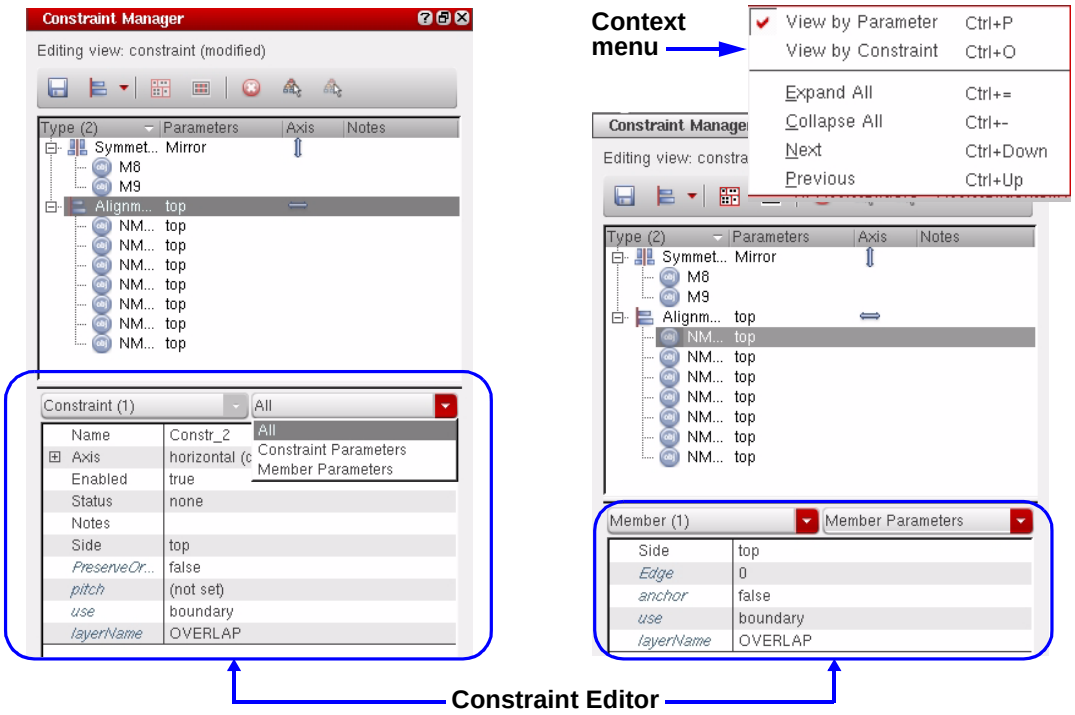


Steps to delete constraints:

- Select one or more constraints from the Constraint Manager table.
- Do one of the following:
  - Right-click in the Constraint Manager table and select **Delete**.
  - Select the **Delete** button from Constraint Manager tool bar.
  - Press the **Delete** key.

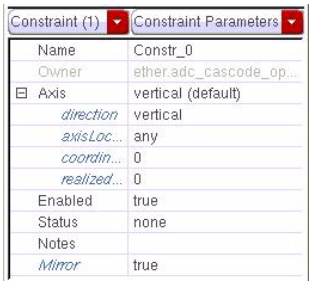
Editing Constraints

Edit constraint parameters and member parameters from Constraint Editor.



- A predefined constraint parameter describes the required electrical and physical implementation to the constraint-aware tools. Example: axis, precedence.
- Individual constraint members can have their own member parameters. Example: each member of an alignment constraint can have a specific *side*.
- Not all constraints allow member parameters.

Editing Constraints (continued)



Constraint	Parameters	Constraint	Parameters
Axis	Horizontal/vertical direction, any/fixed location	Status	Enforced — the constraint has been met/satisfied; impossible — not met; none
Enabled	True/false to control whether a constraint should be ignored	Notes	Notes

Constraint Editor Parameter Status Visualization (lower section of Constraint Manager):

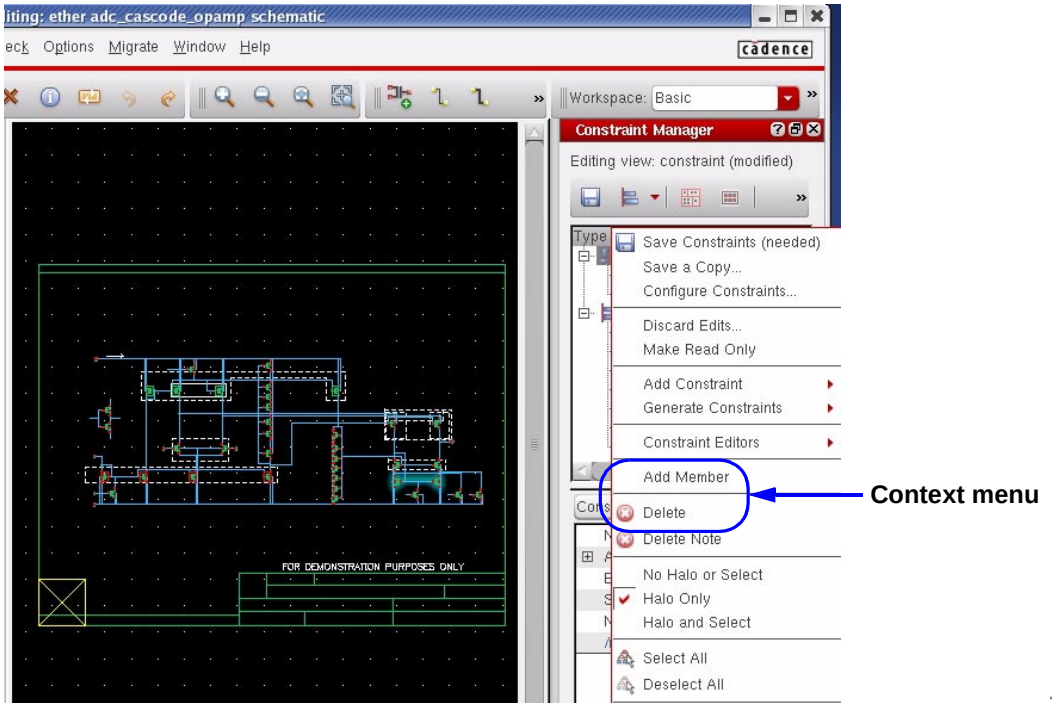
Color/Style	Meaning
Standard black	the value has been modified manually.
Italic blue	current value of constraint parameter is same as the default value.
Grey	read-only
Red	the parameter values across selected members are different.

Constraint status in Constraint Manager table upper section of Constraint Manager). The strike-through of a constraint indicates that the constraint is inactive.

Constraint Status	Text Color	Strike-Through
Enforced	Green	Not available
Impossible	Red	Not available
ReadOnly	Grey	Not available
Enabled/Disabled	Purple	Available
Overridden	Pink	Available
OutOfContext	Orange	Available

Editing Constraints (continued)

Add or remove a member to an existing constraint:



Steps to add a new member to a constraint:

1. Select a constraint from the Constraint Manager table.
2. Select the member in the design.
3. Right-click in the Constraint Manager table and select **Add Member**.

Steps to remove a member:

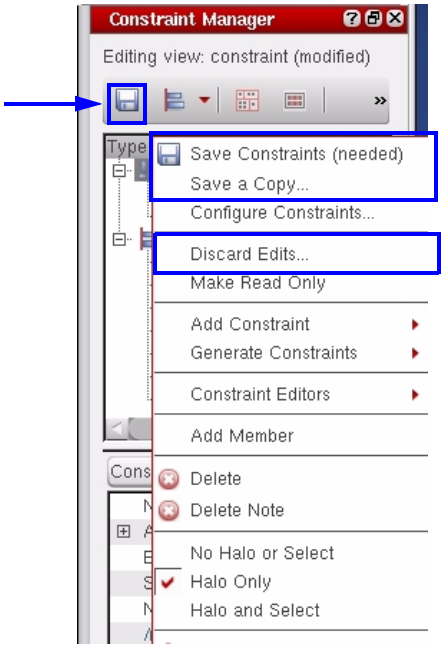
- Select a member under a constraint from the Constraint Manager table.
- Do one of the following:
  - Right-click in the Constraint Manager table and select **Delete**.
  - Select the **Delete** button from Constraint Manager tool bar.
  - Press the **Delete** key.

Saving Constraints

- Front-end constraints are stored in a separate view called *constraint*.

  The *constraint* view cannot be opened directly.

- Physical constraints are stored in the layout cellview using native OA object called *oaConstraint*.



- The pull-down menu commands from the design window:
  - File—Save (Constraint)
  - File—Save a Copy
  - File—Discard Edits (Constraint)
- Why store front-end constraints in a separate view?
  - Read-only schematics.
  - Constraints need to support an entire design hierarchy, not just the current view.
  - Future planning of constraints.
- Environment variable to specify the default constraint view name:

```
constraintdefaultViewName 'string "myConstr"
```

Constraint Manager Assistant UI Features

- Sort columns alphabetically
- Drag and drop
- Reorder columns
- Resize columns
- Expand and collapse the member list under a constraint
- Halos
- Cross-selection with the Navigator
- Tool tips
- Toolbar shortcuts
- Context menus accessed from the right mouse button

Additional bindkeys for Constraint Manager:

Key Binding	Action
Delete	Delete the current selection
Ctrl+A	Select all constraints
Right arrow	Expand
Left arrow	Collapse
Up/down arrow	Move up/down the Constraint Manager tree
Shift/Control+Select	Extended selection

Constraint SKILL API

- License
  - In L, using the constraint API requires a separate license feature: 95400.
  - In XL and GXL, the license permits you to use constraints from either the GUI or the API (that is, the licensing scheme does not check for the presence of 95400).
- Features
  - Using GUI equivalent SKILL API to add, delete, or modify constraints
    For example: *ciConCreate, ciConDelete, ciConSet*, ciConUpdate**
  - Creating user-defined constraints
  - Registering your own constraint generator with Constraint Manager so that an icon for the newly registered constraint is added to the GUI
  - Using a third-party editor to edit constraints.
  - Creating new finders
  - Creating new categories

An example of creating a user-defined constraint:

- Use the Property Dictionary Extension file *propdict.def*. This file may be placed anywhere in the *$path*. Add the following constraint definition in this file:

```
occPropGroupType MyConstraint
{
    legalPropName month
    {
        valueType = int;
        valueDefault = 1;
    }
    legalPropName year
    {
        valueType = int;
        valueDefault = 2006;
    }
    description = "Year and Month Constraints";
    toolTip = "Setup year and month constraint definition";
}
```

  This new constraint type automatically appears in Constraint Manager. To use the API (in the schematic; in the layout, the member names will have to be modified accordingly), do the following:

```
cv=geGetEditRep()
cache = ciCacheGet( cv~>libName cv~>cellName cv~>viewName)
ciConCreate( cache 'MyConstraint
   ?members list( list( "M2" 'inst) list( "M3" 'inst) )
   ? parameters list( list( "month" 3 ) list( "year" 1976 )
        )
```

- The constraint cache is a collection of constraint data that is built on demand for a given design and contains all the constraints related to that design.

Additional Constraint Information

Virtuoso Unified Custom Constraints User Guide:

http://sourcelink.cadence.com/docs/files/Release_Info/Docs/constraints/constraints6.1/preface.html

Virtuoso Unified Custom Constraints Configuration Guide:

http://sourcelink.cadence.com/docs/files/Release_Info/Docs/constraintsCustom/constraintsCustom6.1/preface.html

Labs

Lab 6-1 Creating Constraints Using the Constraint Manager

Lab 6-2 Creating Constraints Using the Circuit Prospector

Lab 6-3 Constraints Propagation

Using ADE L

Module 7

December 14, 2006

Module Objectives

In this module, you will

- Explore the new functionality in ADE L
- Customize the ADE L environment
- Use the ADE L environment to run a simulation

Introducing: Virtuoso Analog Design Environment L

- The ADE user interface enhancements are based on Qt instead of the SKILL® language.
- Stop time, sections can be parameterized using the VAR syntax.
- There are many new calculator functions.
- A hierarchical configuration is now possible. (A configuration can contain a configuration.)
- Virtuoso® Visualization and Analysis is the waveform display (enhanced WaveScan).
- Corners and sweeps have moved to the XL environment.
- The Spectre®, SPICE, and UltraSim® simulators are now in the MMSIM stream.

Using ADE L 7-3

To start ADE L, choose **Launch – ADE L** from the Virtuoso® Schematic Editor (VSE). This command replaces the *Tools – Analog Environment* command.

There are some changes to the license feature numbers. In IC 6.1, the new feature number for ADE L is not 34510 but 95200. Hence, you cannot run ADE L with feature # 34510. Obviously, you need IC 61 licenses to run any IC 6.1 products.

ADE User Interface Enhancements

The user interface includes:

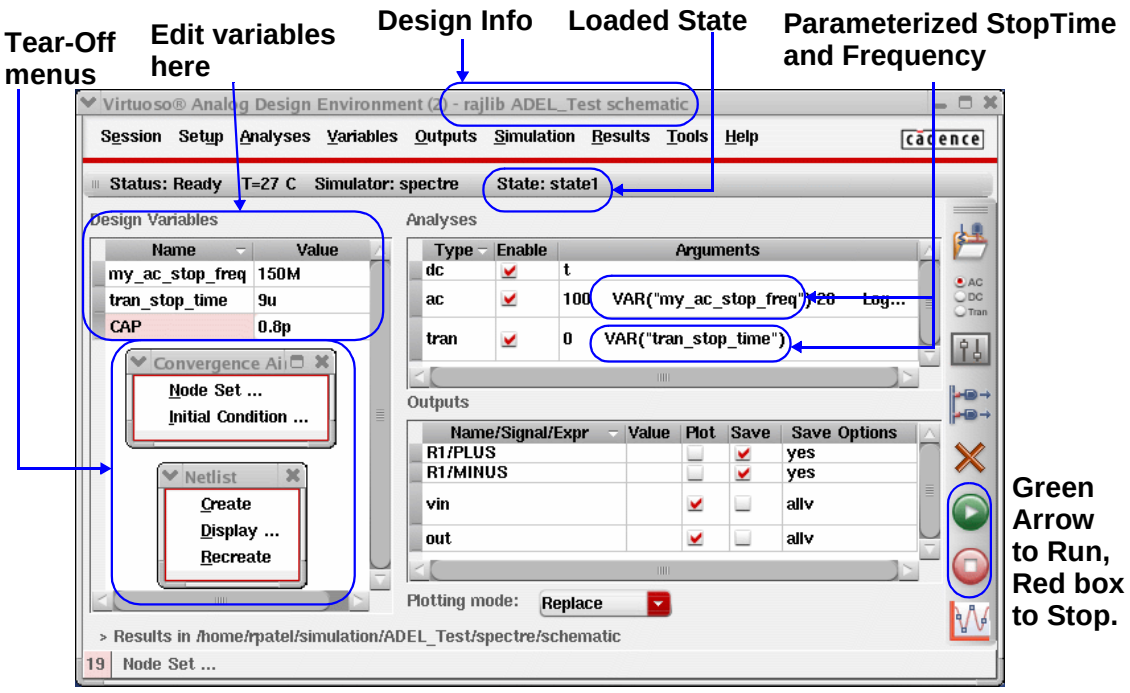
- Dockable menus
- Resizable main window
- Repositionable button bar (left-right, top-bottom)
- Edit-in-place capability for Design Variables, Analyses, and Outputs
- Check boxes in the Outputs field to enable/disable analyses, plotting, and saving

Using ADE L 7-5

Starting with IC 6.1.0, ADE (ADE L) has a new look and feel. The interface is more user friendly. You can increase or decrease the size of the window to your needs.

You can edit some of the fields directly on the main form. For example, you can enable or disable an analysis using the check box displayed along with the analysis on the form. You can also modify the values in the *Design Variables* section directly from the field. You can change order of design variables from A to Z or Z to A. You can change design variables order according to descending or ascending associated values. In the *Outputs* section, you can specify whether the output needs to be plotted and/or saved using the check boxes: Plot and Save.

Introducing ADE L

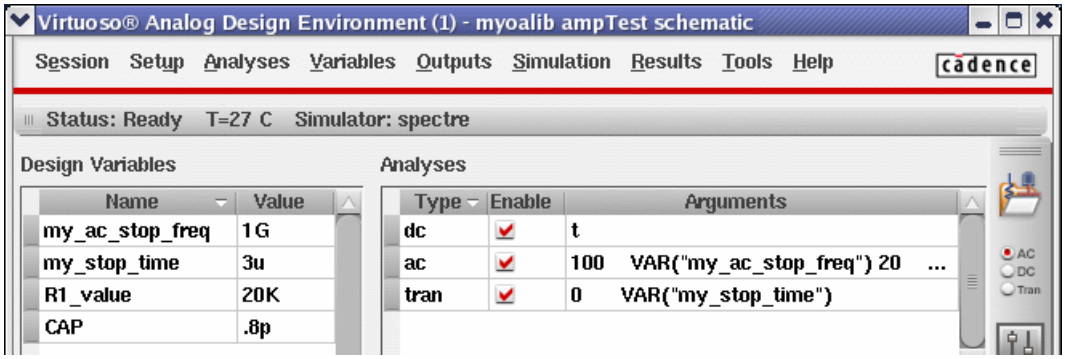


The ADE window has some major changes. However, you can still recognize the old ADE window with a new name, *ADE L*. Here are some changes and improvements:

- The *Design* section in the old ADE window has been removed. Now the design information shows up on the banner of the Virtuoso Analog Design Environment (ADE L) window.
- Your loaded state shows up on the ADE L window.
- You can tear off commonly used pull-down menus by pulling down the menu with the dotted line and then clicking on the dotted line.
- The old Yellow light (Run) is replaced by a Red Box (Stop)

Parameterized Setup

- You can parameterize the environment setup with VAR().
  - VAR(“my_design_variable_name”) for static values
  - VAR(“my_stop_time”) for transient stop time
  - VAR(“my_ac_stop_freq”) for ac stop frequency
  - VAR(“my_model_path”) for model library path
- Static parameterized variables turn into “_EXPR_#” parameters in the netlist.



Starting with IC 6.1.0, you can set variables in several ADE forms using VAR syntax. When the variable is specified in the VAR syntax, ADE automatically adds it as one of the design variables. Most of the forms support the usage of VAR syntax. You can specify any design variables in terms of other design variables using VAR syntax.

Environment setup can now be parameterized using the VAR() functions. By using the VAR(“my_design_variable”) syntax in place of static values, you can parameterize user interface settings such as analysis settings (transient stop time or ac stop frequency) and model file paths.

The support for VAR syntax is simulator independent and works in all default Cadence® simulator interfaces including Spectre®, UltraSim®, AMS, SpectreVerilog, and UltraSimVerilog software. It also works for custom third-party simulator interfaces, provided that the integration uses Cadence recommended guidelines.

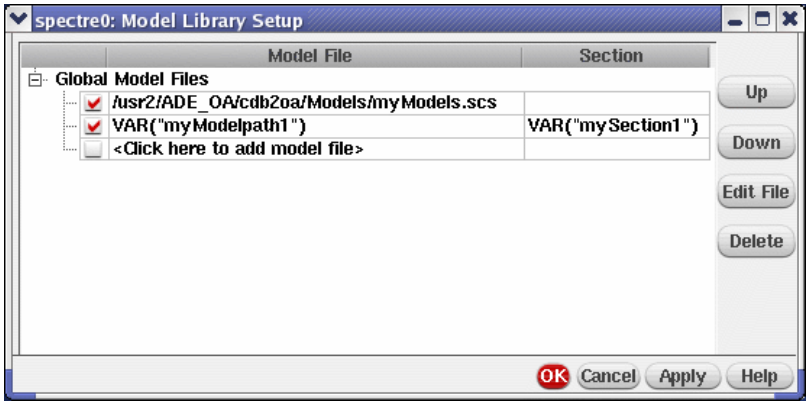
For example, “my_stop_time” and “my_ac_stop_freq” parameters create new parameters like “_EXPR_1” and “_EXPR_2” in the parameters line in the Spectre netlist.

Important

When you enter a design variable value, make sure that you press Return on your keyboard. Otherwise the change in the parameterized value might not take effect.

Parameterized Model Library Path

- Specify model file and section using VAR syntax.
- Assign string values to “myModelpath1” and “mySection1” in ADE L.



Use double quotes in the value field for these string variables.

When you parameterize model file and section using VAR syntax, the parameterized variables automatically show up in the *Design Variables* section of the ADE L window. You need to assign the actual path to the model file (*myModelpath1*) and section name (*mySection1*). These two variable entries need to be enclosed in double quotes in the *Value* field.

New Calculator Functions in IC 6.1.0

- compare
- dnl
- dutyCycle
- evmQAM
- evmQpsk
- freq_jitter
- histo
- peak
- period_jitter
- pzbode
- pzfilter
- spectrum
- unityGainFreq

The new functions above are described in the following pages.

The eyeDiagram, fourEval, sample, and value functions are listed in the *What's New* page from ADE L, but were available in IC 5141.

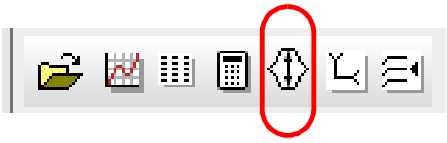
The compare Function

- This function compares the two given waveforms based on the specified values for absolute and relative tolerances.

  Arguments: Signal1, Signal2, Absolute Tolerance, Relative Tolerance

- This function compares only the sections of the two waveforms where the X or independent axes overlap.

- Similar to “Difference of two signals” when the Absolute Tolerance and Relative Tolerance is zero.



Example

```
compare(VT("/out") VT("/vin") 0.05 0.02)
```

Using ADE L 7-15

This function compares the two given waveforms based on the specified values for absolute and relative tolerances. This function compares only the sections of the two waveforms where the X or independent axes overlap.

- Signal1 is the name of the first signal.
- Signal2 is the name of the second signal.
- Absolute Tolerance is the absolute tolerance.
- Relative Tolerance is the relative tolerance.

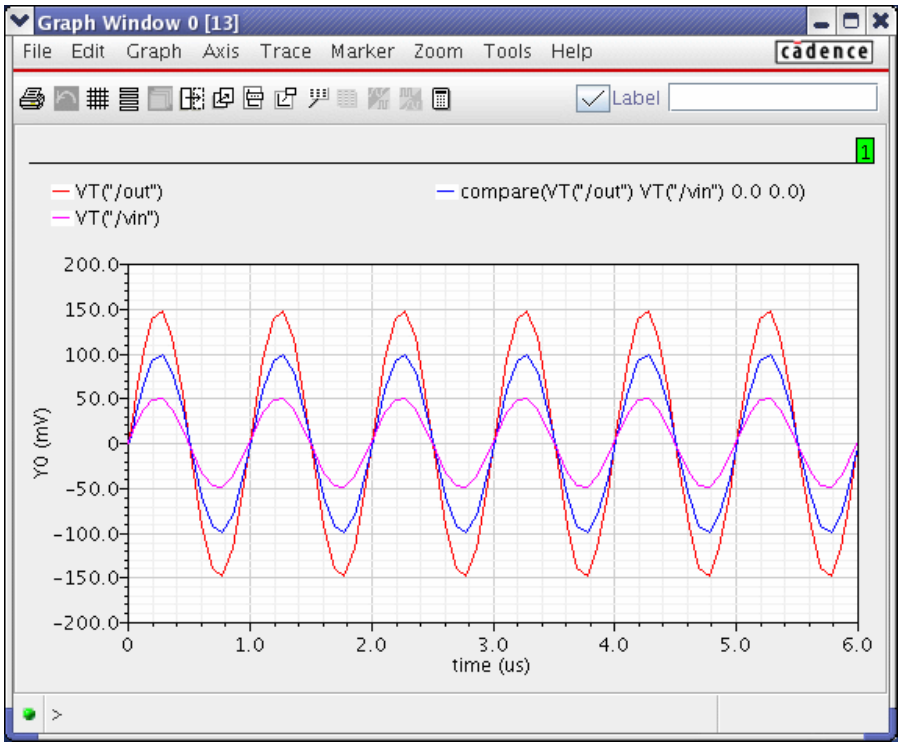
The following situations are possible:

- If neither relative nor absolute tolerance is specified, the function returns the difference of the two waveforms (Signal1 - Signal2).
- If only the absolute tolerance is specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the absolute tolerance (|Signal1 - Signal2| > f_abstol); otherwise it returns a zero waveform.
- If only the relative tolerance is specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the product of the relative tolerance and the larger of the absolute values of the two waveforms (|Signal1 - Signal2| > f_reltol * max(|Signal1|, |Signal2|)); otherwise it returns a zero waveform.
- If both relative and absolute tolerances are specified, the function returns the difference of the two waveforms only when the absolute value of the difference is greater than the sum of the separately calculated tolerance components (|Signal1 - Signal2| > f_abstol + f_reltol * max(|Signal1|, |Signal2|)); otherwise it returns a zero waveform.

Note: The function also compares parametric waveforms. However, for a successful comparison of parametric waveforms, the family tree structures of the two input waveforms need to be the same. For both the input waveforms, the number of child waveforms at each level need to also be the same, except at the leaf level where the elements are simple scalars.

The compare Function (continued)

The output plot of the compare function looks like this.



The dnl Function (differential nonlinearity) for A2D and D2A

- This function computes the differential nonlinearity of a transient waveform.

  The waveform can be a simple or parametric.

- This function is Inherited from Virtuoso Specification-driven Environment.
- Arguments: Waveform, Sampling signal/list/step, Cross Type, Mode, Threshold, Delay, Method, Unit, Number of Samples

Example

```
dnl(VT("/out1") VT("/out2") ?mode "auto" ?crossType "rising" ?delay 0.0
?method "end" ?units "lsb" ?nbsamples nil)
```


This function computes the differential nonlinearity of a transient simple or parametric waveform.

- Waveform is the name of the signal.
- Sampling signal/list/step is the signal used to obtain the points for sampling the Waveform (points at which the waveform crosses the threshold while either rising or falling as specified in the cross Type field with the delay added to them), list of domain values at which the sample points are obtained from the Waveform, or the sampling interval.
- Cross Type specifies the points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising (rising) or falling (falling).
- Mode specifies whether the threshold value is to be calculated by WaveScan (auto) or specified by you (user). The auto threshold is calculated as:

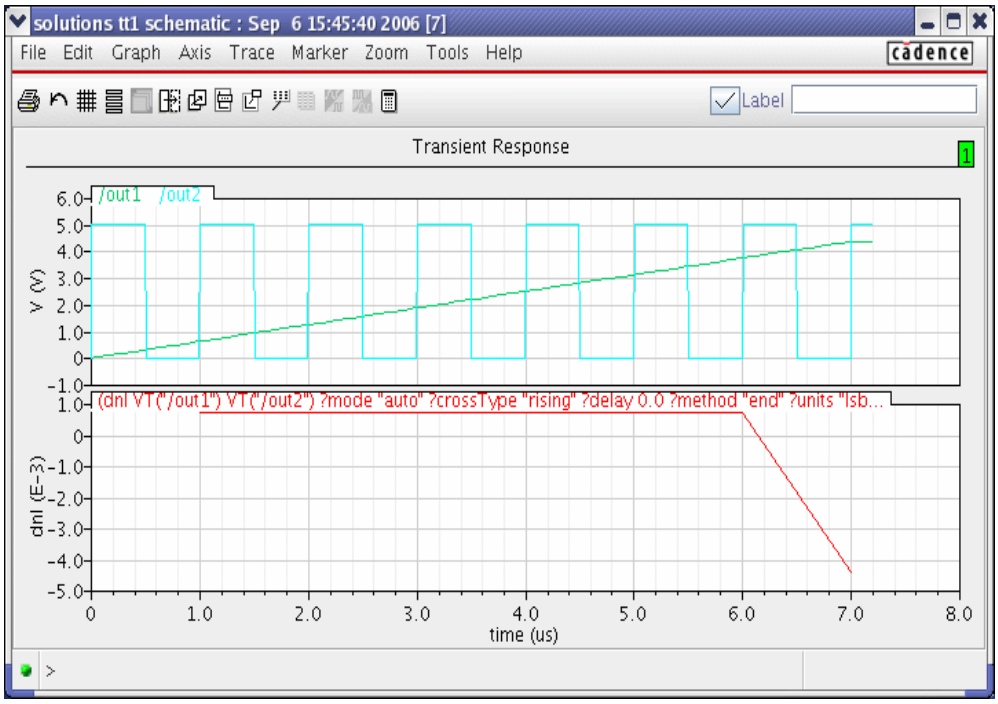
  Auto Threshold Value = integral of the waveform divided by the X range.
  - Threshold is the threshold value against which the frequency is to be calculated. The threshold input will only be considered if mode == "user".
  - Delay is the delay time after which the sampling begins.
  - Method specifies whether the end-to-end (end) or straight line (fit) method is used.
  - Unit specifies whether the output waveform is to be output as an absolute value (abs) or multiples of least significant bit (lsb).
  - Number of Samples specifies the samples used for calculating the nonlinearity. If not specified, the samples are taken against the entire data window.

Note: For each of the three ways in which the sample points can be specified, only a few of the other optional arguments are meaningful.

- For Sampling signal, the fields Cross Type, mode, Threshold, Delay, Method, and Units are meaningful.
- For list, the fields Method and Units are meaningful.
- For step, the fields Method, Units, and No. of samples are meaningful.

The dnl Function (continued)

The output plot of the dnl function looks like this one. The plot of dnl shows the maximum dnl = -4.5e-3 of lsb.



The dutyCycle Function

- This function calculates the ratio of the time for which the signal remains high to the period of the signal.
- Use this function on periodic signals only.
- Arguments: Waveform, mode, threshold, Plot/print vs., Output Type

Example

```
dutyCycle(VT("/out") ?mode "auto" ?xName "time" ?outputType "plot")
```

Using ADE L 7-23

Calculates the ratio of the time for which the signal remains high to the period of the signal. You should use this function on periodic signals only.

In the SKILL mode,

- Waveform is the name of the signal, expression, or family of waveforms.
- mode specifies whether the threshold value is to be calculated by WaveScan (auto) or specified by you (user).

The auto threshold is calculated as:

Auto Threshold Value = integral of the waveform divided by the X range.

- Threshold is the threshold value. The threshold input will only be considered if mode == "user".
- Plot/print versus specifies whether X axis of the output waveform is time (or another X-axis parameter for non-transient data) or cycle.
- Output Type is the type of output. If set to plot, the output is a waveform; if set to average, the output is an average value. The default value is plot.

In the MDL mode,

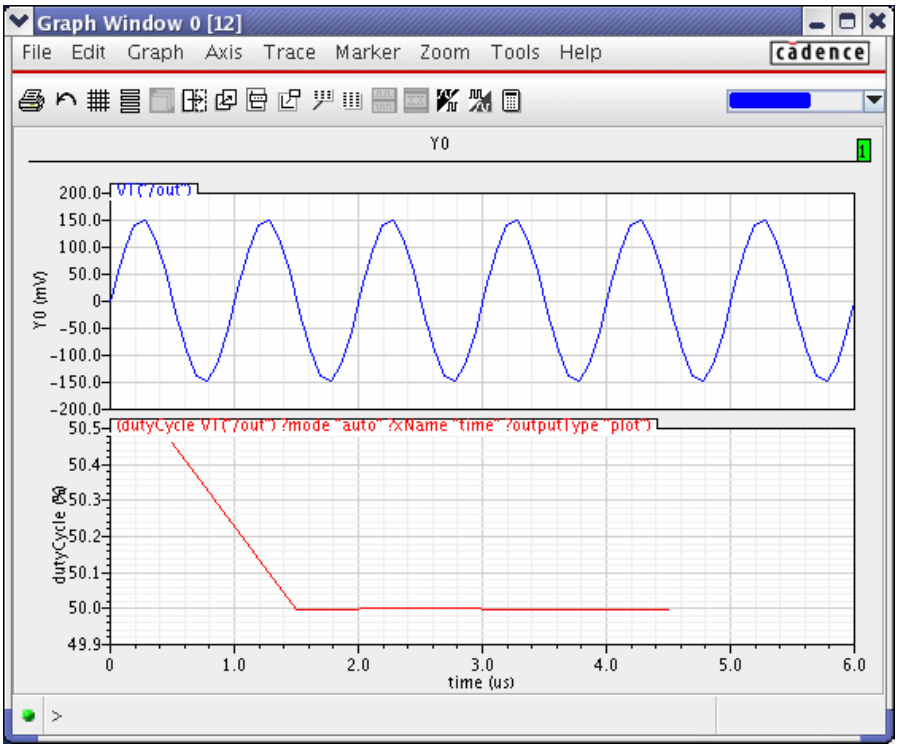
- sig is the name of the signal.
- theta is the percentage that defines the logic high of the signal. A threshold value is calculated as follows:

  yThresh=theta/100*(Ymax+Ymin)

  The portion of the signal above yThresh is taken as high.

The dutyCycle Function (continued)

The output plot of the dutyCycle function looks like this.



Note: This function requires about 1.5 cycles of a waveform to start calculating the correct values of an input waveform before it provides a correct output.

The evmQAM Function

- Calculates the Error Vector Magnitude (EVM) for multi-mode modulations of I and Q waveform outputs from the transient simulation.
- EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality.
- Based on a statistical error distribution normalized from an ideal digital modulation.
- Quadrature Amplitude Modulation (QAM) is a typical modulation scheme where EVM is useful.
- Plots the I versus Q scatter plot.
- Arguments: I-Signal, Q-Signal, Symbol Start, Symbol period, Modulation Level, Normalize Display

Example

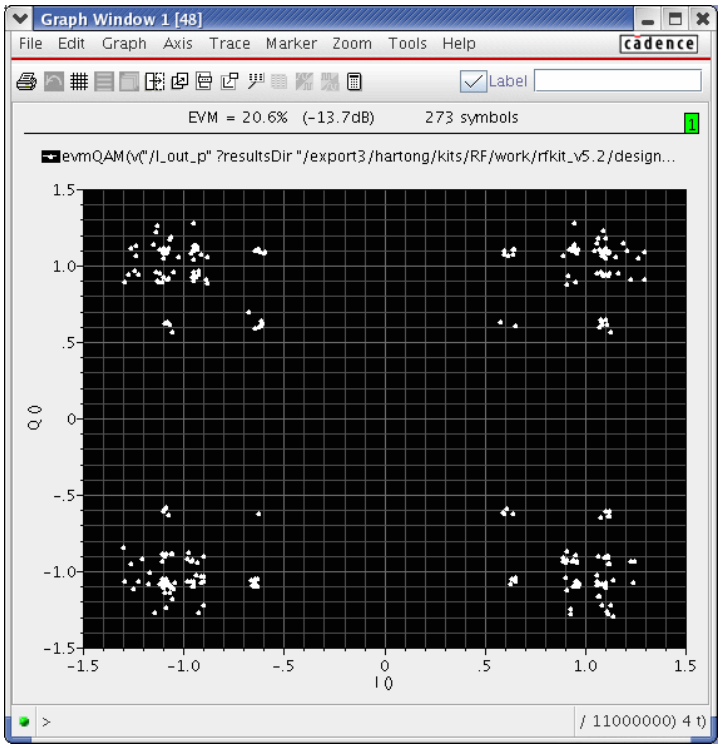
```
evmQAM(VT("/samp_out_I"), VT("/samp_out_Q"), 1.5u, 181.81n, 4, t)
```


Processes the I and Q waveform outputs from the transient simulation run to calculate the Error Vector Magnitude (EVM) for multi-mode modulations. The function plots the I versus Q scatter plot. EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Quadrature Amplitude Modulation (QAM) is a typical modulation scheme where EVM is useful. The EVM is calculated by detecting the I and Q signal levels corresponding to the four possible I and Q symbol combinations and calculating the difference between the actual signal level and the ideal signal level.

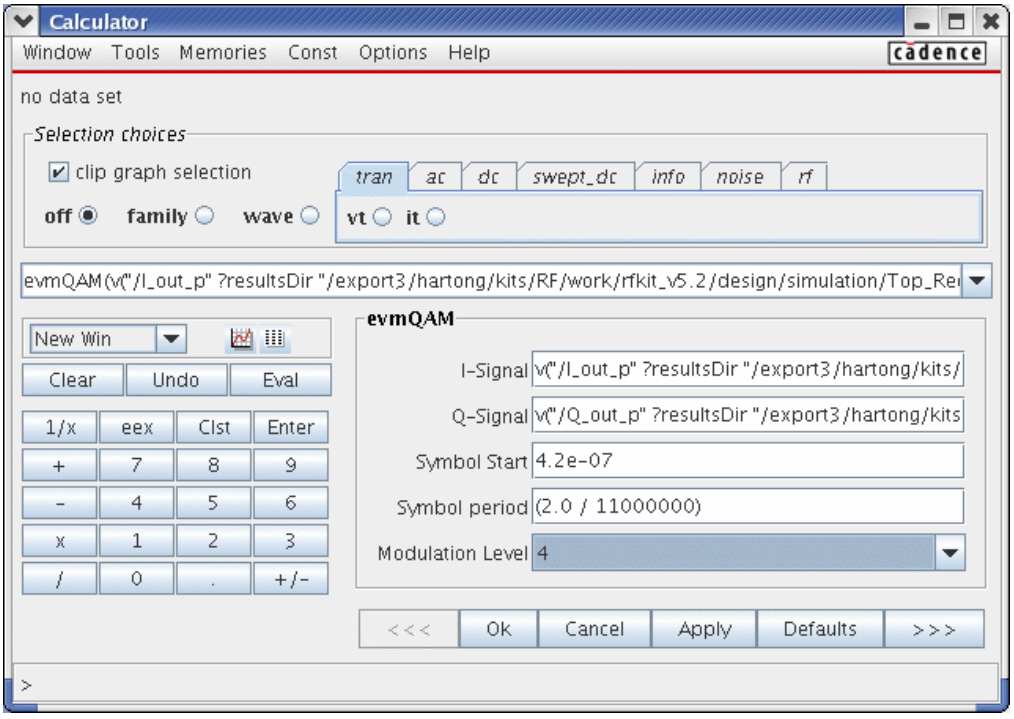
- This function is not supported for families of waveforms.
- The evmQAM function is available only in the SKILL mode.
- I-Signal is the waveform for the I signal.
- Q-Signal is the waveform for the Q signal.
- Symbol Start is the start time for the first valid symbol.
- Symbol period is the period for the symbol. Each period is represented by a data rate. The data rate at the output is determined by the particular modulation scheme being used. For example, if the data rate selected is 5.5 Mbps, it corresponds to a period of 181.8 ns.
- Modulation Level is the modulation level.
- Normalize Display normalizes the scatter plot to the ideal values +1 and -1.

evmQAM Results

The output plot of the evmQAM function looks like this.



The calculator setup for *evmQAM* looks like this.



The evmQpsk Function

- This function calculates the Error Vector Magnitude (EVM) for multi-mode modulations of I and Q waveform outputs from the transient simulation.
- EVM is a useful measurement to describe the overall signal amplitude and phase-modulated signal quality.
- EVM is based on a statistical error distribution normalized from an ideal digital modulation.
- Quadrature Phase Shift Keying (QPSK) is a typical modulation scheme where EVM is useful.
- Plots the I versus Q scatter plot.
- Arguments: I-Signal, Q-Signal, Symbol Start, Symbol period, Auto Level Detect, Amplitude(V), Offset(V), Normalize Display

Example

```
evmQpsk(VT("/samp_out_I"), VT("/samp_out_Q"), 1.5u, 181.81n, t, nil,
nil, t)
```


Processes the I and Q waveform outputs from the transient simulation run to calculate the Error Vector Magnitude (EVM) and plot the I versus Q scatterplot. EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Quadrature Phase Shift Keying (QPSK) is a typical modulation scheme where EVM is useful. The EVM is calculated by detecting the I and Q signal levels corresponding to the four possible I and Q symbol combinations and calculating the difference between the actual signal level and the ideal signal level.

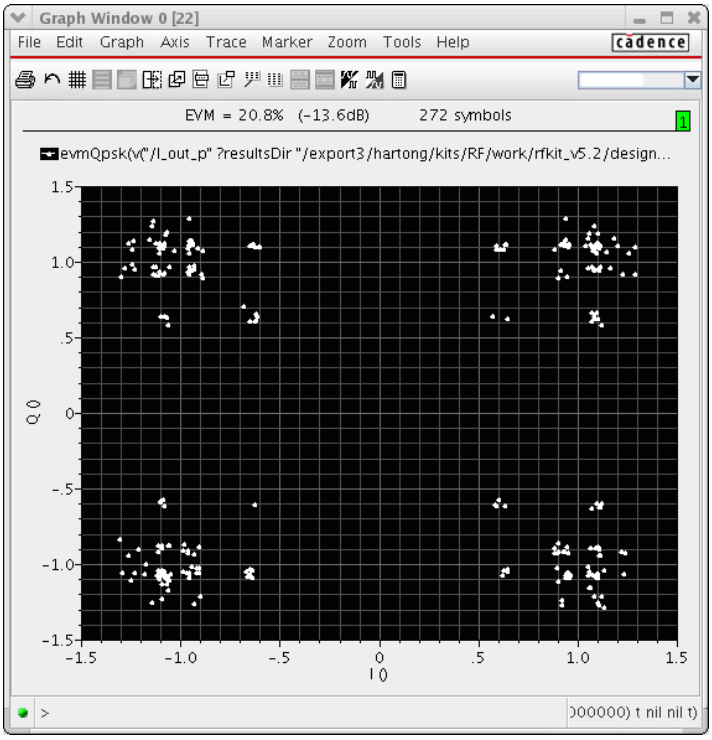
This function is not supported for families of waveforms.

This function is available only in the SKILL mode.

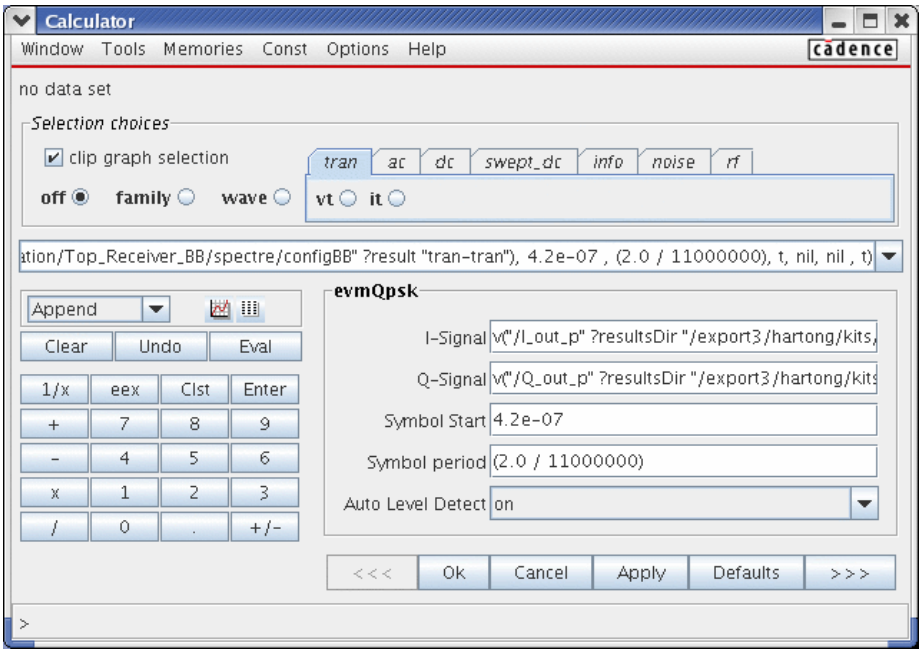
- I-Signal is the waveform for the I signal.
- Q-Signal is the waveform for the Q signal.
- Symbol Start is the start time for the first valid symbol.
- Symbol period is the period for the symbol. Each period is represented by a data rate. The data rate at the output is determined by the particular modulation scheme being used. For example, if the data rate selected is 5.5 Mbps, it corresponds to a period of 181.8 ns.
- Auto Level Detect on indicates that you want the amplitude (Amplitude) and DC offset (Offset) to be calculated automatically. Amplitude is calculated by averaging the rectified voltage level of the signal streams and DC Offset by averaging the sum of an equal number of positive and negative symbols in each signal stream. These values are used to determine the EVM value. If Auto Level Detect is set to off, you must specify values for the Amplitude and Offset fields.
- Amplitude(V) is the amplitude of the signal. You need to specify a value in this field only if Auto Level Detect is set to off.
- Offset(V) is the DC offset value. You need to specify a value in this field only if Auto Level Detect is set to off.
- Normalize Display normalizes the scatter plot to the ideal values +1 and -1.

evmQpsk Results

The output plot of the evmQpsk function looks like this.



The setup for *evmQpsk* looks like this.



The freq_jitter Function

- This function calculates the frequency jitter.
- Returns a waveform representing the deviation from the average frequency.
- Arguments: Waveform, Cross Type, mode, Threshold, Bin Size, Plot/Print vs., Output Type

Example

```
freq_jitter(VT("/out") "rising" ?mode "user" ?threshold 1 ?binSize 2
?xName "cycle" ?outputType "sd")
```


Returns a waveform representing the deviation from the average frequency.

In the SKILL mode,

- Waveform is the name of the signal, expression, or family of waveforms.
- Cross Type is the points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising (rising) or falling (falling).
- Mode specifies whether the threshold value is to be calculated by WaveScan (auto) or specified by you (user). The auto threshold is calculated as:

  Auto Threshold Value = integral of the waveform divided by the X range.
- Threshold is the threshold value against which the frequency is to be calculated. The threshold input will only be considered if mode == "user".
- Bin Size is the width of the moving average window, The deviation of value at the particular point from the average of this window is the jitter.

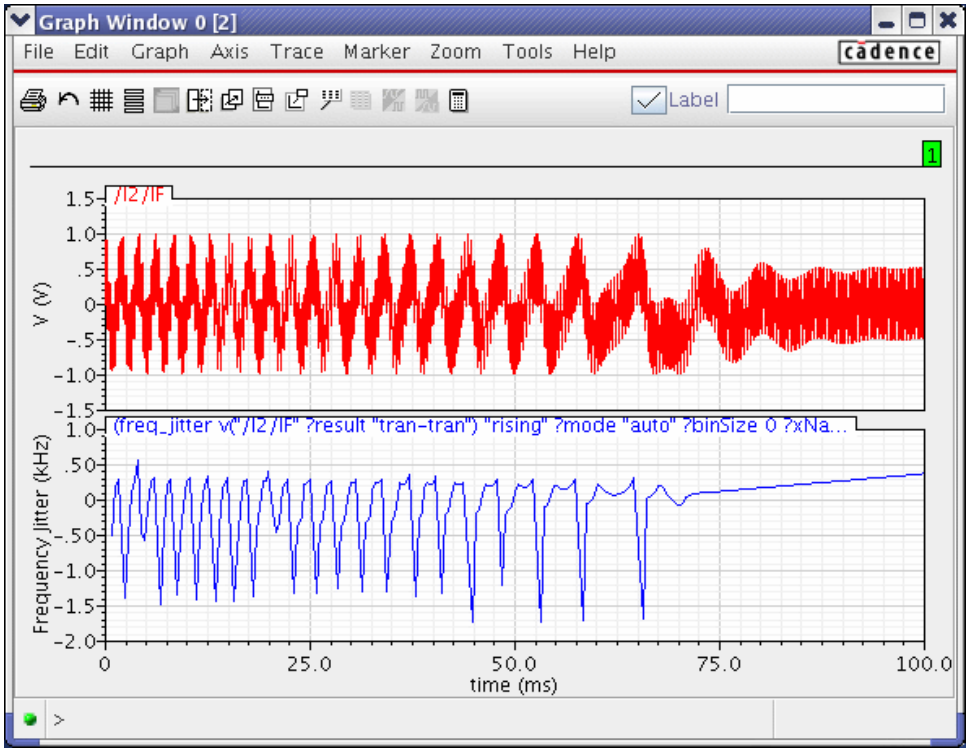
  If binsize=0, all frequencies are used to calculate the average.
  If binsize=N, the last N frequencies are used to calculate the average.
- Plot/print vs. specifies whether you want to retrieve the frequency jitter against time (or another X-axis parameter for nontransient data) or cycle. Cycle numbers refer to the n^{th} occurrence of the delay event in the input waveform.
- Output Type is the type of output. If set to sd, the output is a standard deviation jitter. If set to plot, the output is a waveform. The default value is plot.

In the MDL mode,

- sig is the name of the signal.
- thresh is the threshold Y-axis value to be crossed.
- dir is the direction of the crossing event.
- binsize is the integer used to calculate the average frequency of the signal. If binsize=0, all frequencies are used to calculate the average. If binsize=N, the last N frequencies are used to calculate the average.

The freq_jitter Function (continued)

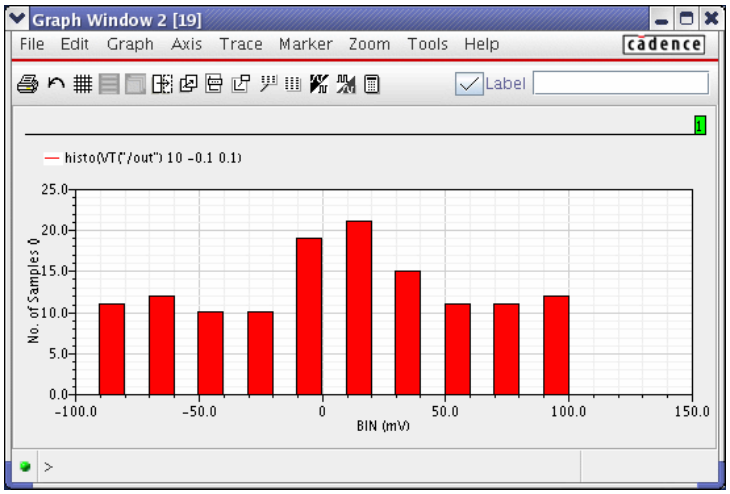
The results for freq_jitter look like this.



Note: This function requires about 1.5 cycles of a waveform to start calculating correct values of an input waveform before it provides a correct output.

The histo Function

This function returns a histogram that represents the statistical distribution of input data.



- The height of the bars (or bins) in the histogram represents the frequency of the occurrence of values within a specific period.
- Arguments: Signal, nbins, min, max

Example: histo(VT("/out") 10 -0.1 0.1)

This function returns a waveform that represents the statistical distribution of input data in the form of a histogram. The height of the bars (or bins) in the histogram represents the frequency of the occurrence of values within a specific period. Using the *histo* function, the range for capturing these frequencies can be specified through the min and max values.

- Signal is the waveform.
- nbins is the number of bins to represent the input data.
- min is the first value on the horizontal axis of the histogram. By default, it assumes the minimum value of the input waveform.
- max is the last value on the horizontal axis of the histogram. By default, it assumes the maximum value of the input waveform.

Example

The input values signal=V(out), nbins=10, min=-0.1, max=0.1 create a display with 10 bins that might look like the histogram above.

The peak Function

- This function detects peaks in the present waveform and outputs peak waveform.
- Output waveform contains the x-points and y-points of the peaks found in the input waveform and according to the filter criteria specified via xtol and ytol.
- Arguments: Signal, From, To, X-Tolerance, Y-Tolerance

Example

```
peak(VT("/out") ?from 1n ?to 20n ?xtol 2n ?ytol 0.5)
```


Detects the peaks in the present waveform and accumulates the X and Y coordinates of the peak points of the input waveform.

Arguments

Signal: The input waveform

From: Starting point selected in the input waveform. Default value is the first point of the waveform.

To: End point selected in the input waveform. Default value is the last point of the waveform.

X-Tolerance: X- tolerance (default value 0.0)

The function will filter all the peaks inside the neighborhood of *xtol* distance on the x axis for a given peak.

Y-Tolerance: Y- tolerance (default value 0.0)

The function will filter all the peaks inside the neighborhood of *ytol* distance on the y axis for a given peak.

Note:

(i) If both *xtol* and *ytol* are specified then the function will try to filter all the neighboring peaks in xtol (x axis) and ytol (y axis) region for a given peak.

(ii) If only xtol or ytol are given then the function will behave for xtol and ytol as per definition in the arguments.

(iii) The function is not defined for waveforms which have the Y-Vector of complex numbers.

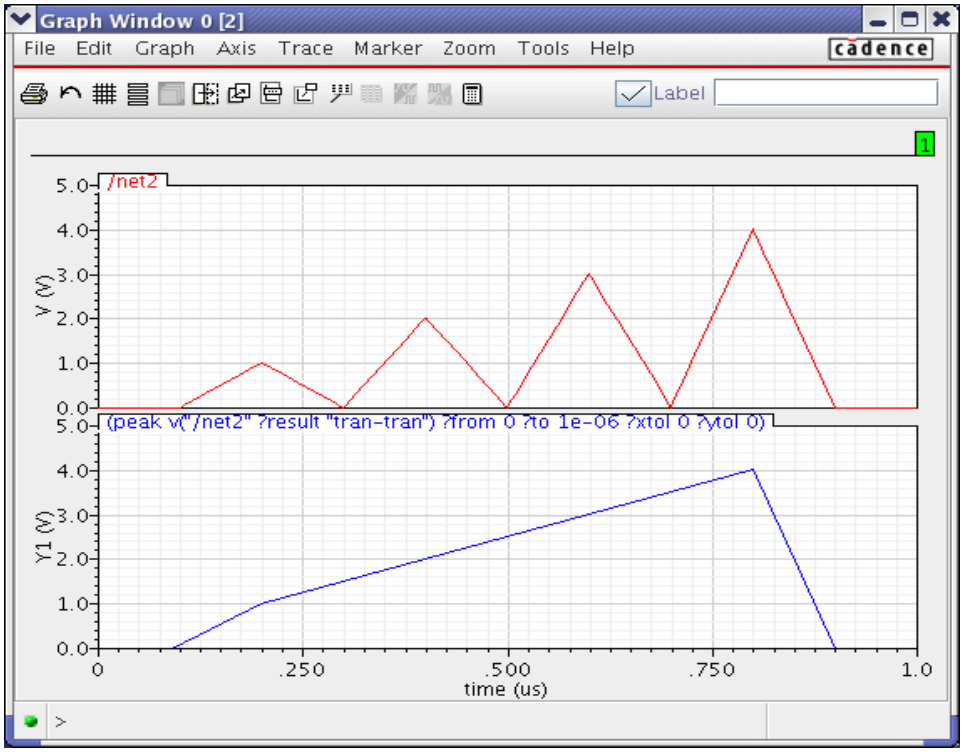
Value Returned

o_waveform is a waveform containing the x-points and y-points of the peaks found in the input waveform and according to the filter criteria specified via xtol and ytol.

nil returns nil and error message other wise.

The peak Function (continued)

The output plot of the peak function looks like this.



The period_jitter Function

- This function calculates the period jitter.
- This function returns a waveform representing the deviation from the average period.
- Arguments: Waveform, Cross Type, mode, Threshold, Bin Size, Plot/print vs., Output Type

Example

```
period_jitter(VT("/out") "rising" ?mode "user" ?threshold 1 ?binSize 2
?xName "cycle" ?outputType "sd")
```

Using ADE L 7-45

Returns a waveform representing the deviation from the average period.

In the SKILL mode,

- Waveform is the name of the signal, expression, or family of waveforms.
- Cross Type is the points at which the curves of the waveform intersect with the threshold. While intersecting, the curve may be either rising (rising) or falling (falling).
- Mode specifies whether the threshold value is to be calculated by WaveScan the function (auto) or specified by you. The auto threshold is calculated as:

  Auto Threshold Value = integral of the waveform divided by the X range.
- Threshold is the threshold value against which the frequency is to be calculated. The threshold input will only be considered if mode == "user".
- Bin Size is the width of the moving average window. The deviation of value at the particular point from the average of this window is the jitter.

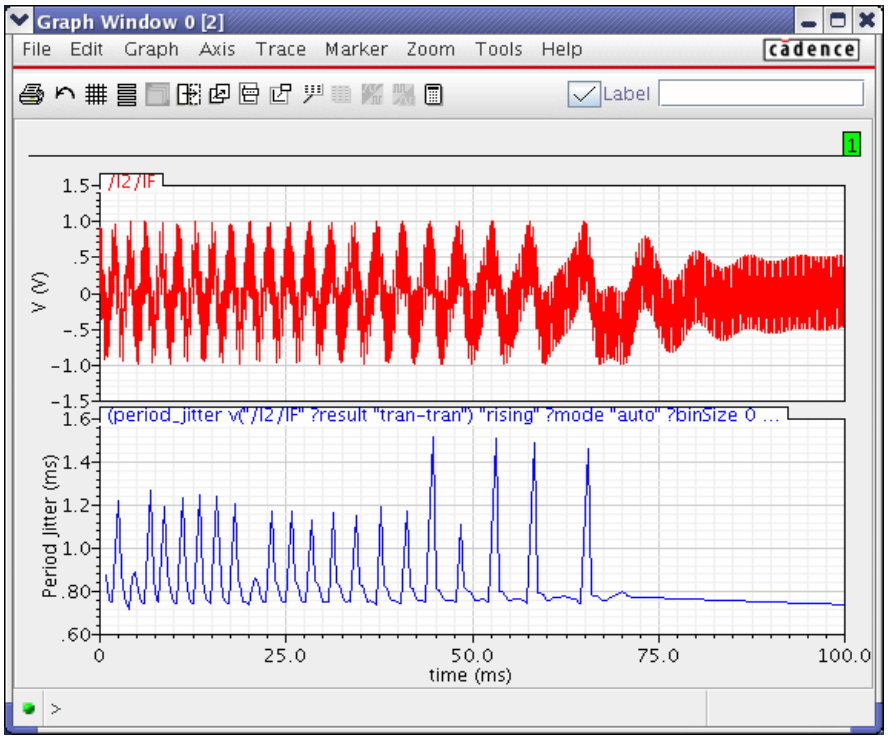
  If binsize=0, all frequencies are used to calculate the average.
  If binsize=N, the last N frequencies are used to calculate the average.
- Plot/print vs. specifies whether you want to retrieve the period jitter against time (or another X-axis parameter for non-transient data) or cycle. Cycle numbers refer to the n^{th} occurrence of the delay event in the input waveform.
- Output Type is the type of output. If set to sd, the output is a standard deviation jitter. If set to plot, the output is a waveform. The default value is plot.

In the MDL mode,

- sig is the name of the signal.
- thresh is the threshold Y-axis value defining the period/frequency of the signal.
- dir is the direction of the crossing event.
- binsize is the integer used to calculate the average frequency of the signal.
- If binsize=0, all periods are used to calculate the average. If binsize=N, the last N periods are used to calculate the average.

The period_jitter Function (continued)

The output plot of the period_jitter function looks like this.



Note: This function requires about 1.5 cycles of a waveform to start calculating the correct values of an input waveform before it provides a correct output.

The pzbode Function

- This function calculates and plots the transfer function of a circuit from pole zero simulation data.
- Arguments: DC Gain, Min. Frequency, Max Frequency, No. of Points, Poles, Zeros

Example

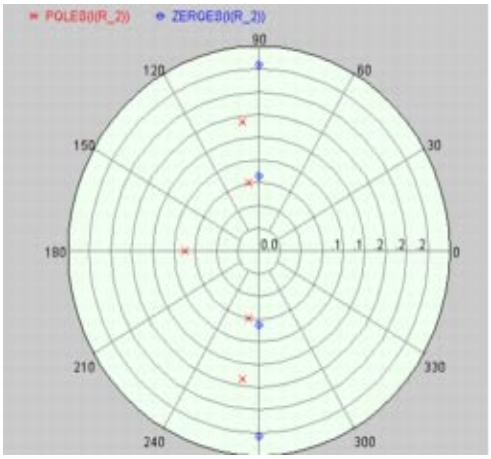
```
pzbode(1 1M 1G 20 ?poles "all" ?zeros "all")
```


This function calculates and plots the transfer function for a circuit from pole zero simulation data.

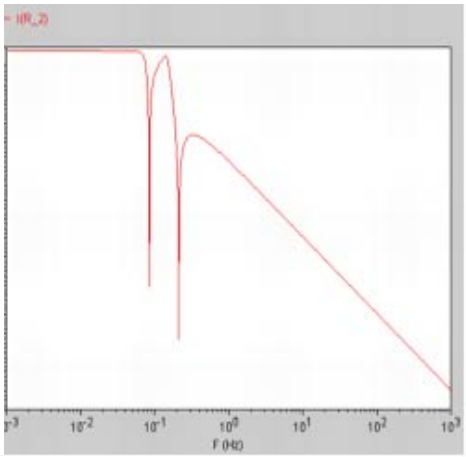
- poles are the poles from the simulation data.
- zeros are the zeros from the simulation data.
- DC Gain is the transfer gain constant.
- minfreq is the minimum frequency for the bode plot.
- maxfreq is the maximum frequency for the bode plot.
- npoints is the frequency interval for the bode plot, in points per decade.

The pzbode Function (continued)

Polar Plot



Corresponding Bode Plot



This diagram illustrates how the result with the values poles=POLESIR_1, zeroes=ZEROESIR_1, c=IR_1\[K\], minfreq=1e-3, maxfreq=1e3, and npoints=1000 is determined.

The pzfilter Function

- This function filters poles and zeros according to the specified criteria.
- This function works only on pole-zero simulation data.
- Arguments: maxfreq, reldist, absdist, minq

Example

```
pzfilter(?maxfreq 1G ?reldist 0.01 ?absdist 0.05 ?minq 10000)
```

Using ADE L 7-53

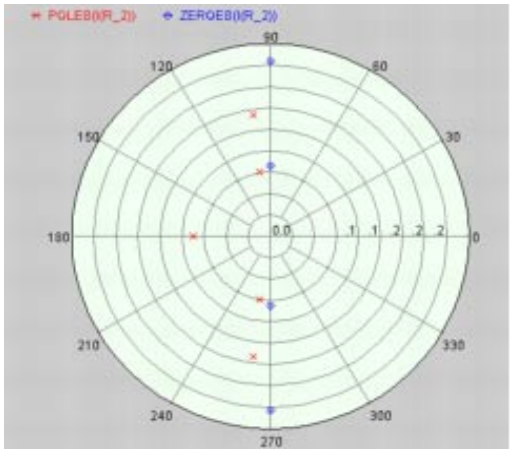
Filters the poles and zeros according to the specified criteria. The *pzfilter* function works only on pole-zero simulation data.

- poles are the poles.
- zeros are the zeros.
- maxfreq is the frequency up to which the poles and zeros are plotted.
- reldist is the relative distance between the pole and zero. Pole-zero pairs with a relative distance lower than the specified value are not plotted.
- absdist is the absolute distance between the pole and zero. Pole-zero pairs with an absolute distance lower than the specified value are not plotted.
- minq is the minimum Q-factor. Pole-zero pairs with a Q-factor less than the specified value are not cancelled.

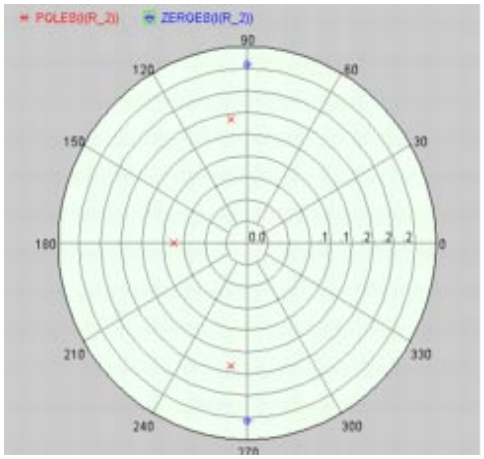
Note: If you do not specify maxfreq, reldist, absdist, or minq, then pzfilter filters out the poles and zeros with a frequency higher than 10 GHz (default value of maxfreq).

The pzfilter Function (continued)

Original Polar Plot



Filtered Polar Plot



Using ADE L 7-55

The values poles=POLESIR_2, zeroes=ZEROESIR_2, absdist=0.05, and minq=10000 filters pole-zero pairs with a relative distance of less than 0.05 Hz from the plot on the left side. In the filtered plot shown on the right side, two pole-zero pairs have been filtered out.

The spectrum Function (for A2D Converters)

- This function calculates following by using discrete Fourier transform of any given input signal:
  - Signal-to-Noise-and-Distortion Ratio (SINAD)
  - Spurious Free Dynamic Range (SFDR)
  - Effective Number of Bits (ENOB)
  - Signal-to-Noise Ratio (SNHR, without distortion)
- This function is available only in the SKILL mode.
- Arguments: Signal, Number of Samples, Number of Noise bins, Start Frequency, End Frequency, Window Type, ADC Span, Measure Type

Example

```
spectrum(VT("/vcoOut") 1K 0 1K 10G "Rectangular" 0 "snhr")
```


This function calculates Signal-to-Noise-and-Distortion Ratio (SINAD), Spurious Free Dynamic Range (SFDR), Effective Number of Bits (ENOB), and Signal-to-Noise Ratio (without distortion) by using discrete Fourier transform of any given input signal.

The spectrum measure is used for characterizing A-to-D converters and is typically supported for transient simulation data.

The spectrum function is available only in the SKILL mode.

- *Signal* is signal to be measured.
- *Number of Samples* is the number of sampled points used for the FFT. Valid values: Any integer power of two greater than zero. For a value that is not a power of two, the function rounds it up to the next closest power of two. Default value: Number of data points in the Signal.
- *Number of Noise bins* is the number of noise bins where the size of one bin is the reciprocal of the data window width. For example, 1 ms of transient data creates a bin size of 1 kHz. Valid values: Any integer power of two greater than or equal to zero. Default value: 0, implying that no signal is spilling into the bins. A frequency band of bin-size times the number of bins is calculated and adjusted as a function of the selected window. Frequency components in each band to the left and right of the fundamental or the harmonics are set to zero and do not contribute to any output result.
- *Start Frequency* is the lower limit of frequency range for the spectrum measures. Default value: First frequency point of the FFT.
- *End Frequency* is upper limit of frequency range for the spectrum measures. Default value: Last frequency point of the FFT.
- *Window Type* is the windowing function applied to o_waveform. Valid values: Blackman, Cosine2, Cosine4, ExtCosBell, HalfCycleSine, HalfCycleSine3, HalfCycleSine6, Hamming, Kaiser, Parzen, Rectangular, and Triangular. Default value: Rectangular.
- *ADC Span* is the full-scale span ignoring any DC offsets. This is used in ENOB calculation. Valid values: Any floating point number. Default value: If ADC Span is not specified or is nil, it is assumed to be 0 and is taken to be the peak-to-peak value of the fundamental.
- *Measure Type* is Result specifier. Valid values: sinad, sfdr(db), enob, and snhr.

The unityGainFreq Function

- This function computes and reports the frequency at which the gain is unity (0 dB).
- This function is available only in the SKILL mode.
- Argument: Gain-Frequency waveform

Example

```
unityGainFreq(VF("/out"))
```

Using ADE L 7-59

Reports the frequency at which the gain is equal to one (0dB).

```
unityGainFreq( o_gainFreqWaveform )
===> n_frequency / nil
o_gainFreqWaveform: Gain-Frequency waveform
```

Before this function was available, the UGF functionality was achieved with the following public functions:

```
cross(mag( VF("out") ),1,1,"either",nil,nil)
```

If the input is a transfer function, then you can have:

```
cross( mag(VF("output")/VF("input")),1,1,"either",nil,nil)
```

Hierarchical Configuration Support

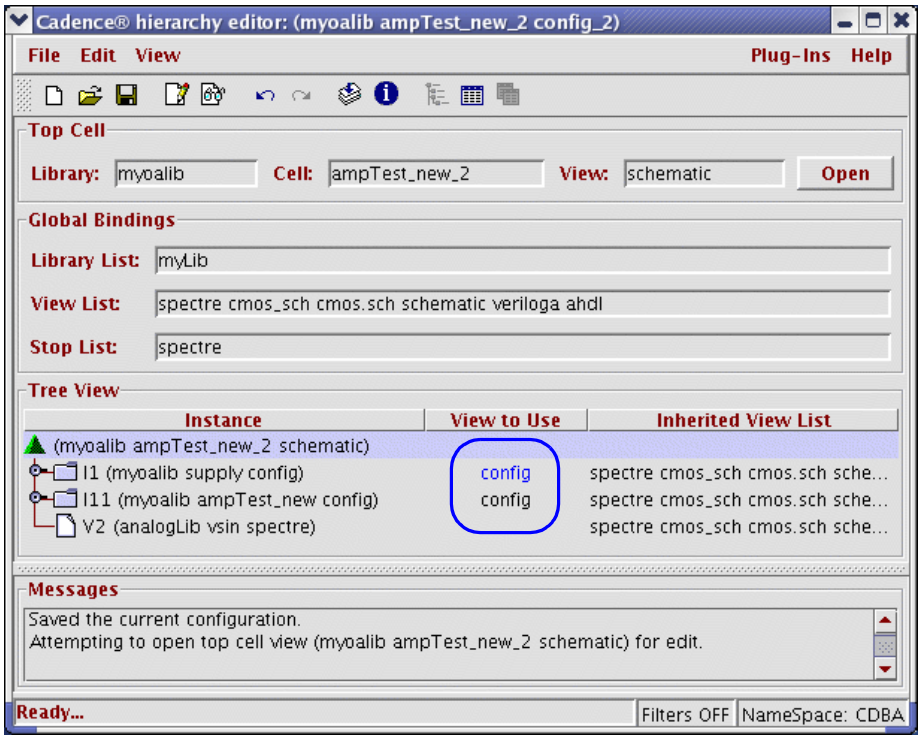
Design Framework II now offers complete support for a hierarchical configuration.

- A hierarchical configuration is a configuration that refers to another configuration.
- All netlisters can netlist hierarchical configuration.
- Hierarchical configuration is crucial for multichip module (MTS) design.
  - Can bring multiple configurations together for simulation.
  - Allows different technologies on different blocks through use of scoped modules.

Complete hierarchical configuration support is now part of the Design Framework II toolset. This support means that all netlisters, including those used in ADE are capable of netlisting hierarchical configurations (configurations that refer to other configurations). Such a capability is crucial to multichip module (MTS) design and other situations where you want to bring multiple configurations together for simulation.

Hierarchical Configuration Support (continued)

Here is an example of embedding a configuration within a configuration.



In this example, the *config_2* view is using a *config* view for the *supply* and *ampTest_new* cell. This was not possible in before the IC 5.2.51 release.

Java and Qt HED

- There are two flavors of HED in IC 6.1.0
  - Qt-based HED
  - Java-based HED
- The AMS plug-in is not available in Qt-based HED.
  - You need to open the Java-based HED.
- To open the Java-based HED
  - From the command line, enter *amsHED*
  - From Design Framework II, enter: *setenv AMS_HED*
- Should resolve to only Qt based HED in IC 6.1.1

In IC 6.1.0, we have two HED: one from Java and another one from Qt.

If you try to install AMS-PLUGIN in QT-based HED, the following error message is observed in the HED message window:

```
ERROR (HED): The AMS Plug-in is not available for the QT based
Hierarchy Editor
in this release. The AMS Plug-in is available for AMS Designer
users by running the legacy Java based Hierarchy Editor using the
amsHED command. To invoke amsHED from DFII you must set the
"AMS_HED" variable in your environment before running DFII.
```

With IC 6.1.0, you can open Java HED using one of the following methods.

1. From the command line, enter:

   ```
   amsHED
   ```

2. In Design Framework II, you set the *AMS_HED* environment variable:

   ```
   setenv AMS_HED
   ```

Also, *amsHED -lib solutions -cell ampTest -view config_AMS -plugin ams* will open JAVA HED with AMS plug-in installed.

License Feature # Change for ADE and ADE L

- ADE (# 34510) is now ADE L (# 95200).
- Obviously, you cannot run ADE L in IC 61 with # 34510.
- You cannot run ADE in IC5141 with IC 61 (# 95200 or # 34510) license until IC5141USR4.

Product Mapping between IC 5.1.41 and IC 6.1.0

IC 5141		IC 61	
Product	**Feature #**	**Product**	**Feature #**
Composer	34500	VSE L	95100
ADE	34510	ADE L	95200
EDFM + ADE	34120 + 34510	ADE XL	95210
EDFM	34120	ADE XL	95210
VSDE	35100	ADE XL	95210
NeoCircuit	3805	ADE GXL	95220
NeoCircuit + DFM	3806	ADE GXL	95220

No Changes in Feature

- The listed product/feature numbers are unchanged in IC 6.1.0.

BU Family	Feature #	Product description
Composer	21060	Virtuoso Schematic VHDL Interface
Composer	21400	Virtuoso Schematic Editor Verilog Interface
Composer	276	Virtuoso Schematic Editor HSPICE Interface
Composer	570	Virtuoso Schematic Composer to design compiler Integration
Analog IC	32100	Virtuoso Analog Oasis Run-Time Option
Analog IC	32101	Cadence OASIS for RFDE
Analog IC	32760	Virtuoso Analog HSPICE Interface Option

- You need an IC 61 license for each of these products.

The products listed above do not have any feature number changes in IC 6.1.0, but you still need IC 6.1.0 licenses to use these features.

Quiz

1. How can you start “awd” in IC 6.1.0?
2. How can you parameterize a model file with a section?
3. What calculator function you use to measure frequency jitter?
4. What calculator function you use to measure “effective number of bits” or “signal to noise ratio”?
5. How can you start a Java-based HED in IC 6.1.0?
6. What is the license feature number for ADE L in IC 6.1.0?
7. What is the license feature number for HSPICE interface from ADE L in IC 6.1.0?

Labs

Lab 7-1 Using ADE L

Lab 7-2 ADE L with ampTest

Multiple Tests in ADE XL

Module 8

December 14, 2006

Module Objectives

In this module, you will

- Learn the flow for defining and executing multiple tests
- Work with the new usage model and new menus
- Learn the importance of the right mouse button in 6.1.0
- See the integration of previously separate functions
- Use the new assistants to manage simulations

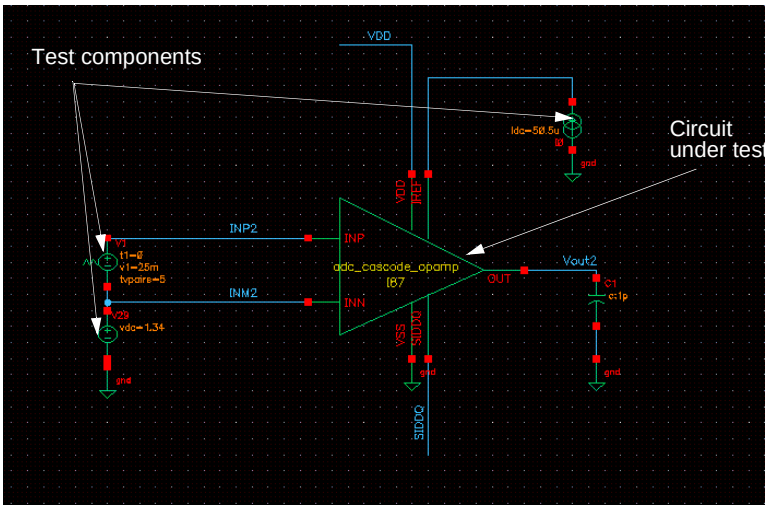
Terms and Definitions

Test	A schematic created for simulation of a particular circuit.
Analysis	The type of simulation to be carried out on a “Test.” There are a number of types of analysis, such as *dc*, *transient*, *ac*.
Assistant	A menu panel giving you information about the design or allowing certain operations to be carried out easily, such as navigation within the design hierarchy or setting up tests and analyses.
CIW	Command Interpreter Window

Defining a Test

A test combines the following elements:

- A framework of stimuli connected to circuit under test
- Functional investigation of a circuit in dc, time, or frequency domains
- A simulator that allows corner sweeps and statistical analysis



Multiple Tests in ADE XL 8-5

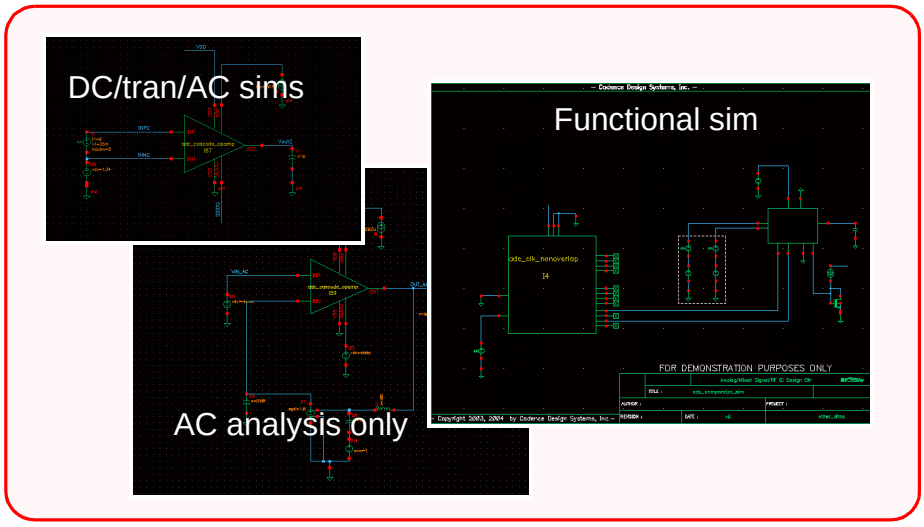
A test is a framework of simulation stimuli that let a circuit design function as required, hence allowing its operation to be studied and verified.

Most designers will be familiar with simulation of single designs probably in the time or frequency domain around corners and with statistics.

IC 6.1.0 Provides Multiple Test Setups

An ADE XL session provides:

- Multiple designs in same simulation environment
- Tests enabled and disabled with one click
- Multiple analyses for each circuit if required



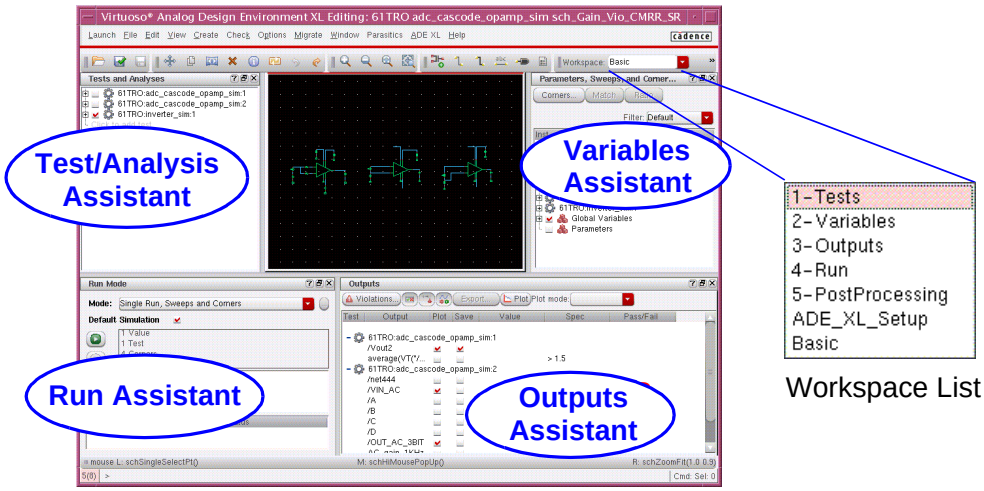
Multiple Tests in ADE XL 8-7

The IC 6.1.0 release provides much greater flexibility to evaluate not just one circuit, but many, at the same time and in the same environment.

Multiple designs can be loaded and multiple analyses can be performed against them. These analyses can easily be enabled and disabled and the overall simulation environment changed to provide different conditions.

ADE XL Environment

The default presentation includes the assistants shown here.



- Start from schematic choosing **Launch – ADE XL** or **Launch – ADE GXL**.
- Multiple tests and analyses are defined on one screen.
- It is easy to switch tests on and off (check boxes).
- Results and simulation progress are all displayed on the same screen.

Multiple Tests in ADE XL 8-9

The new ADE XL interface facilitates the testing of multiple designs from the same environment and permits analysis of all results together. You can start the software either from a schematic and then choose **Launch – ADE XL** or by opening an *adexl* view.

Whereas previous versions of the Virtuoso software enabled just one design to be analyzed at a time, ADE XL lets you specify simulations of multiple designs from the same session. There are check boxes provided so you can easily switch on and off particular analyses or entire tests. If corner simulation is being employed, then all the simulations will run under the same corners.

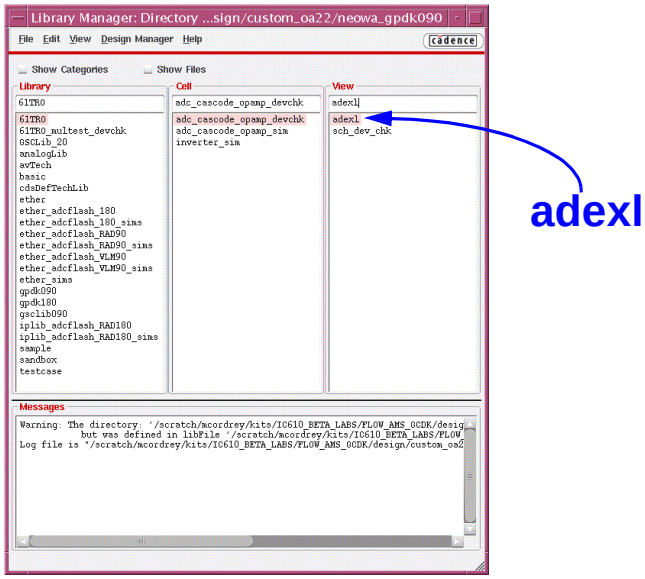
The flow steps through each phase in turn: Tests and Analyses, Variable, Outputs, Run Mode, Post-Processing.

One way to navigate through the simulation environment is to use the workspace list in the top right corner. You work down through the tabs as the tests are set up, culminating in running the simulation and reviewing the results.

Important

The tests that are defined in the ADE XL sessions are not linked to the schematic that is displayed. It is possible to have several simulations without viewing the schematics at all. The environment is completely flexible.

A New View: *adexl*



- The *adexl* view stores all the information on the tests/analyses in the session, including simulation data (*psf* directory).
- This view does not store any *artist* state information of each test.

In ADE XL, a new view has been created that contains all the information relating to the ADE XL session. This is the *adexl* view. When you initially launch ADE XL from a schematic an *adexl* view is created and attached to the schematic.

Now, the clever part is that, as the ADE XL session evolves and more tests are added, the *adexl* view can contain references to other circuits that are not even in the same library as the original one. A “mature” *adexl* view can have several tests defined and these can be all over the design space.

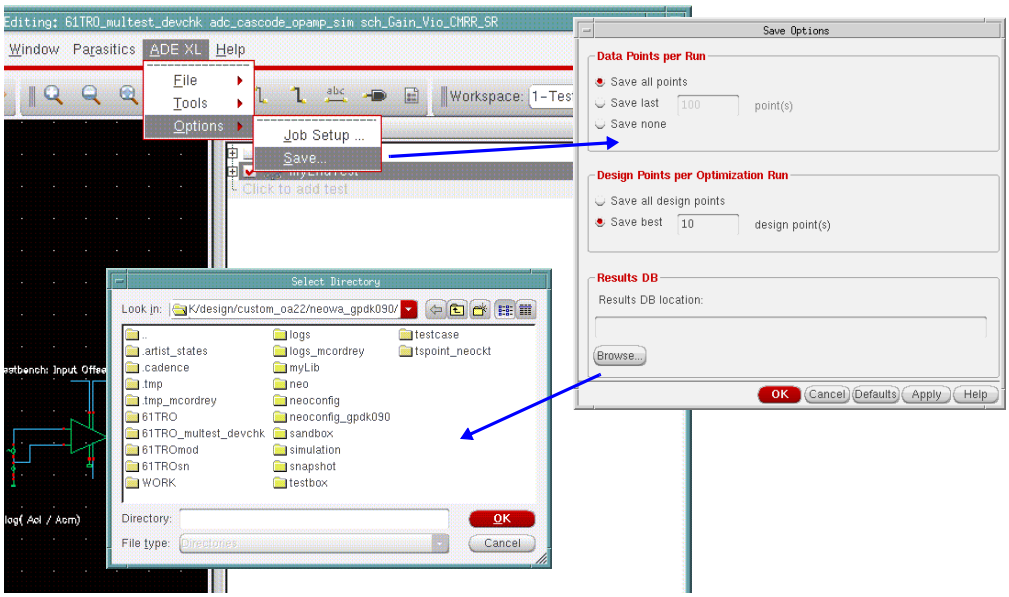
Important

*The **adexl** view DOES NOT CONTAIN any state information about each simulation. **This is still stored as state files associated with each test**.*

When you quit the session, you will be asked if you want to save the changes to the ADE XL setup. This means the tests and analyses defined and the positions of assistants.

Compared to prior versions of ADE, each test corresponds to what was a single ADE session in 5.1.41. Thus, if you have many tests defined, then this is equivalent to many simultaneous ADE sessions all functioning in the same simulation environment. They can even use different simulators and different sets of models. All the same flexibility is present as if they were multiple ADE sessions.

ADE XL Simulation Database

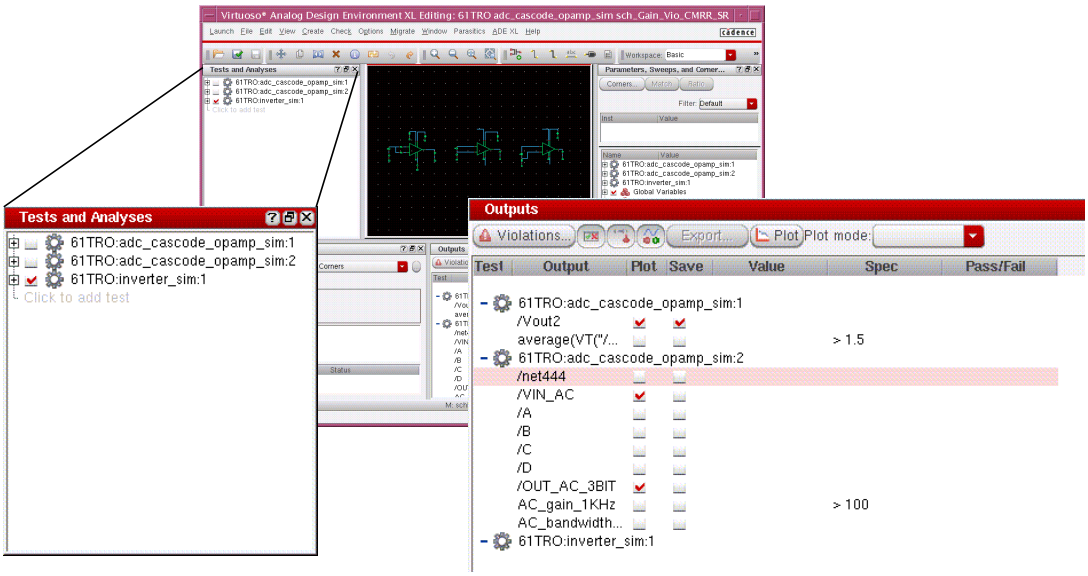


- Location for simulation results can be changed in options.

One important ADE XL option is where the simulation results are stored. This is set from the ADE XL – Options – Save form. This choice brings up the Save Options form and then the location for the results directory can be specified, either directly or by browsing the directory structure.

This information is stored in the *adexl* view.

ADE XL Assistants



- Assistants provide information and feedback on all parts of simulation flow.
- They are detachable. Drag them with the mouse.
- You can also use workspace list.

Multiple Tests in ADE XL 8-15

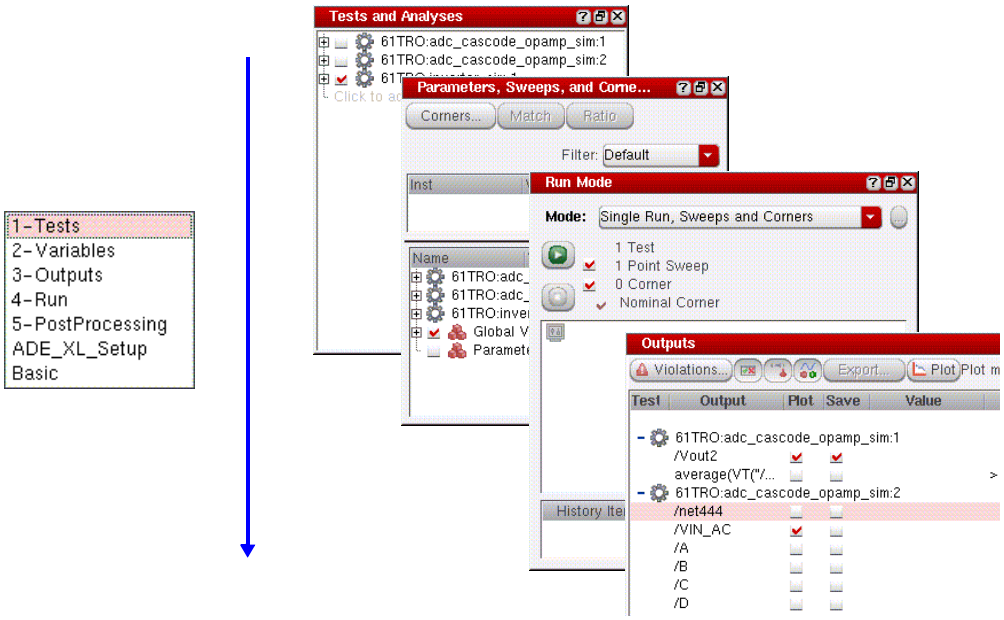
Special menu panels or *Assistants* are used throughout the 6.1.0 release and the same is true for ADE XL. These assistants provide navigation through designs and provide information on design parameters, results, and a host of other features.

The default setup for ADE XL shows a number of the assistants all active in the same session window. If you find this too unwieldy, you can detach the assistants. Click the top bar of the assistant, in the same way you do in a window, and drag the assistant onto the desktop. The assistants can be resized just like any other window.

To reattach the assistant just move it back onto the Virtuoso session. Getting the desired position may take a bit of practice. One way to return to the default is to navigate away and back using the workspace list.

A Word About Flow

The easiest way to use assistants is to set them up one-by-one.



Multiple Tests in ADE XL 8-17

It is good to give some pointers here as to a possible flow to use. ADE XL can let you see all the information about your simulations on the same screen at the same time. This might be a little too much and it might be preferable to follow a Flow down the workspace list as the simulations are defined.

Start with the Test and Analysis assistant to define the tests and analyses. Next move on to the Parameters and Variables section and put those in. After that, do the Outputs, then Run the simulations, and Post-Process the results.

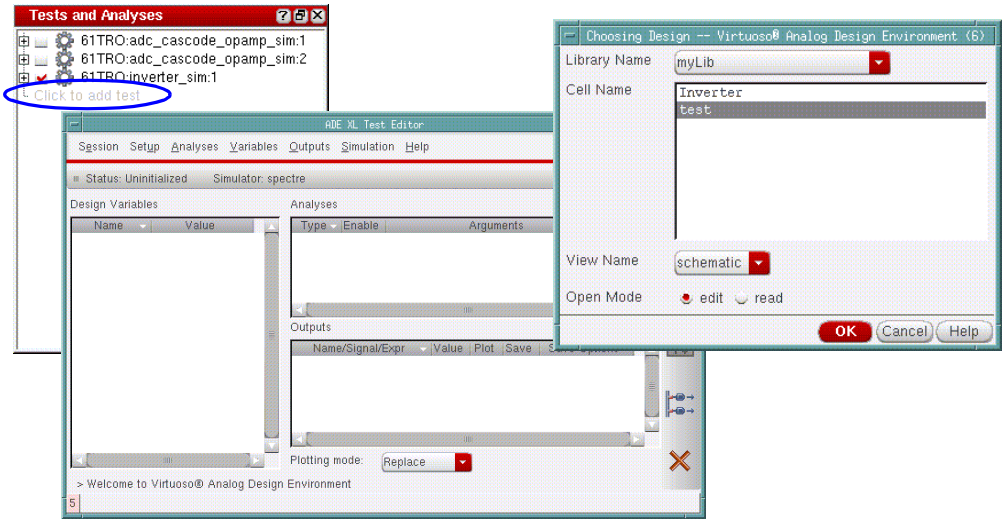
To modify the runs, choose the assistant that you need, then jump back down to the Run Mode assistant to relaunch things.

You will develop your own preferences, but this technique is a good way to start until you are comfortable with all the functionality that is now at you finger tips.

You can define your own set of default assistants in a workspace.

Defining a Test Using an ADE L Style Form

Tests and Analyses Assistant



- **Click to add test** brings up the ADE L style form for selecting a design. (You can't run a simulation from this ADE L style form.)
- All standard pull-down menus are available by clicking the right mouse button.

Multiple Tests in ADE XL 8-19

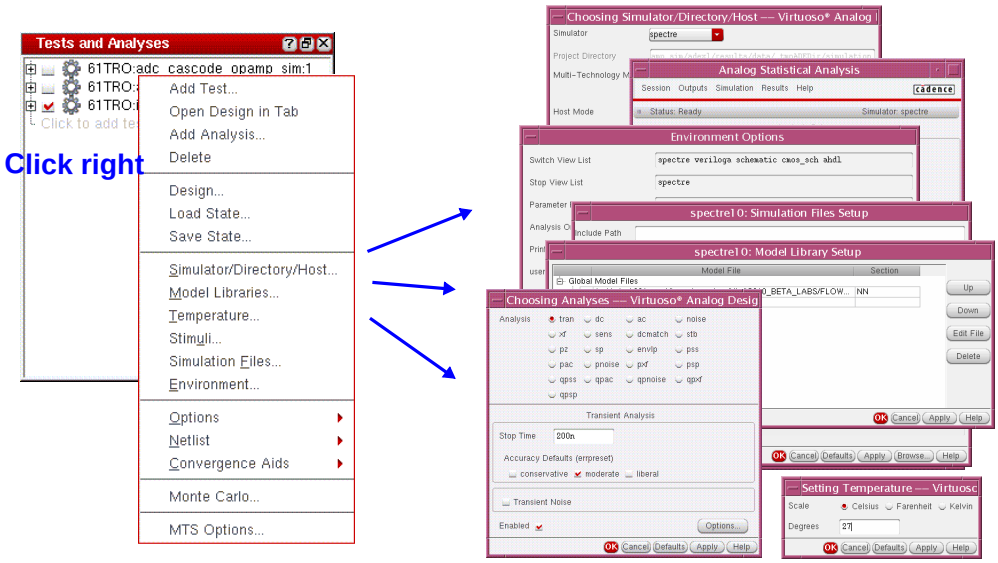
Test definition is done from the Tests and Analyses assistant.

You can define analyses by either left-clicking on **Click to add test** or right-clicking over a test or analysis. The ADE XL Test Editor form appears with all the pull-down menus that are used as part of ADE. Variables, analyses, and outputs can be defined from this form if you want to work in the 5.1.41 way. You are also prompted as to which design to read in.

The usage models of these menus are all fairly similar to before, but with the new Cadence® look and feel. A full explanation of each of these entries is outside the scope of this module.

Creating Analyses from a Menu

Tests and Analyses Assistant



- Click **Add Analysis** to get started.
- **Right**-click to bring up familiar ADE L forms.

As well as using the ADE L look and feel forms, ADE XL allows analysis definition from a right-click and a menu. Here you define each analysis along with the model libraries, outputs, and any other simulation files that are needed.

Important

The right mouse button is a big part of the ADE XL interface.

If in doubt..... right-click.

Multiple Tests on Different Circuits

Tests and Analyses Assistant

- You have independent designs under one environment.
- Just position the mouse pointer over test for details.
- Open schematics in separate tabs using the right mouse button menu.

Multiple Tests in ADE XL 8-23

This example shows two tests that are completely independent, but you can be run them in the same session. Holding the mouse pointer over a test name will display the details of that test, including the design name, simulator, and which state file is being used.

The designs can be opened from the **Open Design in Tab** option on the right mouse button menu.

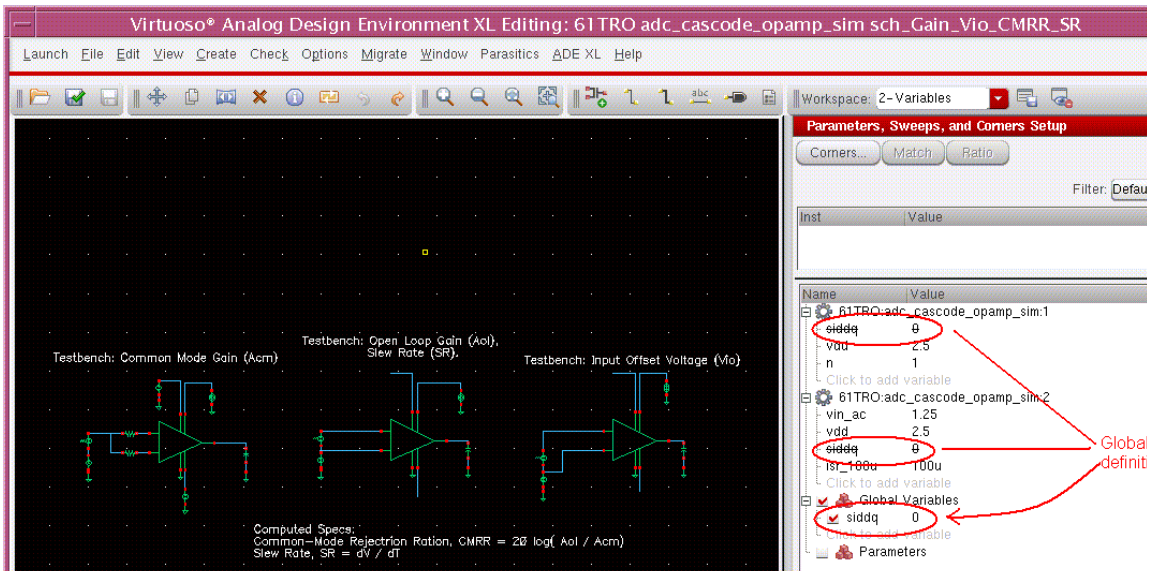
You can change the name of a test directly in the Tests and Analyses form.

Important

Remember that these designs can be completely different. ADE XL lets you simulate them in the same environment. The schematics for the tests can be opened in separate tabs from the right mouse button menu.

Global and Local Variables in Multiple Tests

Variables Assistant



- Design variables associated with tests are automatically defined globally.
- Sweep simulations require global variables to be used.
- Strike-throughs indicate global variable will be used instead of local.

Multiple Tests in ADE XL 8-25

Each test can have its own variables associated and there is also the facility to use global variables across all tests. This allows the same environment to be defined once and reused across tests.

The variables are specified from the menu brought up by right-clicking on the design name. You will see all the standard ADE options; **Add variable, Edit variable, Copy from Cellview, Import Sweep and Toggle View**.

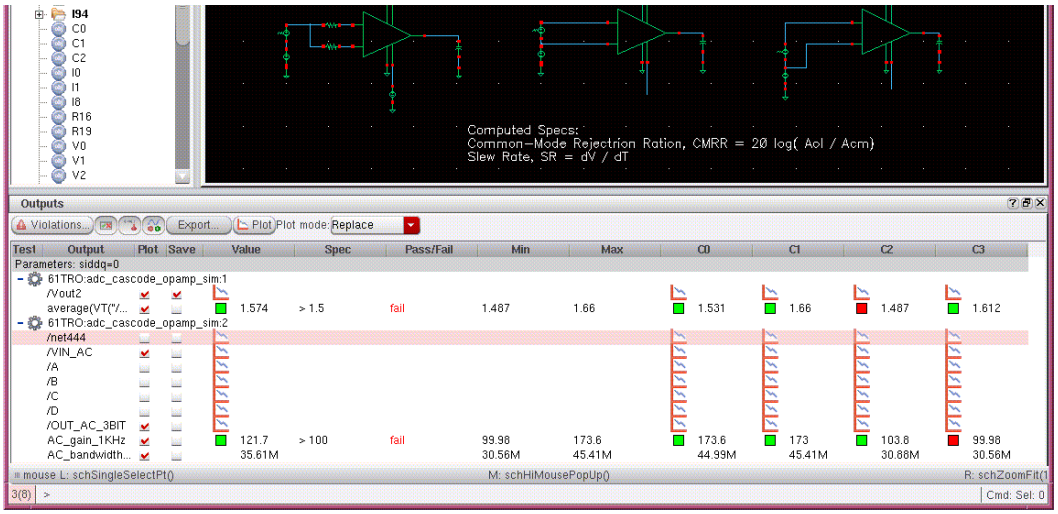
When clicking right on an existing local variable there is also the option **Make variable global**, whereby it becomes visible across all the tests. The **Global Variables** entry shows which global variables are present and each analysis will have the same variable name but struck through to indicate that it is global.

When further tests are created in the session, they automatically become part of the same environment and hence can use the same variables.

Note: Variables are covered in more detail in a later module.

Output Definitions in ADE XL

Outputs Assistant



- Output statements can be shown for all tests on the same screen.
- They can also include expressions and their required values.
- Device checking is also included here.

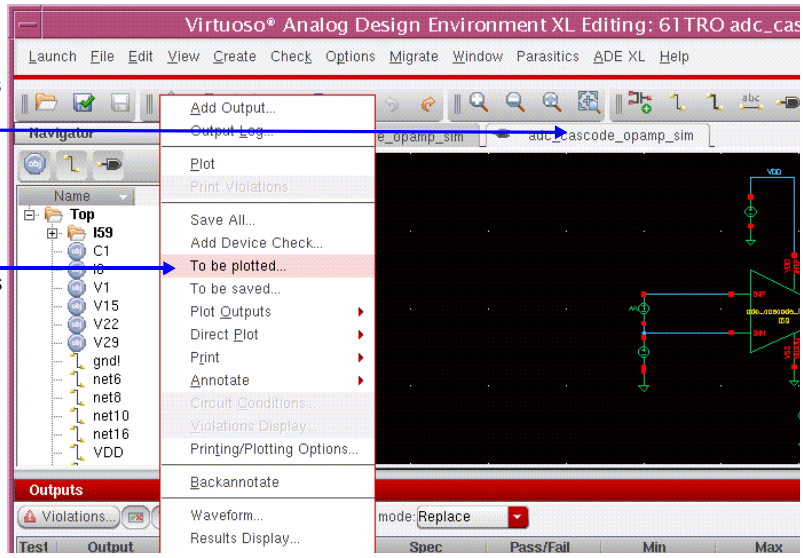
Select the **Outputs** assistant for this phase of the preparation. The list of outputs are shown for each test. These can be opened and closed as part of a “tree” display. The window indicates when waveforms are available to plot or displays the values of output expressions. Specifications for output expressions are also displayed in the Outputs assistant along with device checks. Colored check boxes indicate whether the data passes (green), fails (red), or is within 10% of spec limit (yellow).

Output Selection

Outputs Assistant

Schematic opens in new tab to select nets.

Click **right** to open the outputs menu.

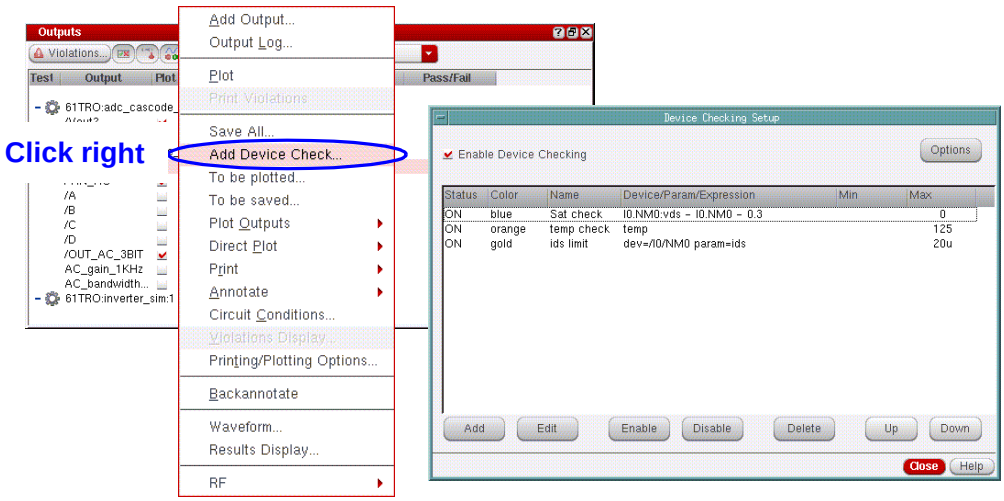


- Right-click **To be plotted** to select outputs.
- Schematic is automatically brought up to allow selection of nets.
- All other output options are found on this menu.

Use the right mouse button to display the output selection menu on the required design. Outputs for the simulation are selected with the **To be plotted** option. If required, the schematic of the design in question is opened in a new tab by ADE XL to allow output selection.

You can also traverse the hierarchy using the normal options under **Edit – Hierarchy** so you can select nets throughout the design.

Device Checking in ADE XL

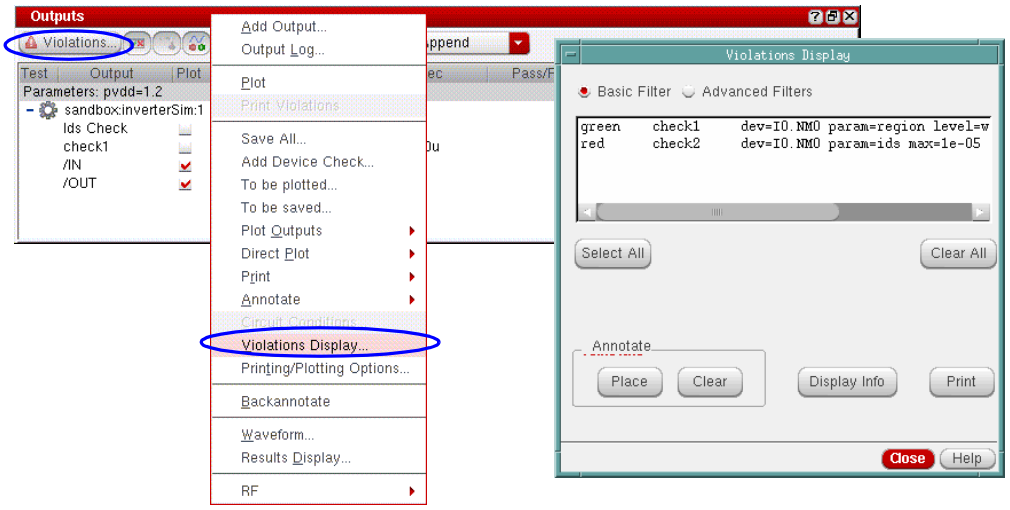


- Right-click in the Outputs assistant.
- The software displays Device Checking Setup form as before.
- The use is the same as in the IC 5.1.41 release.

Device checking is implemented in ADE XL in the same way as it appeared in ADE under the 5.1.41 release. Position the pointer and right-click over the test in question and select Add Device Check from the menu. This action displays the Device Checking Setup form, which is probably familiar to you.

From this point on, the usage model is the same for adding and managing device checks and setting the severity of violations and other options.

Analyzing Check Violations



- Right-click on test and select **Violations Display** or use the **Violations** icon.
- The Violations Display form appears as in 5.1.41.
- From here, the use model is the same as previously.

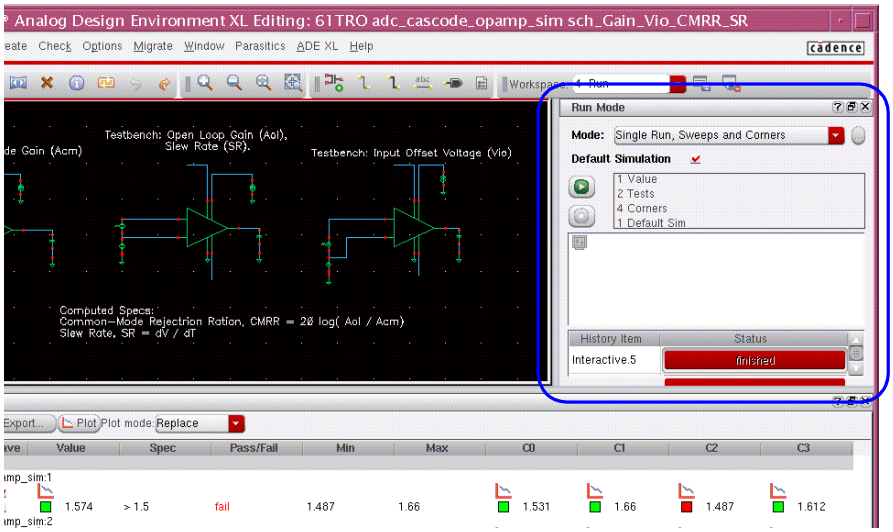
As for 5.1.41, violations are managed from the Violations Display form. This can be found from the right mouse button menu on the test in question in the Outputs assistant. Once again from this point on, the use model is the same is the same as for standard ADE.

Important

Device checks are displayed in a similar way to Specs in the Outputs assistant. However, the actual number shown relates to the number of failures for each check, rather than an actual value of a violation. For example, if a check covers all the pmos_lvt in a circuit and 3 of them fail, then "3" will be displayed against the check, and a Fail entry will be displayed.

Running Multiple Tests

Run Mode Assistant



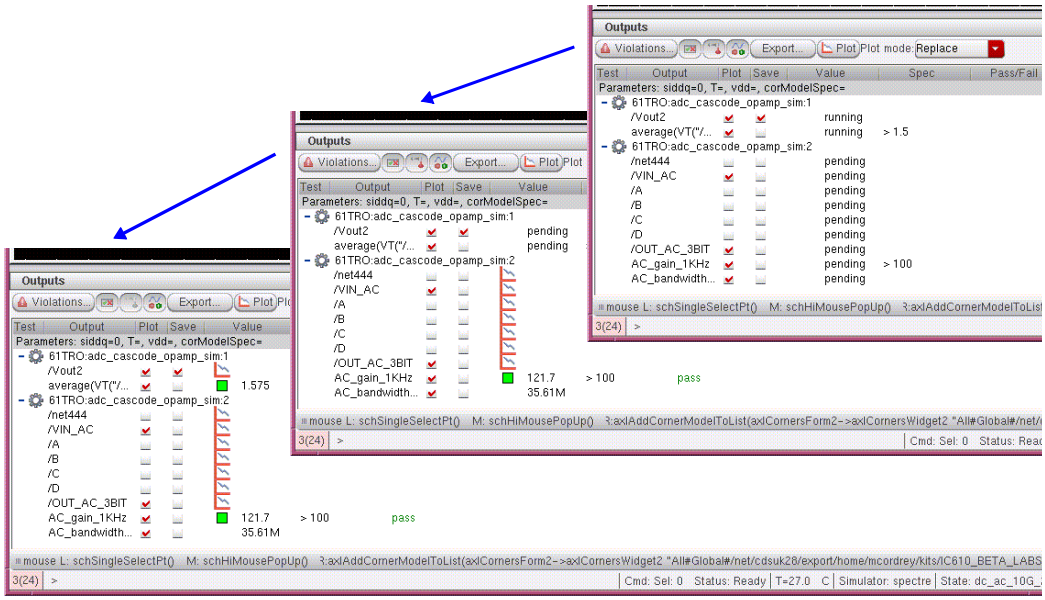
- The number of tests/corners to be run are listed.
- Status of each output is updated in real-time during simulation.

The Run Mode assistant starts a simulation. Depending on how you have specified the session, the simulation run can involve a number of different tests with several corners and sweeps. The illustration here shows two tests, both using four corners of temperature and supply voltage.

After the run is started, click the **Play** button. The Outputs panel is updated in real-time as each test is completed. The Run Mode assistant describes how many simulations have been completed of the total number.

Evolution of Run

Outputs Assistant



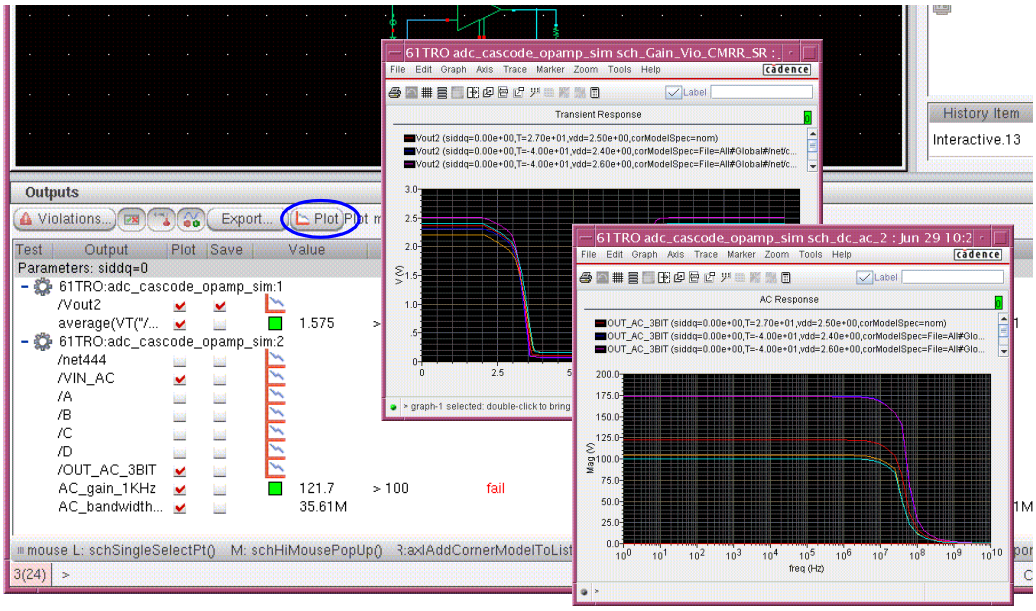
The details of the run results are updated as the run proceeds:

pending ... running ...  plot ready

After a set of simulations are launched, the output window is updated in to show how each analysis is progressing. Status progresses through *pending* to *running* and then a plot icon is displayed or a value shown when the analysis has finished.

Output Plots

For each test, a separate window will display the results.



- The results for all runs are displayed and waveforms are plotted.
- Use the **Plot** icon to plot individual outputs.

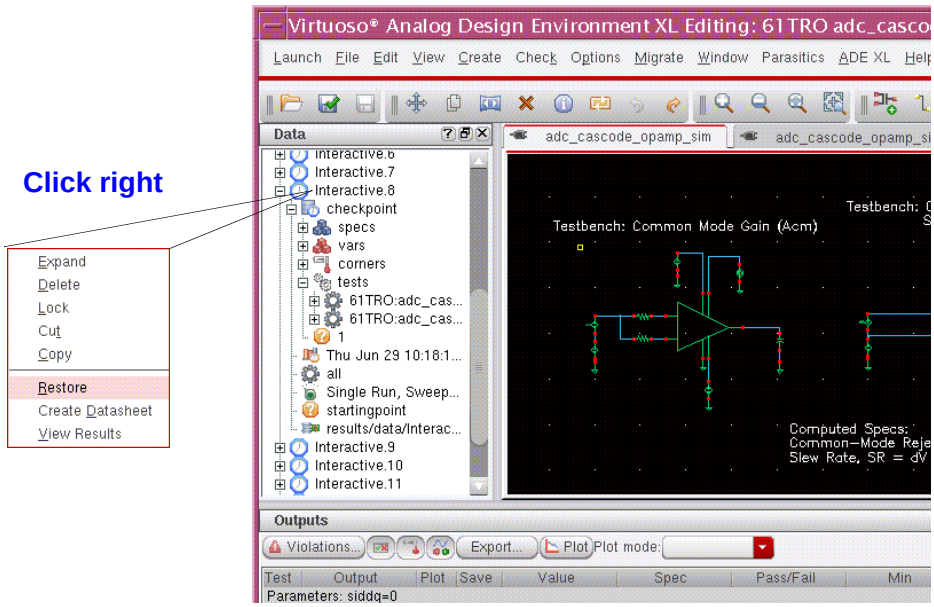
After the set of simulations have finished, the Outputs assistant is updated with any results, and the outputs selected will be displayed in the Waveform Viewer.

The Waveform viewer displays results.

One waveform window per test is opened.

Postprocessing Results

- Select the PostProcessing workspace or assistant.



- ADE XL keeps track of all runs, current and previous.
- You can select previously executed results using the right mouse button.

Multiple Tests in ADE XL 8-41

The PostProcessing workspace in ADE XL lets you work with all previous simulation runs from a given session as well as those previously run in this simulation directory.

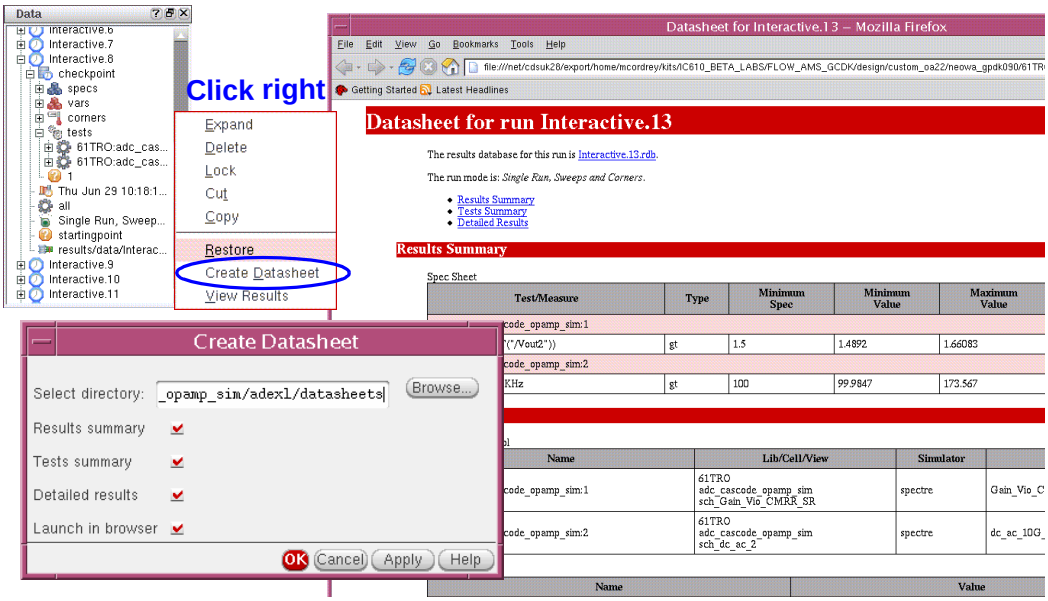
The workspace provides a directory tree in the same manner as the Results browser from IC 5.1.41. Right-click on the run in question and select from the menu to either **Restore** the results to the session or simply **View results**.

Important

*Selecting **Restore** from the menu will bring back the previous simulation settings and make these current. Thus a subsequent simulation run will use these settings. This should not be confused with **View Results** which will just bring back the results for a particular run and insert them in the **Outputs Assistant** in a separate tab. In this way, the use can switch easily between different sets of results to compare outputs.*

Creating a Datasheet from the Analyses

PostProcessing Assistant



- ADE XL can create an HTML datasheet for the results of an analysis.
- The datasheet will contain a summary of the results of tests and corners.

Multiple Tests in ADE XL 8-43

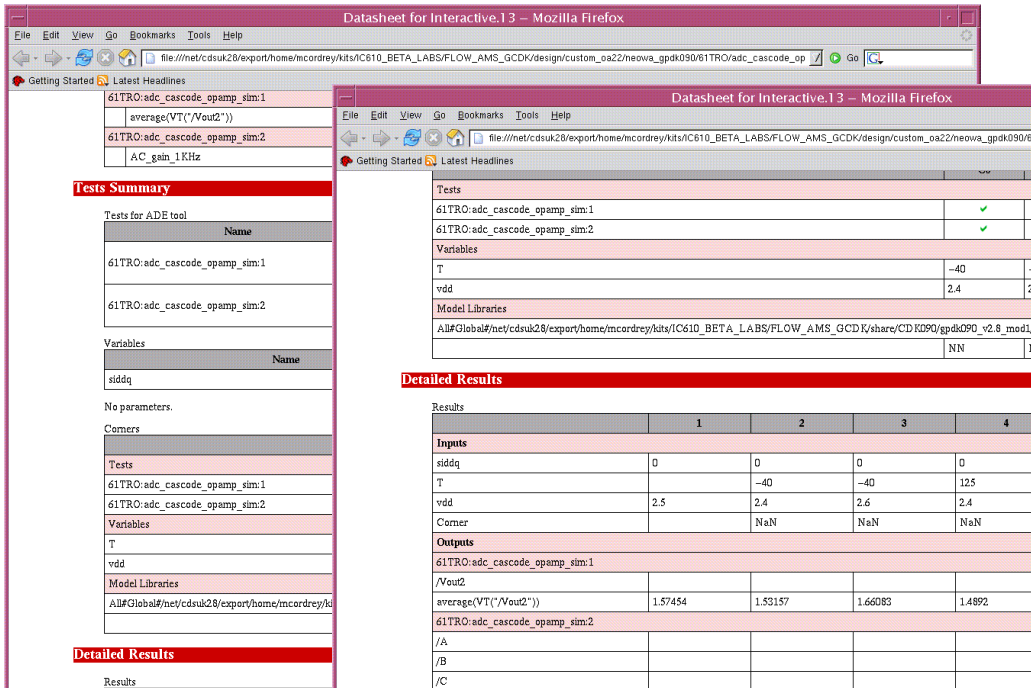
In ADE XL, you can create an HTML datasheet-style summary of an analysis. The results are tabulated and each test and corner is detailed. This feature lends itself ideally to being included in a final document or even being linked to, ensuring up-to-date results without having to update the results document.

Use the right mouse button on the required set of results. Then select **Create datasheet**. In the ensuing form, select **Results summary, Tests summary, Detailed results** and whether to **Launch in browser**.

The results summary shows the values from the simulation for each expression defined. Maximum and minimum values are shown together with any specification.

Data Sheet: Other Sections

Tests Summary and Detailed Summary Sections



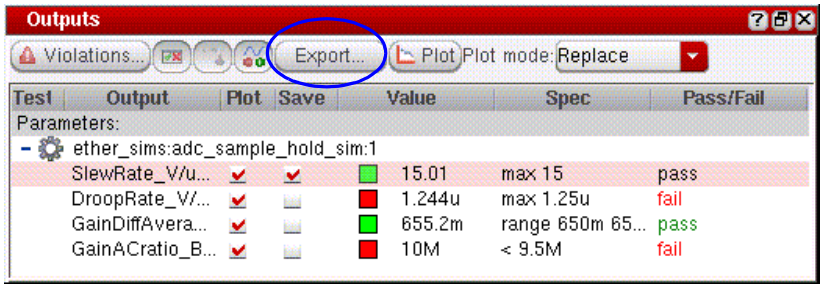
Multiple Tests in ADE XL 8-45

The **Tests Summary** lists the details of all the tests specified in terms of cellview name. Variables and any design parameters are also given with results from Corners and Monte-Carlo runs if present.

The **detailed results** section lists all the inputs and outputs with the values from each of the runs completed.

Exporting Results to a File

- You can save the results from a simulation run to a file in **.csv** or **.html** format.
  - The **csv** file can be opened later in any text editor.
  - The HTML file can be viewed in any browser window.
- All the measurements specified in the Outputs assistant are saved to the file.
- In the Outputs assistant, click the **Export** icon.
- The results file can be kept as a project document for future reference.



Results can be saved in a **csv** (comma separated value) file. This file can be opened in any text editor on UNIX or in Microsoft Excel on WINDOWS.

Results can also be saved in the HTML format. This file can be opened in any browser on UNIX or a WINDOWS platform.

To export the simulation results to the file, click the **Export** icon in the **Outputs** assistant.

Only the measurements specified in the outputs assistant are exported to the file.

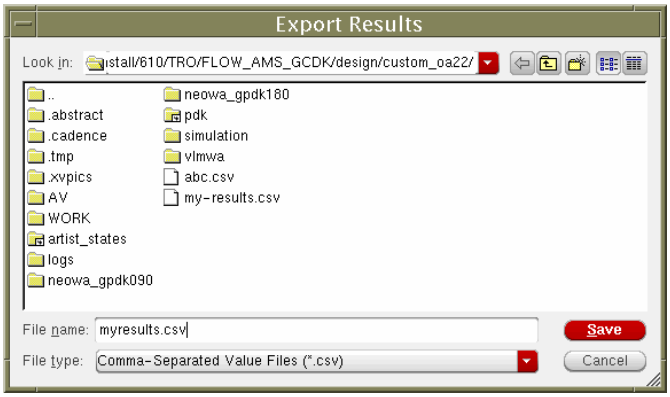
This results file with some key measurements can be kept with other project documents for future reference.

Important

This feature saves only the measurements specified in the outputs assistant. It doesn't save all the simulation results to the file.

How to Export the Results

- Export Results form pops up when you click on the **Export** icon in the Outputs assistant pane.
  - Drop down selection to choose file type with option to save the results in *.csv* or *.html* file.
  - Default file format is *csv*.
  - Make sure to specify the file name with the complete extension, such as *myresults.csv*, when saving a file in the *csv* format or *myfile.html* when saving in the HTML format.
  - The file is stored in the project directory by default.



Default format of the saved file is csv. You can also save it in the HTML format.

Files are by default stored in the project directory. (The current working directory where you started the Virtuoso software.)

Important

Make sure that you specify the file name with the correct extension when saving results.

Summary

- Defining multiple tests
- What is in the adexl view?
- Using assistants
- Setting outputs to be save and/or plotted
- Launching simulations and track progress
- Manipulating results
- Creating a datasheet
- Exporting results to a file

Important

Remember that tests can be for different circuits and you need to be aware that the schematic displayed might not relate to the test you are working on.

Labs

Labs come after the next lecture.

Using Design Variables

Module 9

December 14, 2006

Module Objectives

In this module, you will

- Use design variables in ADE XL and GXL
- Differentiate between local, global, and instance parameter variables
- Use the calculated value of a variable from one test in another test

Working with Variables

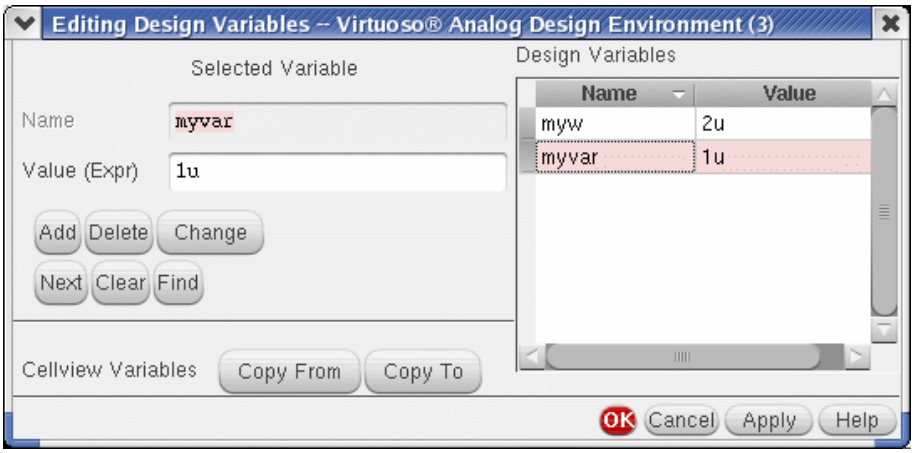
- Set all variables in Parameters, Sweeps, and Corners Setup assistant.
- Design variables are the same as design variables in ADE L
  They appear under test name.
- Global variables override design variables.
  They can change design variables of the same name in multiple tests.
- Parameters are instance variables.
  They can change a design variable for a specific instance only.
- As always, when in doubt, use the right mouse button.

Using Design Variables 9-5

You can use design variables and Component Description Format (CDF) parameters to set component values. You can add, change, and delete design variables on the Parameters, Sweeps, and Corners Setup pane in your ADE XL environment. Design variable values are always global to the design. The scope of a CDF parameter value depends on which Analog Expression Language (AEL) functions you use to refer to the parameter.

Design Variables

- The design variables are the same as ADE L design variables.
- They appear under test name in Parameters, Sweeps, and Corners Setup assistant.
- Right-click the variable name to modify it. This action displays the standard ADE L form.



Using Design Variables 9-7

To change a design variable, follow these steps.

1. In the parameters table on the Parameters, Sweeps, and Corners Setup pane, click in the Value cell of the design variable that you want to change.
2. Type a new value or expression for the design variable.
3. Press Return or click anywhere outside that table cell.
   - The new value appears in the Value cell for that design variable.

Disabling a Design Variable

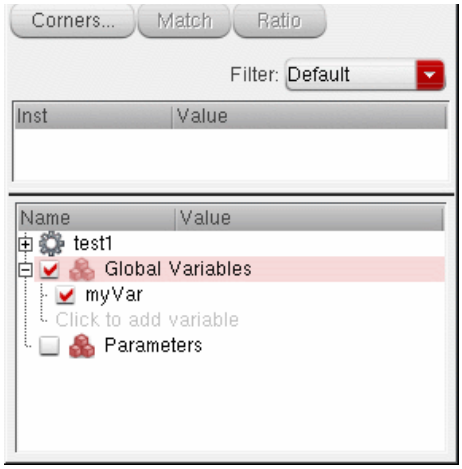
- You can disable a design variable for a particular simulation while retaining its definition on the Parameters, Sweeps, and Corners Setup pane.
- In the Variable column, remove the mark from the check box to the left of its name or identifier prior to starting a simulation.

Saving a Design Variable

- On the environment window menu banner, choose **ADE XL – Save Setup**.
- Your setup information, including design variables, is saved. These values are reloaded the next time you open this design cellview.

Creating a Global Variable

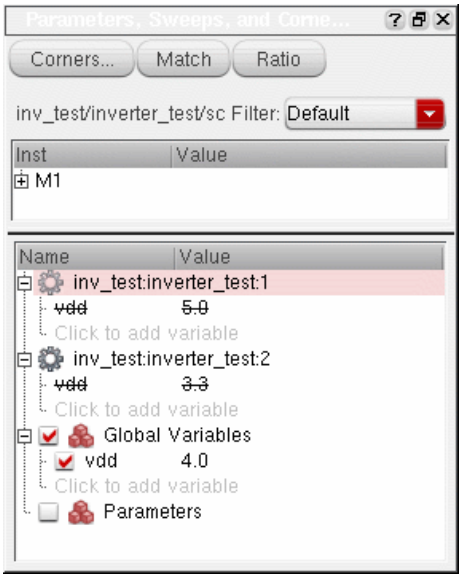
- A global variable overrides a design variable.
  - All design variables become global variables by default.
  - Global variables override the same variable in multiple tests.
  - Local variables for each test will show strikethroughs to indicate that a global variable exists and has priority.
- You can create new variables or use existing design variables by using the right mouse button or clicking to add a new variable.



Global Variables

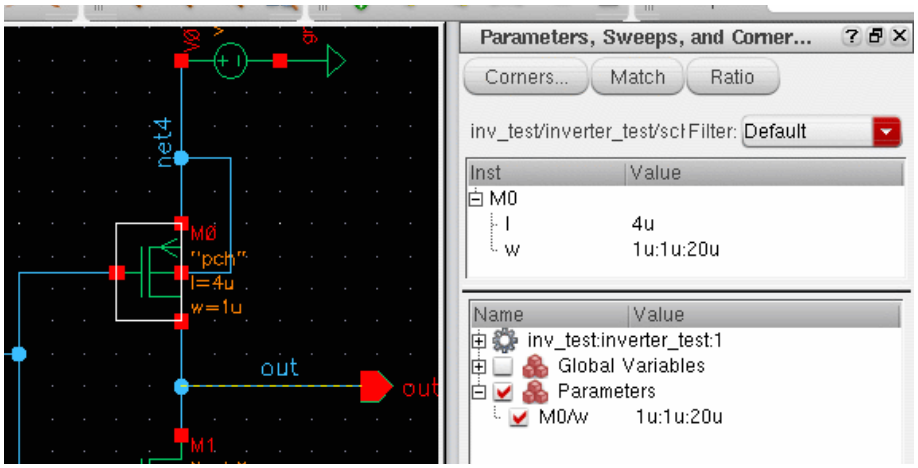
Values of global variables will be used in all tests.

- They can override the same variable in multiple tests.
- Strikethrough will be seen on variable if overridden.

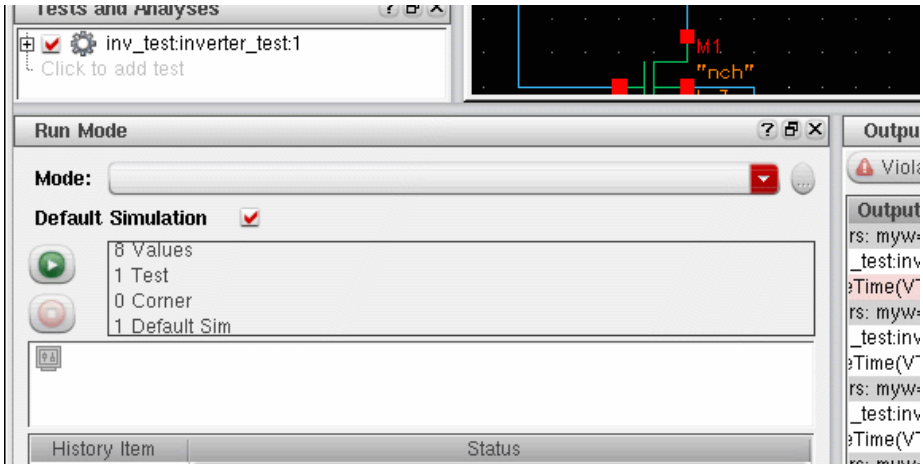


Instance Parameters

- These parameters can modify a variable or create a new variable on a specific instance.
- They can specify an instance parameter as a sweep parameter.
- Select Instance in the schematic and modify parameters in the assistant.
- Instance parameters show up in the input.scs file as DPAR_1, etc.
- Instance parameters can have static or sweep values.



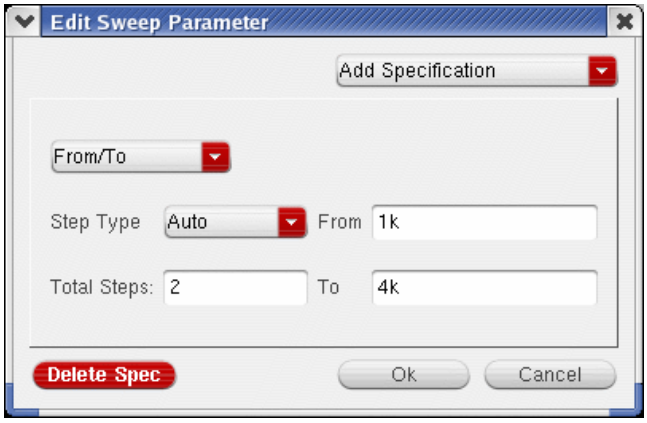
The number of tests shows how many simulations will run:



If you are viewing a tree of parameters, the view changes to the parameter categories view.

Specifying a Design Variable as a Sweep Parameter

- You can set a global variable or parameter as a sweep variable using the built-in parametric analysis.
- To specify a design variable as a sweep parameter, use syntax like: 1u:1u:20u. The default sweep is From/To sweep.
- All your favorite sweeps are here: From/To, Center/Span, Center/Span%, Inclusion list, and Exclusion list.

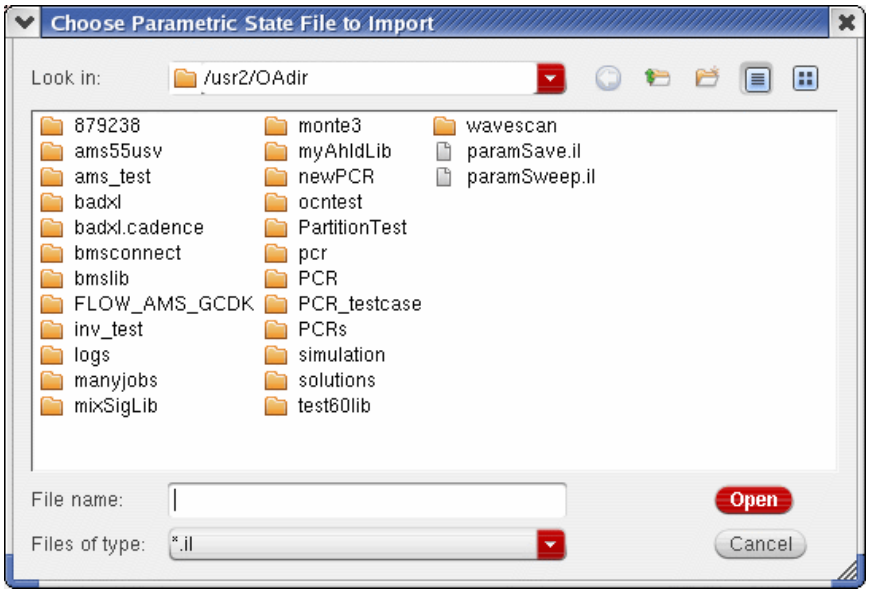


Setting Up the Sweep

- Right-click the design variable that you want to use as a sweep parameter and select **Make Variable Global**. 1u:1u:20u is a sweep from 1u to 20u in 1u steps.
- You can specify that you do not want a design parameter varied for a particular simulation while retaining its parameter definition in the parameters table on the Parameters, Sweeps, and Corners Setup pane.

Import Sweep

- You can import any saved parametric analysis sweep from ADE L.
- Right-click and select **Import Sweep**.



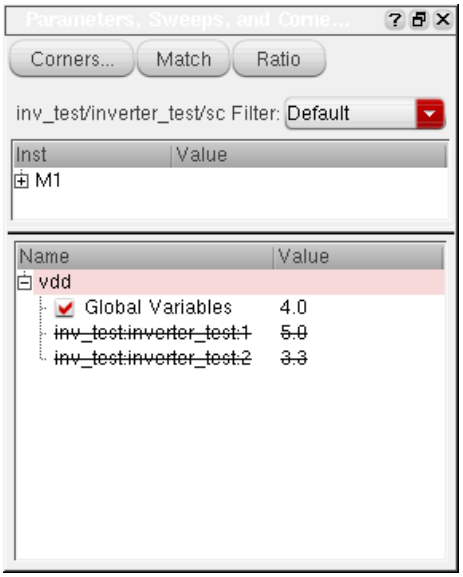
- You can write you own sweep in SKILL as well.

Here is the SKILL® format:

```
variable 0 "myw"
range 0 (?type "From/To" ?from "1u" ?to "6u" ?control "Auto"
?step "3" ?select nil)
```

Toggle View

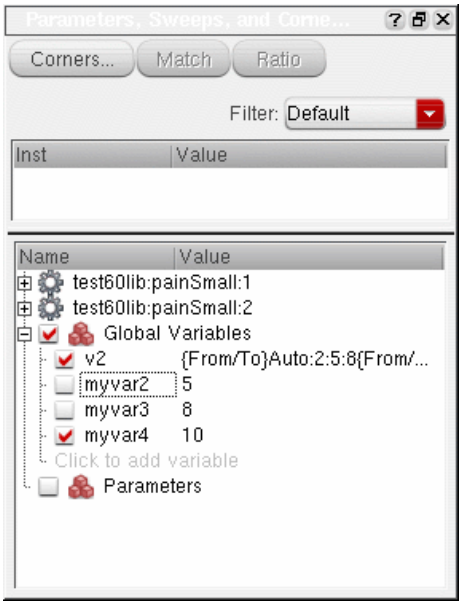
- Toggle View shows all variables and values.
- It also shows variables together, not as a class.



This is an easy way to see all the variables listed singly.

Parameter Specification

- You can enable or disable parameters by selecting the check box.
- Enabled parameters will have a red check mark in the box.



Using Design Variables 9-21

To delete a parameter specification, do the following:

1. On the Edit Sweep Parameter form, click in a field of the row or section for the parameter specification you want to delete.
2. Click **Delete Spec**.

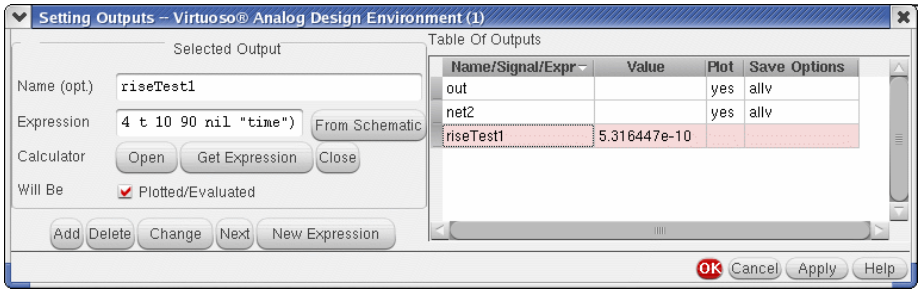
Using Global Variable in Multiple Tests

- You can use a global variable of one test in another test.
- Use calcVal expression:

```
calcVal("riseTime1" , "test")
```

Example Steps

1. Create a calculator expression in Outputs section of test1.

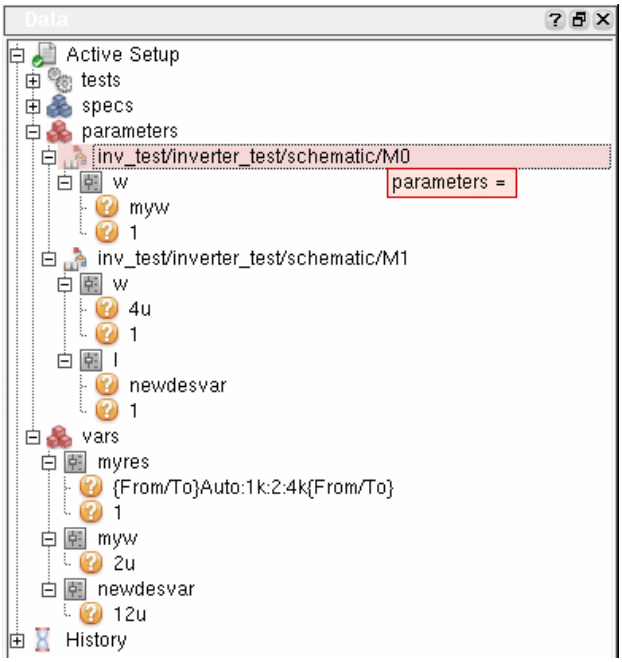


2. In the second test, create an output expression that uses the calculator expression from the first test as in this example:

```
riseTest2 = calcVal("riseTest1", "test1") * 2
```

Data Assistant

- All changes to *adexl* variables can also be viewed from the Data assistant.
- You can edit values here as well.



Quiz

1. How do you know when you are using a Global Variable?
2. What is an easy way to regroup all variables by name rather than by type?

Labs

Lab 9-1 Setting Up Tests in ADE XL

Lab 9-2 Adding a Second Test

Generating Specifications

Module 10

December 14, 2006

Module Objectives

In this module, you will

- Use specifications in the Outputs assistant

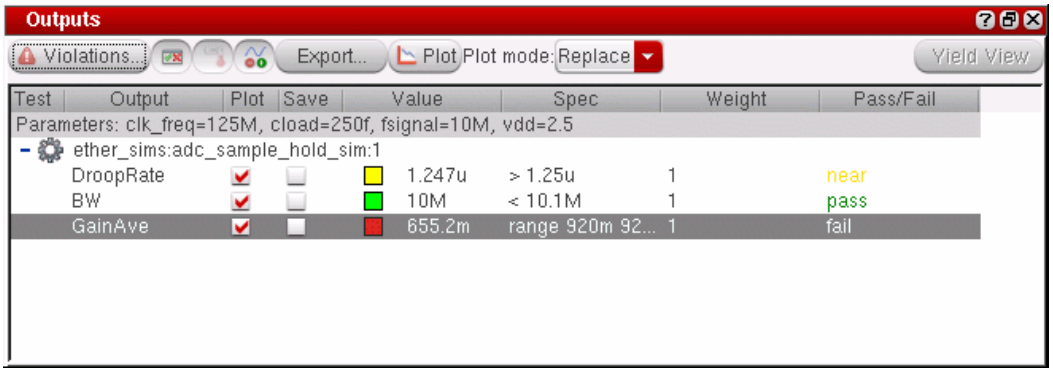
Specifications

- Provide an easy to read visual aid of simulation results versus design specifications.
- You can view and analyze measurement results in pass/fail/near format.
- Specifications can be modified at any time, and the measurements will be compared against the new specifications.
- You can use different specifications with multiple tests or analyses.
- Specifications work with the corners and sweeps.
- Specifications are derived from functional and physical requirements of any analog design. Typical analog design specifications include measurement of key performance parameters. Various types of analyses are run on an analog design to measure these performance parameters.
- In the ADE XL and ADE GXL environments, these performance measurements can be compared with your design specifications. The measurements can be viewed and analyzed in Pass/Fail/Near format. Pass/Fail/Near information is also shown with color coded boxes in front of each measurement value.

- You can change these specifications with no need to run the simulation again. The measurements will be compared with the new specifications as you change them.
- You can set different specifications for different test conditions.
- This feature works with Corners and Sweeps. Measurements are analyzed for every corner/sweep.
- You can use the show/hide toggle button to show or hide the Specs column in the Outputs assistant. Even when the Specs column is hidden, Pass/Fail information is shown.
- Functionality is based on VSDE and works with ADE XL and ADE GXL environments.

Compare the Results with Design Specifications

- Set up a simulation and add measurements for key performance parameters.
- Add these specifications by placing type and value fields in the Spec column of the Outputs assistant.
- Specifications can be added before or after running the simulation.
- Options are: *maximize*, *minimize* <,>, *tolerance*, *range*, and *info*.
- Color coded pass/fail information appears in the Outputs assistant.



- *Near* is within 10% of the value.

The *Spec* column in the Outputs assistant of the ADE XL and GXL provides a mechanism for comparing the measurements with your design specifications.

It is a quick and easy way to verify the measurements against the design specifications.

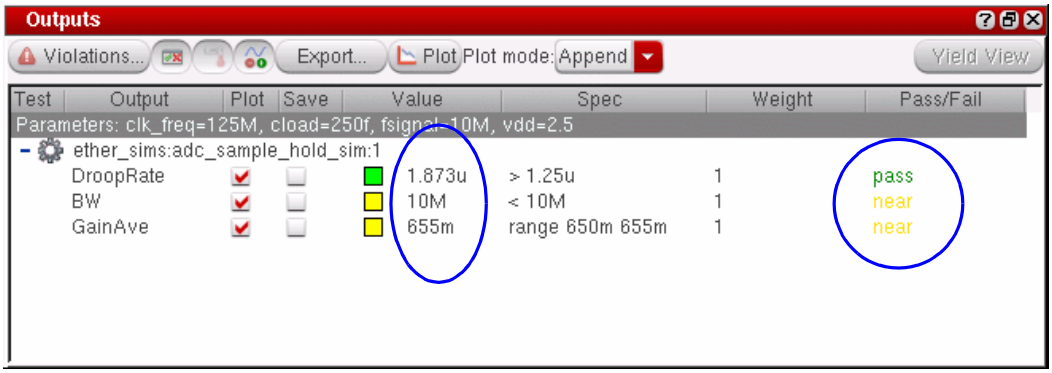
You can view the results in color coded pass/fail format.

The following table describes the different specification types that can be set in the Outputs assistant.

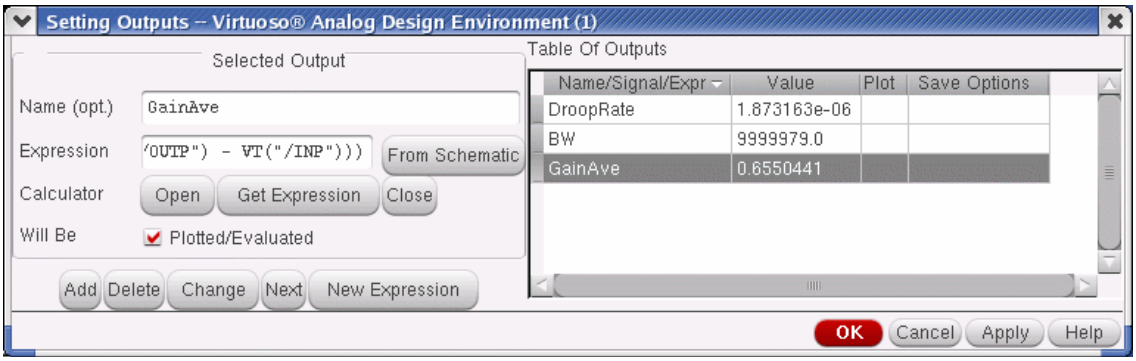
Spec	Description
Maximize	Maximize the value of measurement. The value specified is the minimum value this measurement should have to pass.
Minimize	Minimize the result. The value specified is the maximum value this measurement can have to pass.
(*gt*)	The measurement result needs to be greater than this value.
(*lt*)	The measurement result needs to be less than the specified value.
tolerance	Measurement result can be in the tolerance range (%) specified here with respect to the value specified.
range	Give the range in which value is acceptable.
info	Prints the measurement value only. No pass/fail information is provided.

View and Analyze Measurement Results

- Measurement results are shown with the pass/fail information after the simulation ends.
- Measurement name and values are also shown under the Output and Value columns. The Value field is rounded off, so that pass/fail will display “near” if the value is within 10% of the specification.



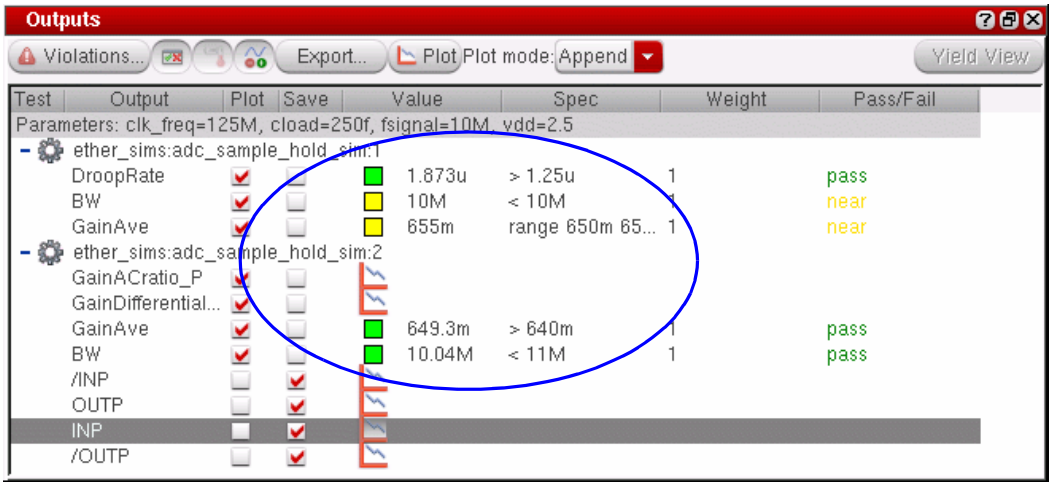
Double-click on the expression name to bring up the Setting Outputs window. From this window, you can see the value without the rounding.



If you compare the numbers, you see that *BW* and *GainAve* are labeled as *near*, because their values are within 10% of the specification.

Working with Specifications

- You can define different specification values or types when running multiple tests or analyses.
- Any measurement can pass for one test and can fail for another, depending upon the type and value of the specification.

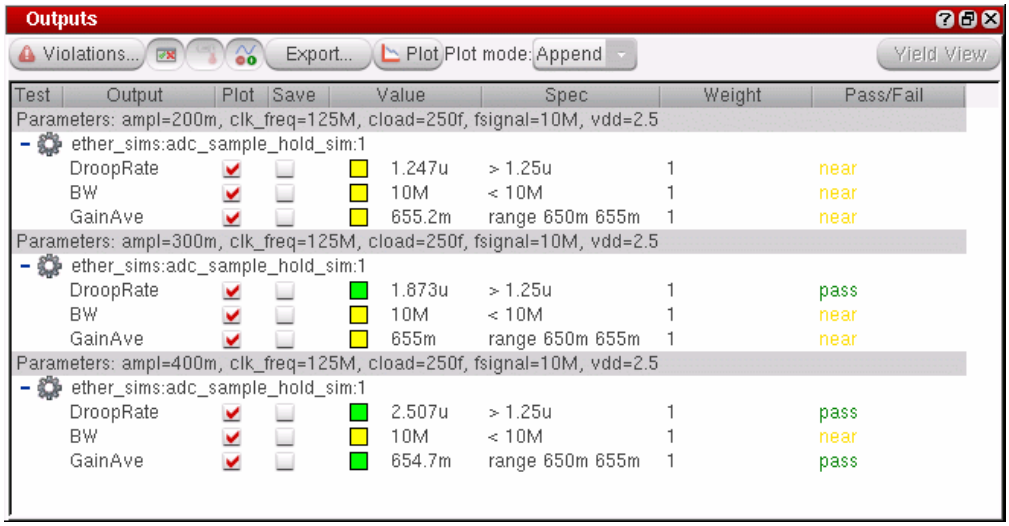


Generating Specifications 10-11

- Different specification values or type or both can be specified for same measurement but for different tests or analyses.
- Any measurement can pass for one set of analyses and can fail for another depending upon the spec type/value specified.
- As you can see above, different specifications are specified for same measurement. This measurement passes for one set of sweep where as it fails for another set of sweep.

Specifications with Corners and Sweeps

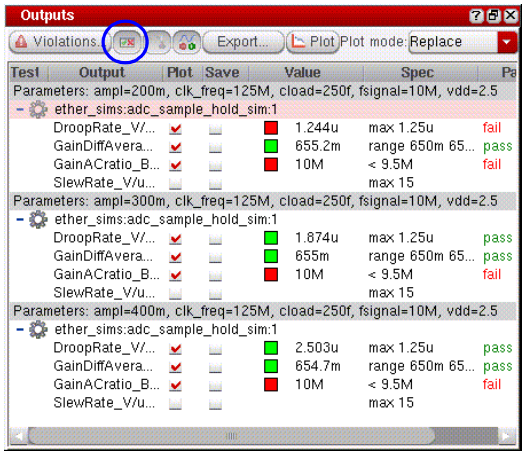
- When running corners or sweeps, the specifications are compared against the measurements for each corner or sweep run.
- You can analyze/view if the measurements are passing for one corner/sweep and are failing for another.

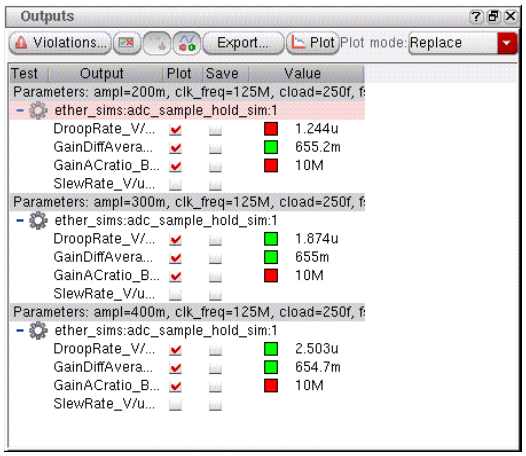


- When you run corners or sweeps, all the measurements are done for each corner and/or sweep.
- Specifications are compared against each measured value in each corners/sweep run.
- You can see if the specifications are passing/failing for each corners/sweep run.

Show/Hide Specifications

You can use the show/hide toggle button to show or hide the Spec column in the Outputs assistant.





- You can show/hide the Spec column by using the toggle button.
- Pass/Fail information is shown even when the Spec column is hidden.

Summary

In this module, you learned

- How to compare the measurement results with the design specifications.
- How to view and analyze the measurement results in the pass/fail format.
- How to compare the specifications with measurement results in case of corners and sweeps.
- How to show/hide the Spec column in the Outputs assistant.

Quiz

1. What are the seven different criteria that you can put on a specification?
2. Can you modify a specification after simulation?

Labs

Lab 10-1 Generating Specifications

Data Assistant

Module 11

December 14, 2006

Module Objectives

In this module, you will

- Show active setup and history tree
- Use checkpoints
- Create a datasheet

Data Assistant

- Data assistant
  - Active setup
  - History tree
- Restoring a checkpoint
- Creating datasheets

All these features are available in ADE XL and ADE GXL environments.

In the Virtuoso® ADE XL and GXL environments, the data assistant contains the current active setup and a list of previous runs in the checkpoints in the history tree.

Active setup tree contains information on the currently active tests, specifications, variables, and corners.

The History tree is managed in checkpoints, which contain information similar to the active setup for pervious runs. Any previous run can be restored to access tests, measurements, specifications, results, sweeps, or corners.

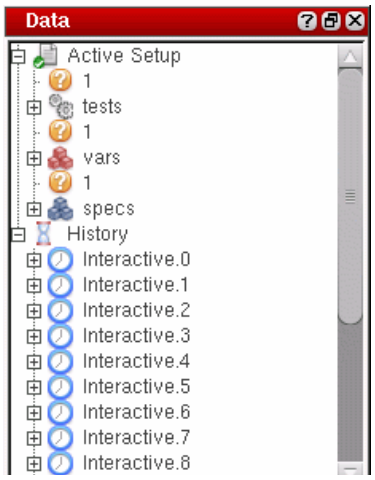
The environment creates the checkpoints prior to running the simulation and stores these checkpoints in the project directory.

You can also restore a part of the checkpoint (just specifications or tests). This restored checkpoint can be modified or kept as is to run the simulation again.

You can also create a *datasheet* from a particular checkpoint.

Data Assistant (continued)

- ADE XL and GXL environments save the current active configuration and a set of previous run configurations in the Data assistant.
- Configuration includes setup information such as corners analyses setup, design parameters, sweeps, and test definitions.
- Two different tree structures are stored in the Data assistant:
  - Active Setup tree
  - History tree



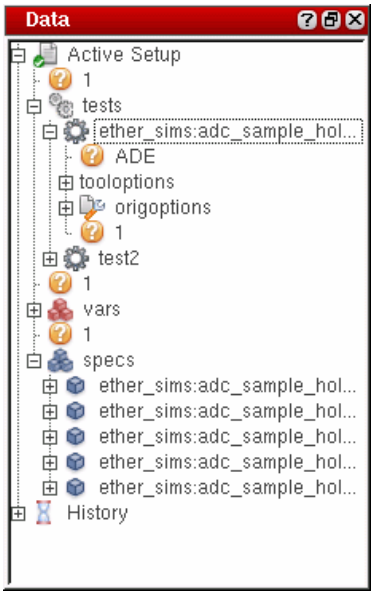
Data Assistant 11-7

The configuration setup in the Data assistant is stored in the form of tree structures.

- To expand a tree branch on the Data assistant, click **+** to expand an item or double-click on the tree. The set of items belonging to that branch appear beneath it.
- Similarly, to collapse a tree branch, click **-** to collapse an item. The set of items belonging to that branch are no longer visible.
- You cannot expand the top branches (interactive.n) in the history tree by double clicking it. You can type new names for the top branches in the history tree by double-clicking them.

Active Setup Tree

- Details of the current active setup appear in the *Active Setup* tree. You can view details, such as tests, specifications, variables, and corners setup, in the active setup tree.
- To view the details of the active setup, click **+** to expand the tree.

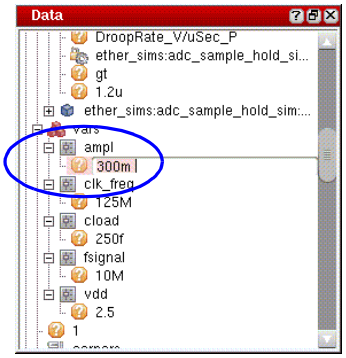


Details of the currently active configuration is stored in the Active Setup tree. It has details on the current setup, including corners analyses setup, design parameters, sweep, and test definitions. The active setup tree can be expanded or collapsed using the +/- symbols next to the active setup.

The name of the tree can be changed. Double-click on the name to edit it. You can give it any name you choose. This name is valid for the current ADE session only. This name is restored to *Active Setup* as soon as you restart the ADE session.

Working with the Active Setup Tree

- You can change the configuration of the current active setup at any time.
- The setup under the following tree branches can be changed at any time:
  - tests
  - specifications
  - variables
  - corners
- After you make changes to the above setup, you can run the simulation with the modified setup.
- There are a couple of places to change parameters:
  - Use the right mouse button in the Parameters assistant
    or
  - Change the parameters in the Data assistant

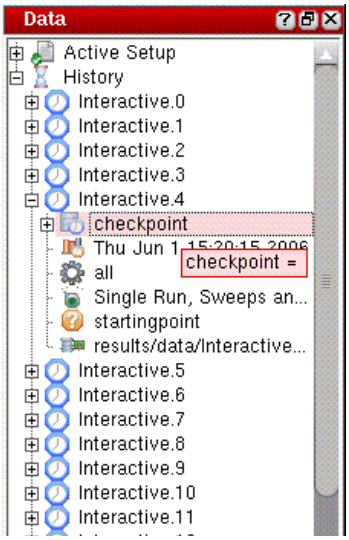


The setup under the following tree branches can be changed:

- tests
  - You can change the test that runs the simulation.
  - You can change the tool options. For example, you can change the cell/lib/view on which the simulation is running at any time. The simulation will now run on the new cell/lib/view.
  - You can change the simulator at any time. The simulation will run with the new simulator.
  - You can change the artist saved state which is used to run the simulation. You need to specify the location and the name of the new artist state to use for running the simulation.
- specs
  - You can change the name of the measurement/spec, type, and value of the specification at any time. The simulation results will be compared against the new specifications.
- vars
  - You can change the variable values at any time. The simulation will run with the new set of variables. As you can see in the assistant above, you can change the value of the *ampl* variable.

History Tree and Checkpoints

- In the ADE XL environment, the active configuration of data, such as corners analysis setup, design parameters, and the sweep and test definitions, are saved in the checkpoint.
- A checkpoint is created for every configuration saved.
- You can restore a partial or a complete checkpoint to run a simulation.



Setup for previous runs are stored in the history tree in the form of checkpoints.

You can change the name of the tree structure to any name you want, say *my_history*. This name stays for the current ADE session only.

Saved setup in checkpoint includes tests, the time stamp when the simulation was run, the type of simulation (single run or optimization), corners analyses setup, design parameters, sweep setup, and results database location for the run.

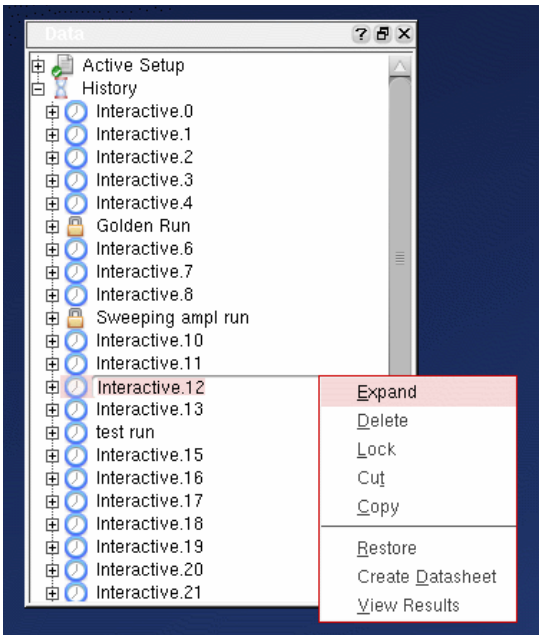
A checkpoint is created for every configuration setup that is saved.

This checkpoint is created prior to running the simulation and is stored in the project directory.

You can restore sections of a checkpoint (such as just the sweep setup), edit the data, and run the simulation again.

Working with Checkpoints

- Use Expand to see more information.
- You can rename checkpoint and then “lock” it.
- Copy does nothing and will be removed.



Data Assistant 11-15

In the data assistant, click **+** to expand the history tree.

The history tree has history entries (Interactive.n), which have checkpoints stored. You can rename these history entries. Double-click on any history entry to edit it.

To view the details for a particular checkpoint, click **+** to expand the checkpoint.

Checkpoint details appear in the expanded branch. The checkpoint stores setup information including tests, tool options (cell/lib/view, simulator used, artist state used), parameters, specifications, corners, time stamp (when the test was run), and results directory information.

You can rename a checkpoint by double-clicking it. When you double-click a checkpoint, you can edit the name.

You cannot change the setup information in a checkpoint. It is stored permanently.

You can lock a history entry so that no one else except the owner can view the details in it.

You can delete a checkpoint. Right-click on a checkpoint and select **delete** from the drop-down menu.

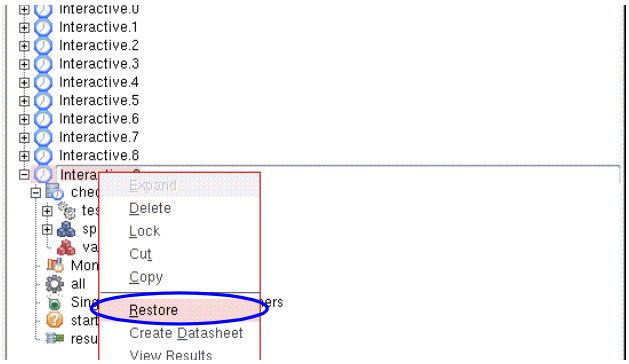
You can restore the setup from a checkpoint for running the simulation again.

You can view results for a particular run by restoring results from checkpoint.

You can also create a datasheet using a checkpoint.

Restoring a Checkpoint

- You can restore the complete checkpoint or only some part of it.
- Right-click on a checkpoint and select **Restore** from the drop-down menu. Complete test information and the parameter information is restored to various assistant panes.
- You can restore a test/spec or any other individual setup from a checkpoint.
  - Right-click on the setup (test/spec) and select **Restore** from the drop-down menu.



Data Assistant 11-17

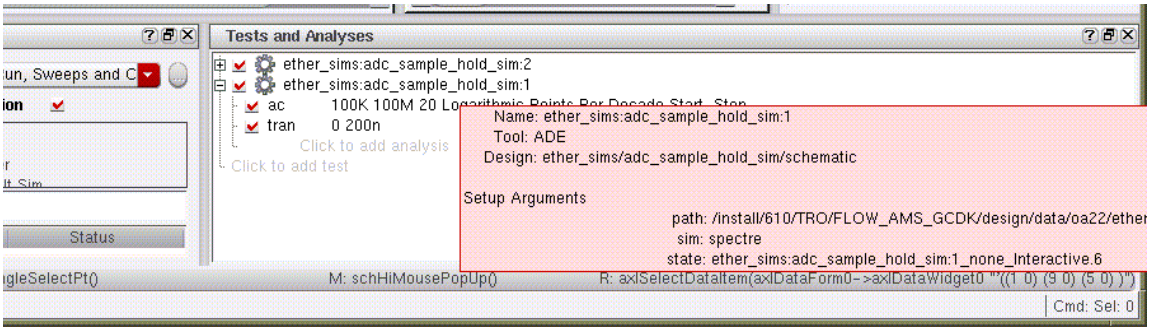
You can restore the configuration from a checkpoint. This restored configuration can be edited to run the simulation again.

You can restore the complete checkpoint, which restores the complete setup from the checkpoint. You can also restore any individual setup, such as test/corner/spec, from a checkpoint.

You can use this feature to rerun the simulation “as is” or after making some changes.

Restoring a Checkpoint (continued)

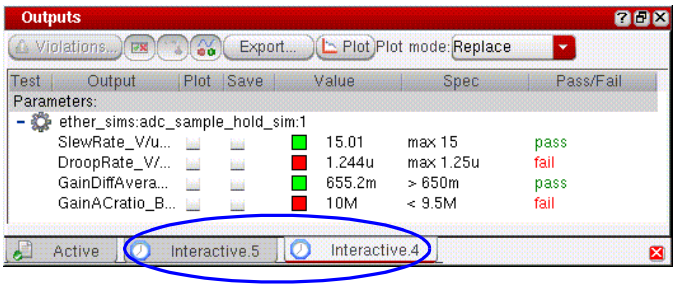
After restoring a test from a checkpoint, you can position your cursor over the test to verify the checkpoint name.



When you restore a test from a checkpoint, put your cursor over the restored test to verify the checkpoint name. This action will ensure that you have restored the correct checkpoint and it is restored successfully.

Viewing Results from a Checkpoint

- Right-click on a checkpoint and select **View Results** from the drop-down menu.
- The results appear on a new tab on the Outputs assistant.
- The name of the tab in the Outputs assistant matches the name of the checkpoint you restored.
- Results tabs appear along the bottom of the Outputs assistant.



- You are viewing results only. The Active Setup stays the same.

You can also use the saved checkpoint to view the results for a run. The results appear on a new tab in the Outputs assistant.

The name of the new tab in the Outputs assistant is same as the checkpoint name.

All the results including the specs information are restored to the Outputs assistant.

This is different from restore. In restore, you restore all the setup for running the simulation again, whereas here you are only viewing the results.

Creating a Datasheet

- A datasheet is an important project document that typically contains a tests and results summary for a design.
- You can create a datasheet from a checkpoint in the Data assistant.
- The datasheet is created in the *<library>*/*<cell>*/*adexl*/*datasheet*s/ directory.
- The datasheet is created in the HTML format.

Data Assistant 11-23

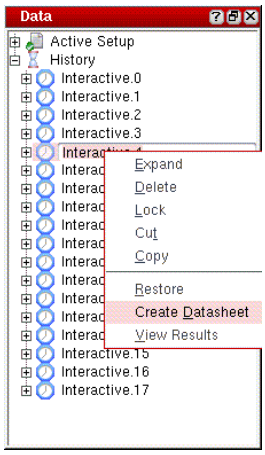
A datasheet is an important project document for any design activity. A typical datasheet contains a tests and results summary for a design. This document can be referred to at any later date to get answers for questions like which tests were run on a design and what were the results.

Datasheet can be kept with other project documents for future reference.

Creating a Datasheet (continued)

To create a datasheet, go to the Data assistant. Then follow these steps:

- From the History tree, right-click on any checkpoint (Interactive.n)
- Select **Create Datasheet**.



Data Assistant 11-25

You can create a datasheet from any checkpoint on the History tree in the Data assistant.

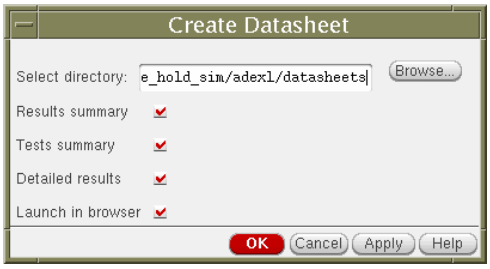
Right-click on any checkpoint and choose Create Datasheet to display the Create Datasheet form.

Creating a Datasheet (continued)

The Create Datasheet form has check boxes with options to save:

- Results summary
- Tests summary
- Detailed results

It also lets you launch the datasheet in the default browser after saving.



While creating a datasheet, you have option(s) to save the Results summary, the Tests summary, and the Detailed results.

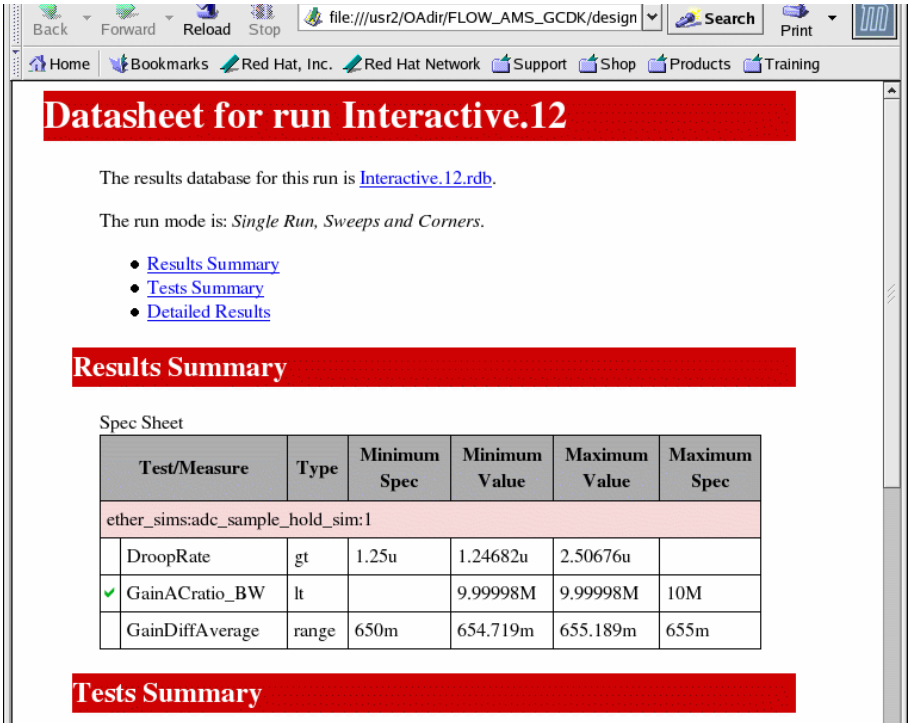
You also have the option to launch the default browser with this newly created datasheet.

By default, all options are turned ON.

Check box	Description
Results summary	When turned on, this option writes a summary of test and measurement results in the datasheet.
Tests summary	When turned on, this option writes the test summary information in the datasheet, including test design, simulator, and state information, variable values, sweep, corners, and model library information.
Detailed results	When turned on, this option writes the detailed results information to the datasheet, including parameter values for each run (sweeps, corners) and the output values for each test.
Launch in browser	When turned on, this option launches the datasheet in an HTML browser.

Creating a Datasheet (continued)

If the Launch in Browser check box is checked, the newly created datasheet is opened in the default browser.



The datasheet can be viewed later in any other HTML browser.

Datasheet is created by default in the <library location>/<*cell*>/*adexl*/*datasheets* directory.

When you save the datasheet, you have options to save Tests summary, results summary and detailed results. If these options are chosen, the created datasheet has links to tests summary, results summary, and detailed results. You can click on any one of the links and it takes you to the appropriate section in the datasheet.

As you can see above, the results summary section of the datasheet shows the measurement results.

Similarly, the Tests summary section has the test summary information, including test design, simulator, and state information, variable values, sweep, corners, and model library information.

The detailed results section of the datasheet has the information on parameter values for each run (sweeps, corners) and the output values for each test.

Summary

In this module, you learned:

- What are the active setup and the history tree
- How the setup information is stored in checkpoints
- How to rename and delete checkpoints
- How to restore setup from a checkpoint
- How to view results from a checkpoint
- How to create a datasheet

Quiz

1. When is a checkpoint saved?
2. How do you load data from a previous run?

Labs

Lab 11-1 Using the Data Assistant

Lab 11-2 Restoring a Checkpoint

ADE XL and GXL Customization

Module 12

December 14, 2006

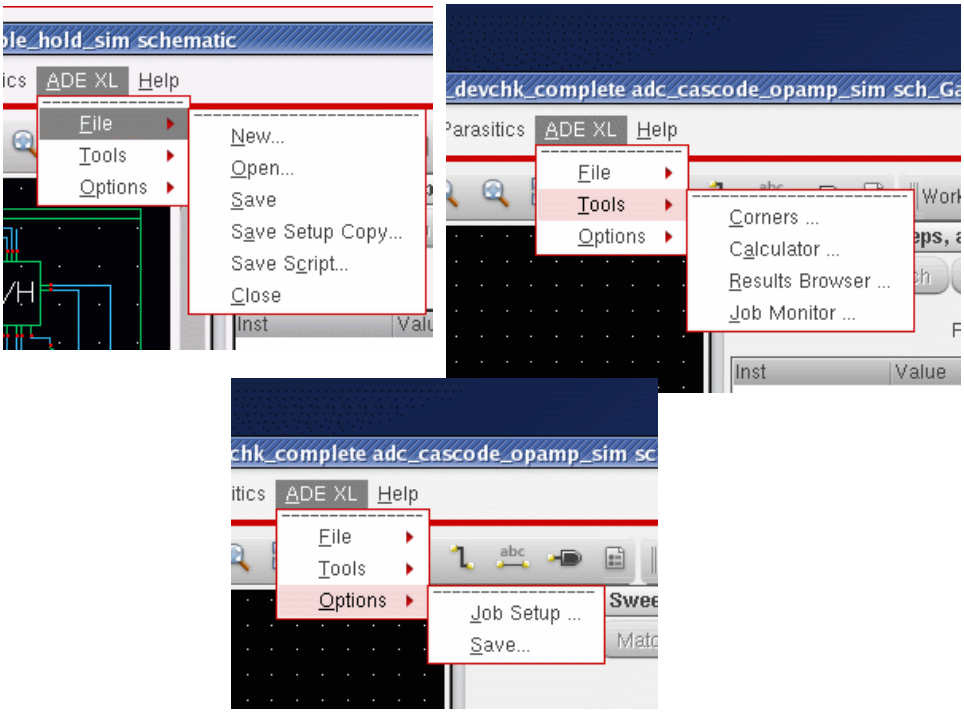
Module Objectives

In this module, you will

- Customize the setup of ADE XL and ADE GXL
- Change the user interface and set variables in the *.cdsinit* or *.cdsenv* files

ADE XL and ADE GXL Setup

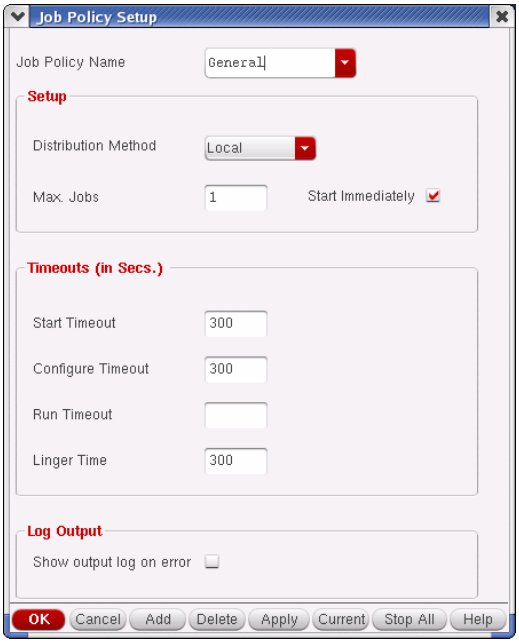
When you launch XL or GXL, an ADE XL pull-down menu is added to the menu bar.



- The ADE XL pull-down permits customized setup of XL or GXL environment.
- The *.cdsenv* and *.cdsinit* variables are also available for setup variables.
- Save Setup Copy is a quick way to copy the *adexl* view.
- Save Script will save an OCEAN script.

ADE XL and GXL Job Policy Setup

- Job Policy defines how ADE XL and GXL runs tests.
- Choose **ADE XL – Options – Job Setup**.



- Job Policy Name is a type-in field. You can save setups here. Click **OK** to save the information in ./.cadence/jobpolicy
- To set the default in your *.cdsenv* file, enter:

```
adexl.icrpStartup defaultJobPolicy string "mySetup"
```

- The set the default in your *.cdsinit* file, enter:

```
envSetVal("adexl.icrpStartup" "defaultJobPolicy" 'string
"mySetup")
```

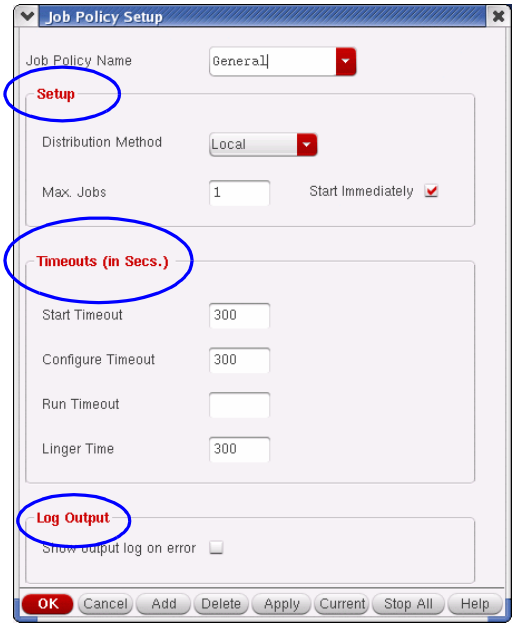
ADE XL Job Policy Setup

Job Policy setup has multiple options which can be broadly put into three categories.

- Setup
  - Distribution methods
- Timeouts
  - Start Timeout
  - Configure Timeout
  - Run Timeout
- Log Output
  - Show output log on error.

All these options can be set from the graphical interface as well as with environment variables.

You can give a name to the job policy with specified options and it can be used by the designer.



- To specify the name of the binary to run on the remote host, put the following line in your *.cdsenv* file:

  ```
  adexl.icrpStartup binaryName string "icms"
  ```

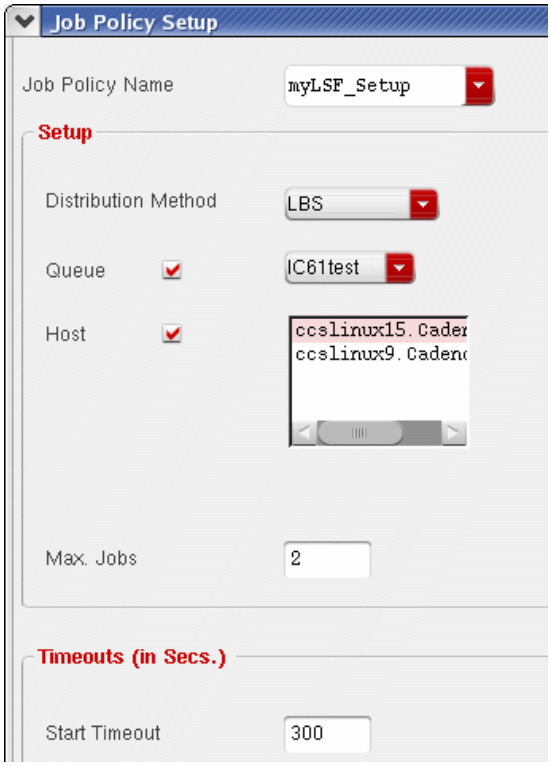
- Alternatively, a similar line for the *.cdsinit* file looks like this:

  ```
  envSetVal("adexl.icrpStartup" "binaryName" 'string "icms")
  ```

  Note: The default is *icms*.

ADE XL Job Policy Setup (continued)

- Setup (Distribution Methods)
  - Local: The simulation runs on the local machine.
  - Remote Host: The simulation runs on a remote host. You also need to specify the remote machine name (similar to ADE L remote simulation).
  - Command: You can specify a command to send to your distributed system (such as type any bsub command here).
  - LBS: Distributed Processing is used for running simulation.
- LBS, LSF, and SGE are supported.



In the form, you can choose the Distribution method to use.

Local Job Policy: When you choose local job policy, you select **local** in the *Distribution Method* cyclic field. You also have the option to specify the maximum number of jobs that can be run at any time in the *Max Jobs* field. In this method, the simulation is run on your local machine.

Remote Host Job Policy: Select **Remote-Host** in the Distribution Method cyclic field. You need to specify the name of the remote host and max jobs. You can also specify job timeouts in this method. In this method, the simulation jobs are submitted to remote machines.

Command Job Policy: Select **Command** in the Distribution Method cyclic field. You have a new field, *command*, in this method. You will type the command you want to use to start the jobs here. This command is prepended to the *icrp* startup command. This entire command is run on the local host. You also have the option to specify max jobs and job timeouts in this method.

If you are using the load sharing facility (LSF), you can type this in the command field:

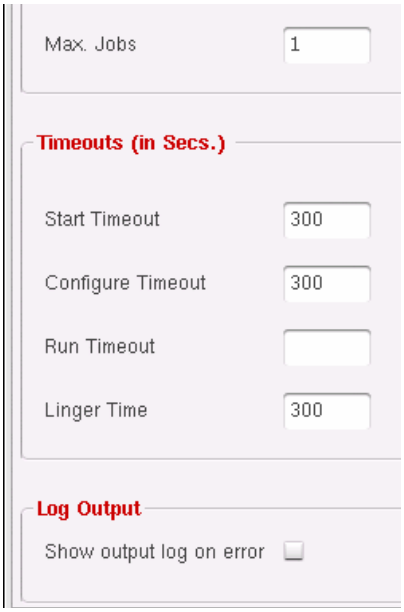
```
bsub -q <name of your queue> <some bsub options>
```

LBS Job Policy: LBS, LSF, and SGE distributed processing systems are supported by ADE XL. To use any of them, choose LBS in the distribution method cyclic field. Select the Queue and Host. You can also specify the max jobs and job timeout options. Valid values are any binary that is valid on the remote host.

LBS: Load Balancing System. Simple job distribution system for setting up queues (collections of host machines) and hosts per queue; each queue has a job-per-host limit.
LSF: Load Sharing Facility. DRMS from Platform Computing
SGE: Sun Grid Engine. Freeware from Sun Microsystems, Inc.

ADE XL and GXL Timeout Setup

- When you submit a job using ADE XL/GXL, a new process (icrp) is started. This process is started in background and it communicates with the ADE environment.
- Start Timeout: Number of seconds of time to wait for the icrp process to report that it has started the job.
- Configure Timeout: Number of seconds to wait for icrp process to report that it has configured the job.
- Run Timeout: Number of seconds to wait for icrp process to report that simulation job completed. A blank on the form equals infinity (no timeout).
- Linger Time: Keeps the remote jobs alive for this much time and then kills them. If you have a busy LSF queue, you won't keep jobs running indefinitely in the queue.



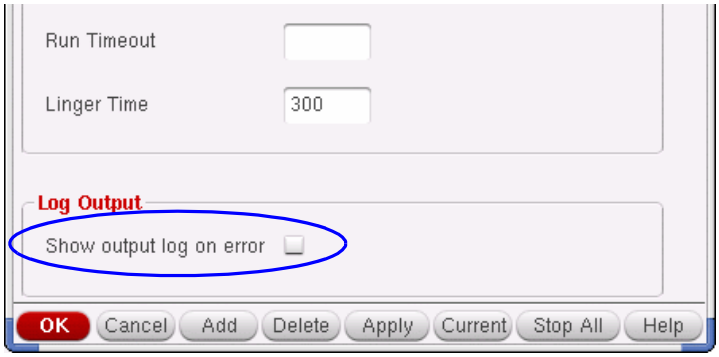
When you submit a job from the ADE environment, the new process called *icrp* is started in the background. This *icrp* process communicates with the ADE environment.

When you submit jobs using ADE XL and you choose any distribution method other than *local*, you have an option to specify timeouts. These timeouts are the time in seconds that ADE waits for the *icrp* process to respond. You can specify these timeouts using the Job Policy Setup graphical interface.

ADE Job Policy Setup

Show output log on error

This option can be set from the graphical interface or by using an environment variable.



- Default job log saved in ./logs.
- For debugging purposes, the display of error information is controlled by the environment variables shown below.

- When setting the ADE job policy, you can also choose an option to display the log file in case of an error. This option can be set using the graphical interface or with an environment variable.
  - In your *.cdsenv* file, add:

```
adexl.icrpStartup showOutputLogOnError boolean t
```

  - Or, in your *.cdsinit* file:

```
SetVal("adexl.icrpStartup" "showOutputLogOnError" 'boolean t)
```

- You can also write standard output messages from a job submit command to the CIW. These options are not available in the graphical interface. They can be set with environment variables only. Similarly, you have options to write output error messages to the CIW.
  - In your *.cdsenv* file:

```
adexl.icrpStartup showJobStdout boolean t
```

  - Or, in your *.cdsinit* file:

```
envSetVal("adexl.icrpStartup" "showJobStdout" 'boolean t)
```

- To write standard output error messages from the job submit command to the CIW,
  - In your *.cdsenv* file:

```
adexl.icrpStartup showJobStderr boolean t
```

  - Or, in your *.cdsinit* file:

```
envSetVal("adexl.icrpStartup" "showJobStderr" 'boolean t)
```

ADE XL and GXL Database Setup

- *Data Points per Run* can limit the number of data sets saved.
- By default, all simulation results are saved under *<lib><cell>adexl*.
- Choose **ADE XL – Options – Save** to change the results directory location.

ADE XL and GXL Customization 12-17

- In ADE XL and GXL environments, the setup and results databases are saved by default at *<lib>/<cell>/adexl*. The *<lib>/<cell>/adexl* directory is created under the directory you specify. The location for output data can be changed using the form or by using an environment variable.
  - In your *.cdsenv* file:

```
adexl.setupdb saveDir string "~/simulation"
```

  - In your *.cdsinit* file:

```
envSetVal("adexl.setupdb" "saveDir" 'string "~/simulation")
```

- **Data Points per Run** refers to the number of *psf** directories saved after each run (including corners and sweeps). So if you specify this number to be 10, it will save the last 10 *psf* directories. In future releases, the default will be 5 in addition to any that are locked.
- **Design Points per Optimization Run** refers to how many "best" points to save for an optimization run. The definition of a point is a set of parameter and global variable values across corners. This number is 10 by default. So even if the optimization generates 3000 points, it will save only the 10 best points.

* psf is an acronym for parameter storage format. It is a Cadence® standard format for storing analog signal numerical results in a binary file.

Quiz

1. What are the four types of Distribution methods available in ADE XL?
2. What is icrp?

Labs

There are no labs for this module.

Corners and Sweeps

Module 13

December 14, 2006

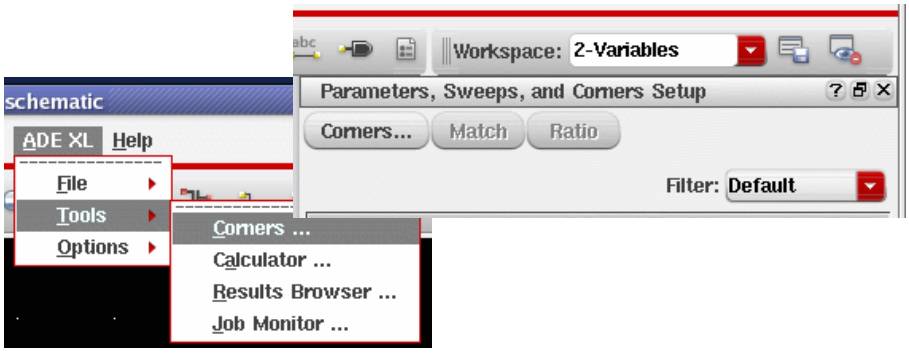
Module Objectives

In this module, you will

- Set up and run a corners simulation
- Specify an instance parameter as a sweep parameter
- Specify a design variable as a sweep parameter

Corners

- Corners and sweeps are now features in Virtuoso ADE XL.
- There is a new interface compared to the IC 5.1.41 release.
- Vary the temperature, device parameters, and design/global variables from any test.
- There are two ways to start the *Corners Setup* graphical interface:
  - From the ADE XL session, choose **ADE XL – Tools – Corners**.
  - From the Parameters, Sweeps, and Corners Setup assistant, click the Corners button.



The Corners tool (along with all EDFM tools) is now part of Virtuoso® ADE XL and not ADE L. The Corners Setup graphical interface is quite different compared to the Corners tool in IC 5.1.41. There are two ways you can display the Corners Setup form:

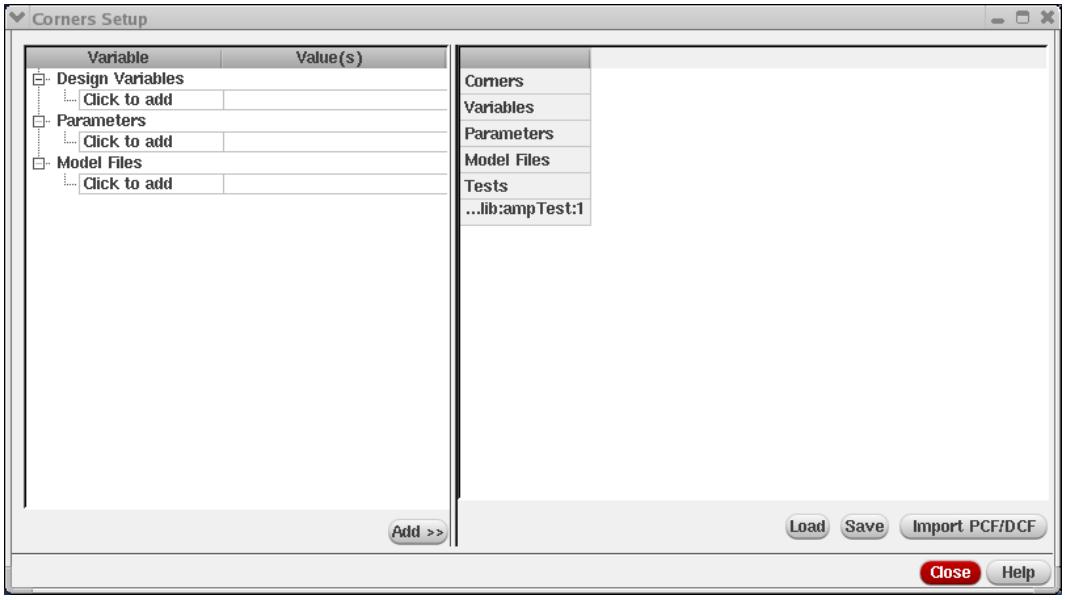
1. From the ADE XL session, choose **ADE XL – Tools – Corners**.

   or

2. From the Parameters, Sweeps, and Corners Setup assistant, click the **Corners** button. You can choose the Variables workspace to get the Parameters, Sweeps, and Corners Setup assistant in your ADE XL session.

Corners Setup Form

This is just a setup form. You cannot run Corners simulation from this form.



When you display the Corners Setup form either from the ADE XL session by choosing **ADE XL – Tools – Corners** or from the Parameters, Sweeps, and Corners Setup assistant by clicking the **Corners** button, you get the Corners Setup form similar to the one above.

From the Corners Setup form, you can add design variables, parameters, and model files (with or without sections). You can also load and save corners setup and can import previously defined PCF/DCF files to populate the Corners Setup form.

Unlike in IC 5.1.41, this is just a setup form. You cannot run the corners simulation from it. The corners simulation can be run from the ADE XL testbench.

Corners Setup

- Add design variables
- Add parameters
- Add model files
- Create corners
- Disable/Enable corners
- Remove corners/variables/parameters/model files
- Exclude a corner model file
- Specify model file section name
- Rename corners
- Import PCF/DCF files
  - PCFs are process customization files.

    Typically, a process engineer or process group defines processes, groups, variants, and corners in a PCF so that everyone in an organization uses the same definitions.
  - DCFs are design customization files.

    You can import a set of predefined corner variables and measurements from one or more DCF files. Typically, a design engineer or design group defines corner variables and measurement information for a particular design or for several designs within a design group in a DCF.

When you first display the Corners Setup form, unless you have loaded or imported PCF/DCF files, then the Corners Setup form does not have the necessary data to run corners simulation. You need to set up this form properly before running a corners simulation.

You can add corner variables for design variables, parameters, and model files that you have in your test setup. To add corner variables, do the following:

1. (Optional) Add design variables.
2. (Optional) Add parameters.
3. (Optional) Add model files.
4. Click the Add>> button.

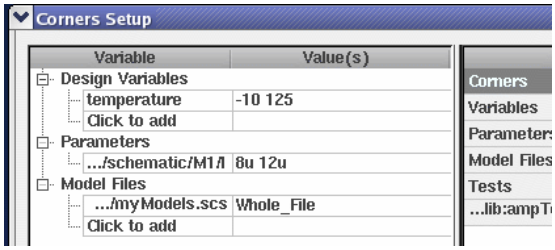
At this point, the corners you specified appear in the table on the right side of the form, one corner for each combination of values.

You can also add design variables and global/device parameters to an existing corner. You can enable/disable a particular set of corners. You can exclude some model files and specify different sections of a model file. You can enable/disable any set of corners by using the right mouse button and selecting **Enable for All Tests** or **Disable for All Tests**. You can also uncheck or check the red check button to disable/enable corners from the Corners Setup form. You can remove any corners/variable/parameters/model file by right-clicking on them and then using the right mouse button to select the appropriate command. You can rename any corner by double-clicking that corner and then typing in a new name. You can specify the model file section name by double-clicking the section name and then selecting a different section name from the available cyclic field.

Adding Variables/Parameters

For each of the Variables/Parameters that you want to add as a corner, do the following:

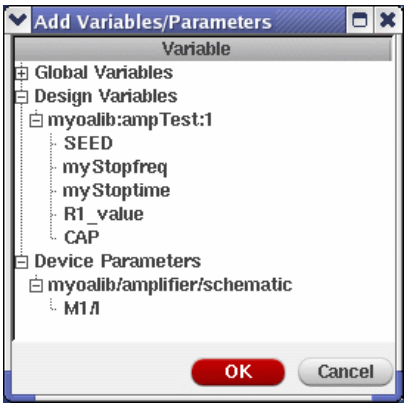
- In the Design Variables (Parameters) tree, double-click Click to add.
  - The field in the Variable column becomes a cyclic field from, from which you can select one of the design variables available in your test setup.
- In the cyclic field in the Variable column, select a design variable.
  - The design variable appears in the field.
- Double-click in the Value column field for the variable and type one or more comma-separated corner values.
- Click the **Add>>** button to create the corners. On the right side of the form, select the combined Corner, and use the right mouse button to choose **Expand Corner**.



Adding Design Variables/Parameters to Corners

To add design variables and parameters, do the following:

- Left-click in the corner you want to change, and use the right mouse button select **Add Variables/Parameters** from the pop-up menu.



- Select variables/parameters you want to add to corners and click **OK**.

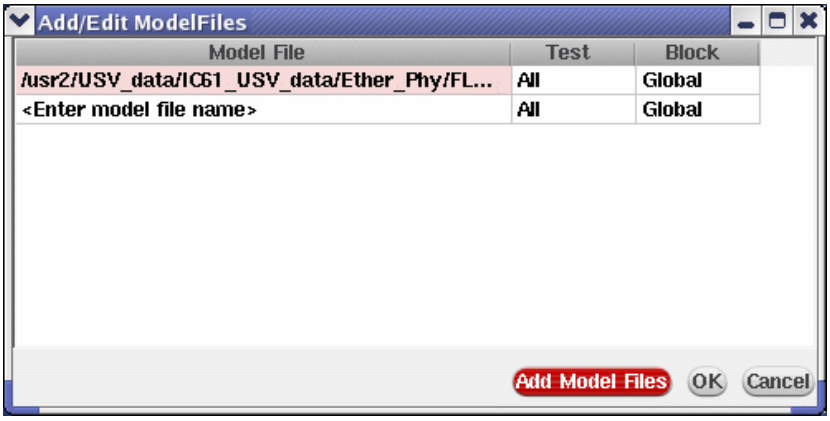
To add design variables and parameters to corners, do the following:

1. Right-click in a corner column on the right side of the Corners Setup form, use the right mouse button and select either Add Variables/Parameters from the pop-up menu. The Add Variables/Parameters form appears.
2. While holding down the Control key, select the variables and parameters to add.
3. Click OK.

Each variable and parameter appears as a new row on the right side of the Corners Setup form. Their values appear in the column you right-clicked. You can change or add a value by double-clicking in a column and typing a new value.

Adding Model Files

To add a model file, in the Model Files tree, double-click Click to add. The Add/Edit ModelFiles form appears.



Corners and Sweeps 13-15

For each model file you want to add as a corner, do the following:

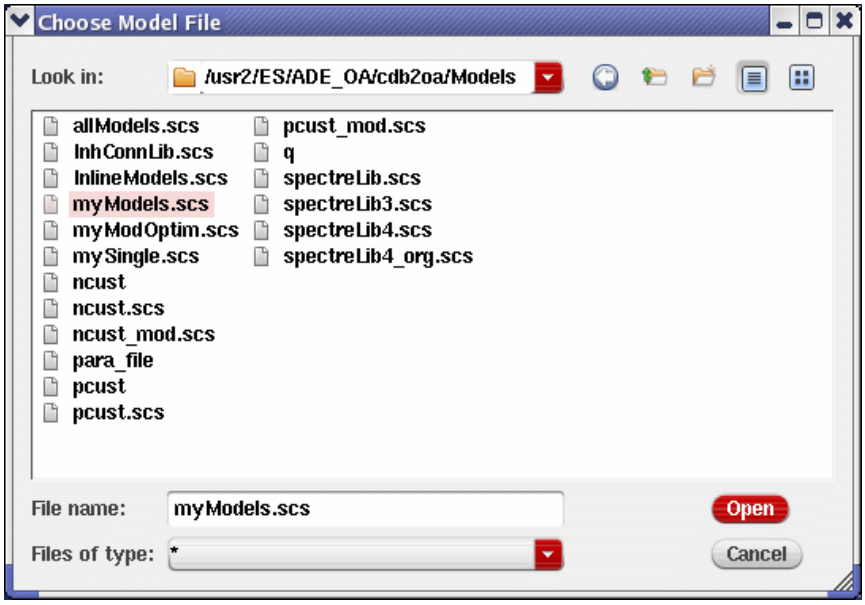
1. In the Model Files tree, double-click Click to Add. The Add/Edit Model Files form appears.

   This form also appears if you right-click in a corner column on the right side of the Corners Setup form and select Add/Edit Model Files.

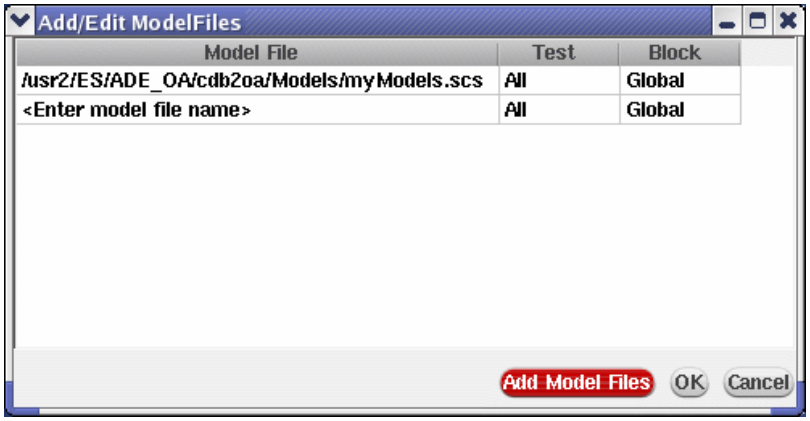
2. Click the Add Model Files button. The Choose Model File form appears.
3. Navigate to and select a model file.
4. Click Open.
   The model file appears on the Add/Edit Model Files form.

Adding Model Files (continued)

Click the Add Model Files button to display the Choose Model File form.



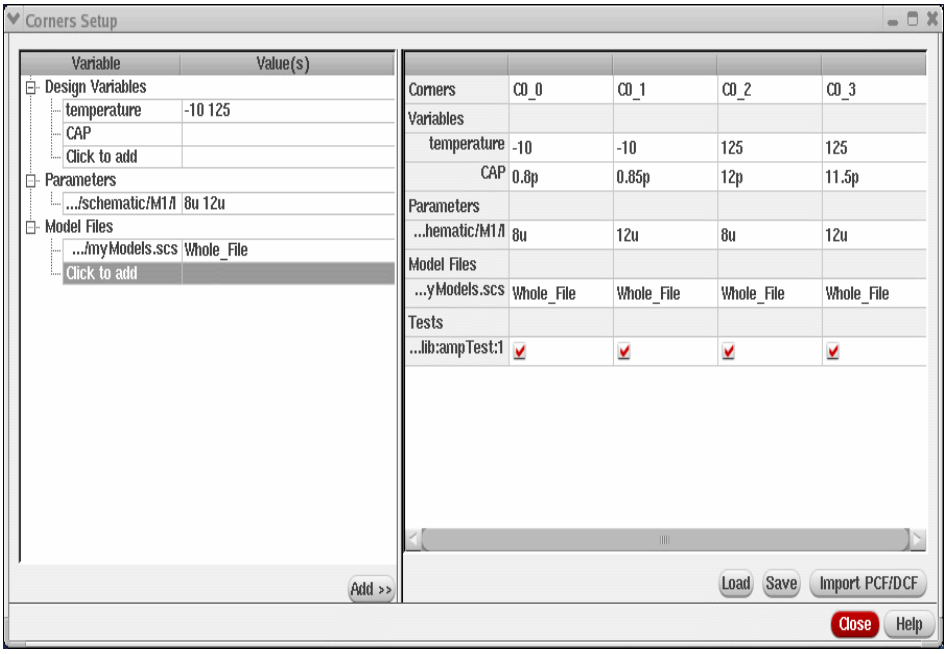
The chosen model file shows up in the Add/Edit ModelFiles form:



1. (Optional) In the Test column, double-click and select a particular test to which you want to apply the model file from the drop-down menu. By default, the program applies the model file to all tests.
2. (Optional) In the Block column, double-click and select a particular section of the design to which you want to apply the model file (when using MTS) from the drop-down menu. By default, the program applies the model file to all sections of the design: Global.
3. Click OK.

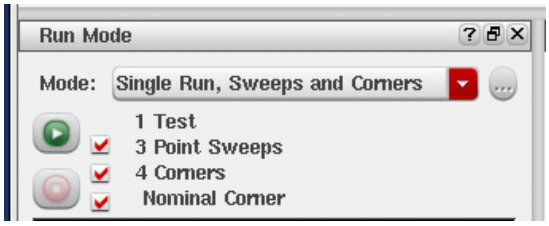
Running a Corners Simulation

When the corners setup is complete, use the Run button in the Run Mode to run the simulation for each corner.



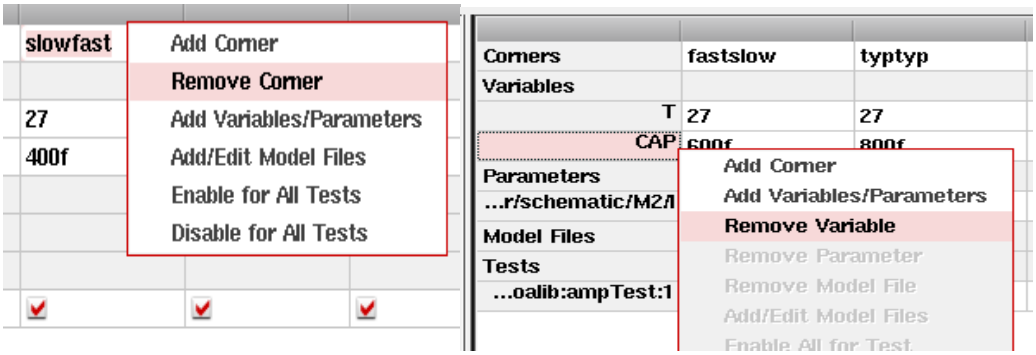
A fully setup Corners Setup form looks something like the one above. To run a Corners simulation, you need to use the run button in the Run Mode assistant pane (as shown below). There is no run button in the Corners Setup form.

At this point, in your Run Mode assistant pane, you see following:



Removing Corners, Variables, and Parameters

- To remove a Corner:
  - Right-click on the corner, hold, and select Remove Corner.
- To remove a Variable:
  - Right-click on the variable, hold, and select Remove Variable.
- To remove a Parameter:
  - Right-click on the parameter, hold, and select Remove Parameter.



- To remove a corner, do the following:

  Right-click in the corner column, hold the mouse button, and select Remove Corner. The corner column disappears from the form. The number of Corners reflected on the Run Mode assistant pane decreases by one.

- For each corner variable you want to remove, do the following:

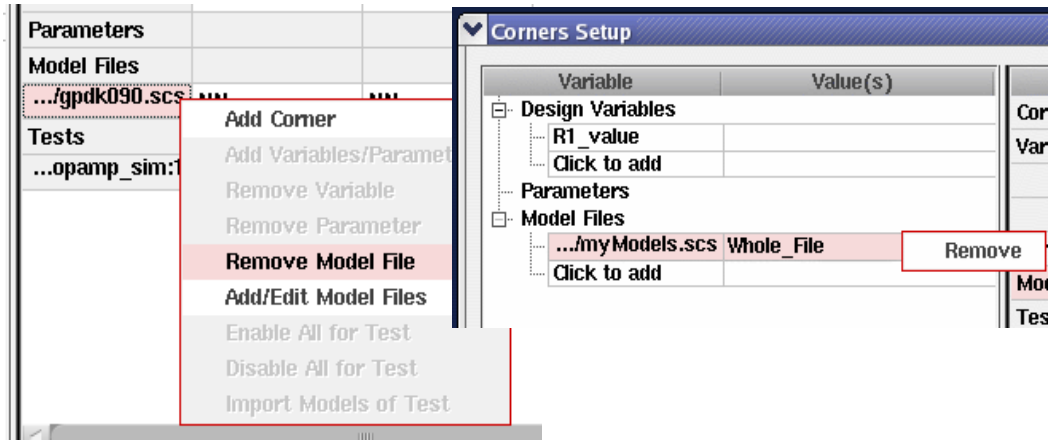
  Right-click the variable name in the table on the right side of the Corners Setup form, hold the mouse button, and select Remove Variable. The program removes the corner variable row from the table.

- For each corner parameter you want to remove, do the following:

  Right-click the parameter name in the table on the right side of the Corners Setup form, hold the mouse button, and select Remove Parameter. The program removes the corner parameter row from the table.

Removing Corner Model Files

- To remove a Corner Model File:
  - Right-click on the model file, hold, and select **Remove Model File**.
  - Right-click on the model file (without section), hold, and select **Remove**.



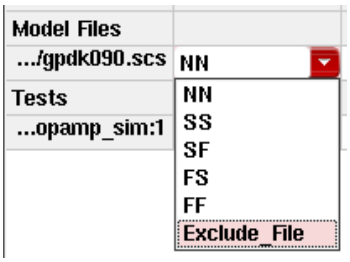
For each corner model file you want to remove, do the following:

Right-click the model file name in the table on the right side of the Corners Setup form, hold the mouse button, and select **Remove Model File**. The program removes the model file row from the table. If you are using a model file without any sections (a whole file), then you need to select the model file on the left side of the Corners Setup form, hold the mouse button, and select **Remove**.

Sometimes it is convenient to just exclude a corner model file rather than removing it. This is covered on next page.

Excluding Corner Model Files

- To exclude a Corner Model File
  - Double-click in the value column. Select **Exclude_File** in the cyclic field.
  - When you click **Add>>**, the program excludes that model file.
- To restore the excluded model file
  - Double-click in the Value column. Select **Whole_File** in the cyclic field.
  - When you click **Add>>**, the program includes that model file.
- To specify sections of a model file
  - Double-click in the value column. Select any section in the cyclic field.

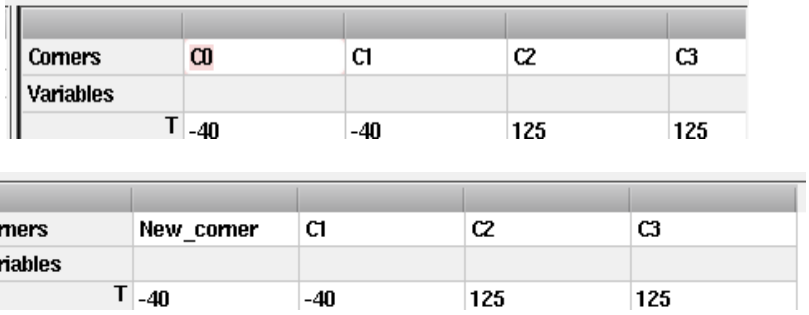


Corners and Sweeps 13-25

- For each model file you want to exclude from corner creation, do the following:
  - In the Value column, double-click and select **Exclude_File** in the cyclic field that appears. When you click **Add**>>, the program excludes that model file.
- To restore the model file for corner creation, do the following:
  - In the Value column, double-click and select **Whole_File** in the cyclic field that appears. When you click **Add**>>, the program includes that model file.
- If your model file has more than one section, you can specify each section you want to include in corners creation by doing the following:
  - In the Value column, double-click and select any section you want to include in the cyclic field that appears.

Renaming a Corner

- By default, the program names corners C0, C1, C2, and so on.
- To change the default name of a corner:
  - In the Corners row, double-click the default name.
    You can edit the highlighted name.
  - Type a new name for the corner.
  - Press Return.
    The new corner name appears as the column heading in the Corners row.



Corners	C0	C1	C2	C3
Variables				
T	-40	-40	125	125

Corners	New_corner	C1	C2	C3
Variables				
T	-40	-40	125	125

- By default, the program names corners C0, C1, C2, and so on. To change the default name of a corner, double-click the default name in the Corners row.
- You can edit the highlighted name.
  - Type a new name for the corner and press Return.
  - The new corner name appears as the column heading in the Corners row.

Disabling and Enabling Corners

- You can selectively disable or enable the nominal corner, corners for one or more tests, or all corners.
- To turn off the nominal corner simulation:
  - On the Run Mode assistant pane, remove the mark from the *Nominal Corner* check box.
  - The simulator does not run the nominal corner.
- To turn on the nominal corner simulation:
  - On the Run Mode assistant pane, mark the Nominal Corner check box.
  - The simulator runs the nominal corner.
- To disable a corner for all tests:
  - Right-click in the corner column and select **Disable for All Tests**.
  - The program removes the marks from the check boxes for all tests. The simulator does not run that corner.
- To enable a corner for all tests:
  - Right-click in the corner column and select **Enable for All Tests**.

Corners and Sweeps 13-29

- Disabling and Enabling the Nominal Corner

  The nominal corner is the one that runs without varying the corners variables.
  - To turn off the nominal corner simulation, do the following:

  On the Run Mode assistant pane, remove the mark from the Nominal Corner check box. The simulator does not run the nominal corner.
  - To turn on the nominal corner simulation, do the following:

  On the Run Mode assistant pane, mark the Nominal Corner check box. The simulator runs the nominal corner.
- Disabling and Enabling One Corner for All Tests
  - To disable a corner for all tests, do the following:

  Right-click in the corner column and select **Disable for All Tests**. The program removes the marks from the check boxes for all tests. The simulator does not run that corner.
  - To enable a corner for all tests, do the following:

  Right-click in the corner column and select **Enable for All Tests**. The program marks all check boxes for all tests. The simulator runs that corner.

Disabling and Enabling Corners (continued)

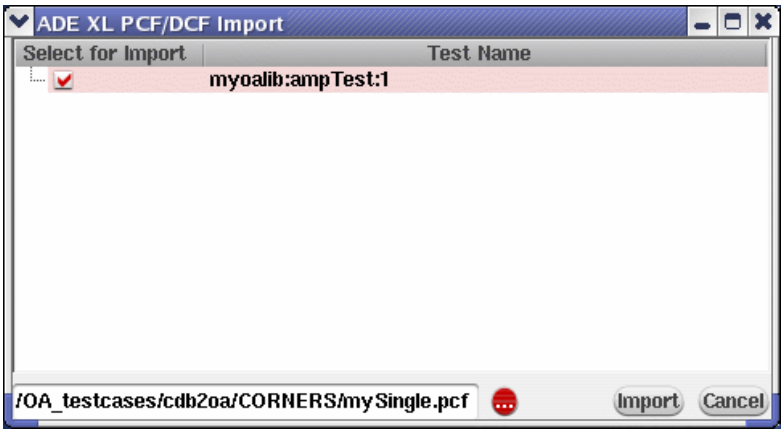
- To disable all corners for a test:
  - Right-click a test name and select **Disable All for Test**.
  - The program removes the check marks from the check boxes for all corners for that test. The simulator does not run any corners for that test.
- To enable all corners for a test:
  - Right-click a test name and select **Enable All for Test**.
  - The program marks all check boxes for all corners for that test. The simulator runs all corners for that test.
- To disable all corners:
  - On the Run Mode assistant pane, remove the mark from the # Corners check box (where # indicates the number of corners).
  - The Nominal Corner check box becomes unavailable. The simulator does not run any corners.
- To enable all corners:
  - On the Run Mode assistant pane, mark the # Corners check box.
  - The simulator runs all corners.

- To disable all corners for a test, do the following:
  - Right-click a test name and select **Disable All for Test**. The program removes the check marks from the check boxes for all corners for that test. The simulator does not run any corners for that test.
- To enable all corners for a test, do the following:
  - Right-click a test name and select **Enable All for Test**. The program marks all check boxes for all corners for that test. The simulator runs all corners for that test.
- To disable all corners, do the following:
  - On the Run Mode assistant pane, remove the mark from the # Corners check box (where # indicates the number of corners). The Nominal Corner check box becomes unavailable. The simulator does not run any corners.
- To enable all corners, do the following:
  - On the Run Mode assistant pane, mark the # Corners check box (where # indicates the number of corners). The simulator runs all corners.

  (Optional) Remove the mark from the Nominal Corner check box to turn off the nominal corner simulation.

Importing Customization Files

- On the Corners Setup form, click **Import PCF/DCF** button.

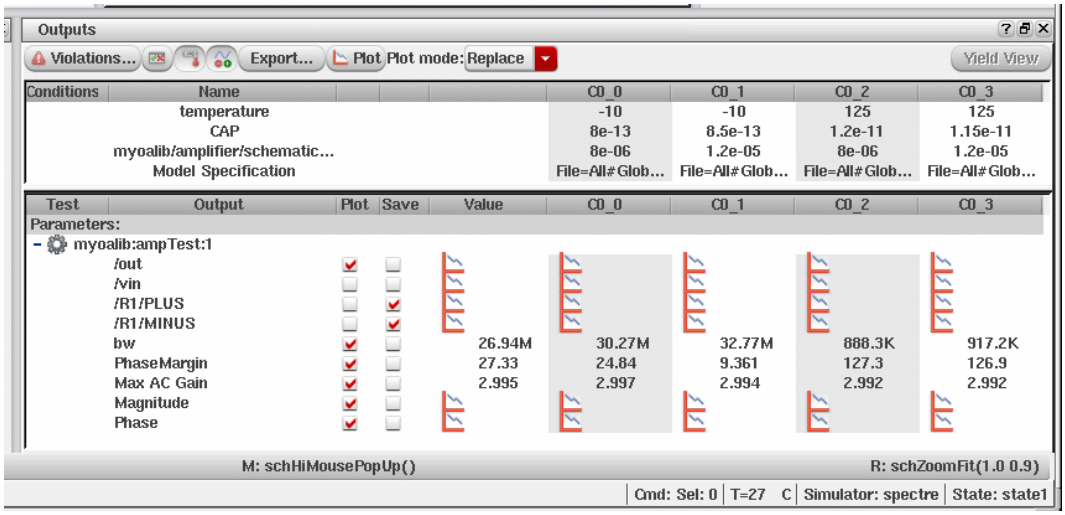


- Click the red ellipsis (**...**) button, select a file in the Choose PCF/DCF File for Import form and click **Open**. The file appears in the field in the lower left corner of the ADE XL PCF/DCF Import form.
- Mark the check boxes next to each test name and click **Import**. If there are any errors, they appear in the CIW.

- You can import a set of predefined corners from one or more process customization files (PCFs). Typically, a process engineer or process group defines processes, groups, variants, and corners in a PCF so that everyone in an organization uses the same definitions.
- You can import a set of predefined corner variables and measurements from one or more design customization files (DCFs). Typically, a design engineer or design group defines corner variables and measurement information for a particular design or for several designs within a design group in a DCF.
- Usually, you need to load process customization information before you can import corners or measurements from a design customization file. Customization must refer to definitions that you have already loaded.

Corners Outputs

After a successful corners simulation, the Outputs section looks like this.



After a successful corners simulation, the Outputs pane looks something like the one above.

You also see multiple waveforms in a WaveScan window if you have any signal set for plotting.

Sweeps

- The ADE XL environment has a sophisticated parametric analysis.
- You can run corners and sweeps together.
- You can display sweeps from the Variables workspace in ADE XL.
- You can sweep instance parameters, design variables, and global variables.
- You can add or change a parameter specification.

Now you can run Corners and Sweep together for any tests. You can sweep any instance parameters, design variables, and global variables.

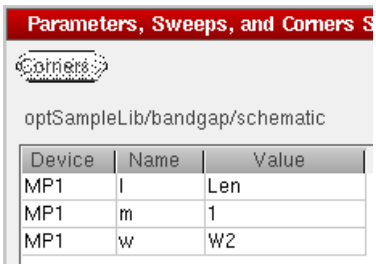
Creating a Global Variable

- To create a global variable:
  - Right-click the Global Variables check box in the Parameters, Sweeps, and Corners Setup assistant pane and select **Add Variable**.
  - The Create Design Variable form appears.

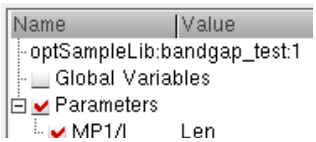
  - In the Variable Name field, type a name for your global variable.
  - Click **OK**.
- The global variable appears in the Global Variables tree on the Parameters, Sweeps, and Corners assistant pane.
- All design variables are promoted to global variables by default.

Specifying an Instance Parameter as a Sweep Parameter

- To specify an instance parameter as a sweep parameter,
  - On the schematic pane, select an instance.
  - The parameters associated with the instance appear in the device table on the Parameters, Sweeps, and Corners Setup pane.

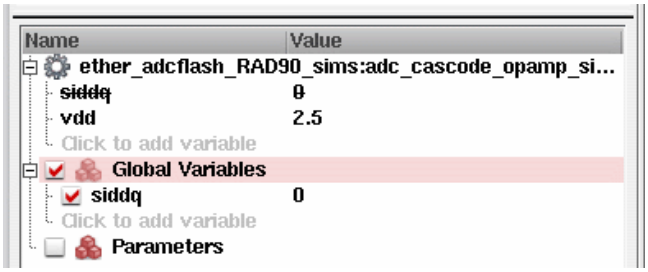


  - Right-click the instance parameter you want to use as a sweep parameter, hold the right mouse button down, and choose Create Parameter.
- The instance parameter and its value appear in the parameters table as shown below.



Specifying a Design Variable as a Sweep Parameter

- To specify a design variable as a sweep parameter,
  - Right-click the design variable to use as a sweep parameter and choose Make Variable Global.

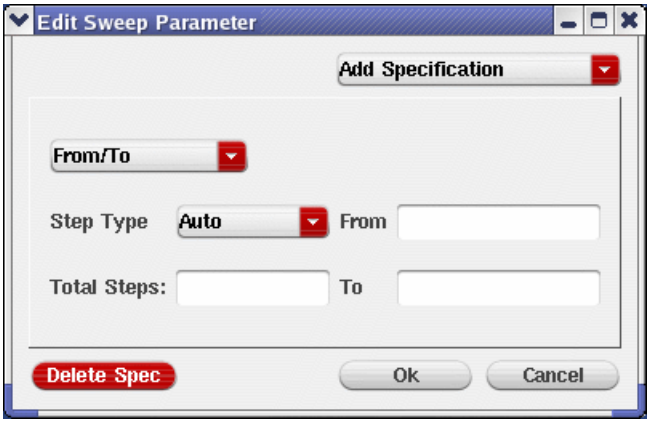


  - The program redraws the design variable name with a strikethrough for the test and adds the name to the set of global variables.
  - You can now define sweep values for this parameter.
- You can specify that you do not want a design parameter varied for a particular simulation while retaining its parameter definition in the parameters table of the Parameters, Sweeps, and Corners Setup pane by removing the mark from the check box to the left of the parameter name or identifier prior to starting your simulation.

Adding or Changing a Parameter Specification

To add or change a parameter specification for a design variable you want to vary/sweep

- On the Parameters, Sweeps, and Corners Setup pane, right-click the global or sweep parameter for which you want to add or change the specification and select **Edit Variable**. The Edit Sweep Parameter form appears.
- From the Add Specification menu, choose specification range.



Corners and Sweeps 13-45

To add or change a parameter specification for a design variable you want to vary (sweep), do the following:

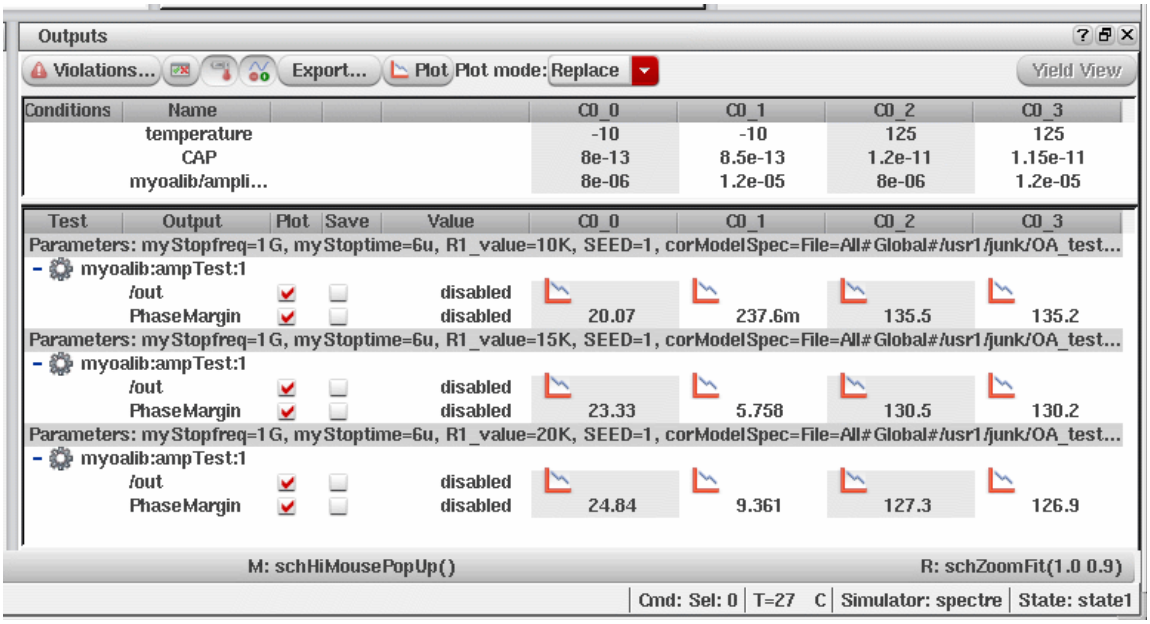
1. On the Parameters, Sweeps, and Corners Setup pane, right-click the global or sweep parameter for which you want to add or change the specification and select Edit Variable. The Edit Sweep Parameter form appears (as shown above). The name of your design parameter appears in the Variable Name field.
2. From the Add Specification menu, choose one of the following:

Add Specification	Definition
Inclusion List of values	You can specify a set of values through which you want to sweep your design variable.
Exclusion List of values	You can specify a set of values to exclude from your parametric sweep.
From/To range of values	You can specify a range of values through which you want to sweep your design variable and a method for stepping through that range.
Center/Span values	You can specify a center value and a span value for varying your design variable.
Center/Span% values	You can specify a center value and a percentage span value for varying your design variable.

Note: You can also type in the sweep range directly into the value field of the global variable or parameter using either start:step:stop or space/comma separated syntax.

Corners and Sweep Outputs

After successful Corners/Sweep simulation, the Outputs pane shows these results.



After successful corners and sweep simulation, the Outputs pane looks something like the one above. In this Outputs assistant pane, corners results are displayed horizontally while sweep results are displayed vertically.

Quiz

1. How do you display the “Corners Setup” window?
2. What are the different ways to create Corners?
3. What steps do you need to perform to sweep a design variable?
4. Which assistant provides information regarding the number of sweeps and corners simulation runs?
5. How can you disable all corners for a test?

Labs

Lab 13-1 Corners and Sweeps in ADE XL

Lab 13-2 Corners and Sweep in ADE XL with ampTest

Virtuoso Design Characterization and Modeling

Module 14

December 14, 2006

Module Objectives

In this module, you will

- Learn how to use Virtuoso® Design Characterization and Modeling (DCM) to create a Top-down Verilog®-A model of a large set of useful analog design blocks
- Use Virtuoso DCM to create a bottom-up calibrated Verilog-A model of a phase frequency detector
- Run a verification test using built-in measures of Virtuoso DCM for a selected design block and view the results against specifications

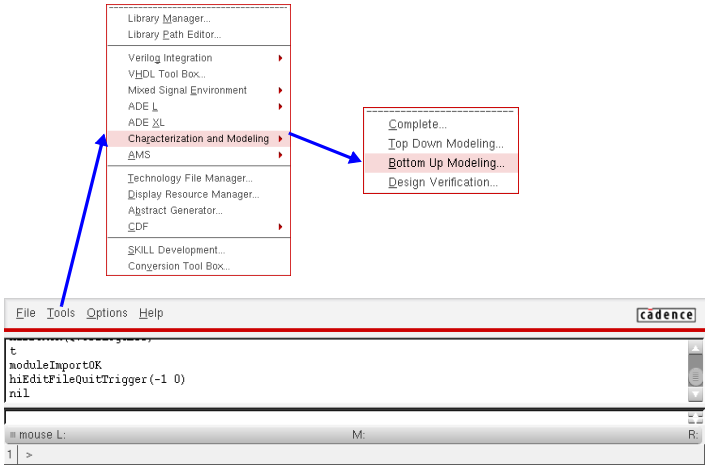
Acronym	Definition
DCM	Design Characterization and Modeling
HDL	Hardware Description Language used to describe digital circuits
Verilog	A commonly used HDL, often referred to as Verilog-D to differentiate it from Verilog-A
Verilog-A	Analog modeling syntax extension to the Verilog HDL language
VSDE	Virtuoso Specification-driven Environment
VCME	Virtuoso Characterization and Modeling Environment
Liberty	Calibrated Verilog models with a .lib extension
CIW	Command Interpreter Window, the top-level control window opened in Virtuoso
DFF	Delay Flip-Flop
PFD	Phase Frequency Detector
VCO	Voltage Controlled Oscillator

What Is Virtuoso DCM?

- Virtuoso® DCM is an ADE GXL component for characterizing and modeling designs.
- DCM is derived from VSDE, VCME, and Modelwriter.
- DCM creates behavioral models (Verilog-A, Verilog-D, Liberty)
  - Top-down: create a parameterized Verilog-A model to enable system simulation prior to detailed design.
  - Bottom-up: speed up system simulations after the detailed design is completed by creating a calibrated Verilog-A model using table models to match simulated performance, or calibrated Verilog-D models, or Liberty models.
- DCM runs a design verification set of simulations over conditions (corners) and against specifications.
- When to use models?
  - Top-down model: beginning of a project to fill in a block in a system with a simple model that lets you prove the system concept and interface specifications.
  - Bottom-up model: after the detailed design is done on a block, to replace it with a calibrated model that simulates faster.

Use Modes

- Starts by you choosing **CIW – Tools – Characterization and Modeling**
- Three basic modes:
  - Top-down
  - Bottom-up
  - Design verification
- Can do all modes (complete) on designs with compatible top-down and bottom-up models (DFF, PFD).
- The taps in the Virtuoso DCM form change, based on the mode chosen.



The Complete mode lets you set up any or all of top-down, bottom-up, or design verification using the same Function setup, as long as compatible models exist for top-down and bottom-up. (Currently that means DFF and PFD only.)

Top-Down Model Generation

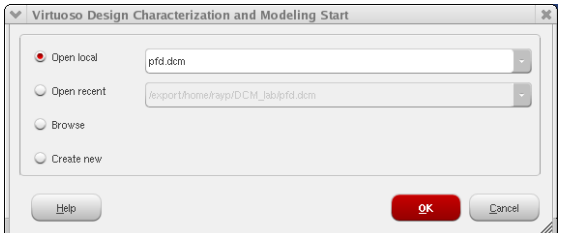
Creates Verilog-A models based on Modelwriter “canned” models:

Analog Models	**Switch with delay, ideal switch, switch**
Components	**Thyristor**
Continuous Time	**Abs. value amp, log. voltage amp, circ. integrator, integrator, diff. voltage amp, differentiator, voltage controlled delay, delay element, dead band amp, voltage amp**
Digital	**DFF**
Discrete Time	**Sample and hold**
Instruments	**Voltmeter**
Op-amp Models	**Clocked comparator, ideal/non-ideal op-amp, Tanh comparator**
PLL	**AMS voltage controlled oscillator, phase frequency detector**
PLL_Components	**Single block PLL, VCO**
Sources	**Multitone source**
System Level	**Voltage attenuator, voltage exponent, voltage controlled voltage amplifier, soft voltage clamp, peak detector, multiplier, controlled integrator**
Telecom	**AM demodulator, random bitstream, QPSK modulator, 16-QAM modulator, AM modulator**

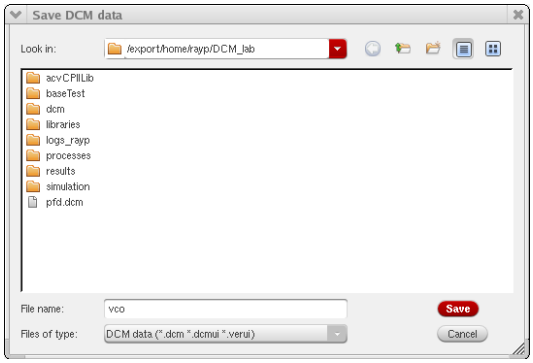
- Parameter values in model can be set prior to generation.
- If the symbol does not exist, it can be automatically created from the model pins.

Virtuoso DCM Configuration Files

- Choose a previous Virtuoso DCM configuration, or start a new <design>.dcm file.

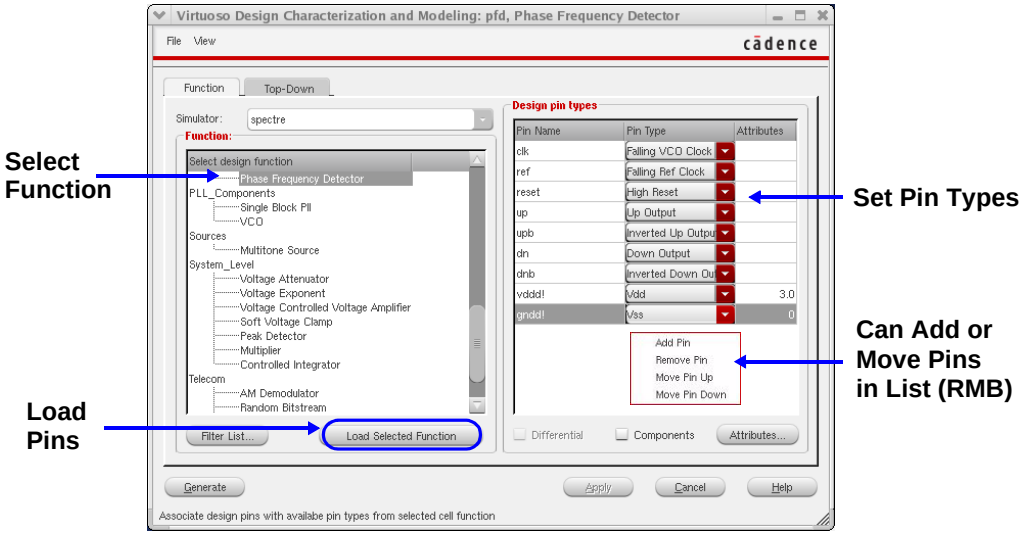


- When you click **Apply** on any of the Virtuoso DCM forms, the software checks the data, reports discrepancies, and prompts you to create a configuration file if there isn't one.



Top-Down Function Selection Tab

1. Select the function of the cell to model (opamp, vco, pfd, etc).
2. Click the **Load Selected Function** button to get pins from the design and pin types from the template. Pin names can be renamed.
3. Set the pin types (with a right-click) to match the design pin functionality (such as output, input, vdd, vss). For vdd and vss, set the attribute to be the high and low voltage levels. Do not use the Components option.

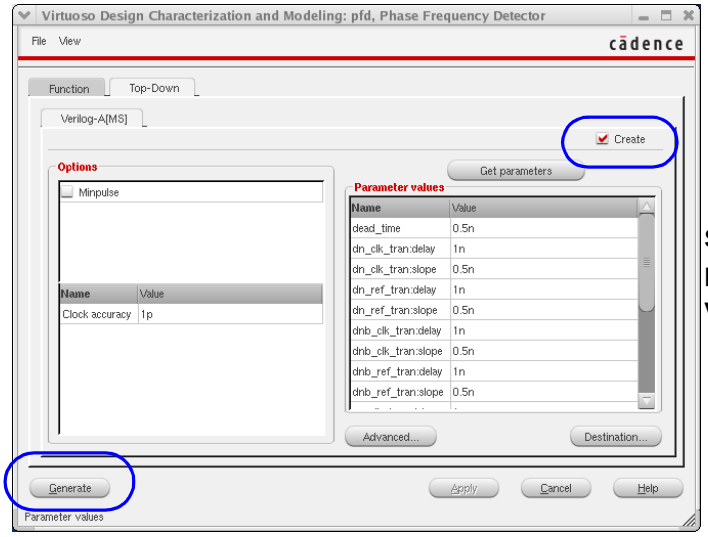


- Click **Apply** after filling out forms to save the configuration to the *.dcm* file.
- Double-clicking in the Attribute field will bring up the form to set the values for that pin.
- The Filter button reduces the list of available functions to those that are relevant to the mode (Top-down, Bottom-up, or Design Verification).

Top-Down Parameter Setting Tab

1. **Be sure to check the Create box!**
2. Set the default values for the parameters in the model here.
3. Set the destination for the model with the **Destination** button.
4. Click the **Generate** button to create the Verilog-A model.

`If the symbol does not exist, a Create Symbol form appears.`



Set parameter values.

Virtuoso Design Characterization and Modeling 14-15

- View the created model by choosing **View – Top Down – Generated model**.
- Click **Apply** to save any changes to the *.dcm* file, and then click **Generate**. Values changed in forms after generating the model will not be used until the next Generate.
- Click **Advanced** allows comments and include files to be added to the model header.

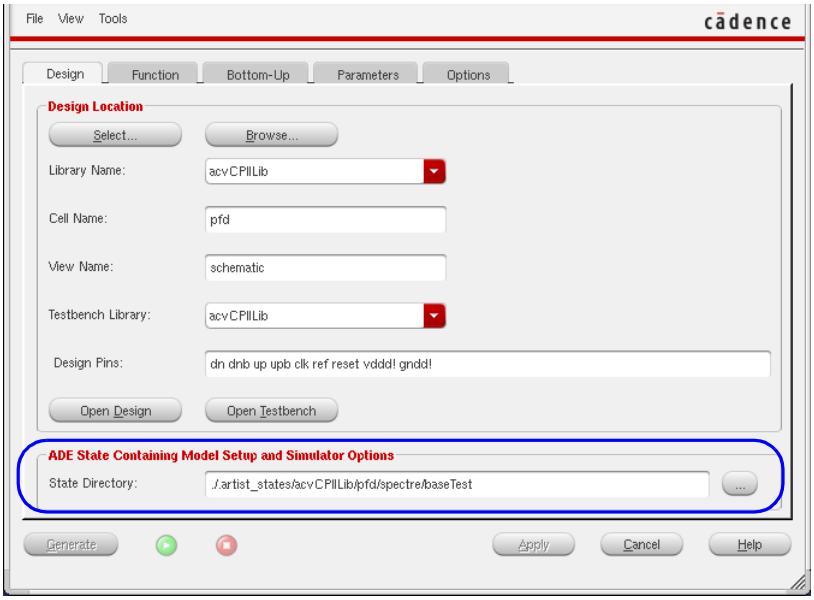
Bottom-Up Model Generation

- Bottom-up mode creates a calibrated Verilog-A model for a limited set of designs that exist as schematics or extracted views.
  - DFF (also Verilog-D and Liberty models)
  - Divider
  - VCO
  - PFD
  - Baseband Filter, Mixer, LNA or IQ-Mod/IQ-Demod
- This mode creates the test bench.
- The bottom-up mode uses ADE state file for simulation information, such as models.
- This mode runs selected simulation sweeps of the design to produce tables of behavior for use in models.
- You can also compare behavior of original design with the model (Verify).
- You can add additional models over time, or you can create your own using Open DCM, a protocol for creating model templates for use with DCM.

Open DCM is described in the *Virtuoso ADE GXL User Guide*, Appendix A.

Bottom-Up Design Tab

- Use the same design configuration file (.dcm) as top-down or create one for the design being modeled in the same way (select, browse, enter by hand).
- Add the path to the Base Test (Artist State). Bottom-up mode requires an ADE state file that includes the path to the model files.

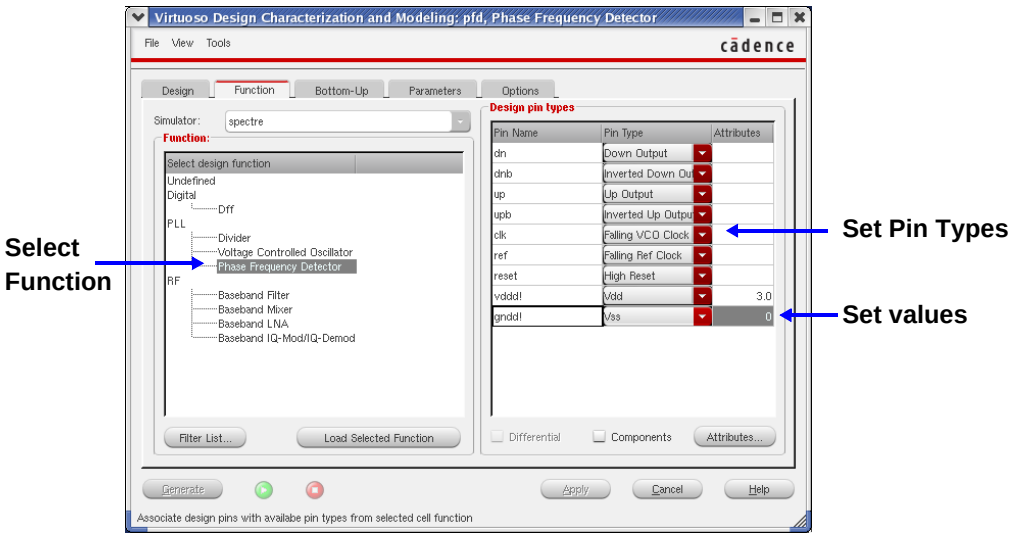


Browsing to find the state file brings up the *artist_states* selection form from ADE.

If the design includes inherited connections or global signals like vdd! or vss!, you need to select the design symbol from the schematic or browse to find the view rather than entering the pins by hand.

Bottom-Up Function Tab

1. Select a function to match the design from the short list.
2. Unless already loaded from the .dcm configuration file, load the function pins and pin types. If necessary, set the pin types to match the pin functions as explained for top-down.
3. Use the **Components** option to add sources and loads to the test bench if needed. Add values (attributes) for them.

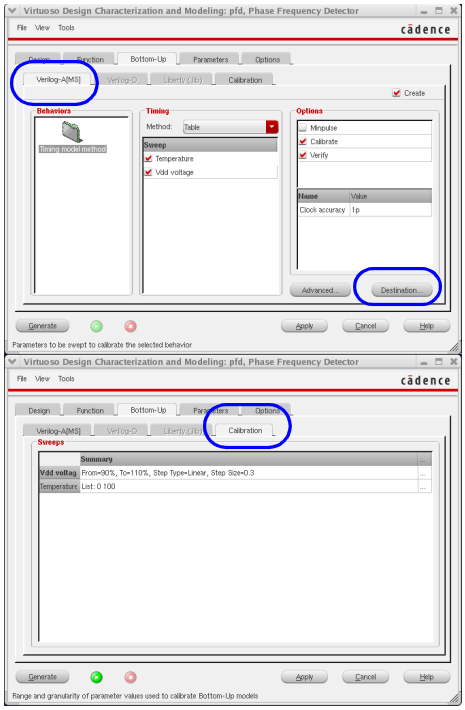


- More functions will be added over time. New model templates can be created using Open DCM.
- Don’t forget to click **Apply** after changing values.
- Choices for simulator are *spectre*, *ultrasim*, and *ams*.

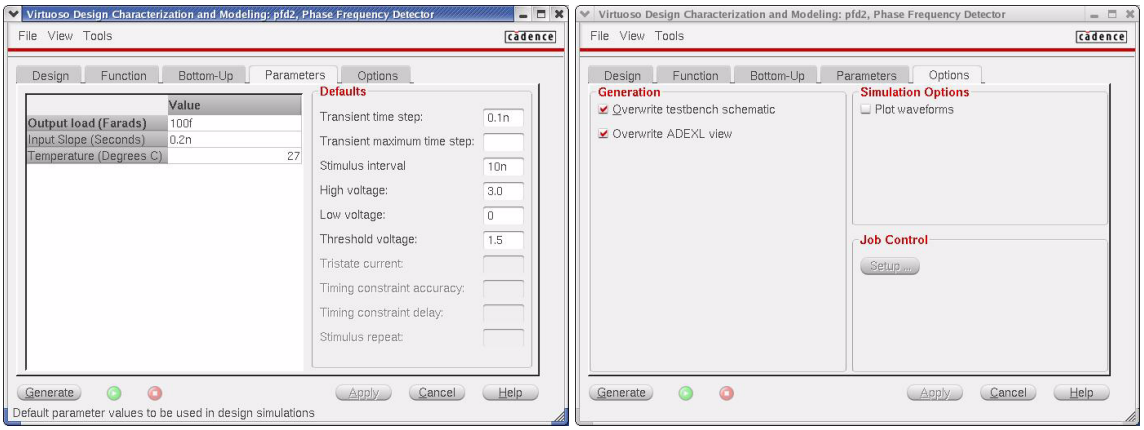
Bottom-Up Calibration Setup Tab

The table models used in the bottom-up model can be multidimensional, with temperature and supply voltage being the most commonly swept.

1. **Be sure to check the Create box!**
2. Check the desired sweeps (Temperature, Vdd voltage).
3. Check the options (Calibrate, Verify). Verify will compare the simulation results of the model with the original schematic or netlist.
4. Set the destination *lib.cell:view* for the calibrated model.
5. Set the step sizes for VDD and Temp on the Calibration tab.

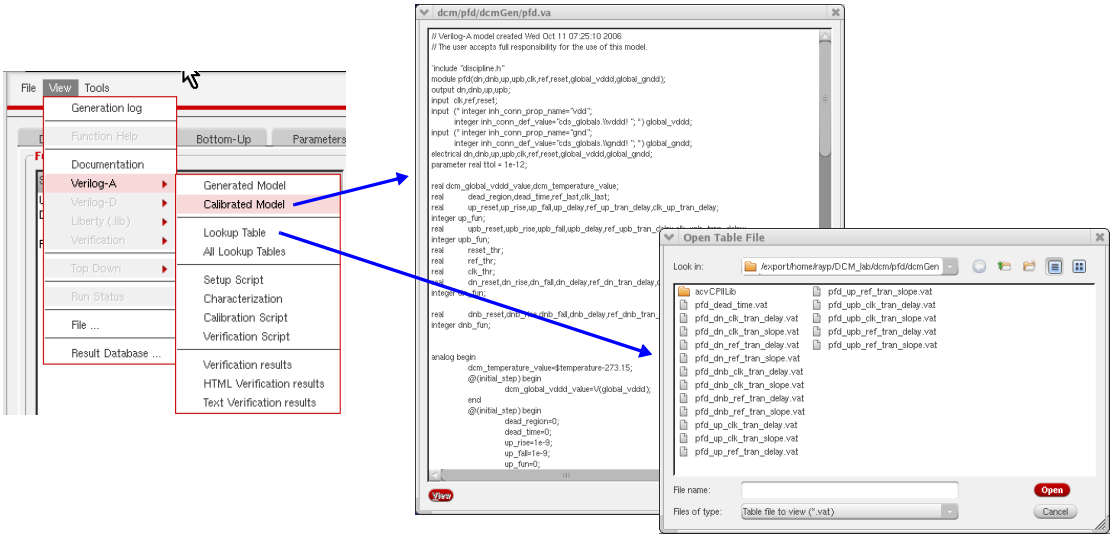


General default values and control options are set on these remaining tabs.

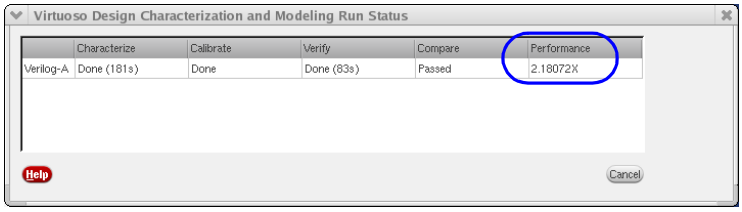


Bottom-Up Model Creation

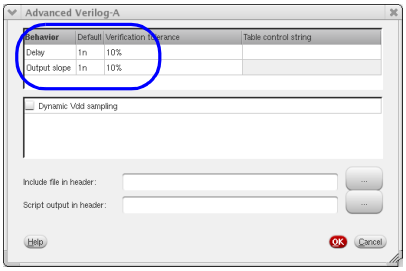
1. Click the **Generate** button to create the basic Verilog-A model with placeholders for table models based on simulation results.
2. Click the **Green** arrow key to begin the simulation runs. (Click on the Red arrow key to stop the simulation).
3. When the simulation ends, examine the results from the **View – Verilog-A** menu.



- The speed-up of the model over the original schematic or netlist simulation is shown in the status window.



- Percentage verification thresholds can be changed by clicking the **Advanced** button on the Bottom-up form.



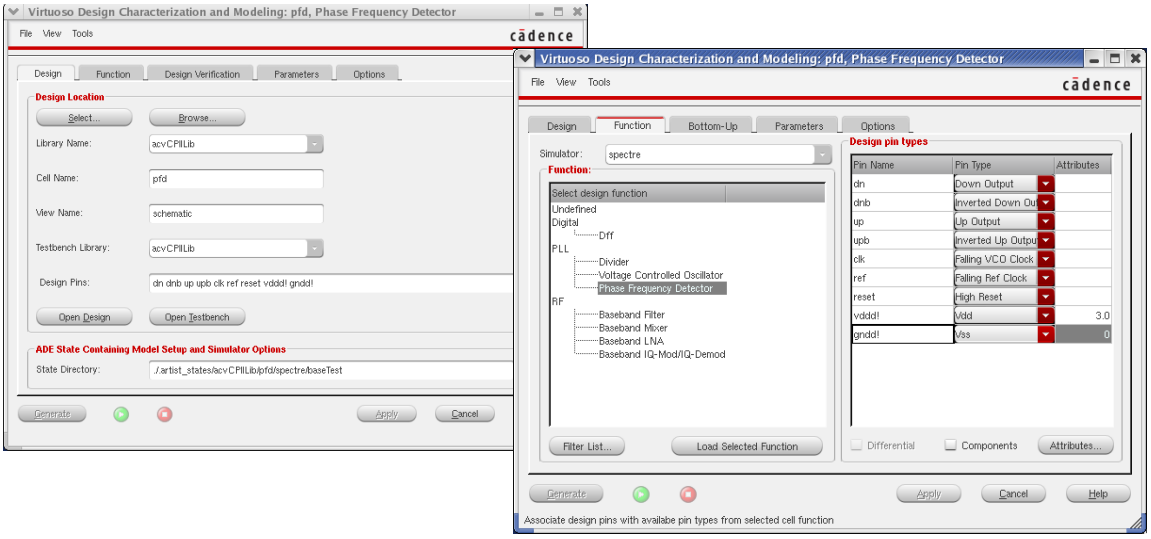
Design Verification

Design verification provides the following functions:

- Simulates the same limited set of designs over external factors, such as voltage, temperature, and load capacitance, and compares the results to the specification.
  - DFF
  - Divider
  - PFD
- Creates the test bench and applies selected measures to the results.
- Uses ADE state file for simulation information, such as models.
- Can output results to HTML.

Verification Design Tab

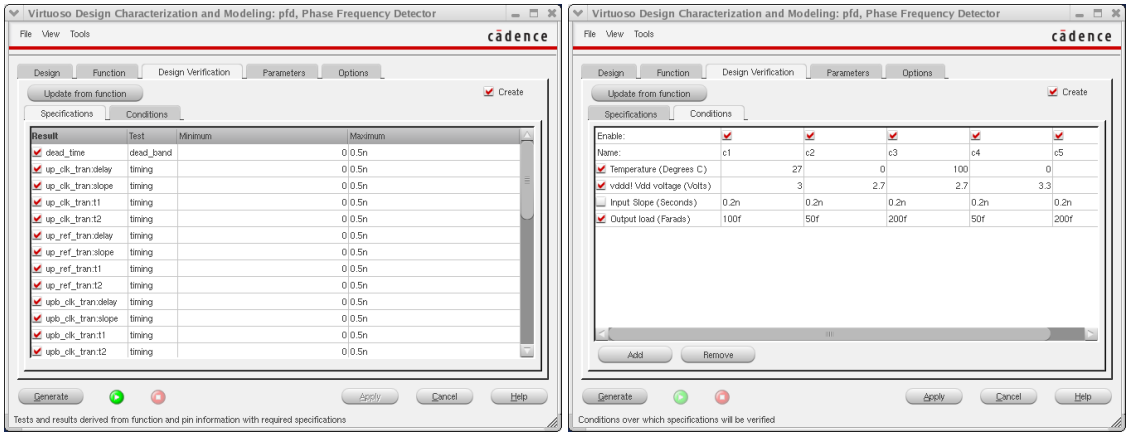
- This tab uses the same design configuration file (.dcm) as top-down or creates one for the design being modeled in the same way (select, browse, enter by hand).
- The path to the Base Test (Artist State) is also required to provide the path to the model files.
- The function setup is the same as for the bottom-up mode.



Don’t forget to click **Apply** after changing the settings.

Design Verification Tab

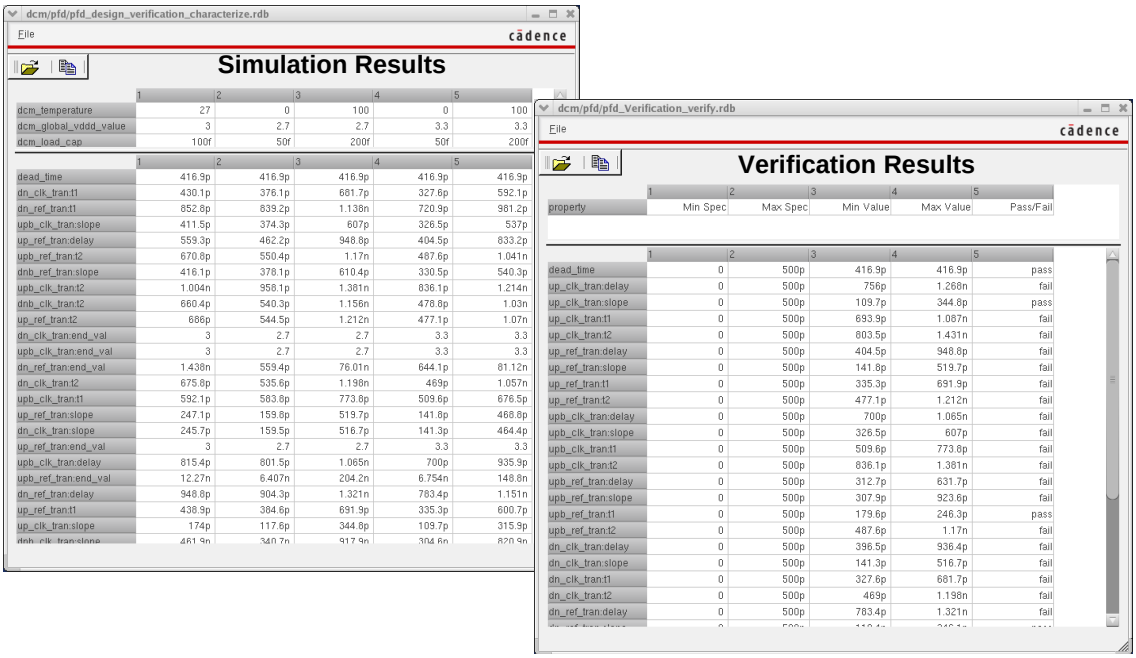
1. **Be sure to check the Create box!**
2. Set the specification limits that pertain to the model template on the Specifications tab. They can be enabled or disabled individually.
3. Set up the simulation corners to test on the Conditions tab. You need to enable each condition (c1, c2, etc) to test. Also, you need to enable each variable (Temperature, Voltage, Load capacitance) to be used.



- Don’t forget to click **Apply** after making changes to save them to the *.dcm* file.
- Don’t click **Generate** until all the form settings have been made.
- The Parameter and Options forms are the same as for the Bottom-Up mode.

Design Verification Results

1. Click **Generate** and then the Green arrow to start the simulations.
2. When the simulation ends, choose the results to examine from the **View – Verification** menu.



Quiz

1. To create a Verilog-A model for an opamp from a “canned” template, use:
   a: Top-down mode of Virtuoso DCM
   b: Bottom-up mode of Virtuoso DCM
   c: Verification mode of Virtuoso DCM
2. To generate a calibrated Verilog-A model an existing circuit by creating table models that reflect simulation performance, use:
   a: Top-down mode of Virtuoso DCM
   b: Bottom-up mode of Virtuoso DCM
   c: Verification mode of Virtuoso DCM
3. To check the performance of a selected circuit against specifications using prebuilt measures, use:
   a: Top-down mode of Virtuoso DCM
   b: Bottom-up mode of Virtuoso DCM
   c: Verification mode of Virtuoso DCM
4. A customer can write their own templates for use with Virtuoso DCM Bottom-up and Verification modes using:

   _____

Labs

Lab 14-1 Top-Down Modeling

Lab 14-2 Bottom-up Modeling

Lab 14-3 Design Verification

ADE GXL Parameterization and Optimization

Module 15

December 14, 2006

Module Objectives

In this module, you will

- Set up a parameterization flow in ADE GXL by setting matching devices, device properties, and values for optimization
- Set up the performance specifications that ADE GXL must meet during synthesis and optimization
- Learn when to use ADE GXL local and global optimization methodologies
- Access, display, and backannotate the results to the schematic

In addition to parameterizing a design for sizing and optimization using ADE variables, you can also use a parameterization flow in ADE GXL. To use this flow, you can parameterize a design and set specifications; then run a local or global optimization and backannotate the results of the optimization.

For a non-optimization flow, the main advantage of the parameter flow is the capability to make a virtual change to your schematic and see the effect of the change through simulation. The change is not registered on the schematic, and thus doesn't require schematic changes or check-and-save.

Introduction

ADE GXL parameterization and optimization flows let you address several challenges.

- Designer productivity: Compresses the time to market.
- IP reuse: Process migration often requires complete redesign for Analog, mixed-signal, and RF circuits.
- Parasitic closure: Electrical and physical design concerns must be addressed simultaneously to meet performance requirements.
- Yield Improvement: Failure to address yield early in the design cycle results in costly respins, and increased time to volume.

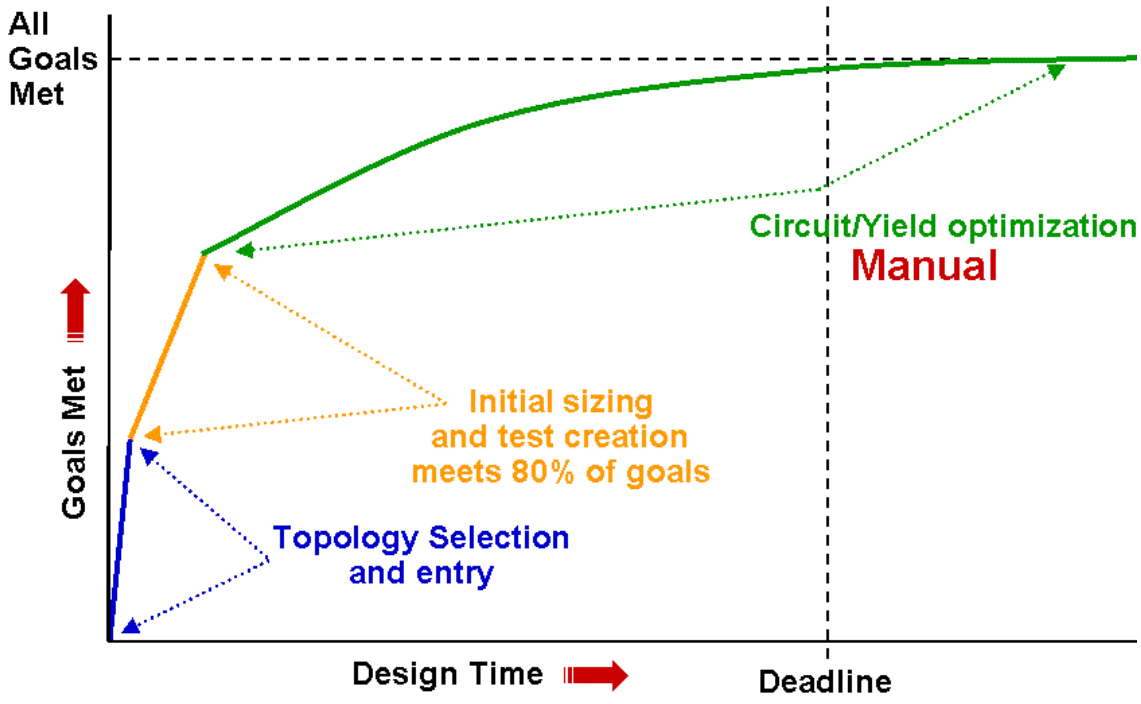
There are three key drivers in custom design as depicted here: economics, complexity, and physical effects.

First, economics drives designs toward smaller process geometries over time. If it is possible to integrate more functionality on a chip at a lower process technology, you can bet that the competition will be doing just that. In fact, many custom design teams are skipping a process node, moving from 0.35 micron or 0.25 micron down to 0.18 micron or 0.13 micron to integrate their designs with massive digital logic. That said, there is still a lot of custom design above 0.5 micron.

Each new process technology enables you to integrate exponentially more complexity on a single die. At the same time, exponentially more physical effects come into play. Both the number and the type of physical effects are increasing.

What is interesting is that these same drivers affect you no matter what process node you're currently at. Your current designs are much more complex and deal with many more physical effects than in the past. At the same time, your future designs will be exponentially more challenging.

The Basic Optimization Design Flow



ADE GXL Parameterization and Optimization 15-7

The basic design flow can be split into three sections.

1. First, is the topology selection that is necessary for the overall architecture. This is strongly dependent on the expertise of the designer and requires a lot of inventiveness.

2. Second, is the initial sizing, which might take place as the topology is entered. This phase is also the time when the test benches are developed and the performance requirements are measured against the goals. Often the design will meet nominal specifications, but not minimum or maximum specifications. You can use ADE GXL parameterization in this phase of the design to rapidly explore different topologies before going on to the last phase.

3. Third, is the optimization of the design to best fit the design to all the goals and over all the process and environment conditions.
   It is a very simulation dependent phase and can often take a significant amount of time. Occasionally in the interest of time, a circuit might be overdesigned to ensure that all specifications are met, but this might be at the expense of power dissipation or area for example.

   Adjusting the design to meet a goal for the lowest power can take quite a long time. ADE GXL Optimizer provides a benefit in speeding up this last phase of the design cycle and it is especially effective in optimizing the performance where a trade-off exists. The classic trade-off is the balance between speed and power and many sacrifice one in the interest of time.

ADE GXL Simulation-Based Technology

- Optimization can be used at any point in the design process.
  - Use for initial sizing to explore the whole design space for the final solution.
  - Use for finishing a partially sized design, working with the designer's knowledge.
  - Use to determine sizes that remain compliant with the PDK over all tests and corners.
- Optimization is circuit-type independent: linear or nonlinear, dc or rf.
  - Analog: Opamps, bandgaps, Comparators, GmC filters, PFDs
  - RF: LNAs, mixers, VCOs
  - Memory: Sense amps, memory cells
  - LCD: LCD Drivers
  - I/O: I/O cells

You can use the ADE GXL optimizer for multiple applications.

It can take an unsized design and find a valid nominal design as well as a nominal design and finish it over all corners.

ADE GXL Parameterization

Parameterize a given design for sizing and optimization:

- Matching devices and device properties
  - Matching all device properties
  - Matching specific device properties (most commonly used)
  - Ratio-matching device properties
- Defining values for optimization
  - Ranges and lists
  - Sweep variables

ADE GXL Parameterization

When you parameterize the design for optimization, you capture the critical device relationships in the circuit. For example, the input transistors of an opamp must be matched, and the transistors of a current mirror must be ratioed. To run optimization, you must define a range of legal values for optimization for each transistor that is the “master” in a matching relationship.

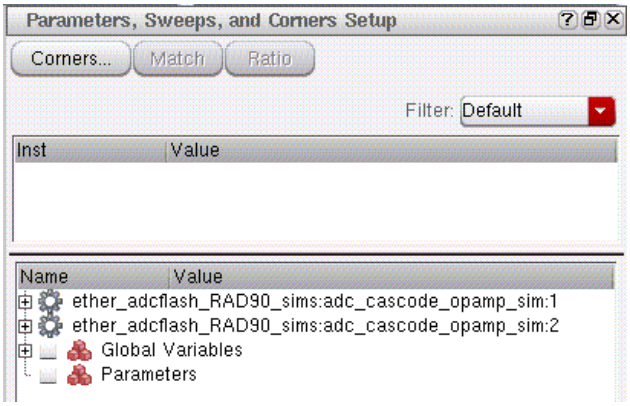
You can match a specific property of a device to another device property value, match all properties for a device to the properties of another device, or ratio-match devices properties.

Parameter ranges for optimization can be defined in one of two ways. You can set the ranges in the value field for the device parameter or a global variable (similar to a parameter sweep). This section covers the first use model.

Parameterizing a Design

- To begin parameterizing the design, launch the Parameters, Sweeps and Corners Assistant.
- From the CIW, choose **Launch – ADE GXL**.
- In the Workspace field, choose the Variables workspace.

  The Parameters, Sweeps, and Corner Setup Assistant appears.



Matching Options for Devices

Three options are available for matching:

- Match all device properties for a device
- Match only specific properties
- Ratio-match

After specifying device relationships, you can define legal values for device properties.

Parameterizing the Design by Matching Devices

To match devices, follow these steps:

1. Select the devices to match in the schematic.

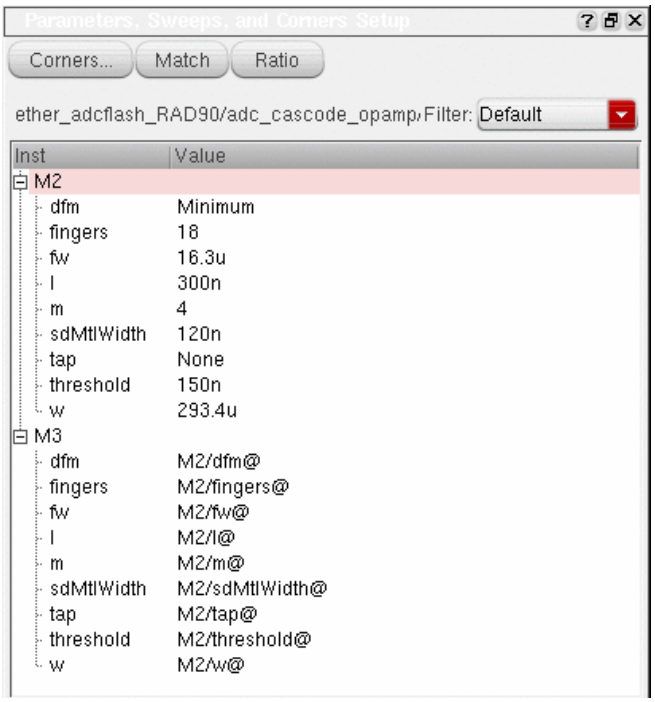
   The instance names are copied to the Parameters assistant.

2. In the Parameters assistant, select the device you want as the “master device.” The master device is the device that all the other selected devices will match.

3. Click **Match**.

   You can now modify any of these values to vary them during the optimization process.

The following figure shows the M3 device being matched to the M2 device, which is set as the master device:



Setting Matched Device Properties

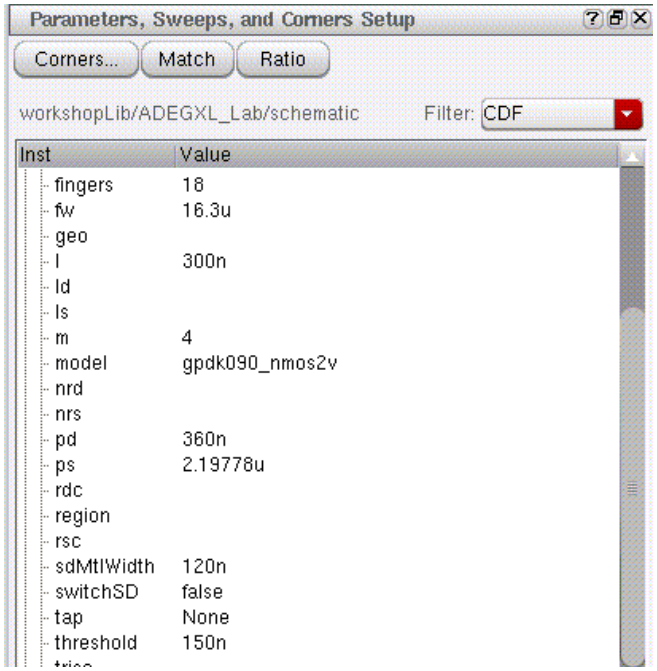
To match specific device properties, follow these steps:

1. In the schematic, select the devices for which you want to match properties.
2. The devices are listed in the Parameters assistant.
3. In the Parameters assistant, ensure *Default* is chosen from the *Filter* field.
4. Select the **+** next to a device to edit.

   The list of properties for the device appears, as well as the schematic value of each property.
5. Right-click and select **Toggle View**.
6. The grouping is now based on property name, rather than device name.
7. Select the property to match.
8. Click **Match**.

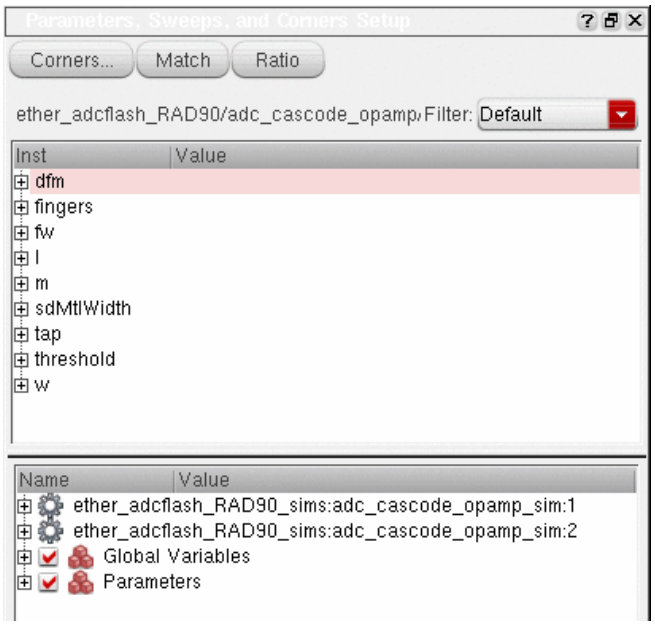
You can now modify any of these values to vary them during the optimization process.

Using the default filter is not required in the Parameters, Sweep and Corners assistant (as opposed to CDF) but it makes it cleaner.



Using Toggle View to Examine Properties

The following figure shows the toggle view for a specific device:



There is an alternative method for matching and/or ratioing device properties, which is similar to matching the entire device, but instead you select just the specific property name in the top part of the Parameters, Sweeps and Corners assistant and click "Match" (or "Ratio").

You'll use this approach in the ADE GXL parameterization/optimization lab.

Parameterizing Devices by Ratio-Matching

To ratio-match specific device properties, follow these steps:

1. In the schematic, select the devices for which you want to match properties.

   The devices are listed in the Parameters assistant.

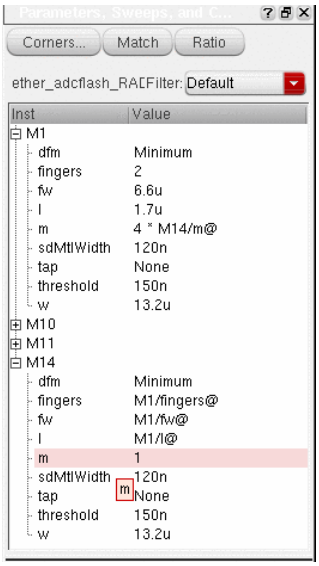
2. In the Parameters assistant, choose **Default** from the Filter field.
3. Select the **+** next to the device to edit.

   The list of properties for the device appears, as well as the schematic value of each property.

4. If necessary, right-click and select **Toggle View** to base the sorting on the property name, rather than on the device name.
5. Select the property to match.
6. Click **fwsf**.

ADE GXL automatically ratios the properties based on the lowest value.

This window displays the properties being ratioed based on the lowest valued device.



You can now modify any of these values to vary them during the optimization process.

Parameterization: Defining Values for Optimization

To define a legal range of values for devices, follow these steps:

1. Select the devices for which you want to specify values.

   The devices are listed in the Parameters assistant.

2. In the Parameters assistant, choose **Default** from the Filter cyclic field.
3. Select the **+** next to a device to edit.

   The list of properties for the device appears, as well as the schematic value of each property.

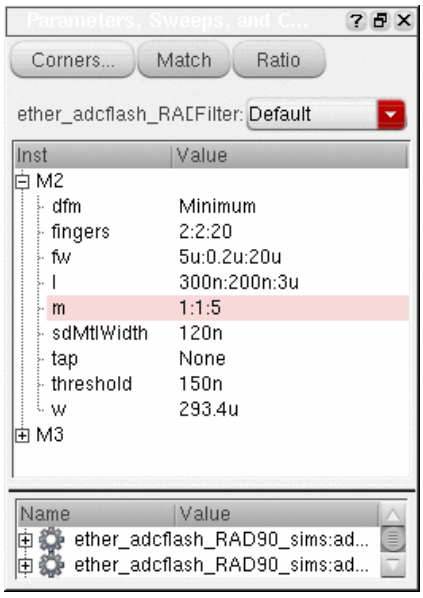
4. Double-click the Value column for the property to modify.
5. In the Value field, enter a range or list of values.

Specify ranges in the format **MIN:STEP:MAX**. You can separate a list of values with either commas or spaces.

In this flow, select a device then add the range to the property. The parameter then appears below with the associated range.

Parameters: Defining Values

As shown here, the list of properties for a given device is displayed with the associated range.



In the example above, you select a device then add the range to the property. The parameter appears below with the associated range.

If you have already done matching or ratio, then the parameter already exists below and this is likely the place where you will input the range because it doesn't require selection from the schematic.

Even if you don't perform any matching or ratio, you can create a parameter by selecting in the Value field by pressing Return or by using the right mouse button on the device name or value in the upper field. This lets you specify a range in the lower field.

Therefore, you can enter ranges using those two methods, before the parameter is created or after.

Setting Up Specifications

- Specifications are design goals that ADE GXL must meet during optimization. These specifications are used as benchmarks for candidate circuit quality during optimization.
- For each specification, specify an objective, or performance constraint, and then specify a target value for that goal.
- There are five types of specifications:
  - open-ended
  - close-ended
  - range
  - tolerance
  - info

When you set specifications on outputs, you establish the performance specifications that ADE GXL must meet during synthesis. These specifications are used as benchmarks for candidate circuit quality during optimization.

If you run a simulation prior to setting up specifications, ADE GXL displays whether the output passes the specification as you add specifications.

An open-ended specification is when you choose either minimize or maximize, then specify an ideal value. For an open-ended specification, the algorithm tries to continually improve upon the ideal value. So, if the ideal value for a given specification is to maximize beyond 1e+8, 3e+8 is better than 2e+8.

A close-ended specification is when you specify a target value, then specify that the resulting specification must be greater than or less than that target value. For a close-ended specification, an ADE GXL algorithm attempts to meet the target value, and exceeding the target is no better than meeting the target. So, if the target value for a specification is >1e+8, then 3e+8 is no better than 2e+8.

You can also specify a range specification, in which you specify both an upper and a lower boundary for the target value.

For a tolerance specification, you specify a target value and an allowable percentage deviation.

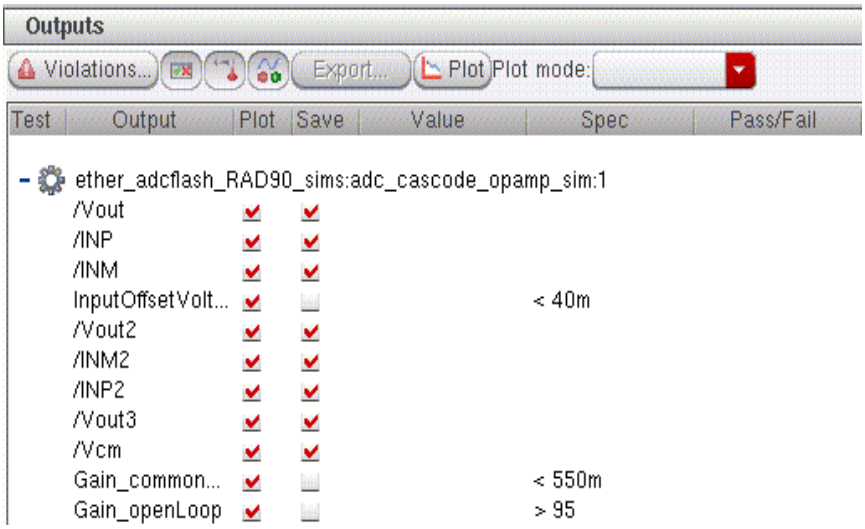
You can specify an info specification when you want to see the value of a measurement, but don’t want that goal to affect sizing.

Setting Up Specifications (continued)

To set up specifications

- In the Workspace field, choose the Outputs workspace.

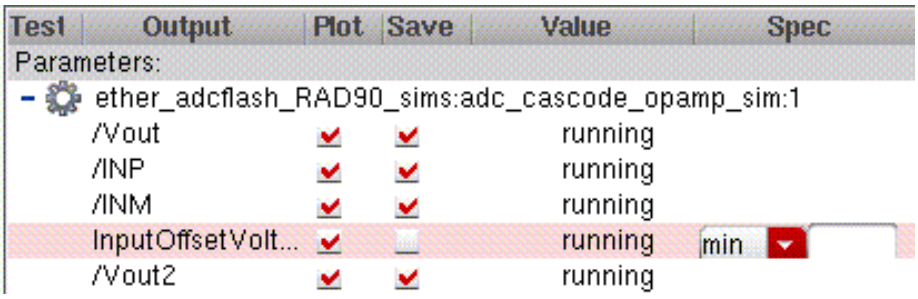
  The Outputs assistant appears.



Setting Up Specifications (continued)

- Select the output for which you want to set a specification.
- Select the Spec column for that row.

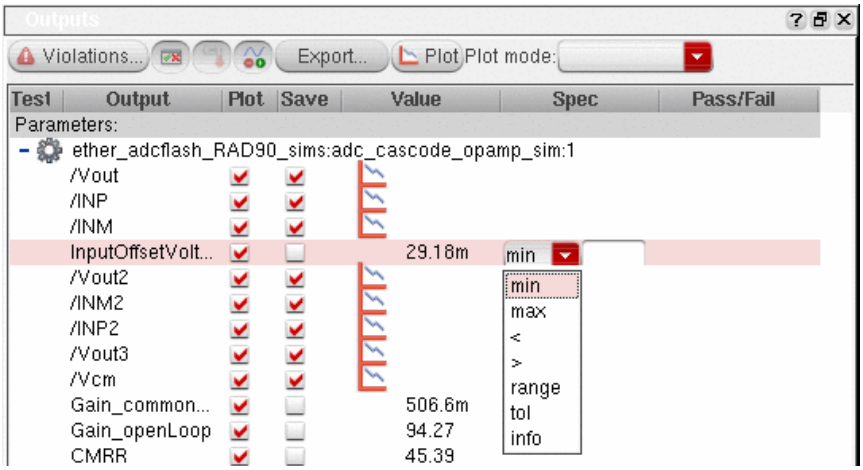
A cyclic field of objectives appears.



A single click just selects the entire row. Another single click is required to show the cyclic field.

Setting Up Specifications (continued)

- Choose an objective from the cyclic field (open-ended, close-ended, range, tolerance, or info).
- Select the field next to the cyclic field.
- Enter a target value in the field.



- Repeat the previous steps for all outputs that you want to set specifications for.

Regarding the specification target value, ADE GXL does not allow a target value equal to zero. For example, even if you assume that the ideal value is zero and you want to minimize the specification accordingly, setting zero as a target value is not supported. In this case, a possible work-around is to use a range specification centered around zero.

Local or Global Optimization

You have two options for optimization:

- Local optimization
  - Runs the schematic design around a specified design point.
  - Deterministically improves that design.
- Global optimization
  - Does not require a starting point.
  - Efficiently searches over all of the variables defined to find a good solution for your design.

You have two options for optimization: local or global.

Local optimization runs the schematic design, then deterministically improves that design. Local optimization is useful if:

- You have a starting design that you want to improve.
- A design meets specifications at the nominal corner but not across all corners.
- Models in a PDK have changed just enough to cause the design to no longer meet all specifications.

Note that local optimization only searches the design space around the specified point, so ADE GXL might not locate the best point possible to meet the design specifications. In addition, local optimization does not guarantee small perturbations to your variable values.

Unlike local optimization, global optimization does not require a starting point. It efficiently searches over all of the variables defined to find a good solution for your design.

In addition, because local optimization only searches the design space around the specified starting point, the tool might miss an acceptable solution, even if it does exist in the design space. In that case, you might want to try global optimization instead.

Local Optimization Options

- Two options are available for a local optimization effort:
  - Fine
    The type fine searches in a fine grid around the starting point.
  - Coarse
    The type *coarse* searches in bigger steps, but it can miss a local minimum that might have been found by the *fine* option.
- Use the fine option if your starting point is already close. (For example, use it in a case where only a few specifications are not met).
- Use the coarse option if your time is limited and the starting point is not as close. With the coarse option, ADE GXL runs more points, but it is faster on multiple machines than when running with the fine option because of parallel points.

Two options are available for local optimization: *fine* and *coarse* (in previous versions of the NeoCircuit software, a combination of the two was also available as a third choice). The type *fine* searches in a fine grid around the starting point. The type *coarse* searches in bigger steps, but it can miss a local minimum that might have been found by the fine option.

Use the *fine* option if your starting point is already close. With this type, the ADE GXL algorithm finds the local minimum around the selected point, but it might also quit once that local minimum is found and then miss a more optimal solution in the local design space. Because it only runs one point at a time, the ADE GXL local optimizer is slow with this option, and it runs the least number of points on average.

Use the *coarse* option if your time is limited and the starting point is not as close.With this option, the ADE GXL algorithm is less likely to quit prematurely, but it also might miss a local minimum that the fine option would have found. With the coarse option, the ADE GXL optimizer runs more points, but it is faster on multiple machines than when running with the fine option because of parallel points.

Setting Local Optimization

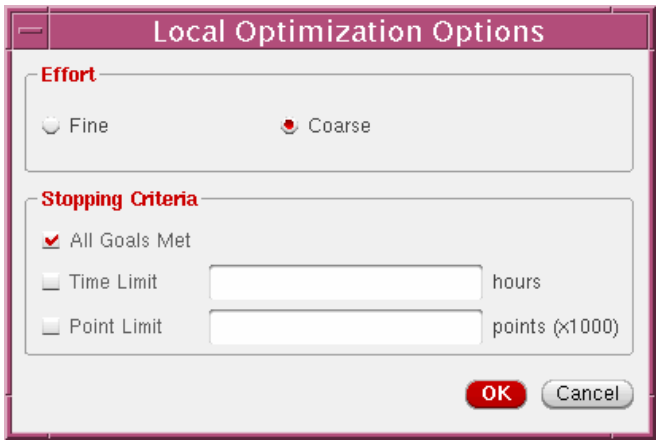
To access the ADE GXL local optimizer, follow these steps:

- From the Workspace cyclic field, choose the Run workspace.

  The Run assistant appears.

- From the Mode cyclic field, choose **Local Optimization**.

  The Local Optimization Options form appears.



When setting the effort to *Fine*, ADE GXL local optimizer uses an algorithm based on Brent-Powell. It adaptively adjusts the step size used by the algorithm for each of the variables.

This is a good algorithm to use when having a starting point that is suboptimal, trying to refine the design to improve performance or making changes like new models or specifications.

When setting the effort to *Coarse*, the ADE GXL local optimizer uses the VSDE Explore algorithm based on Hooke-Jeeves (for more information on this algorithm, you can refer to the VSDE optimization guide).

When doing an optimization, the Hooke Jeeves algorithm submits as many design points as there are variables for each step. Brent-Powell, at the other end, does one variable and one value at a time. So as a result, the Coarse algorithm typically runs more points than the Fine one.

The ADE GXL local optimizer uses the existing schematic design as the starting point.

Setting Local Optimization (continued)

- Specify the effort level by selecting one of the following buttons:
  - Fine
  - Coarse
- If desired, select criteria for the length of time local optimization needs to run.
  - To run only until all goals are met, select the **All Goals Met** check box.
  - To set a time limit for the run, select the **Time Limit** check box and enter a value in hours.
  - To set a limit on the number of points run, select the **Point Limit** check box and enter the number of points.The number of points is calculated x1000.
- Click the **OK** button.

As a side note, when choosing the All Goals Met option, maximize effectively becomes >, and minimize effectively becomes <.

Setting Global Optimization

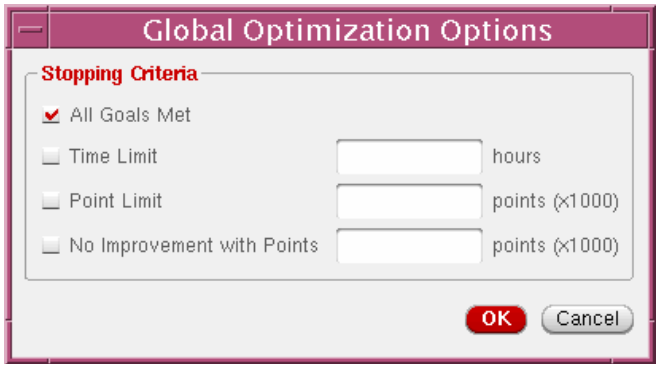
To run a global optimization, follow these steps.

- In the Workspace cyclic field, choose the Run workspace.

  The Run assistant appears.

- In the Mode cyclic field, choose **Global Optimization**.
- Click **...** to specify optimization options.

  The Global Optimization Options form appears.



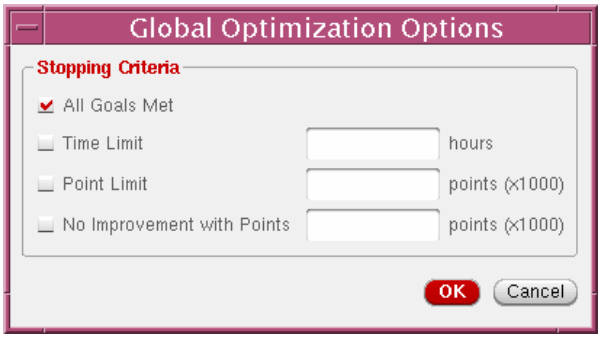
ADE GXL Global optimizer uses a Population-based stochastic algorithm. It samples and explores multiple parts of the design space simultaneously to find an optimal solution

This algorithm supports large design spaces (40+ variables) and finds a solution relatively quickly. The design space size can indeed be on the order of 10^50 and the algorithm finds a solution in a few thousand simulations

Previously, the NeoCircuit Global Optimizer supported three different efforts: Low, Recommended, and High. Because many designers were confused by the effort level and because most of the time people used the *Recommended* effort, there is now only one effort level in ADE GXL Global Optimizer: *Recommended* from the NeoCircuit product.

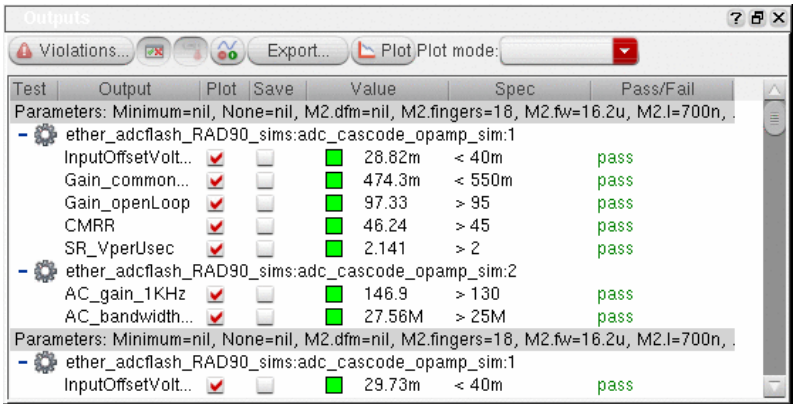
Setting Global Optimization Limits

- You can select criteria for the length of time local optimization needs to run.
  - To run only until all goals are met, select the **All Goals Met** check box.
  - To set a time limit for the run, select the **Time Limit** check box and enter a value in hours.
  - To set a limit in the number of points run, select the **Point Limit** check box and enter the number of points.
  - To stop sizing when no improvement is seen, select **No Improvement with Points** and enter the number of points.



Accessing Results

- After clicking **OK** on the Global or Local Optimization form, ADE GXL starts optimizing the design.
- The Outputs window displays the 10 best points as it progresses. The best design point is at the top.
- The optimization stops when it meets all specifications. (Everything is green in the Outputs assistant).



In the ADE GXL environment, when you run optimization, the Output window displays the ten best points by default. However, you have the option to specify the maximum number of best points to appear in the Outputs pane by specifying an environment variable *numberOfBestPointsToView* in either the *.cdsenv* or *.cdsinit* file as described below:

In the *.cdsenv* file, enter:

```
adexl.gui numberOfBestPointsToView int 10
```

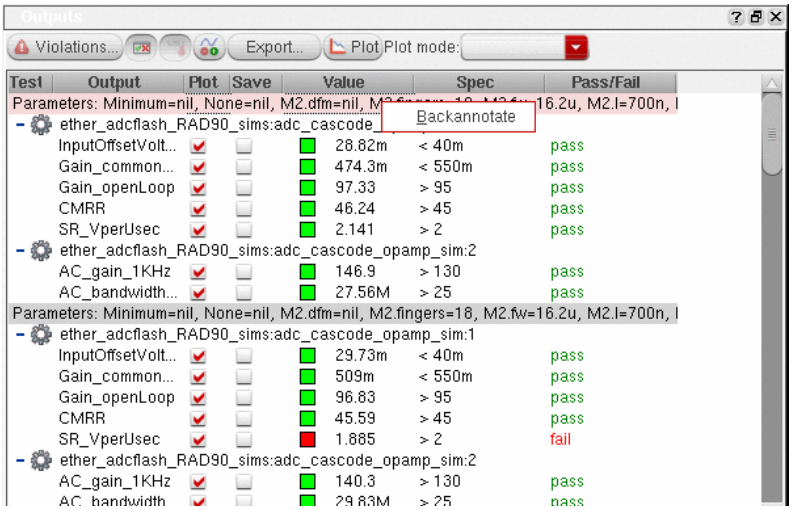
In the *.cdsinit* file, enter:

```
envSetVal("adexl.gui" "numberOfBestPointsToView" 'int 10)
```

Any integer value is a valid input here.

Backannotate the Results

The final step is to backannotate the design to the schematic by right-clicking on the Parameters line in the Outputs window and selecting **Backannotate**.



ADE GXL Sampling Algorithm

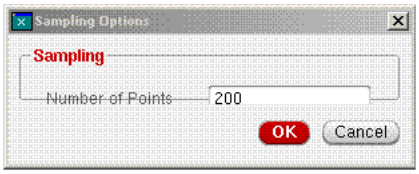
To access the ADE GXL sampling algorithm, follow these steps:

- In the Workspace cyclic field, choose the Run workspace.

  The Run assistant appears.

- From the Mode field, choose **Sampling**.
- Click **...** to specify Sampling options.

  The Sampling Options form appears.



ADE GXL sampling mode is based on Hammersley algorithm. This application samples your design space by choosing multiple points in your variable space and gets the design point that best matches with your specifications.

For example, if a given design contains 10 variables, each with 10 values, the ADE GXL sampling algorithm chooses points in the 10 variables space and runs selected simulations instead of looking at a combination of points of 10^10. It basically samples the design space.

The number of simulations run is directly controlled by the number of points specified in the sampling option: even if 10 variables with ten swept values are specified, the ADE GXL sampling algorithm runs 50 combinations if the number of points is set to 50.

The ADE GXL sampling algorithm lists all the points with the best one on the top (according to specifications provided). By comparison, the ADE GXL optimizer provides by default only the 10 best points with the best one at the top.

As such, it can be used to provide a good starting point for the local optimizer. To use the best point found by ADE GXL sampling as a starting point for the ADE GXL local optimizer, you just need to backannotate the result to the schematic and launch the local optimizer.

Parameterization Tips

While parameterizing your design, you can’t currently undo any wrong/incorrect mismatch/ratio correlations and need to delete all the incorrect devices/parameters relationships.

Quiz

- Which type of mismatch definitions can you perform in a parameterization flow?
- Which optimizer can you use in ADE GXL? Which options are available?
- How many points do you get by default when running ADE GXL optimizer?

- You can define three types of matching:
  - Match all device properties for a device
  - Match only specific properties
  - Ratio-match
- You can start a local and global optimizer. For the ADE GXL local optimizer, two efforts are available, *coarse* and *fine*.
- By default, the output pane will display the ten best points found during optimization.

Lab

Lab 15-1 Optimizing a Design in ADE GXL

Using Virtuoso Parasitic Estimation and Analysis with ADE GXL

Module 16

December 14, 2006

Module Objectives

In this module, you will

- Use the estimated parasitic flow in ADE GXL to improve simulation accuracy
- Refine an extracted parasitic view using filters for selective parasitic simulation
- Compare estimated and extracted parasitics and examine the differences

Acronym	Definition
MSPS	Mixed-Signal Parasitic Simulation
PEA	(Virtuoso) Parasitic Estimation and Analysis

Introduction

- Before IC 6.1.0, mixed-signal parasitic simulation (MSPS) concentrated on postlayout parasitic simulation and debugging, supporting the following features:
  - Out-of-context probing for net parasitics
  - Refinement of extracted views to exclude parasitics
  - Parasitic backannotation
- Starting with IC 6.1.0, Virtuoso® Parasitic Estimation and Analysis (PEA) has replaced MSPS and supports:
  - New prelayout parasitic estimation flow
  - Enhanced postlayout parasitic simulation flow
  - New parasitic compare flow
  - Tight integration into the ADE GXL environment

Availability of Virtuoso PEA Features in IC 6.1.0

The following table shows which parasitic simulation features are available in each application that parasitic simulation can be started from:

Feature	Virtuoso SE L and XL	ADE L and XL	ADE GXL
Show parasitics	Yes	Yes	Yes
Report parasitics	Yes	Yes	Yes
Out of context probing	Yes	Yes	Yes
Refine extracted view	Yes	Yes	Yes
Probe instance design/nets	Yes	Yes	Yes
Parasitic filters using the Constraints Manager	No	No	Yes
Specify R estimates	No	No	Yes
Specify coupled C estimates	No	No	Yes
Specify decoupled C estimates	No	No	Yes
Browse estimates with the Constraints Manager	No	No	Yes
Build estimated schematic view	No	No	Yes
Compare estimates with extracted view	No	No	Yes

New In ADE GXL

- The new functionality available in ADE GXL includes:
  - Parasitic Constraints
  - Parasitic assistants
    - Parasitic Estimates assistant
    - Parasitic Filters assistant
  - Parasitic workspaces for the 3 Virtuoso PEA flows
    - Parasitics-Estimated
    - Parasitics-Extracted
    - Parasitics-Compare
  - Parasitic Estimates Flow
  - Compare Parasitic Flow
- Tight integration lets you easily make changes to the parasitic resimulation setup and run simulations in ADE GXL.

Parasitic constraints are used to set parasitic estimates. Constraints are created using a modified version of the *Constraint Manager* assistant pane, the *Parasitic Estimates (Constraint Manager)* assistant.

When you access ADE GXL, you can use two related parasitic simulation assistant panes:

- The *Parasitic Estimates (Constraint Manager)* assistant
  - You can set parasitic estimates using constraints created by this assistant.
- The *Parasitic Filters* assistant
  - You can add or remove parasitics from the extracted view using this assistant.

ADE GXL provides 3 workspaces.

- Parasitics-Estimated workspace is used to aid the specification of the parasitic estimates. It includes 3 additional panes:
  - “Parasitic Estimates”, “Outputs,” and “Run” assistants
- Parasitics-Extracted is used to aid generation of the extracted (refined) parasitics. It includes 3 additional panes:
  - *Parasitic Filters*, *Outputs*, and *Run* assistants
- Parasitics-Compare is used to aid comparing the estimated parasitics against the actual layout parasitics. It includes 3 additional panes
  - *Parasitic Estimates*, *Parasitic Filters*, and *data* assistants

Virtuoso PEA Use Models

- Estimated Flow
  - Define parasitic estimates on critical nets and simulate circuits.
  - Parameterize parasitic estimates on critical nets and sweep to see the impact on circuit performance.
  - Optimize design over parasitic corners.
    - Specify the worst case parasitic estimates and produce robust designs.
- Extracted Flow
  - Filter out parasitics on noncritical nets and view impact on circuit performance.
  - Backannotate effective R and C on critical nets to trace problematic nets in postlayout.
- Compare Flow
  - Compare the differences between estimated parasitics and actual parasitics.
  - Easily discover problematic nets in the layout and
    - Ask the layout engineer to retune layout for problem nets
    - Resize or reoptimize circuit prelayout after changing estimated parasitics based on comparison report

Parasitic Estimates Flow

- The parasitic estimates flow lets you run estimated parasitic simulations to ensure that your design is robust across parasitics on critical nets before laying out the design.
- You can set parasitic estimates by using constraints that are created with a modified version of the *Constraint Manager* assistant pane, the *Parasitic Estimates (Constraint Manager)* assistant.
  - Estimates for R, Coupled C, and Decoupled C are supported.
  - Estimates are entered for various nets in a design using the new constraint management system.
  - All constraints are disabled by default.
    You can selectively enable/disable estimates.
- After entering constraints, you can create an estimated schematic view with the *Build Estimated Schematic* form.
- You can then use the *estimated* view for prelayout estimated parasitic simulation.

Using Virtuoso Parasitic Estimation and Analysis with ADE GXL 16-13

In Virtuoso® ADE GXL, the estimated parasitic flow uses parasitic values specified through *estimates*. These estimated values are stored in parasitic estimate cellviews separate from, but still associated with, the schematic view.

After layout, parasitic simulation can compare these parasitic estimates to the actual parasitic values, highlighting any parasitics that exceed their estimated values or differ from them by more than optionally specified tolerances. To achieve this, you can control the comparison method, which treats estimates as either *limits* or *targets*, with a specified tolerance expressed as a percentage.

Estimated parasitics, through the performance of a presimulation, will help ensure that your design functions correctly with specific estimated parasitic values. After laying out the design, you can compare the actual extracted parasitic values to the estimates.

The parasitic estimates use a star-shaped model of each net. Parasitic resistances are associated with the leaf instance terminals of the net. Coupled and decoupled capacitance associated with the net as a whole is connected to the center of the star.

The initial values of the *R*, *Coupled C*, and *Decoupled C* fields can be controlled by variables in the *.cdsenv* file, as in this example:

```
msps.estimates defaultR string "20a"
msps.estimates defaultCC string "2a"
msps.estimates defaultDC string "3a"
```

Steps of the Parasitic Estimates Flow

1. Open schematic or config view in ADE GXL.
2. Open and Fill out the Setup Parasitic Estimates form.
3. Select net(s) to create parasitic estimates (constraints) for.
4. Select **Create Estimated Resistance** or **Create Estimated Capacitance** from the Parasitic Estimates (Constraint Manager) assistant.
5. After entering all constraints, create an *estimated* schematic view with the form.
6. Use the Hierarchy Editor to select the *estimated* view for simulation.
7. Run estimated parasitic simulation.
8. Run out-of-context probing.

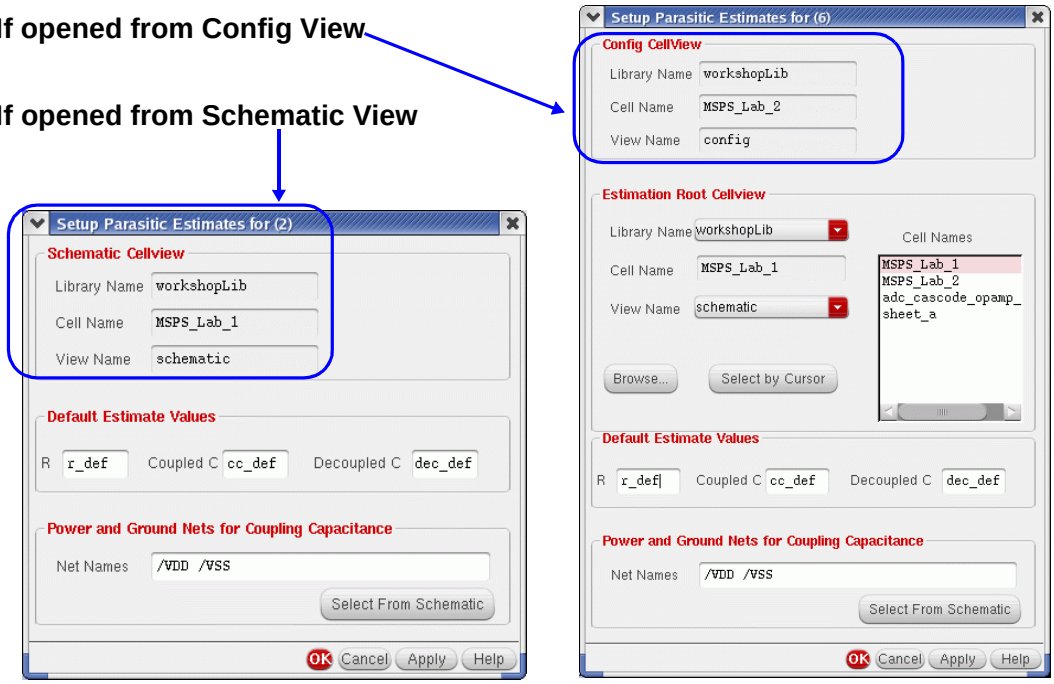
For details of keyboard shortcuts that you can use with parasitic simulation, see the *dfII/samples/local/mspsBindKeys.il* file in the Cadence installation hierarchy.

Setting Up Parasitic Estimates Flow

Setup the default parasitic values by choosing **Parasitics—Estimated —Setup**.

If opened from Config View

If opened from Schematic View



Note: When setup information has been added, it is preserved with the cell.

Schematic/Config Cellview: These values are auto-filled.

Estimation Root Cellview: The default estimation root will be the top-level cell. You must change this to point to the DUT before creating estimates.

Default Estimate Values: This section sets the parasitic *Default Estimate Values* for *R*, *CoupledC*, and *DecoupledC*. They can be a numerical constant or an algebraic expression that contains ADE variables.

Power and Ground Nets for Coupling Capacitance: Power and ground nets **must** be specified for coupled/decoupled parasitics reporting to work correctly. If no power and ground nets are specified, all capacitors are considered to be coupled capacitors.

Note: When you select power and ground nets for the capacitance report from a configuration, these selections must be made out of context. That is, you need to make your selections while in the schematic view of a block that is bound to an *extracted* view of a configuration.

- *Net Names* identifies those power and ground nets that you want to use when differentiating between coupled and decoupled nets. A slash, /, at the beginning of each net name is required for coupled and decoupled parasitic reporting to work correctly.

- *Select from Schematic* selects the power and ground nets directly from the schematic rather than physically typing them in. You can also use the *Navigator* assistant.

Important

Values entered on this form are saved as schematic properties. To remove them completely, open the schematic properties from the Library Manager, and delete *_mspsSetup*.

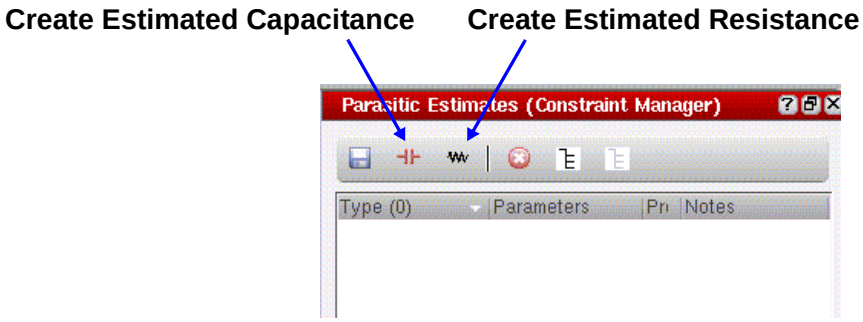
Selecting Nets

Select net(s) that you want to create parasitic estimates (constraints) for.

- Select net(s) using the design canvas or the Navigator assistant.
- Select a minimum of:
  - A single net to create estimated resistance.
  - A pair of nets to create estimated coupling capacitance.
  - One, three, or more nets to create estimated decoupling capacitance(s).

Choosing the Estimated Parasitic

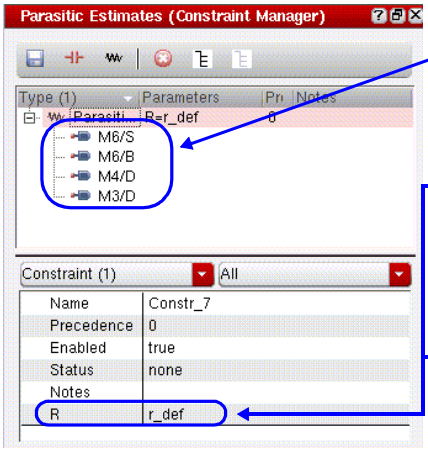
After selecting the net, click the Create Estimated Capacitance or Create Estimated Resistance icon on the Parasitics Estimates toolbar.



Note: All estimates are stored under the *constraint* view and are reusable in subsequent designs.

Creating Estimated Resistance

The Create Estimated Resistance option creates a constraint for each selected net.



Each constraint is automatically populated with the leaf instance terminals that are connected to a net.

There is a single constraint specific parameter, *R*, which represents the resistance associated with each instance terminal.

The initial estimate value, *r_def*, is taken from the Setup Parasitic Estimates form.

Note: The *R* parameter value can be a numerical constant or an AEL expression that contains ADE variables, such as R=1 or R=r_def.

The top section of the Parasitic Estimates (Constraint Manager) assistant shows details of the current Parasitic Estimate constraints, which can be expanded to reveal the current constraint members. (This is the only constraint *Type* that will be listed in this form.)

Each constraint member inherits the specified *Default Estimate Values*, as shown in the Parameters column, unless these values are overridden (edited) in the lower section of the assistant.

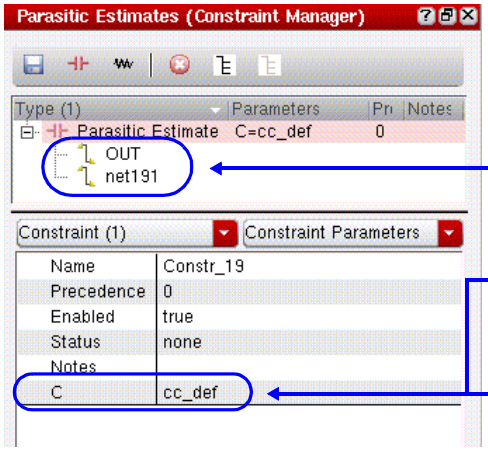
You can also use the lower section of the Parasitic Estimates (Constraint Manager) to edit a constraint's *Precedence*, any constraint Notes you want to include, as well as a variety of other editable constraint parameters.

Details of the constraint's parameters are found in the *Virtuoso Unified Custom Constraints User Guide*.

Creating Estimated Capacitance

The Create Estimated Capacitance option gives you two options: between nets (coupled) and net to reference (decoupled).

- When two nets are selected, the estimated coupling capacitance is created.



Each constraint is automatically populated with the net names connecting terminals of the coupling capacitance.

There is a single constraint-specific parameter, C which represents the capacitance between a pair of nets.

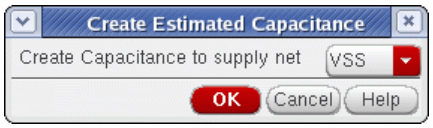
The initial estimate value, cc_def, is taken from the Setup Parasitic Estimates form.

Note: The C parameter value can be a numerical constant or an AEL expression that contains ADE variables, such as C=1 or C=cc_def.

Setting Decoupling Capacitance

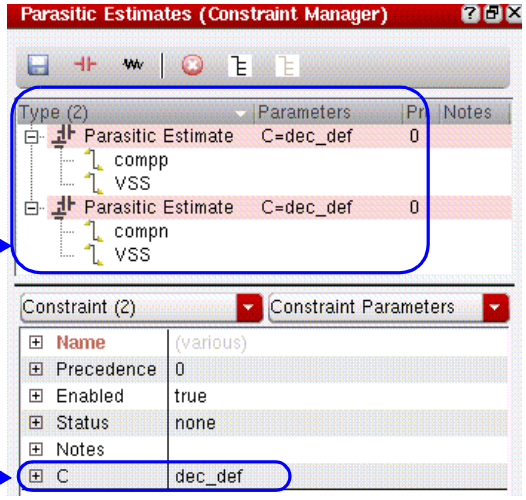
When one, three, or more nets are selected, the decoupling capacitance form pops up.

Select the supply net to which the decoupled capacitances are referenced.



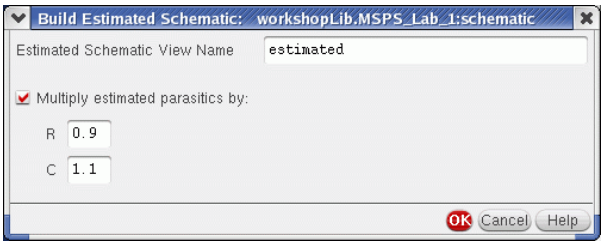
Decoupling Capacitance C=dec_def is added between the compn and VSS nets and also between the compp and VSS nets.

There is a single constraint specific parameter C, which represents the capacitance between a pair of nets.



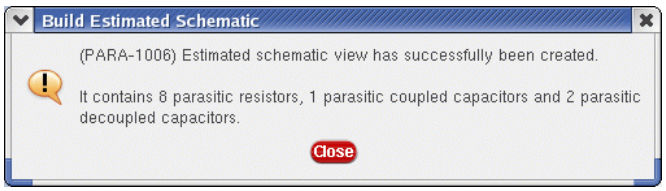
Building the Estimated Schematic

- After entering all constraints, an *estimated* schematic view is created using the Build Estimated Schematic form.
  - You can globally multiply all estimates by a factor.



Choose **Parasitics – Build Estimated Schematic**.

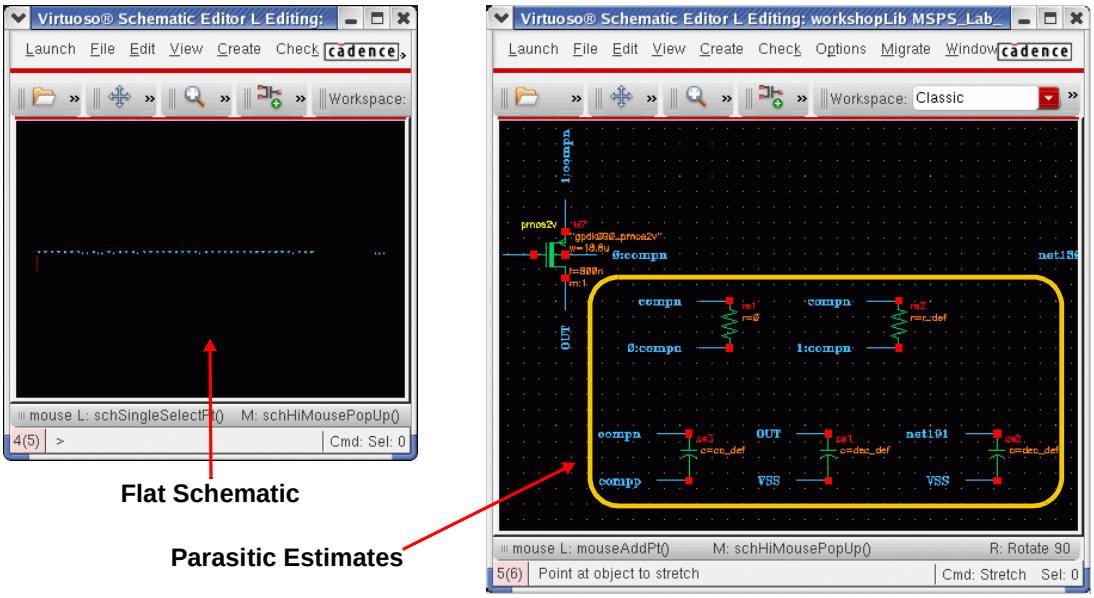
- After you create the estimated view, you will see a summary message detailing how many resistors, coupled capacitors, and decoupled capacitors were created for the selected nets.



Understanding the Estimated Schematic

The new *estimated* view is for netlisting purposes only.

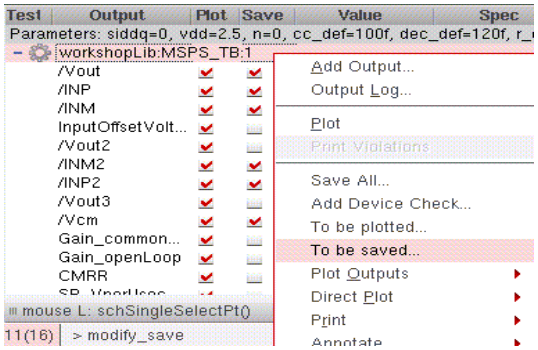
- It redefines connectivity to include the parasitic estimates.
- Hierarchical schematics are flattened to a flat estimated schematic cellview stored at the top level of the hierarchy.

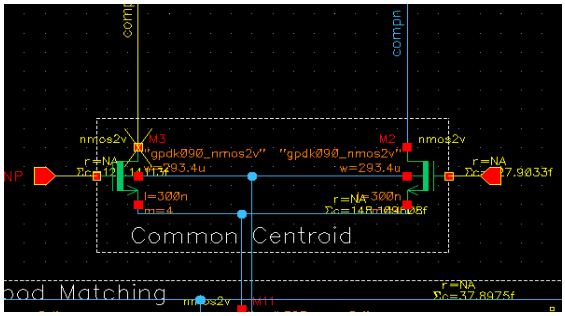


Simulating a Design With Estimated Parasitics

- Use the Hierarchy Editor to select *estimated* view and then run a simulation.
- Probe the results:
  - Probe using the original schematic because the flattened schematic is only used for netlisting.
  - Probe out-of-context where parasitic simulation will map between the hierarchical and flattened schematics.
  - Currently if you want to probe the nets that are inside a specific block you have to save those nets first.

1. In the output section of the current session window, save nets that you want to probe.

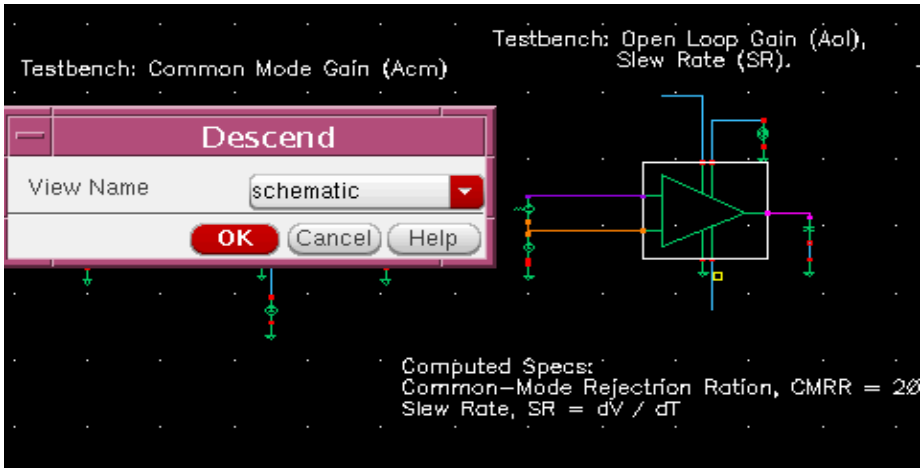




2. You are prompted to select nets in which case you have to use out-of-context probing to make the selection on the schematic.

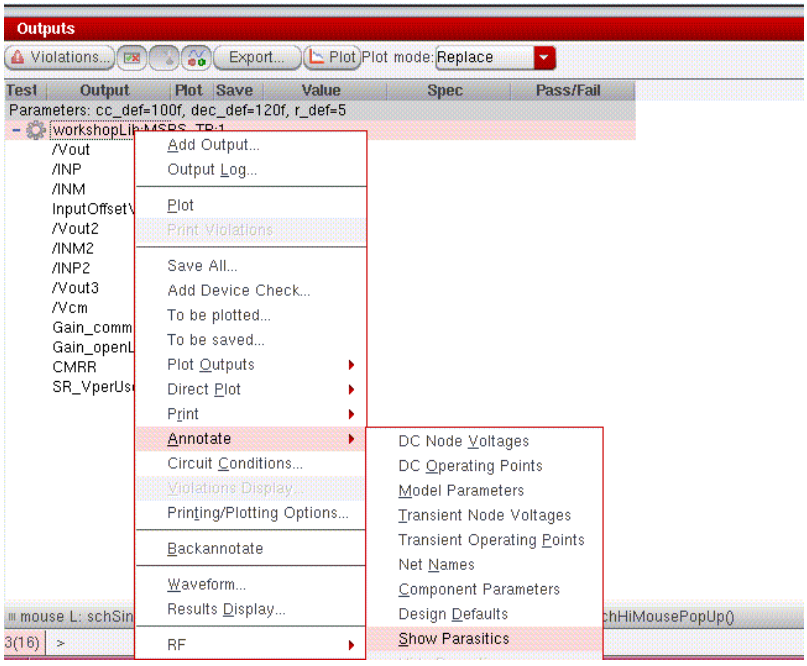
Backannotation of Estimated Parasitics

- Backannotation is accurate only when a simulation is run from the test bench (TB) configuration view.
- Use out-of-context probing to descend into the schematic view of the Design Under Test (DUT).



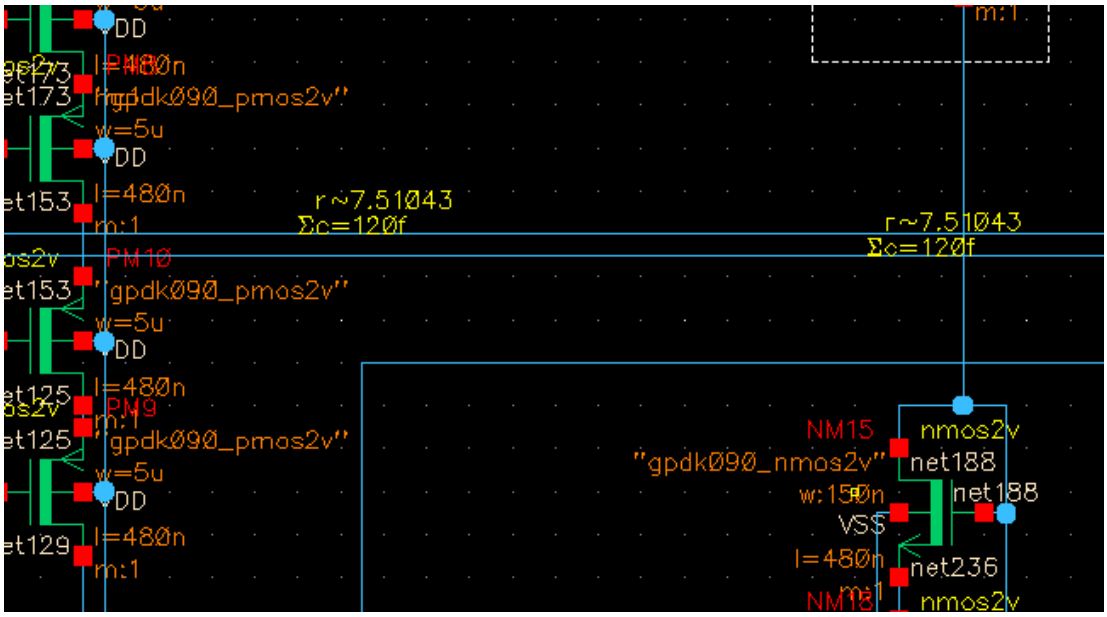
Backannotation of Estimated Parasitics (continued)

Right-click on the test in the Outputs assistant, and choose **Annotate – Show Parasitics** from the output menu to backannotate the estimated values.



Backannotation of Estimated Parasitics (continued)

Estimated parasitics are now displayed on the schematic.



Parasitic Refine Extracted Flow

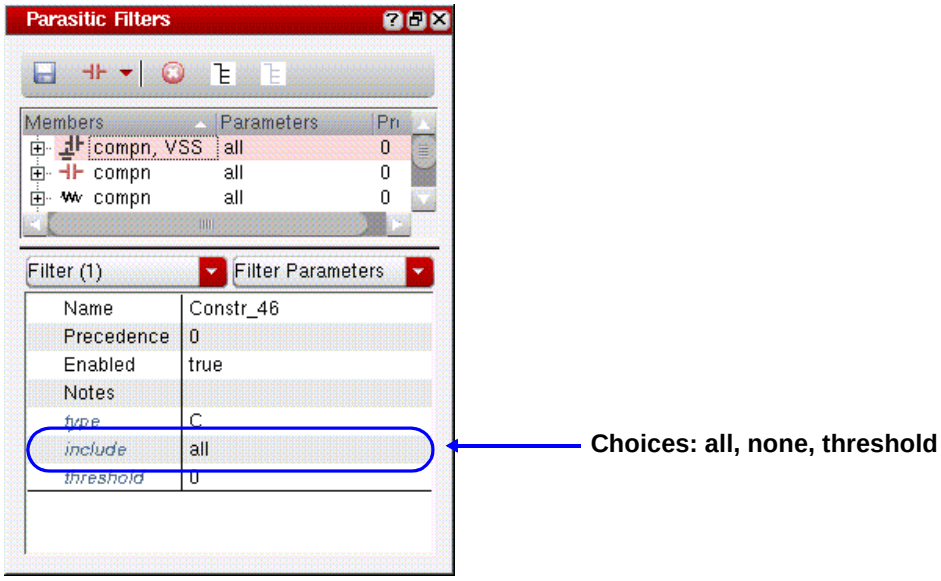
- In ADE GXL, the Parasitic Refine Extracted Flow lets you debug the sensitivities of various nets to parasitics by adding/removing parasitics from the extracted view using the parasitic filter assistant.
- Filters are used to include “all”, “none” or a selected set of parasitic instances.
  - Default value for all parasitic filter types is “none”.
  - You can set filters based on threshold value on nets or groups of nets. Parasitic instances (resistors, capacitors, and inductors) above the specified threshold on a net are kept in a refined extracted view.
- After setting filters, you can create a refined extracted view (*av_analog_extracted*) with the Parasitics Refined Extracted View form.
- Use the refined extracted view for postlayout parasitic simulation.
- You can access the refined extracted parasitics flow by choosing the **Parasitics—Extracted—Refine Extracted View** option from the menu bar.

Note: The Parasitic Refine Extracted flow is also supported in ADE L and XL. However, advanced features, such as Filters and Filter assistant pane, are unique to the ADE GXL environment.

An *extracted* view can contain thousands of parasitics that can have a dramatic impact on analog simulation of the laid out design. Refinement lets you specify which net parasitics to consider significant and then create a new version of the *extracted* view containing only these. When you generate the *refined extracted* view, it named *av_analog_extracted* by default.

The Parasitic Filters Assistant

The parasitic filters are stored in the constraints database, separate from other constraints, and are only retrievable by parasitic simulation. Parasitic simulation uses these filters to determine which parasitics to maintain in a refined *extracted* view.



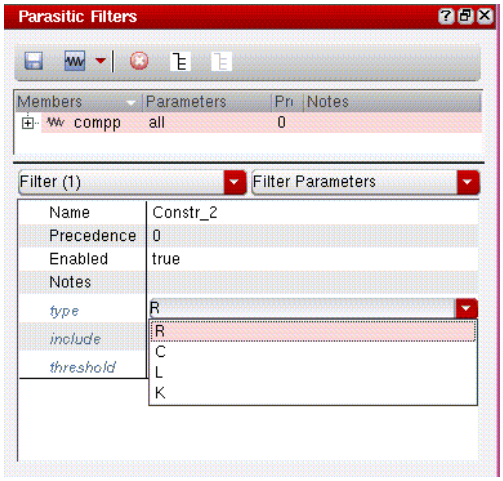
The use of parasitic filters replaces the *sbaLib* component method of identifying which parasitics to include in the refined *extracted* view. These filters provide greater flexibility and control as in these examples:

- You are not required to alter the schematic (as you did with *sbaLib*). In addition, you are not required to have write access to create new filters.
- When determining what parasitics to include in the refined *extracted* view, you can set a filter to include, all or none, or set a lower threshold parasitic value to specify parasitic inclusion. Any parasitics with values less than the threshold set will be omitted from the refined *extracted* view. You can therefore choose to ignore parasitics that will have little or no impact.
- Scoped filters (cell, instance, or net filters) can also assist when setting filters on all nets in a cellview, or below an instance, without having to set filters on each individual net. This overriding facility lets you set different values on particular nets or instances that fall within those groups.
- You can also use occurrence-based pathnames for nets/instances, which let you place different filters on individual instances of cellviews. Previously, with *sbaLib*, if you included the parasitics for a net, they were included for every instance of that cellview. You can now, however, create several instances of a cellview and place different thresholds on each for examining the different levels of parasitics on the design.

Parasitic Filter Types and Scopes

Each parasitic filter has a type and scope.

- The type is controlled by the parameter type that you set when you create a parasitic filter.
- The type can be R, C, L, or K.
- The scope is determined by the filter members, such as nets or instances.



Parasitic Filter Types and Scopes (continued)

The available filter scopes are:

- Cell filters
- Instance filters
- Net filters
- Net-to-net filters (for CoupledC only)

Note: You can only set one of the above scope parameters per filter. Although you can create multiple filters on the same object if you need more than one filter type.

Important

The precedence of filters with different scopes from lowest to highest is: Cell Filters, Instance Filters, Net Filters, Net-to-Net Filters.

- A Cell filter will apply to all nets in a specified lib/cell.

  Filters for those cells higher in a hierarchy will override those filters placed on cells lower in the hierarchy. For example, if CellA has an R cell filter and it contains an instance of CellB, which also has a R cell filter, then the CellA filter will override CellB. The reason for this is that as cells are developed, they might have local cell filters, but in a design hierarchy, filters higher up will apply to any lower cells.

  Cell filters have the lowest precedence of all filter scopes and can specify default values for nets that do not have any other filters on them. Cell filters placed on the top level cell are effectively global default filters having a master as its sole member.

- An Instance filter specifies thresholds on all nets that are wholly contained within an instance occurrence.

  The occurrence path for the instance is displayed relative to the design under test. Instance filter thresholds are applied hierarchically to all nets below the instance in the design hierarchy unless another instance filter has been placed on a lower-level instance. In this case, the latter filter will take precedence on all nets below the lower-level instance.

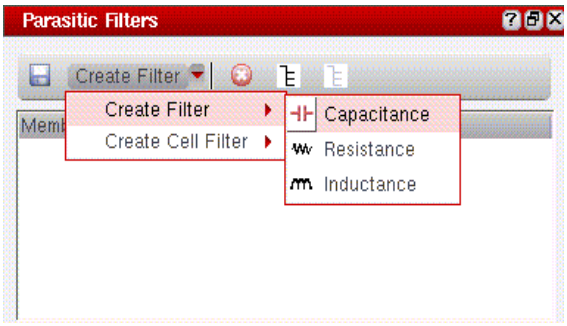
- A Net filter specifies thresholds on individual nets.

  The threshold set will apply to the entire net regardless of which hierarchy level the filter is placed on. When a net filter is created, the selected net will be resolved to the name at the highest level of the hierarchy, preventing the creation of multiple net filters for the same net at different levels of the hierarchy.

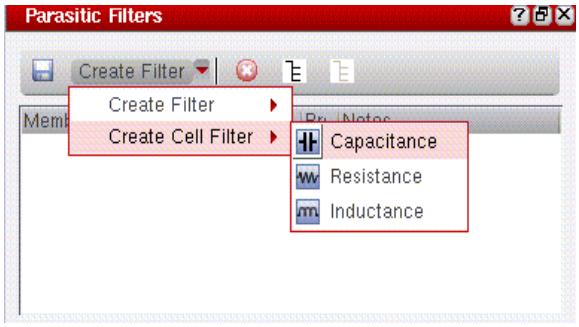
- A Net-to-Net filter sets coupled parasitic capacitance thresholds between two nets.

Using the Parasitic Filters Assistant

- You can make selections from the schematic view or the Navigator assistant to populate a filter.
- Parasitic filters can be defined for:
  - Entire cell (cell filters)
    Example: to filter out all resistances from an extracted view
  - Instance/Net (filters)
    Example: to keep only the parasitics on a critical net



Net/Instance Level Filters



Cell Level Filters

Important

Net/Instance filters override the cell filters.

The *Parasitic Filters* assistant displays all of the parasitic filter types that are in use in the current design.

Because there can be several filters associated with an object, parasitic filters will resolve the precedence between them, indicating which filter to apply.

Running the Parasitic Refine Extracted Flow

1. Open a schematic or config view in ADE GXL.
2. Open and fill in the Parasitics—Extracted—Setup form.
3. Fill out the Filters Parasitics assistant with cell/instance/net level filters
4. After setting all the needed filters, create a refined extracted view by using the Parasitics—Extracted—Refine Extracted View form.
5. Use the Hierarchy Editor to select the refined extracted view for simulation.
6. Run a postlayout refined extracted parasitic simulation.
7. Show parasitics (parasitic backannotation).
8. Report parasitics.
9. Probe parasitics.

Select **Parasitics – Extracted – Refine Extracted View** to display the Refine Extracted View form.

The purpose of the Refine Extracted View form is to generate an *extracted* view for circuit simulation. The refined extracted view is based on the full *extracted* view created in Assura RCX.

The default Assura® full extracted view name is *av_extracted*, while the default parasitic simulation refined extracted view name is *av_analog_extracted*.

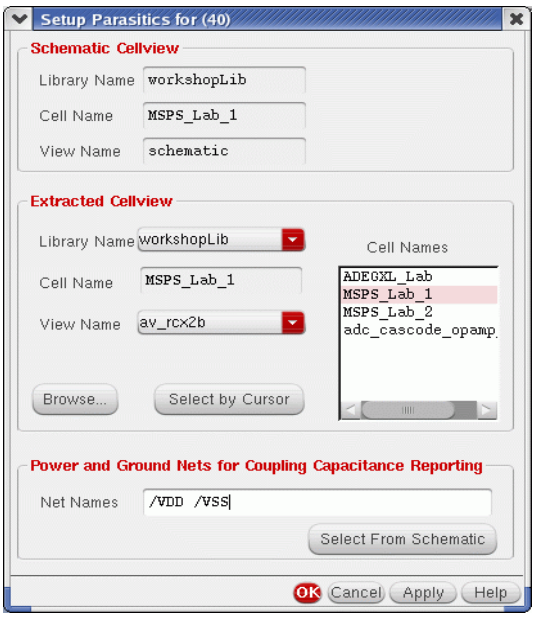
For additional information on this process, refer to "Parasitic Simulation in ADE L/XL and Schematics L/XL" in the *Virtuoso Parasitic Simulation User Guide*.

In the Refine Extracted View form, you can specify the extracted parasitics required for the extraction.

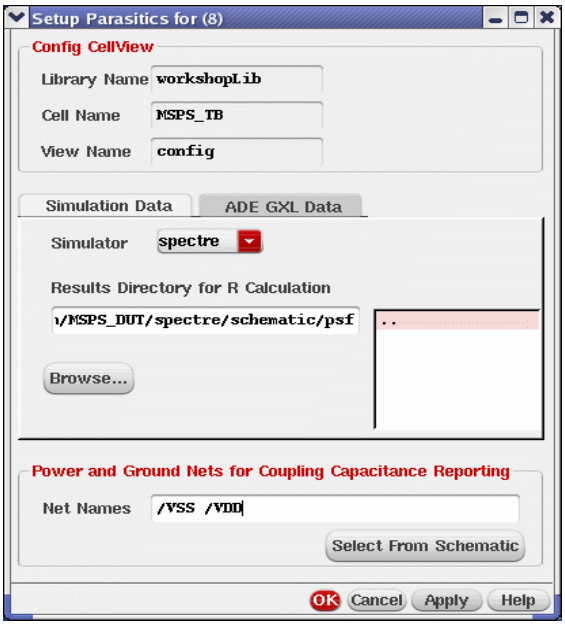
Setting Up the Refined Extracted Flow

Open “Setup” by choosing **Parasitics—Extracted—Setup**.

From Schematic/Extracted View



From Config View



Parasitics Setup for Schematic and Extracted Views

The Setup Parasitics form shown above appears if you access extracted parasitic simulation functionality from a *schematic* or *extracted* cellview.

Note: If you display the Setup Parasitics form from a *schematic* window, you only have to specify an extracted view name, and vice versa.

Parasitics Setup for Configuration Views

When you choose **Parasitics—Extracted—Setup** from a *configuration* view, you are requested to enter simulation/ADE database information.

- Simulation Data/Results Directory for R Calculation: Enter or choose a directory that can store the results of resistance calculations.
- ADE GXL Data: Alternatively, from directly selecting or entering simulation data, you can choose data from the ADE results history.

Not all items from the ADE history appear in the tree display. Only those results generated for a Spectre® or UltraSim® dc op point simulation of the current design are included. After you select data from here, the Simulation Data tab is completed with the appropriate values.

Note: The version of the Setup form shown above will also be displayed if you use parasitic simulation functionality after descending into a subblock of the configuration.

If you have opened a schematic via a simulation configuration, you do not need to specify either view, because this information will be derived when you run out-of-context probing.

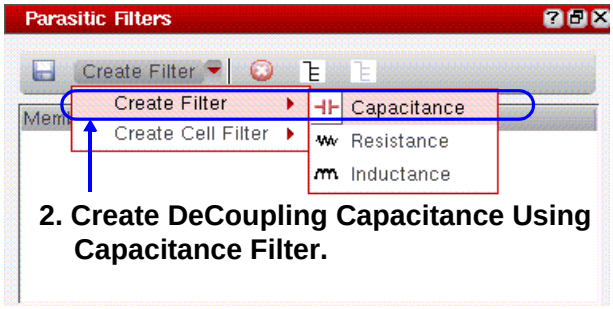
Choosing Filters

- Filter parasitics using cell filters
  - You do not need to select an object to create Cell filters, because these parasitic filter types are automatically applied to the current cell.
- Filter parasitics using net/instance filters
  - Select net(s) using the design canvas or the Navigator assistant.
  - Select from the following:
    - One or more nets to create resistance filter for each selected net. (Select a minimum of 1 net).
    - One or more nets to create a coupling capacitance filter from each selected net to any other net in the design or between each pair of selected nets. (Select a minimum of 2 nets.)
    - Three nets or more are needed to create decoupling capacitance filters. (Select a minimum of 3 nets: two signal nets and a reference net to which the capacitance is decoupled.)

Creating the Parasitic Filters

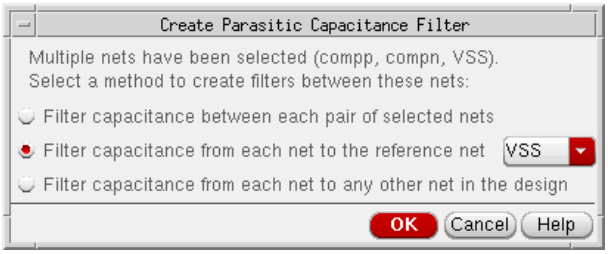
Create Parasitics Filters (decoupling capacitance filter)

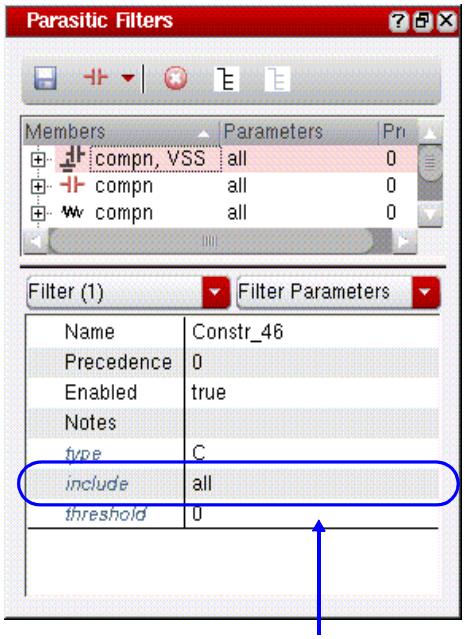
1. Select nets (such as compn,compp,VSS).



2. Create DeCoupling Capacitance Using Capacitance Filter.

3. This form pops up.

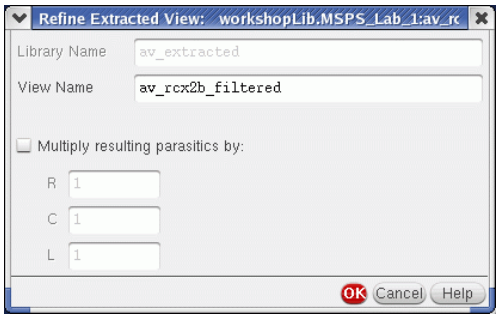




4. Select Parasitics to include: *all*, *none*, *threshold*.

Creating the Refined Extracted View

Create a refined extracted view by choosing **Parasitics—Extracted—Refine Extracted View**.



This form appears. You can choose to globally multiply all parasitics by a factor.

On the Refine Extracted View form, you can specify the extracted parasitics required for the extraction.

Library Name

Specifies the name of the library where the refined cellviews will be created. The library will be created if it does not already exist. The Library Name field is only active when refining hierarchical extracted views and will be disabled if the extracted view is flat. If this is the case, the refined view will be created in the current library.

View Name

Specifies the name of the refined extracted view to be created.

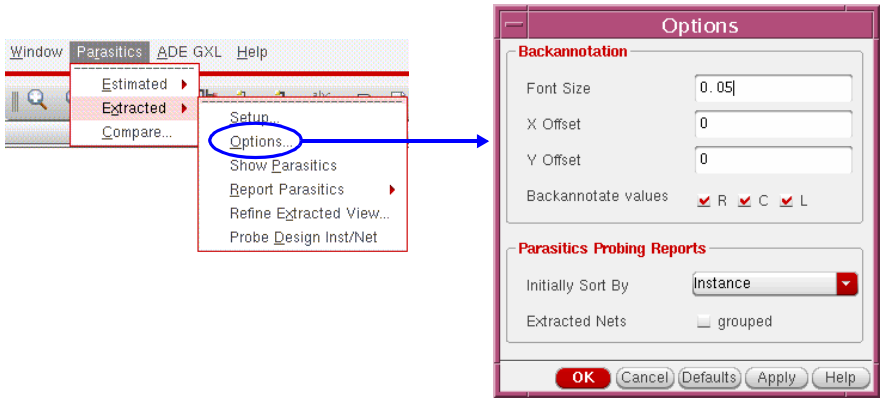
Multiply resulting parasitics by

Lets you simulate with different parasitic values. For example, the minimum and maximum expected parasitic values (based on the same extracted view).

- R: Sets the multiplication factor for resistance parasitics.
- C: Sets the multiplication factor for capacitance parasitics.
- L: Sets the multiplication factor for inductance parasitics.

Simulating the Refined Extracted Flow

- Simulating the design using the refined extracted view
  - Use Hierarchy Editor to select the refined extracted view.
  - Run simulation.
- Running postlyaout analysis (Choose **Parasitics – Extracted**.)
  - Show Parasitics
  - Report Parasitics
  - Probe Design Inst/Net



The extracted parasitics options form contains a selection of options related to Backannotation and Parasitic Probing Reports. The values entered on this form will determine some of the content of other parasitic simulation forms.

GUI Item Description

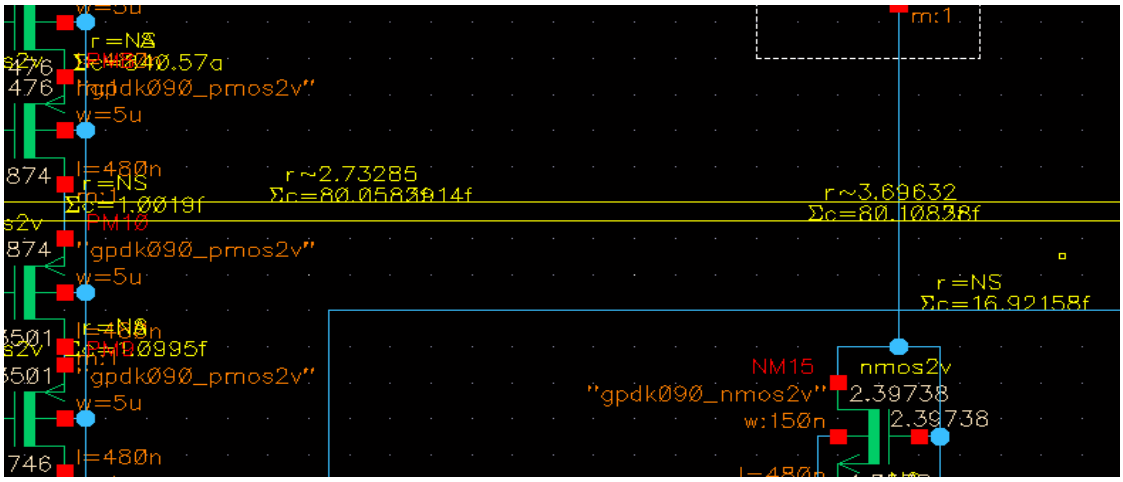
- Backannotation
  - Font Size: Specify the label font size for displaying parasitic backannotation.
  - X Offset: Set the horizontal offset from the centre of the net when displaying parasitics.
  - Y Offset: Set the vertical offset from the centre of the net when displaying parasitics.
  - Backannotate values: Specify the backannotation values that you want to view (R, C and/or L).
- Parasitics Probing Reports
  - Initially Sort By: Specify the initial sorting method (Instance, Type, Value, From, orTo) to be used when probing parasitics on nets.
  - Extracted Nets: Check the grouped option to report parasitics for the whole design net when probing an extracted net. (This will work as if probing in the schematic). For detailed debugging however, uncheck the grouped option and the report will be specific to the selected extracted net fragment.

Note: If Parasitics is in a config set up, with simulation results available, probing in-context in the extracted view will also display effective R results in the corresponding report.

Backannotating Parasitics

You can choose to turn on or off the display of generated parasitic values in the current schematic view.

- Use the Show/Hide Parasitics menu option in the schematic view. You must be running Out-Of-Context Probing for the Show/Hide Parasitics option to be enabled.
- Right-click on the test in the Outputs assistant and choose **Annotate—Show Parasitics**.

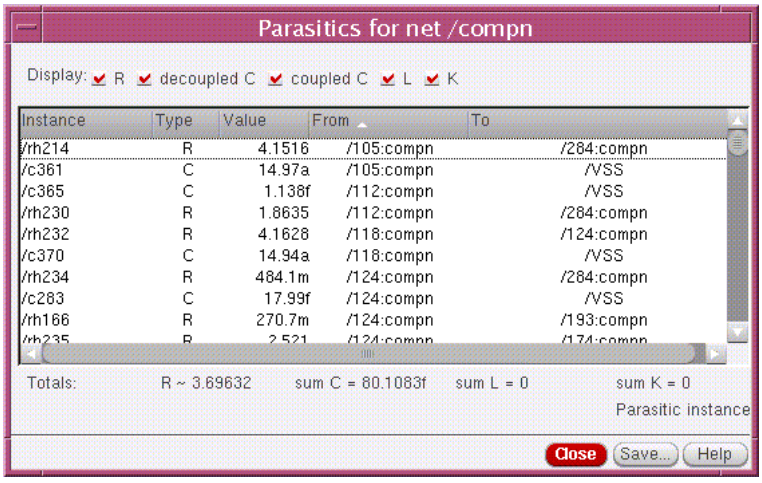


If you show/hide parasitics on a particular schematic (which must be defined in the schematic hierarchy in the Schematic Cellview section of the Parasitics Extracted Setup form, or by going out-of-context), and descend/ascend the hierarchy, the state will be remembered so that, in the new level, parasitics will continue to be displayed as in the previous level.

Parasitics Reports

The following parasitic probing report options are available under the Report Parasitics menu option:

- Report - Net
- Report - Net to Net
- Report - Terminal to Terminal
- Report - Net Capacitors
- Report - All Nets



Using Virtuoso Parasitic Estimation and Analysis with ADE GXL 16-65

Parasitic probing in ADE GXL uses internal hierarchical selection management functionality which enables you to make object selections for report generation from both the Navigator assistant and directly from the design canvas.

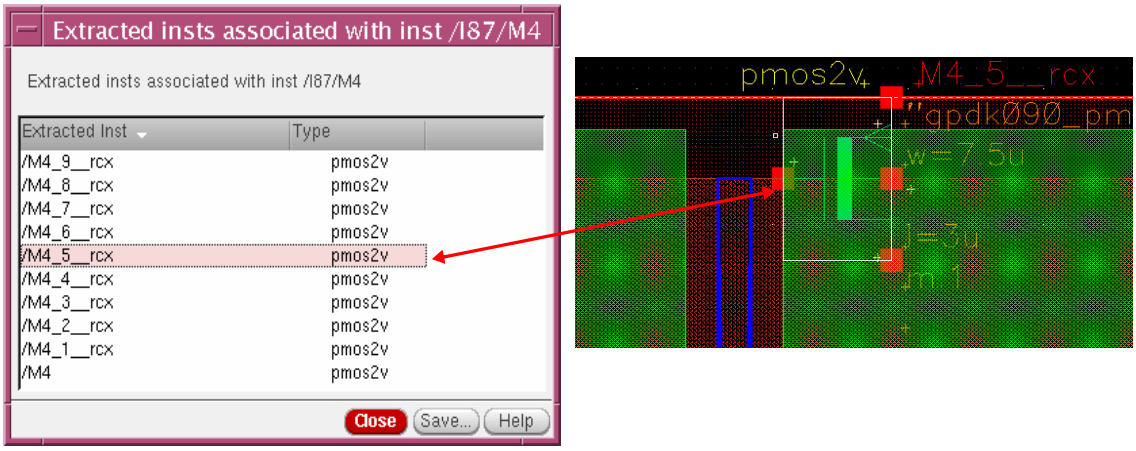
When you select a report menu option, the report form automatically displays. If you have any appropriate objects (such as nets or terminals) currently selected, then the chosen report form will initially report on these current selections. However, once a report form is on display, you can then proceed (prompted by the appropriate title bar instruction) to select objects that you want to report on from either the Navigator or from the design canvas. The report form will update dynamically as you continue to select or deselect objects.

You can save the content of a report table to a specified report file by clicking the **Save** button at the bottom of the applicable form. You have the option to save the information as either a text file (*.txt*), or a comma separated value file (*.csv*).

Details of the Report Parasitics feature are found in the *Virtuoso Parasitic Simulation User Guide*.

Probing Design Instances and Nets

- Ensure that both the schematic and extracted views are open.
- Probe from a design instance or net, in the current schematic/extracted view, to instances in the extracted/schematic view.
  - The probe from schematic will pan to and zoom the extracted instances.
  - The probe from extracted will highlight, but not zoom the schematic instance.



- When probing from the extracted view, highlighting in the schematic view will only work when the *schematic* is open. It will not work when the schematic is opened through a *config*.
- When probing from an extracted view, if the schematic instance is within a hierarchical block, in an open schematic, then the block will be highlighted.
- If attempting to probe from hierarchical blocks/instances, a warning message is displayed in the CIW. To resolve this, you must descend the hierarchy (Edit Hierarchy Descend Edit) and probe at the leaf level.
- If you are probing a schematic net that exits the current hierarchy level, the probe will be restricted to the instances attached to that net in the current level and not the full design. Probing therefore remains localized.
- For a design net, the displayed list contains all of the extracted or schematic instances that are attached to that net in the other view type.
- If you select one or more extracted instances from the list (using the Ctrl/Shift keys), the software causes the extracted view to automatically zoom into those instances, highlighting the selected items in both the schematic and extracted views.

Details of the Probe Design Inst/Net feature are found in the *Virtuoso Parasitic Simulation User Guide*.

Parasitic Compare Flow

- Compare flow allows a comparison of estimated parasitics and actual parasitics based on an extraction.
- For comparing resistances, you have to provide DC operating point simulation data for the extracted and the estimated views.

Important

Prior to comparing resistances, you must have run dcOp analysis on the estimated and the actual parasitics.

- For comparing capacitances (decoupled and coupled), you have to provide pointers to an extracted view and an estimated schematic view created using Virtuoso PEA.
- The estimated and extracted views have to correspond to the same instance path.

  If the estimation was run on instance I5 in the workbench, then the extracted view simulated needs to correspond to I5.

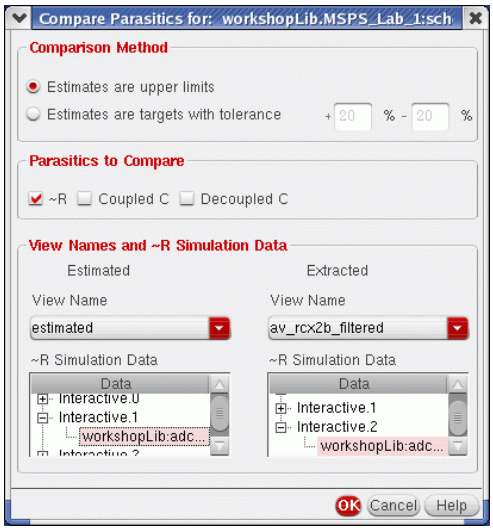
Before you attempt to make a parasitic comparison, you have to run a DC operating point (oppoint) simulation for both the estimated parasitic view and the extracted layout. This will enable the parasitic simulation to calculate and compare effectiveR values for the two views. If you do not want to compare ~R resistance values, a DC oppoint simulation is not required.

For information on running a DC oppoint analysis see “Setting Up for an Analysis” in the *Virtuoso ADE L User Guide*.

To compare estimated parasitics against extracted parasitics, you must use the original schematic design (not the estimated view created using Parasitics Estimated Build Estimated Schematic) and create an extracted view. Because only resistance (R) and capacitance (C) values are currently supported in the estimate flow, only these parasitic types are required to be created in the extracted view.

Parasitic Compare Flow (continued)

1. Open the schematic or configuration (or the design under test (DUT)) that was used for simulation.
2. Descend, not necessarily out-of-context, to where the parasitic estimates were placed. That is, descend to the estimation root.
3. Open and fill in the Compare Parasitics... form (choose **Parasitics – Compare**.)



Comparison Method: Controls how the estimates and extracted parasitics are compared.

Estimates are upper limits: Specifies that estimates must represent an upper limit.

Estimates are targets with tolerance +/-: Specifies that estimates represent a target. Here, you can specify a positive (or upper) and negative (or lower) percentage tolerance within which extracted parasitics must fall.

Parasitics to Compare: Use this option to select which classes of parasitics to validate and compare:

~R (effective resistance), Coupled C, Decoupled C

View Names and ~R Simulation Data

Estimated View Name: Specifies the estimated view name to compare.

Estimate ~R Simulation Data: Specifies the simulation results to use for the estimated view (only enabled if ~R values are to be compared).

Extracted View Name: Specifies the extracted view name to compare.

Extracted ~R Simulation Data: Specifies the simulation results to use for the extracted view (only enabled if ~R values are to be compared).

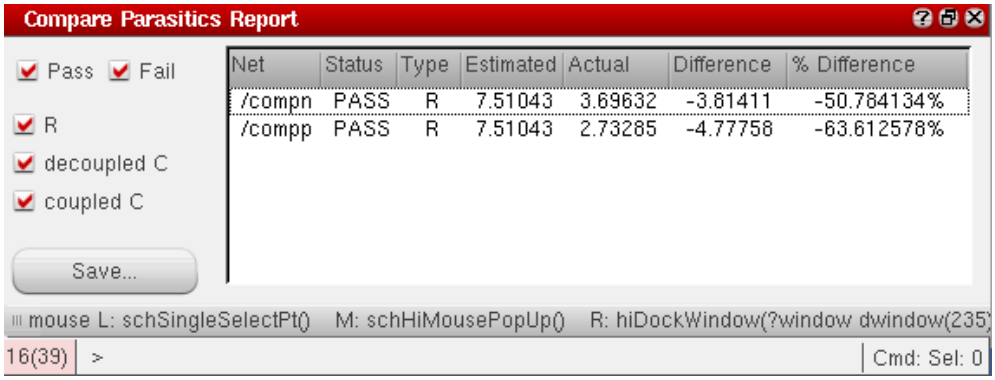
Because only resistance (R) and capacitance (C) values are currently supported in the estimate flow, only these parasitic types are required to be created in the extracted view.

Note: The tables used for Estimated/Extracted Simulation Data are populated with entries from the ADE GXL result history. From here, you locate the results for the simulation made by using the estimated/extracted views. To help locate these, only those results from a Spectre® or UltraSim® DC oppoint simulation of the current design is included in the table.

Parasitic Compare Report

The compare report contains information about the following items.

- Status of each net: PASS or FAIL
- Type of parasitic: R, Decoupled C, or Coupled C
- Estimated Value
- Extracted Value
- Difference
- % Difference



Quiz

1. List the 3 different PEA flows supported in ADE GXL?
2. What are the advanced PEA features used in ADE GXL?
3. Explain how you can access IC5.1.41 MSPS functionalities in IC 6.1.0?
4. How many nets do you have to select to set up net-level parasitic decoupling filter?
5. What type of parasitics can you compare?

Labs

Lab 16-1 Parasitic Estimates Setup and Simulation in ADE GXL

Lab 16-2 Refining Extracted Views

Lab 16-3 Comparing Estimated Parasitics with Extracted Parasitics