Hercules™
Getting Started Tutorial

Version B-2008.09-SP4, October 2011

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader s responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AEON, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CODE V, CoMET, Confirma, Design Compiler, DesignSphere, DesignWare, Eclypse, EMBED-IT!, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ARC, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCSi, VHDL Compiler, Virtualizer, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

2. An Error-Free Design for DRC

Appendix A. Using Enterprise With Hercules-VUE

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *Hercules Release Notes* in SolvNet.

To see the *Hercules Release Notes*,

1. Go to the Download Center on SolvNet located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select Hercules, and then select a release in the list that appears.

About This Tutorial

Hercules is a powerful software package that speeds the process of verifying integrated circuit layouts. Hercules is not just a single tool, but a suite of programs. Hercules can verify layout Design Rule Checks (DRC), perform Electrical Rule Checks (ERC), extract layout structures and compare them to an original design netlist by using the Layout versus Schematic (LVS) application, and generate or modify data for mask preparation. The hierarchical checking of algorithms make Hercules particularly well-suited for large and complex IC verification.

Audience

The *Hercules Getting Started Tutorial* assists the first-time user of Hercules in installing and running the Hercules software.

Using the tutorials, you quickly learn to read Hercules runset files and to debug problems. This tutorial includes a detailed explanation of hierarchical design structure and how Hercules takes advantage of it. You also explore advanced Hercules applications, runset debugging information, and hints on how to get the most out of Hercules.

This tutorial contains directions for using Hercules in the Synopsys Milkyway™ and Cadence® Virtuoso® Layout Editor environments and includes steps for translating all data from Cadence Dracula® physical verification. See Chapter 1, "Installation and Setup" for a complete description.

Note to Dracula Physical Verification Users

Hercules contains a Dracula physical verification runset translator, Drac2He. Chapter 5, “Hercules DRC Migration,” explains how to use this translator and access files. The tutorial shows you how to convert two simple DRC Dracula physical verification rule files to Hercules runsets, and then execute Hercules. However, you must first be familiar with Hercules-VUE, the graphical debug tool that interfaces to Virtuoso Layout Editor. Follow the recommended learning flow described below.

Document Outline

The tutorial is divided into two sections.

- The first section covers the Hercules HDRC tool. It is divided into six chapters, outlined in the last section of this preface.
- The second section covers the Hercules HLVS tool. It is divided into four chapters.

Section 1 - Getting Started with Hercules HDRC

Chapter 1, “Installation and Setup,” contains the File Transfer Protocol (FTP) installation and tutorial setup instructions for Hercules and Enterprise. Enterprise is one of the Synopsys layout editors and a view-only version of the layout tool is provided with Hercules-VUE for looking at errors. Instructions for using Enterprise with Hercules-VUE are provided in Appendix A, “Using Enterprise With Hercules-VUE.”

Chapter 2, “An Error-Free Design for DRC,” discusses the first tutorial design, a very basic, error-free example designed to introduce you to Hercules and the files created. The example is a building block intended to help you understand the more complex examples that follow. You learn about the contents of runset files, how to process runsets, and how to read the output.

Chapter 3, “Single Design Error for DRC,” continues the tutorial with a single design error in the design of an inverter (INV) in a full adder (ADDER). We note the differences between the error-free Hercules run from Chapter 2 and the run with errors in this chapter. We also view the graphical output files in Enterprise, the Synopsys layout tool, provided with Hercules-VUE for viewing errors.

Chapter 4, “A Complete Design for DRC,” presents a multiple-error design. The activities in this chapter are comparable to the ones in Chapter 3 but here you create layout errors similar to those encountered in real designs and learn how to correct them. This chapter introduces Hercules-VUE, a Synopsys graphical user interface debugger.

Chapter 5, “Hercules DRC Migration,” shows you how to convert two simple Dracula physical verification rule files to Hercules runsets, using various translation options to familiarize you with operational and syntactical differences between the two tools.

Chapter 6, “Hercules Migration With Hercules-VUE,” has you convert a simple Dracula physical verification rule file to a Hercules runset using the translation options you learned in Chapter 5. You then run Hercules on this runset and view the results using Hercules-VUE interfaced to Virtuoso Layout Editor, the Cadence layout tool.

Section 2 - Getting Started with Hercules LVS

Chapter 7, “Introduction to Hercules HLVS,” discusses the basic principles of Hercules Hierarchical Layout versus Schematic Checking (LVS). We also review the tutorial design used in this section of the document. You learn about the components of a Hercules LVS job and the contents of runset and output files created for an error-free LVS job.

Chapter 8, “HLVS Advanced Concepts,” introduces you to concepts used in a strict Hercules LVS comparison, as well as to the Hercules LVS advanced options and commands required to make the flow work. We run two examples with a few errors in order to better illustrate the advanced features of Hercules used in this flow.

Chapter 9, “Hercules HLVS Debugging,” goes through a complete Hercules LVS debugging example. You learn how to use the HTML interface and Hercules-VUE, a graphical user interface (GUI) debugging tool.

Chapter 10, “HLVS Migration with Hercules-VUE,” shows you how to convert a simple HLVS Dracula physical verification rule set to a Hercules runset using the translation options you learned in Chapter 5, “Hercules DRC Migration.” Then you run Hercules on this runset and view the results using Hercules-VUE interfaced to Virtuoso Layout Editor and Cadence Virtuoso Schematic Editor, layout and schematic tools.

Learning Schedule

We suggest you complete the chapters of this tutorial in the order given below.

Learning Schedule for DRC

- Begin with Chapter 1, “Installation and Setup,” to ensure that you have the correct installation of the Hercules software and tutorial examples.
- Complete Chapters 1–3 on installation, setup, and basic execution of Hercules.

  Estimated completion time: 30 minutes.
- If you are currently a Dracula physical verification user, once you have completed Chapters 1–3, go to Chapters 5 and 6.

Estimated completion time: 30 minutes.

- If you are currently a Dracula physical verification user and you are learning to write Hercules runsets, after you have completed Chapters 1–3 and Chapters 5 and 6, go back and complete Chapter 4, “A Complete Design for DRC.” It is important that you follow this order of study so that you get an understanding of Hercules in general (Chapters 1 and 2), of how Dracula physical verification and Hercules relate to each other (Chapters 5 and 6), and, finally, learn detailed information on Hercules (Chapter 4), which you need for debugging the runsets you write.

  Estimated time to complete Chapter 4: 20 minutes.

- If you do not use Dracula physical verification in any way, go from Chapter 3 directly to Chapter 4 on detailed debugging in the Hercules environment. You can then skip Chapters 5 and 6.

  Estimated completion time: 20 minutes.

Learning Schedule for LVS

- If you plan on using Hercules for LVS and you have not completed Chapters 1–3, start the tutorial with these chapters. They are necessary for the correct installation of the Hercules code and tutorial directories. They also give you a basic understanding of Hercules.
- Once you have completed Chapters 1–3, continue on to Chapter 7, “Introduction to Hercules HLVS,” for an introduction to Hercules LVS.
- If you are currently a Dracula physical verification user, once you have completed Chapter 7, skip to Chapter 10, “HLVS Migration with Hercules-VUE,” for a complete tutorial on migrating from Dracula physical verification LVS to Hercules LVS and how to debug in the Virtuoso Layout Editor environment.
- If you are currently a Dracula physical verification user and you plan on learning to write Hercules LVS runsets, go back and complete Chapters 8 and 9.
- If you are not a Dracula physical verification user, continue from Chapter 7 on to Chapters 8 and 9 to learn the advanced features of Hercules LVS and how to use the HTML and Hercules-VUE interfaces for debugging LVS. You can skip Chapter 10.

Related Publications

For additional information about Hercules, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

In addition, the documentation is installed with the Hercules software and is available through the Hercules VUE Help menu.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com, entering your user name and password and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Installation and Setup

This chapter contains the File Transfer Protocol (FTP) instructions for installation setup of the product, its documentation, and tutorials. After installation, be sure to read the release notes for important product information. After the tutorial setup, check to see if you have all the correct directories and files, after which you are ready to start the tutorial.

About This Chapter

The following installation instructions assume that you know UNIX operating system commands. You should also have the GNU software to unzip files.

Installation has three groups of steps:

- Creating directories and getting the zipped applications files, documentation, and tutorials.
- Extracting the zipped files.
- Ensuring that your environment accommodates Hercules and Enterprise.

Setup has four groups of steps:

- Creating directories and moving files to them
- Sourcing the setup file.

- Converting data files to the correct formats.
- Checking to see if you have the correct directories.

Creating Directories and Getting the Files

In this section, you create the proper directories and download files.

Create a Target Directory for Hercules Setup Files

If you do not have one already, create a single target directory into which you will transfer Hercules installation files. Choose your own directory name. All the installation files will be in this target installation directory. Our example refers to this target directory as *your_path*.

After the installation is complete, the $HERCULES_HOME_DIR environment variable is created, as well as the address *your_path*, which points to the location where you installed Hercules.

Download the Software

Go to the Synopsys public website (http://synopsys.com/) and do the following:

1. Click the SolvNet Online Support in the QUICKLINKS section in the right margin (you must be a registered user).
2. Choose Download Center.
3. Choose Hercules from the list of products.
4. Follow the instructions to retrieve software from the Synopsys EST system. For more information about downloads, use the documentation links on this page.

Copy Install and Code Files to Target Directory

From /auth/hercules_version.auth/ copy the Hercules code:

- hercules_*version*_*SYS*.tar
- hercules_*version*_common.tar

where *SYS* refers to the platform to be used for your installation and where *version* refers to the version number. For example, if you will be running version 2006.12-SP1 on the SUN.64 platform, download the file

hercules_vY-2006.12-SP1_SUN.64.tar

Note:

A contrib directory included with this release provides useful scripts for analyzing Hercules performance. The contrib directory is at the top level of the release install directory. See the README file for basic instructions. These scripts are not supported.

Enterprise is the layout editor used in Hercules tutorials and trainings. An Enterprise (view-only) license is part of your Hercules package. Enterprise is available via authenticated FTP at ftp.synopsys.com.

Extracting the Zipped File

Once the tar file is present in your target directory, follow the steps below to extract the installation directory and execute the installation script.

Untar File

At the UNIX prompt, enter:

```
tar xvf your_path/hercules_version_SYS.tar
tar xvf your_path/hercules_version_common.tar
```

Install Files in Proper Directory and Execute

The Hercules product is installed using the Synopsys Common Installer. You can find an INSTALL_README.txt file on the product CD describing the common installer in more detail or the README.1ST file in *your_path* directory where the above tarfile is extracted.

Get the Manuals and Release Notes

Go to *your_path*/doc/hercules to get the manuals and release notes.

You will see several directories: the pdf directory contains PDF versions of all Hercules user guide and reference manual documents, while the document-specific directories contain HTML versions of these documents.

Get the Tutorial

The file needed for the tutorial is located in *your_path*. You should see the file *hercules-Examples*.tar.gz.

At the UNIX prompt, enter:

```
gzip -dc your_path/hercules-Examples.tar.gz | tar -xvf -
```

Ensuring Your Environment Accommodates Hercules

Follow the steps below to ensure your environment accommodates Hercules.

Edit Your .cshrc or .profile Files

If you want Hercules and the tutorial to be active in every UNIX shell you create, edit your .cshrc or .profile file to source the hercules_setup.csh (or hercules_setup.sh) configuration file. First, edit the hercules_setup.csh (or hercules_setup.sh) configuration file to specify the proper HERCULES_HOME_DIR value by replacing the string *TOP-LEVEL-INSTALL-DIR* with *your_path*.

In your .cshrc file, enter:

```
source /your_path/setup/hercules_setup.csh
```

Or, in your .kshrc or .profile file, enter:

```
. /your_path/setup/hercules_setup.sh
```

Or, rather than edit your .cshrc or .profile files, you can add the applicable environment variables that are defined in the hercules_setup.csh file directly to your .cshrc, .kshrc, or .profile configuration files.

Initialize Your Configuration Files

Initialize your configuration files by sourcing them. This allows the software to recognize the new Synopsys paths.

If you are in the UNIX C-shell (csh), enter:

```
source .cshrc rehash
```

Or, if you are in the UNIX Korn-shell (ksh), enter:

```
. .kshrc
```

Licensing Hercules

Synopsys bundles licensing software necessary for all Synopsys products within the Synopsys Common Licensing (SCL) standalone product package, also downloadable from SolvNet. It is necessary to install SCL prior to using Hercules.

To install Synopsys Common Licensing:

- Go to the Synopsys public website (http://synopsys.com/).
  1. Click the SolvNet Online Support in the QUICKLINKS section in the right margin (you must be a registered user).
  2. Choose Download Center.
  3. Choose Licensing QuickStart Guide.
  4. Follow the instructions to download and install SCL.

To license or authorize the Synopsys products either:

- Go to the SolvNet Download center as described above.
- Choose SmartKeys.
- Follow the instructions for key retrieval.
- Or, contact Synopsys licensing (http://www.synopsys.com/contactus.html).

Getting Hercules to Run: Setup

You should already have an installation of Hercules on your system before you begin this setup. If not, be sure to load the software, using the instructions provided above. After the Hercules installation, set up your workstation account, directories, and files.

If you have already set up a Hercules account on your network, go to your home directory. You may already have several files in the account. To keep your design files separate, create a new project directory for each example.

Note:

When you install Hercules, you automatically install Drac2He, the runset translator. You execute the translator from your UNIX command line using your Dracula physical verification DRC, LVS, or ERC rule file as an argument.

How to Set Up Your Account

With the installation package, all the Hercules files needed for the tutorials are located in the following directories:

your_path/hercules-Examples/Getting_Started_Hercules_DRC/addertest1/
your_path/hercules-Examples/Getting_Started_Hercules_DRC/addertest2/
your_path/hercules-Examples/Getting_Started_Hercules_DRC/addertest3/
your_path/hercules-Examples/Getting_Started_Drac2he_DRC/migration1/
your_path/hercules-Examples/Getting_Started_Drac2he_DRC/migration2/
your_path/hercules-Examples/Getting_Started_Drac2he_DRC/migration3/
your_path/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs1/
your_path/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs2/
your_path/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs3/
your_path/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs4/
your_path/hercules-Examples/Getting_Started_Drac2he_LVS/migration_lvs/

What's Next?

If you have completed all the steps in this chapter, you are now ready to get started with Hercules. Before you begin Chapter 2, "An Error-Free Design for DRC," we suggest that you read the Document Outline and Learning Schedule sections in "About This Tutorial" for a general idea of the tutorials in this manual and how they apply to your design environment.

2

An Error-Free Design for DRC

In this chapter we start with a very basic, error-free Design Rule Check (DRC) example to introduce you to Hercules and the files created. The example is a building block designed to help you understand the more complex examples that follow. You learn about the contents of runset files, how to process runsets, how to read the output, and how hierarchy processing works.

Learning Objectives for This Chapter

The first DRC orientation example does not contain any errors, but has the following objectives:

- To learn the major components of a simple, yet complete, runset file
- To run Hercules on an error-free design and examine file output, in order to establish a reference for clean design files. This step also verifies that your Hercules software installation and licensing are correct
- To learn to analyze hierarchical design structure and the tree files created by Hercules
- To examine briefly other Hercules output files

Before You Start

Before you start this tutorial, make sure that you have FULLY COMPLETED the installation and setup procedures described in Chapter 1, “Installation and Setup.”

Learning Method: the Design Example

To learn Hercules, you use a four-bit full adder as the example design. This relatively straightforward circuit is an arithmetic building block that can be used inside a CPU or some other computational function. The adder is built hierarchically, making it well-suited to take advantage of Hercules checking capabilities. As we go through the tutorial, we introduce more information about hierarchical design and the adder in relevant sections.

The design is configured so that you can use independent runsets on variations of the full adder. Each variation is in its own directory, to avoid any possible confusion. Since your design copy is in your own account, you are free to experiment with the files. Examine the graphical files not specifically covered in this tutorial and make changes to see what effect they have on the error polygons created. In fact, since this tutorial cannot possibly cover every aspect of the design or the design structures, we encourage you at several points along the way to explore on your own. Independent experimentation reinforces the learning experience.

Layout Editors in This Tutorial

During your Hercules installation, you also should have installed the Synopsys layout editor, Enterprise. As mentioned earlier, all Hercules users who have a Hercules-VUE license also have an Enterprise view-only license. We use Enterprise as the layout editor in most of our examples. The exceptions include those chapters where Dracula physical verification translation is involved. In these cases, directions are given for running Virtuoso Layout Editor. There is also a separate example of viewing error structures with Enterprise in Appendix A, “Using Enterprise With Hercules-VUE”.

Overview of the Runset File

Runset files are ASCII (text) files containing instructions for determining where Hercules gets its files and how it runs. Each runset file has sections with variables and commands defining the functions Hercules performs. The following sections provide explanations of each part of the runset file and its variables and commands.

The Runset File Format

Files with the .ev extension are runset files in this tutorial. Example 2-1 on page 2-3 shows our first runset example, adderdrc1.ev. We go over each line so that you understand the basics of runset file creation.

Example of a Runset File

Study the example of an actual runset file and note the various sections, descriptions of which appear to the right of the series of dashes (----). The keywords for a section appear in the far left margin, followed by a space and an open brace ({). In the sample runset in Example 2-1, the keywords for each section (listed below) are emphasized to make it easier for you to see them.

- Runset Header information
- Runset options
- Preprocessing options
- Layer assignments
- SNAP command
- GRID checks
- DRC checks

Following Example 2-1, there is a general discussion of each of these sections, followed by details of the settings. Note that the setting types include:

- Information
- Options
- Commands
- Assignments
- Checks

Example 2-1 adderdrc1.ev Runset File

```
/* HDRC EXAMPLE RUNSET */

/* ------------------------------ Runset Header information */

HEADER {
   LAYOUT_PATH = ./
   INLIB = EX_ADDER_1
   BLOCK = AD4FUL
```

```
   OUTLIB = EX_ADDER_1
   OUTPUT_BLOCK = TOP_ERR
   GROUP_DIR = group
   FORMAT = MILKYWAY
}

/* ------------------------------ Runset options */
OPTIONS {
   WIDTH = 2
}

/* ------------------------------ Data Preprocessing options */
PREPROCESS_OPTIONS {
   CHECK_PATH_90 = false
   }

/* ------------------------------ Layer assignments */
ASSIGN {
   tox  (1)
   poly (5)
   well (31)
   psel (14)
   cont (6)
   met1 (8)  text (110)
   met2 (10)
   via  (19)
}

/* ------------------------------ SNAP Command */
SNAP {
      assign_layers = 0.010
}     temp = snap_out

/* ------------------------------ GRID checks */
GRID_CHECK {
        assign_layers = true
        check_45      = true
} (100)

/* ------------------------------ DRC checks */
EXTERNAL met1 {SPACING<3.0} (101)
```

Runset Header Information

The Runset Header information section is defined by the keyword HEADER, found in the left margin of the file. The HEADER section contains variables that define where Hercules can find the layout libraries and other input files. The HEADER section also contains information about where to write intermediate processing files and output files. HEADER information

variables that are not shown are set to their default values. Many of these variables are used with other programs in the Synopsys suite of tools; please refer to the *Hercules Reference Manual* for a complete list.

LAYOUT_PATH

This variable tells Hercules where to find the Milkyway database on your system. If you are reading a GDSII input file, which we do later in the tutorial, this variable is not used. The path you set must match the database path you set for your Milkyway libraries during the installation process. Your libraries should be in the directories *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest*. (The asterisk [*] is used as a wildcard to represent multiple files.) Since you will actually execute your Hercules jobs from the same directory in which the libraries are located, the LAYOUT_PATH is set to ./ (the current directory). This line is optional, because you can also set the LAYOUT_PATH environment variable to point to the libraries (this would be set to *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest1/ for our first example).

INLIB

The INLIB variable tells Hercules the name of the input library you want to check. EX_ADDER_1 is our first tutorial example library. If you use GDSII-formatted data, the INLIB line needs to specify a complete path and GDSII file name, such as *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest1/EX_ADDER_1.GDS.

BLOCK

BLOCK tells Hercules the name of the top-level input structure you wish to verify. Our four-bit adder block is named AD4FUL, and we have placed the design in the tutorial library, EX_ADDER_1.

OUTLIB

The OUTLIB setting tells Hercules the name of the output library, which contains all the permanent output layers (created by Hercules), error output, and newly-created graphical layers. Chapter 4, “A Complete Design for DRC,” explains permanent layers. In our example the line is redundant, because Hercules defaults to EX_ADDER_1 (the INLIB library) as the output library if it is not declared. Be aware that there are trade-offs to using the same library for input and output files. We illustrate this with an alternate approach for storing output files later on in the tutorial.

OUTPUT_BLOCK

The OUTPUT_BLOCK setting tells Hercules what to call the top-level output structure that holds all of your error hierarchy and permanent output layer placements. You can use this structure as an overlay on your design block to assess quickly the results of your DRC check run. TOP_ERR is the structure name. The setting is optional, with Hercules defaulting to the name EV_OUTPUT if an explicit name is not declared.

GROUP_DIR

The GROUP_DIR setting tells Hercules where to place all the group files it temporarily creates. Group files contain the data on which Hercules works during runset execution. The setting is optional, with Hercules creating a run_details/group/ directory in the path of the current directory as the default. In our example the group files are stored in the directory ./group created directly inside the current directory. The directory should be local to the machine executing Hercules.

FORMAT

This setting indicates that we are using Milkyway-formatted input data. We could have used GDSII as an alternative.

If you want to try using the GDS data as input:

1. Make sure that the EX_ADDER_*.GDS files are in the appropriate addertest directories.
2. Set FORMAT to GDSII.
3. Use the INLIB path for GDSII input (described in “INLIB” on page 2-5).

You should try this after you have completed your first Hercules run.

Runset Options

The OPTIONS section allows you to specify global assignments for various Hercules processes. Hercules also has a DRC_OPTIONS section. All variables in the DRC_OPTIONS section may also be placed in the OPTIONS section. For example, WIDTH is a DRC_OPTION, and you find it in the DRC_OPTIONS section of the *Hercules Reference Manual*. To simplify our example, we have placed the WIDTH variable under the OPTIONS section. You need to specify an option value only when you wish to override the Hercules default. In our example, we have selected just a few of the many Hercules options you can use. Refer to the*Hercules Reference Manual* for a complete OPTIONS list.

WIDTH

This option specifies the path width, in microns, to assign to polygons generated by checks that produce edges rather than rectangles (such as those output by the CUT command). In this design, a setting of 2 makes it easier to see the error polygons than the default setting of 0, which is the width of a drawn line. The appropriate width is determined by the design rules used for the layout, where small geometry layouts require smaller path widths.

Preprocessing Options

The PREPROCESS_OPTIONS section allows you to specify the output of information and the setting of path grid checking options. You can set options for increased information in the *block*.LAYOUT_ERRORS file and the tree files. You can also set options for path grid checking, which is done when the layers are read during the ASSIGN section. The remainder of the grid checking is done during the GRID_CHECK command, which is discussed later in this tutorial.

CHECK_PATH_90

Setting this option to FALSE allows you to have 45-degree path data in your design. If you look at our design in a layout editor, you will notice that we have 45-degree POLY paths. As a result, you must set CHECK_PATH_90 to FALSE to avoid false errors. Although we do not explicitly write the CHECK_PATH_45 option in our runset, it is set to TRUE by default. Therefore, we are checking to make sure these 45-degree paths are on grid.

Layer Assignments

The ASSIGN section assigns names to the database layers found in the design. Our design uses eight polygon layers and one text layer. All input polygon and text layers must be named and defined in this section before they can be used in the remainder of the runset. In our example, there is a one-to-one correlation between a layer name and a layer number.

SNAP Command

The SNAP command section forces data to conform to a grid resolution during a run. Snapping resolution can be specified for layers individually, or the ASSIGN_LAYERS command can define a snapping resolution for all input layers. If a SNAP command is not issued, Hercules automatically creates one as long as SNAP_RES and RESOLUTION are

both specified within the OPTIONS section. In this example, we have chosen to define explicitly the SNAP command and its output (temp=snap_out). See Chapter 4, “A Complete Design for DRC,” for an explanation of temporary files).

ASSIGN_LAYERS

We have set the SNAP command variable, ASSIGN_LAYERS=0.010, causing all of the input layers to be snapped to a resolution of 0.010. The AREF and SREF origins are not snapped. Refer to INSTANCE_RESOLUTION in the *Hercules Reference Manual* for snapping of this data. Text is typically not snapped unless the snapping of a polygon causes the text over that polygon to move outside the boundary of the polygon. In such a case, the text is shifted so that it is inside the polygon’s boundary, but no effort is made to place the relocated text on a grid point.

GRID Checks

The GRID_CHECK command performs grid checking after group file creation and appears in the runset after the ASSIGN section. We have included two options for this section.

ASSIGN_LAYERS

Setting ASSIGN_LAYERS to TRUE verifies that all layers in the ASSIGN section are grid checked to the database default. Individual layers may also be specified after ASSIGN_LAYERS inside the GRID_CHECK command, in order to override the grid checking behavior on a layer-by-layer basis.

CHECK_45

Setting the CHECK_45 value to TRUE verifies that all layout data is either orthogonal or at 45-degree line angles. This is a requirement for many DRC rule sets. With this option in the file, Hercules performs a line angle check and reports the results.

DRC Checks

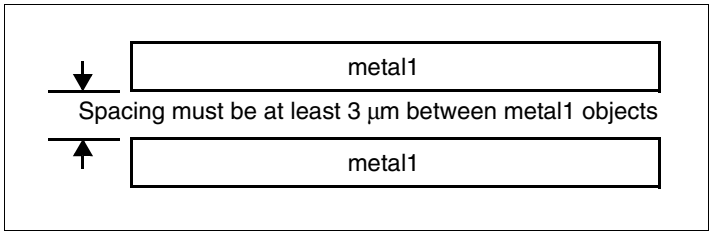
For our basic example, we have one command to check a DRC violation, EXTERNAL.

EXTERNAL

The EXTERNAL check is a spacing check between adjacent metal1 polygons that reside on the same layer (Figure 2-1). The rule creates an error polygon for any metal1 polygons that are spaced less than 3 µm apart. Specifying a number in parentheses, for example, (101), at the end of the statement results in any error vectors or polygons appearing in the error hierarchy for the design output layer 101. The error hierarchy contains all the errors for the entire design. (This concept is demonstrated more thoroughly later in the tutorial.) In addition, data output that is created from design rule statements can be sent to database locations other than the error hierarchy. We explore this in detail when we show examples of statements that create data other than errors.

The diagram in Figure 2-1 illustrates our runset design rule.

Figure 2-1 EXTERNAL Check - 3-µm Spacing Rule



Running Hercules

Now that we have explained the contents of the runset file, you need to run Hercules on the adderdrc1.ev runset in order to evaluate AD4FUL in the EX_ADDER library as a check against the design example.

How to Run Hercules

Be sure that you are in the directory where your adderdrc1.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest1. Enter:

hercules adderdrc1.ev

Your active window displays the execution process. While the screen contents may scroll quickly (freeze the screen with Control-s and restart with Control-q), you should be able to note any specific actions or warnings that appear. All of this information appears in the run_details/AD4FUL.sum file (see “AD4FUL.sum” on page 2-13) for your examination. The sample runset file should take only a few seconds to run.

Run File Results

Now that you have run the adderdrc1.ev file, look in the addertest1 directory to see which files Hercules has added. You will find some files that always appear, and others that appear depending on the statements in your runset file. Enter the command:

ls -R

at the *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest1 command line to get a directory listing similar to Example 2-2 (some directories and files are omitted for clarity).

Example 2-2 Directory Listing After Running Hercules

```
AD4FUL.LAYOUT_ERRORS adderdrc1.ev EX_ADDER_1/ group/
AD4FUL.RESULTS adderdrc1_gds.ev EX_ADDER_1.GDS run_details/

./run_details:
AD4FUL.sum   AD4FUL.tree0  AD4FUL.tree3  evaccess
AD4FUL.tech  AD4FUL.tree1  AD4FUL.vcell

./run_details/evaccess:
AD4FUL  AD4FUL.errbin  AD4FUL.errstr  AD4FUL.ev  VA.libs
```

What If My Output Is Not Correct?

If your directory does not match the one described, here is a list of possible problems and solutions.

Problem: Licensing error appears in your hercules.RESULTS file.

Solution: Refer to “Licensing Hercules” on page 1-5 for instructions on installing and verifying your license daemon.

Problem: License daemon is activated incorrectly.

Solution: Check your license file to make sure you have an active Hercules DRC license in your Synopsys license.

Problem: You get the error message:

```
Fatal error #109: Initialization of MILKYWAY input library EX_ADDER_1
failed. Aborting Hercules.
```

Solution: Refer to “Creating Directories and Getting the Files” on page 1-2 in Chapter 1.

Problem: You get a message saying you do not have permission to write.

Solution: Check your file permissions for the current run directory.

Problem: You get a message saying that the disk is full.

Solution: Hercules requires 180 MB of disk space. Check your disk space to make sure there is room.

If you have a problem that is not listed here, please contact Synopsys support.

Explanation of Error and Summary Files

Now we take a closer look at the Hercules output files. The *block*.RESULTS and *block*.LAYOUT_ERRORS files are written to the directory where the Hercules run is executed. If your run was aborted before reading the name of the top block and an error is generated, you see the hercules.RESULTS file.

AD4FUL.RESULTS

The AD4FUL.RESULTS file, shown in Example 2-3, is a high level summary file of the Hercules run. It includes the:

- version of Hercules
- line used to invoke Hercules
- summary of the class of checks (for example, DRC or LVS)
- overall runtime for your entire Hercules job

Be sure to verify that the

- version of Hercules listed in the log file references the latest software you installed
- correct checks were made
- runtimes were within acceptable limits

Example 2-3 AD4FUL.RESULTS File

```
Hercules (R) Hierarchical Design Verification, Release ... DATA OMITTED

  ... RELEASE VERSION OMITTED
  Synopsys.  All rights reserved.

Called as: hercules adderdrc1.ev

- Parsing runset "adderdrc1.ev" ... DONE

- Checking input conditions ... DONE

- Performing DRC checks and/or device extraction ... DONE
    No layout errors

Hercules Run: Time=  0:00:14 User=11.42 System=0.43
Hercules is done.
```

If the Hercules run results in an error, the *block*.RESULTS file tells you to refer to the *block*.LAYOUT_ERRORS file to get more details on this error.

AD4FUL.LAYOUT_ERRORS

The DRC/ERC/EXTRACT checks on the design block generate this error file. In the example, we specified AD4FUL in the HEADER section of the adderdrc1.ev runset file. There are no errors, so the file contents are minimal and contain only some repeated runset information.

Example 2-4 AD4FUL.LAYOUT_ERRORS File (No Errors)

```
                      LAYOUT ERRORS RESULTS: CLEAN

                       #### #     #####   ##   #   #
                      #     #     #      #  #  ##  #
                      #     #     ####  ###### # # #
                      #     #     #     #    # #  ##
                       #### ##### ##### #    # #   #

=====================================================================

Library name:  EX_ADDER_1
Structure name:  AD4FUL

                           ERROR SUMMARY

                           ERROR DETAILS
```

Additional files generated during the Hercules run are stored in the run_details directory. These files include the detailed run summary files, all of the hierarchy related files, and log files. You should first examine the files mentioned above. The following sections contain more information on some of files found under run_details directory.

AD4FUL.sum

The AD4FUL.sum summary file also gets its name from the top block (AD4FUL) specified. The summary file contains information about the steps Hercules executed and the resources used in each step. It repeats much of the information that appeared in the runset file, with the addition of a full list of options in the various runset sections, complete with user and default settings.

Example 2-5 shows the summary file AD4FUL.sum, which you should have in your run_details directory after your Hercules run is completed. Various parts of the summary file are explained in the following sections, with notes throughout the file. Notice that many sections and options appear that were not in your original runset file. These are default settings provided for your information. At the end of the file we find the most important information: that no errors were found during the check (shown in emphasized type at the end of the file).

Example 2-5 AD4FUL.sum File (No Errors)

```
Hercules (R) Hierarchical Design Verification, Release DATA OMITTED

    Synopsys, Inc.  All rights reserved.

The information here shows the source of Hercules, followed by the
command
you typed (hercules adderdrc1.ev) to process the runset.

Called as: hercules adderdrc1.ev

The following line shows the version of EV_Engine used to run the DRC
checking. All polygon processing is done with EV_Engine. The naming
convention for each release of this executable is as follows:
year.quarter.compile#. If there is a patch release the naming convention
changes to: year.quarter.patch#.compile#

EV_Engine (R) Hierarchical Design Rule Checker, Release DATA OMITTED
 Synopsys, Inc.  All rights reserved.

Running Single-Threaded code

The following information shows the names of the input and output files,
 as well as the sources and destinations of each.

Runset file ......................  adderdrc1.ev
Current Directory ................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
Hostname .........................  ddthp40
Platform type ....................  HP64_U11
MILKYWAY input library path ......   /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
MILKYWAY input file name .........  EX_ADDER_1
MILKYWAY block name ..............  AD4FUL
MILKYWAY output library path .....   /remote/wwas1/hercules/venu/
```

```
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
MILKYWAY output file name ........  AD4FUL
MILKYWAY output_block name .......  TOP_ERR
Run details ......................    /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1/
run_details
Group file directory .............   /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1/group
Retain group files ...............  smart
Check reference structures .......  yes
The OPTIONS section lists the settings you used to analyze and display
data, including the default settings your runset did not alter.
OPTIONS {
   ALL_TEXT_GLOBAL=FALSE
   ASCII_ONLY=FALSE
   ATTACH_TEXT=FALSE
   BOX_CORNER=FALSE
   CHECK_REF_LIB=TRUE
   COUNT_TRAPS=FALSE
   ERR_LIMIT_PER_CHECK = UNLIMITED
   ERR_PREFIX = ERR
   EXPLODE_AREFS=FALSE
   EXPLODE_HOLDING_CELL_LIMIT=0
   EXPLODE_LIMIT=0
   FLAG_ALL_AREF_ERRORS=FALSE
   FLAT_COUNT=FALSE
   FLAT_ERROR=FALSE
   GENERATE_INSTANCE_NAME=TRUE
   HIERARCHICAL_DELIMITER = \
   IGNORE_CASE=FALSE
   INCREMENTAL_CELLS=FALSE
   INCREMENTAL_CELLS_FILE =
   INSTANCE_PREFIX =
   NET_PREFIX =
   SELECT_CELL_TO_NO_EXPLODE=TRUE
   EQUIV_TO_NO_EXPLODE=TRUE
   SCHEMATIC_TO_NO_EXPLODE=TRUE
   BLACK_BOX_TO_NO_EXPLODE=TRUE
   PROTOTYPE_PLACEMENTS=FALSE
   NO_MERGE=FALSE
   GENERATE_INSTANCE_NAME=TRUE
   PRINT_ERRSUM_FILE=TRUE
   MAXIMUM_CELLNAME_LENGTH=32
   SIZE_ENDPOINTS=FALSE
   SNAP_RES=TRUE
   SQUARE_CORNER=FALSE
   STOP_ON_GROUP_ERROR=TRUE
   TEXT_RECT=0
   USE_EXPLODED_TEXT=FALSE
   WIDTH=2
   MAGNIFICATION_FACTOR=1
   OUTPUT_MAGNIFICATION_FACTOR=1
   POLYGON_COUNT_IN_ASSIGN = FALSE
```

```
   FLAT_POLYGON_COUNT = FALSE
   CREATE_VUE_OUTPUT = FALSE
   REMOVE_DANGLING_PORT = UNTEXTED
}
The PREPROCESS_OPTIONS allow you to specify 1) the printing of
information
about Hercules' efficiency at processing the design hierarchy; and 2)
the setting of options for path grid checking. This section lists all
available options.

PREPROCESS_OPTIONS {
   CELL_PROFILE = FALSE
   CELL_PROFILE_CNT=20
   CHECK_PATH_ENDPOINTS = TRUE
   CHECK_PATH_45 = TRUE
   CHECK_PATH_90 = FALSE
   DESIGN_STATS = TRUE
   TREE = TRUE
   CELL_STATS = TRUE
   PRINT_PREPROCESS_FILES = TRUE
}
Because we did not change any of the TECHNOLOGY_OPTIONS in our runset,
they are all default hierarchy optimization options.
TECHNOLOGY_OPTIONS {
   VIA_AUTO_EXPLODE = TRUE
   SUBLEAF_AUTO_EXPLODE = 6
   ALLOW_EXPLODE_WITH_TEXT = TRUE
   POST_VCELL_EXPLODE_CELL_SIZE <= 10
   EXPLODE_CELL_SIZE_PERCENT = 70
   CELL_SIZE_AUTO_EXPLODE <= 10
   EXPLODE_AREFS = FALSE
   EXPLODE_1XN_AREFS = FALSE
   EXPLODE_DATA_CELL_LIMIT = 4
   POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
   EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
   EXPLODE_BIG_SPARSE_CELL = TRUE
   EXPLODE_HOLDING_CELL_LIMIT = 1
   EXPLODE_PLACEMENT_LIMIT = 1
   POST_VCELL_EXPLODE_HIER_SPARSE_CELL = TRUE
}

EVACCESS_OPTIONS {
   PATH =  /remote/wwas1/hercules/venu/HERCULES_DOC/tutorial/
Getting_Started_Hercules_DRC/addertest1/run_details/evaccess
   LIBRARY = AD4FUL
   CREATE_MSG_VIEW = TRUE
   CREATE_NETLIST_VIEW = TRUE
   CREATE_XREF_VIEW = TRUE
   CREATE_GRAF_VIEW = TRUE
}
ASSIGN {
   tox    (1)
   poly   (5)
```

```
   well    (31)
   psel    (14)
   cont    (6)
   met1    (8)    text(110)
   met2    (10)
   via     (19)}

DATATYPE_OFFSET=FALSE. There will be no datatype difference
between FRAM and CEL views.
Using existing technology table!
Input Library Format:  127 char cellnames
Output Library Format:  No output library

Preprocessing group files...

NOTE: If no options are specified, Hercules uses the default values for
the TECHNOLOGY options, as explained above.

For each command executed, Hercules reports actual time, user (CPU) time,
system time (I/O and other non-CPU events) and memory usage in megabytes
(MB). This example uses runset explode and flatten default options from
the TECHNOLOGY_OPTIONS section. Later in "Design Structure" on page 29
we will examine the concept of explode and look at the tree1 file for
what is exploded.

Preprocess Step 1 : Exploding
  Preprocessing time = 0:00:00  User=0.13 Sys=0.00 Mem=17.982

VCELL_PASS statements are on by default. They manipulate the hierarchy
for optimal Hercules processing.
TECHNOLOGY_OPTIONS {
   VCELL_PASS   {
      STYLE = PAIRS
      ITERATE_MAX = 15
      ARRAY_ID = TRUE
      EXPLODE_INTO_VCELL = SMART
      MIN_COUNT = 20
      TOP_PERCENT_OF_VALUE = 40
   }
}

Preprocess Step 2 : Vcell_pass Arrays and Pairs Iteration 1
   Pairs time = 0:00:00  User=0.01 Sys=0.00 Mem=6.951

VCELL_PASS 1, no changes.

  Combined VCELL time = 0:00:00  User=0.01 Sys=0.01 Mem=6.951

Preprocess Step 3 : Post-VCell Explodes
  Post VCell time = 0:00:00  User=0.00 Sys=0.00 Mem=6.873

  Determine Region time = 0:00:00  User=0.00 Sys=0.00 Mem=11.239
```

```
0 Total self_intersect errors found.
  Create Layer Setup time = 0:00:00  User=0.00 Sys=0.00 Mem=11.239

  Preprocessing time = 0:00:00  User=0.00 Sys=0.00 Mem=11.317

Snapping for all layers is done within individual CREATE_LAYERS commands
generated internally by Hercules.
Checking database:

CREATE_LAYER tox  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=tox
6 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.395

CREATE_LAYER poly  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=poly
16 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.427

CREATE_LAYER well  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=well
3 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380

CREATE_LAYER psel {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=psel
6 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380

CREATE_LAYER cont {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=cont
38 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380
```

```
CREATE_LAYER met1 met1.TEXT {
   CREATE_TEXT = TRUE
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met1 met1.TEXT
75 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.489

CREATE_LAYER met2 {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met2
25 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.458

CREATE_LAYER via {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=via
53 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.427

HIERARCHY_CLEANUP HIERARCHY = ERR_2.HIERARCHY
   Check time = 0:00:00  User=0.00 Sys=0.01 Mem=9.162

REMOVE_HIERARCHY { HIERARCHY = ERR_1.HIERARCHY }
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.130

CLEAN_PREPROCESSOR
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.130

   Total Create Layer time = 0:00:00  User=0.01 Sys=0.01 Mem=10.130

GRID_CHECK  {
   ASSIGN_LAYERS=TRUE
   tox={ CHECK_45=TRUE }
   poly={ CHECK_45=TRUE }
   well={ CHECK_45=TRUE }
   psel={ CHECK_45=TRUE }
   cont={ CHECK_45=TRUE }
   met1={ CHECK_45=TRUE }
   met2={ CHECK_45=TRUE }
   via={ CHECK_45=TRUE }
}(100)
```

The number of violations appears here after commands that perform checks.

```
In this case, the GRID_CHECK command and EXTERNAL met1 (testing for a
spacing of less than 3 um) yielded no violations.

0 non-45 violations found.
No output written.

   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=6.302

EXTERNAL met1{
   SPACING<3.000 } (101)
0 spacing violations found.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=8.208

smart delete of "met1"

Checks complete.

The "Total check time" shows the total time and the maximum memory used
for completing the checks.

  Total check time = 0:00:00  User=0.03 Sys=0.01 Mem=8.208

  Saving ERR & PERM data time = 0:00:00  User=0.00 Sys=0.00 Mem=8.803
Here is the total time and the maximum memory used for the run (checks,
plus preprocessing):

 Overall ev_engine time = 0:00:12  User=11.44 Sys=0.18 Mem=17.982
```

Because we did not find any errors in our example, there is no need to look at any graphical output. Hercules does not create any graphical output files for a runset file which produces no errors.

Design Structure

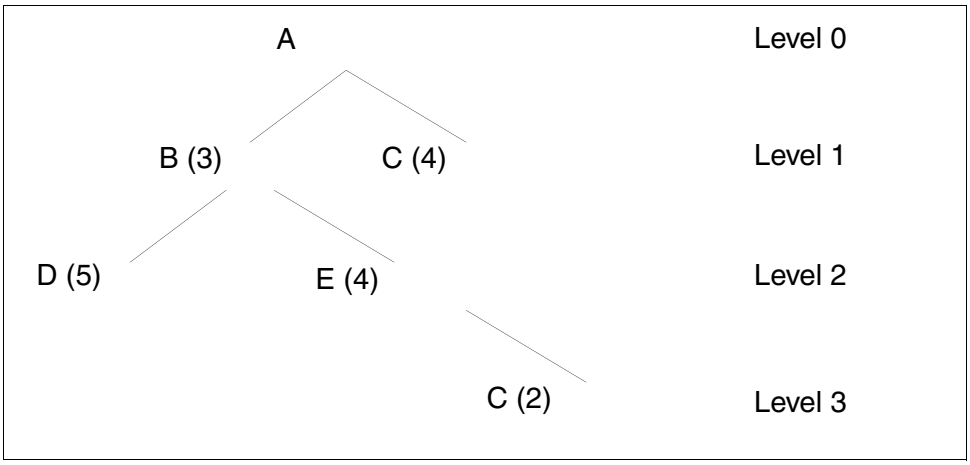
So far, we have discussed individual cells and individual output files. At this point we explain hierarchy so you will better understand the concept of hierarchical output.

As noted previously, our test design is a four-bit full adder arithmetic building block. The adder is built hierarchically, meaning that cells or blocks are placed inside other cells, thus creating design levels. One significant output is the tree file, that analyzes the design hierarchies, or levels. When examining tree files, you can appreciate the difference between true hierarchical design, and inefficient hierarchical designs. Now we discuss the concept of hierarchical design in order to interpret the tree files.

Hierarchical Design

A hierarchical design has nested cells; a cell contains one or more instances of another cell, which can contain one or more instances of still another cell, and so forth. The top-level structure holds all other structures and represents the overview of the entire design. Any structure at the top level is called Level 0. Any structure nested within the Level 0 structure, but one level down, is called a Level 1 structure. Nested within the Level 1 structure can be other structures at Level 2. Level 2 can contain Level 3 structures, and so forth. In essence, you develop a tree of structures, with each structure containing other structures. In Figure 2-2, numbers in parentheses indicate the number of placements, or instances, of a cell in the structure above it. Cell A contains four Cs and three Bs. Cell B contains five Ds and four Es. Cell E contains two Cs.

Figure 2-2 Vertical Hierarchy

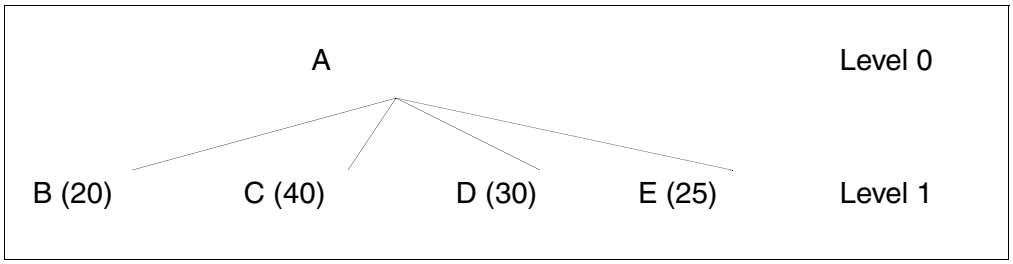


Notice that there are four hierarchical levels (Level 0 to Level 3) of nested cells. There are five unique cells, A to E. To find the number of hierarchical cell placements, add all the placements of a cell at all levels. For example, C is 4+2=6, as there are four C cells at Level 1 and two at Level 3. If you wanted to find the TOTAL or FLAT number of C cells, you would 1) multiply the number of C cells found within the holding cell; and 2) add all the instances of this situation. For example, C = two C cells multiplied by four E holding cells, multiplied by three B holding cells, plus four C cells held directly under A, making a total of 28 C cells. That is, (2 x 4 x 3) + 4 = 28.

Horizontal Design

In a horizontal design, you place all instances of a cell directly inside the Level 0 top cell; that is, there is very little nesting. Therefore, only a few levels of hierarchy exist. In Figure 2-3 there are 30 instances of Cell D directly under A at Level 0. Again, the numbers in parentheses refer to the number of times the cell appears in the structure immediately above.

Figure 2-3 Horizontal Hierarchy



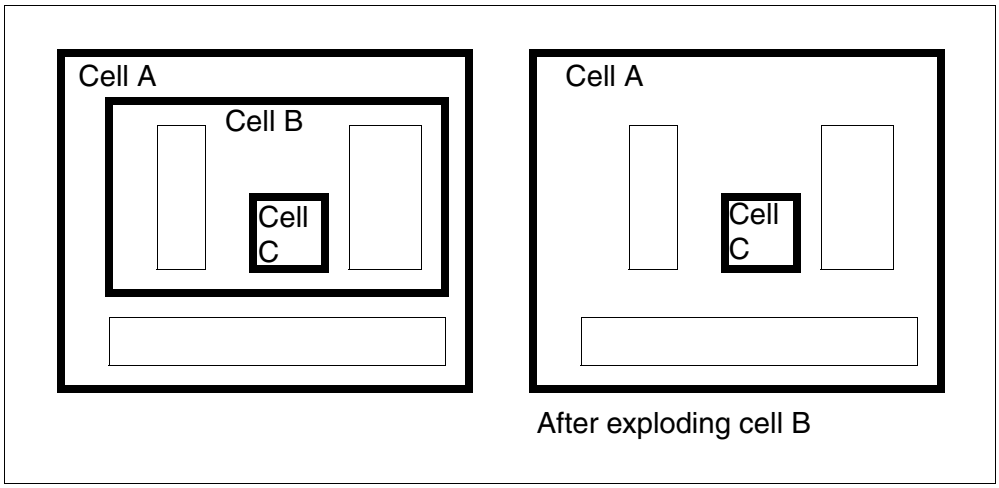
In Figure 2-3, very little nesting is present, with multiple instances of a cell being directly under Level 0. The more horizontal a hierarchy, the closer the number of hierarchical placements is to the number of flat placements. In contrast, the more vertical a hierarchy, the smaller the number of hierarchical placements are in comparison to flat placements. Notice that if the hierarchical design in Figure 2-2 on page 2-20 were flattened out, the 28 C cells would be directly under A.

Efficient Versus Inefficient Hierarchy

For Hercules, the definition of hierarchy implies a repetition of design structures. A hierarchical tool must check the actual polygon data in each cell, as well as the cell-to-cell interactions. To see performance gains through hierarchical processing, cells of a significant size must be repeated. If all cells are unique, so that the number of unique cells equals the number of flat placements, then the design is not really hierarchical, even if it is built using 15 nested levels. Hercules must check all of the polygons in each of the unique cells. Also, if all of the cells placed consist of only a few polygons, then the tool spends as much time checking cell-to-cell interactions as it does checking the actual polygons. This is considered inefficient hierarchy. As you will see in the following sections, Hercules generally explodes inefficient hierarchy for better processing.

A word about explosion is in order here, since the concept appears throughout generated text files. Explode means moving all the polygons to the next level of the hierarchy. The cell holding polygons at a certain level is eliminated, and its polygons move to the next level up, as Figure 2-4 shows.

Figure 2-4 Exploding a Cell - Cell B



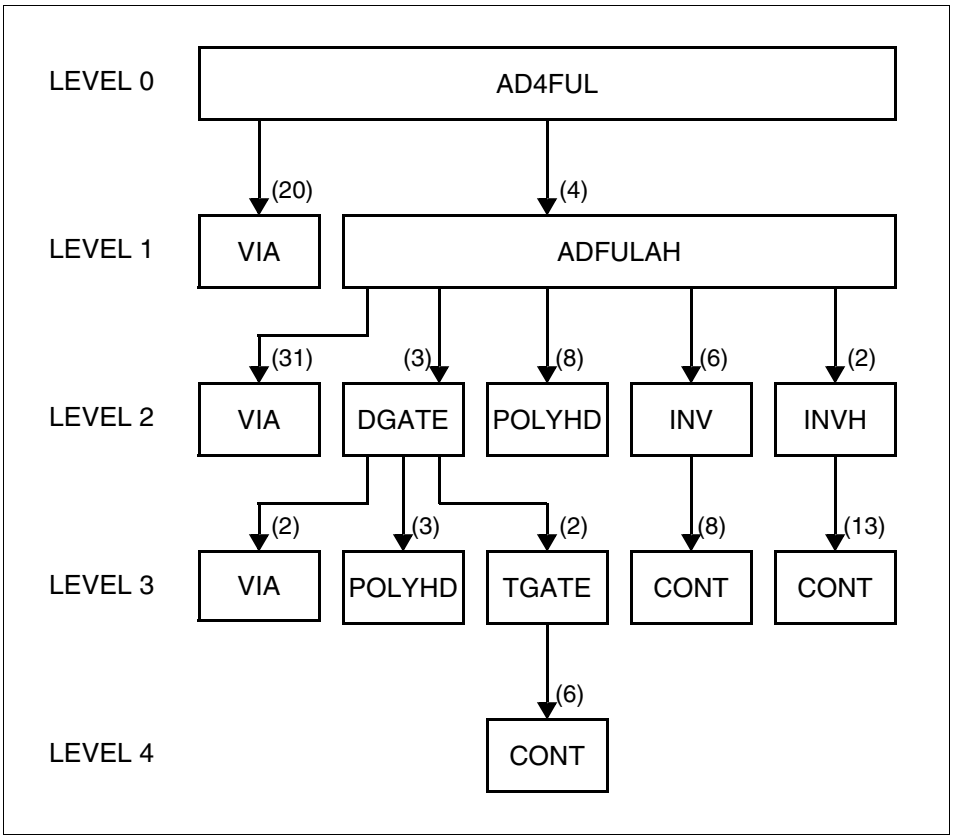
Explanation of Output Tree Files

In light of the discussion above, we examine the AD4FUL tree files in this section.

Hierarchy Tree Files

Figure 2-5 shows a graphical equivalent of the information in the run_details/AD4FUL.tree0 file. The figure contains an extraordinary amount of information, much more than you ever need when analyzing error output. It indicates that the design has five levels of hierarchy, which are chosen by the designer before or during the time the layout is created. It identifies each type of structure in the design, indicating in parentheses how many occurrences of each structure are in AD4FUL. For our tutorial, such a detailed diagram helps you to understand thoroughly all the information Hercules creates. Compare the style of Figure 2-5 to that of Figure 2-2 on page 2-20, and note the similarities of hierarchical structures. For example, there are four ADFULAH cells in AD4FUL, and two INVH cells in each ADFULAH cell.

Figure 2-5 AD4FUL Hierarchy Tree



Assume you have an error in the INV cell. Locate INV in Figure 2-5 at Level 2 in the diagram. The adder uses 24 instances of the INV cell in the design (six INV cells in ADFULAH, and four ADFULAH cells in AD4FUL). But notice that there is really only one structure that contains an error, and this error is repeated in 24 different places on the layout. Hercules checks the design hierarchically, determines that the INV cell has an error, and reports a single error in its output files.

AD4FUL.tree0

The tree0 file lists information about the original hierarchy of the design, such as cell names and where structures appear in the hierarchy. Detailed placement information is shown for the AD4FUL structure in Example 2-6; the tree file includes the placement information for all the structures in the design. Only a portion of this file is shown here.

Example 2-6 Partial AD4FUL.tree0 File

```
*** AD4FUL

PRE_TREE
```

```
      DESIGN STATISTICS DEFINITIONS

Hierarchical levels      The number of levels in the design.
Unique Cells             The number of unique cells in the design.
Hierarchical placements  Total number of placements (srefs and arefs)
                            at each level in the design.
Total Design placements  Total number of placements (srefs and arefs)
                            in the design.
Exploded references      Total number of exploded references in the
                            design.
Exploded placements      Total number of exploded placements in the
                            design.
Total Data count    Total number of all polygons, paths, vectors,
                            and rectangles in the design.
Sref Placements     Total number of cell references in the design.
Arefs               Total number of array references in the design.
Aref Placements     Total number of references in each array
                            reference in the design.
Data           Number of data primitives in each cell of the design.
Text           Number of text primitives in each cell of the design.

*********************************************************************

      DESIGN STATISTICS FOR "AD4FUL".

Compare the number of levels, unique cells, and placements with the
layout
styles in Figure 7, Figure 9, and Figure 10.

   Hierarchical levels        =         5
   Unique cells               =         9
   Hierarchical placements    =       109
   Total design placements    =       749
   Exploded references        =         0
   Exploded placements        =         0
   Total data count           =      2061
   SREF placements            =       749
   AREFS                      =         0
   AREF placements            =         0
   DATA                       =       120
   TEXT                       =         2

*********************************************************************

      HIERARCHICAL TREE FOR BLOCK "AD4FUL".

                                                 TOTAL SREF AREF
 AD4FUL                      Level=0  Count=    1    1    0
  ADFULAH                    Level=1  Count=    4    4    0
   DGATE                     Level=2  Count=    3    3    0
    POLYHD                   Level=3  Count=    3    3    0
    TGATE                    Level=3  Count=    2    2    0
     CONT                    Level=4  Count=    6    6    0
```

```
    VIA                          Level=3  Count=   2    2    0
   INV                           Level=2  Count=   6    6    0
    CONT                         Level=3  Count=   8    8    0
   INVH                          Level=2  Count=   2    2    0
    CONT                         Level=3  Count=  13   13    0
   POLYHD                        Level=2  Count=   8    8    0
   VIA                           Level=2  Count=  31   31    0
  VIA                            Level=1  Count=  20   20    0

*********************************************************************

      CELL STATISTICS
```

Notice that you can compare the number of hierarchical placements to the number of flat placements for each cell. For example, AD4FUL has the same number of flat and hierarchical placements, 1.

```
CELL = AD4FUL
   Cell Status                    = USED
   MILKYWAY Library               =  /remote/wwas1/hercules/venu/
      HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
   MILKYWAY Cell Extents (LL,UR) = (0.000, 0.000) (394.000, 348.000)
   Cell Area                      = 137112.00
   Hierarchical Placements        = 1
   Flat Placements                = 1
   SREFS                          = 24
   AREFS                          = 0
   AREF placements                = 0
   Exploded references            = 0
   Exploded placements            = 0
   DATA                           = 29
   TEXT                           = 0
   Via                            = 0
   Layer Usage of Original Cell:
 Name Data/Text Layer:Dtype Layer Extents (LL,UR) Lyr Area VIEW Ver.Obj
 ---- -------- ---------- -------------------  -------  ---- ------
 met1   20/0     8:0    (0.00, 18.50) (394.00, 337.50)  1458.50  CEL  1
 met2    9/0     10:0   (0.50, 0.00)  (372.00, 348.00)  3348.00  CEL  1
```

For the VIA cell, there are 53 hierarchical placements and 163 flat placements, but the cell only has 3 polygons. You will notice in later tree files that the VIA cell is AUTO_EXPLODED. Because of the large number
of placements versus the small polygon count, it is considered inefficient
hierarchy

```
CELL = VIA
   Cell Status                    = USED
   MILKYWAY Library               = /remote/wwas1/hercules/venu/
      HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
   MILKYWAY Cell Extents (LL,UR) = (0.000, 0.000) (4.000, 4.000)
   Cell Area                      = 16.00
```

```
   Hierarchical Placements       = 53
   Flat Placements               = 168
   SREFS                         = 0
   AREFS                         = 0
   AREF placements               = 0
   Exploded references           = 0
   Exploded placements           = 0
   DATA                          = 3
   TEXT                          = 0
   Via                           = 1
   Layer Usage of Original Cell:

 Name Data/Text Layer:Dtype Layer Extents (LL,UR) Lyr Area VIEW Ver.Obj
 ---- --------  ----------- --------------------  -------  ---- ------
 met1   1/0      8:0        (0.50, 0.50) (3.50, 3.50) 9.00  CEL  1
 met2   1/0      10:0       (0.00, 0.00) (4.00, 4.00) 16.00 CEL  1
 via    1/0      19:0       (1.00, 1.00) (3.00, 3.00) 4.00  CEL  1

..................... DATA OMITTED ....................

Notice the complete summary of all the polygon and text data found in
each cell. If you want to verify that you have read in the correct
polygon
layer or text layer, you can check this file to make sure that the number
of elements matches the number you think are in your design.
```

ADFUL.tree1

A partial ADFUL.tree1 file in Example 2-7 describes the structures after explicit explode and flatten operations; see the explode and flatten options in the TECHNOLOGY_OPTIONS section. Since all of the hierarchy changes are made during the tree1 phase, Hercules automatically determines that it would be more advantageous to process this resulting hierarchy than the original hierarchy it was given. The 1 after tree represents the number of preprocess steps that Hercules has performed on your design. In Example 2-7, we show the salient aspects of the tree1 file. (Sections that are the same as the tree0 file have been omitted.) Pay particular attention to emphasized items and compare the text closely with Figure 2-5 on page 2-23.

Example 2-7 Partial tree1 File

```
*** AD4FUL
POST_TREE
      DESIGN STATISTICS DEFINITIONS

Compare the following to the tree0 file

      DESIGN STATISTICS FOR "AD4FUL".

   Hierarchical levels      = 4
   Unique cells             = 6
   Hierarchical placements  = 18
```

```
   Total design placements  = 73
   Exploded references      = 8
   Exploded placements      = 91
   Total data count         = 2061
   SREF placements          = 73
   AREFS                    = 0
   AREF placements          = 0
   DATA                     = 332
   TEXT                     = 2

*************************************************************************
      HIERARCHICAL TREE FOR BLOCK "AD4FUL".
                                          TOTAL SREF AREF
 AD4FUL                   Level=0  Count=   1    1    0
  ADFULAH                 Level=1  Count=   4    4    0
   DGATE                  Level=2  Count=   3    3    0
    TGATE                 Level=3  Count=   2    2    0
   INV                    Level=2  Count=   6    6    0
   INVH                   Level=2  Count=   2    2    0

*************************************************************************

      CELL STATISTICS
CELL = AD4FUL
   Hierarchical Placements        = 1
   Flat Placements                = 1
   SREFS                          = 4
   AREFS                          = 0
   AREF placements                = 0
   Exploded references            = 1
   Exploded placements            = 20
   DATA                           = 89
   TEXT                           = 0
   Via                            = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   met1     40      0
   met2     29      0
   via      20      0

CELL = ADFULAH
   Hierarchical Placements    = 4
   Flat Placements            = 4
   SREFS                      = 11
   AREFS                      = 0
   AREF placements            = 0
   Exploded references        = 2
   Exploded placements        = 39
   DATA                       = 156
   TEXT                       = 0
   Via                        = 0
   Current Data Counts for Cell:
```

```
   Name    Data   Text
   -----   -----  -----
   poly     12      0
   cont     8       0
   met1     59      0
   met2     46      0
   via      31      0

CELL = INV
   Hierarchical Placements      = 6
   Flat Placements              = 24
   SREFS                        = 0
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 1
   Exploded placements          = 8
   DATA                         = 19
   TEXT                         = 0
   Via                          = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   tox      2       0
   poly     3       0
   well     1       0
   psel     2       0
   cont     8       0
   met1     3       0

CELL = DGATE
   Hierarchical Placements      = 3
   Flat Placements              = 12
   SREFS                        = 2
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 2
   Exploded placements          = 5
   DATA                         = 23
   TEXT                         = 0
   Via                          = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   poly     7       0
   cont     3       0
   met1     8       0
   met2     3       0
   via      2       0

CELL = INVH
   Hierarchical Placements      = 2
   Flat Placements              = 8
   SREFS                        = 0
```

```
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 1
   Exploded placements          = 13
   DATA                         = 29
   TEXT                         = 0
   Via                          = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   tox      2       0
   poly     7       0
   well     1       0
   psel     3       0
   cont     13      0
   met1     3       0

CELL = TGATE
   Hierarchical Placements      = 2
   Flat Placements              = 24
   SREFS                        = 0
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 1
   Exploded placements          = 6
   DATA                         = 16
   TEXT                         = 2
   Via                          = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   tox      2       0
   poly     2       0
   well     1       0
   psel     1       0
   cont     6       0
   met1     4       2

CELL = VIA
   Hierarchical Placements      = 0
   Flat Placements              = 0
   SREFS                        = 0
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 0
   Exploded placements          = 0
   DATA                         = 3
   TEXT                         = 0
   Via                          = 1
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   met1     1       0
```

```
   met2     1       0
   via      1       0

CELL = POLYHD
   Hierarchical Placements      = 0
   Flat Placements              = 0
   SREFS                        = 0
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 0
   Exploded placements          = 0
   DATA                         = 3
   TEXT                         = 0
   Via                          = 1
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   poly     1       0
   cont     1       0
   met1     1       0

CELL = CONT
   Hierarchical Placements      = 0
   Flat Placements              = 0
   SREFS                        = 0
   AREFS                        = 0
   AREF placements              = 0
   Exploded references          = 0
   Exploded placements          = 0
   DATA                         = 1
   TEXT                         = 0
   Via                          = 1
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   cont     1       0
..................... DATA OMITTED ....................
```

AD4FUL.tree3

The tree3 file lists information about the hierarchy of the design after Hercules has internally optimized the way it processes the hierarchy. Here, as indicated by 3 after tree, Hercules has performed three preprocess steps on your design. For very large designs the number is higher. Hercules does not print out all of these files, but instead prints the first two, tree0 and tree1, and then the final statistics in the last file, which is tree3 in this example. You notice that there are usually fewer levels of hierarchy in the tree3 file than there were in the tree0 file. However, in our example, the tree1 file and tree3 file show the same hierarchy. Hercules automatically runs five preprocessing passes on all designs to optimize hierarchy, but, in the case of this design, Hercules found the hierarchy to be optimal after the first step of

preprocessing. For a more detailed explanation of hierarchy processing and how you can control it, refer to the TECHNOLOGY_OPTIONS section in the *Hercules Reference Manual*. Only a portion of this file is shown in Example 2-8.

Example 2-8 Partial AD4FUL.tree3 File

```
*** AD4FUL

AFTER removing temp no_explodes & post-vcell processing

      DESIGN STATISTICS DEFINITIONS

Hierarchical levels       Number of levels in the design.
Unique Cells              Number of unique cells in the design.
Hierarchical placements   Total number of placements (srefs and arefs)
                             at each level in the design.
Total Design placements   Total number of placements (srefs and arefs)
                             in the design.
Exploded references  Total number of exploded references in the design.
Exploded placements  Total number of exploded placements in the design.
Total Data count     Total number of all polygons, paths, vectors,
                             and rectangles in the design.
Sref Placements      Total number of cell references in the design.
Arefs                Total number of array references in the design.
Aref Placements      Total number of references in each array
                             reference in the design.
Data             Number of data primitives in each cell of the design.
Text             Number of text primitives in each cell of the design.

*********************************************************************

      DESIGN STATISTICS FOR "AD4FUL".

Compare the following levels to the tree0 file:
   Hierarchical levels      = 4
   Unique cells             = 6
   Hierarchical placements  = 18
   Total design placements  = 73
   Exploded references      = 7
   Exploded placements      = 71
   Total data count         = 2061
   SREF placements          = 73
   AREFS                    = 0
   AREF placements          = 0
   DATA                     = 332
Notice that the DATA count stays the same as it was in the tree0 file.

   TEXT                     = 2

*********************************************************************

      HIERARCHICAL TREE FOR BLOCK "AD4FUL".
```

```
                                          TOTAL SREF AREF
 AD4FUL                  Level=0  Count=   1    1    0
  ADFULAH                Level=1  Count=   4    4    0
   DGATE                 Level=2  Count=   3    3    0
    TGATE                Level=3  Count=   2    2    0
   INV                   Level=2  Count=   6    6    0
   INVH                  Level=2  Count=   2    2    0

*********************************************************************

      CELL STATISTICS

CELL = AD4FUL
   Cell Status              = USED
   MILKYWAY Library         = /remote/wwas1/hercules/venu/
      HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
     MILKYWAY Cell Extents (LL,UR) = (0.000, 0.000) (394.000, 348.000)
   Cell Area                = 137112.00
   Hierarchical Placements  = 1
   Flat Placements          = 1
   SREFS                    = 4
   AREFS                    = 0
   AREF placements          = 0
   Exploded references      = 0
   Exploded placements      = 0
   DATA                     = 89
   TEXT                     = 0
   Via                      = 0
   Current Data Counts for Cell:
   Name    Data   Text
   -----   -----  -----
   met1     40      0
   met2     29      0
   via      20      0

CELL = ADFULAH
   Cell Status              = USED
   MILKYWAY Library         = /remote/wwas1/hercules/venu/
      HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
   MILKYWAY Cell Extents (LL,UR) = (0.000, -18.500) (376.000, 70.500)
   Cell Area                = 33464.00
   Hierarchical Placements  = 4
   Flat Placements          = 4
   SREFS                    = 11
   AREFS                    = 0
   AREF placements          = 0
   Exploded references      = 2
   Exploded placements      = 39
   DATA                     = 156
   TEXT                     = 0
   Via                      = 0
   Current Data Counts for Cell:
   Name    Data   Text
```

```
   -----    -----   -----
   poly      12       0
   cont      8        0
   met1      59       0
   met2      46       0
   via       31       0

..................... DATA OMITTED .....................

CELL = VIA
   Cell Status              = AUTO_EXPLODED (DATA_CELL_LIMIT CELL)
   MILKYWAY Library         = /remote/wwas1/hercules/venu/
      HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest1
   MILKYWAY Cell Extents (LL,UR) = (0.000, 0.000) (4.000, 4.000)
   Cell Area                = 16.00
   Hierarchical Placements  = 0
   Flat Placements          = 0
   SREFS                    = 0
   AREFS                    = 0
   AREF placements          = 0
   Exploded references      = 0
   Exploded placements      = 0
   DATA                     = 3
   TEXT                     = 0
   Via                      = 1
   Current Data Counts for Cell:
   Name     Data    Text
   -----    -----   -----
   met1      1        0
   met2      1        0
   via       1        0

..................... DATA OMITTED .....................
```

Miscellaneous Output Files

The following sections explain the other output files in the run_details directory.

AD4FUL.tech and AD4FUL.vcell

As we mentioned earlier, Hercules automatically analyzes the input hierarchy of your design to determine the best hierarchy and to generate optimal Hercules results. The AD4FUL.tech and AD4FUL.vcell files are summaries of the hierarchy processing that Hercules performed on your design. For details of the options referred to in these files, refer to the TECHNOLOGY_OPTIONS section in the *Hercules Reference Manual*.

evaccess/

An evaccess/ directory is created for EVaccess database files. The only file of general interest to the user is AD4FUL.ev.

evaccess/AD4FUL.ev

Examine the AD4FUL.ev file, shown in Example 2-9, noting all explicit assignments of variables (such as HEADER variables) and runset programming options. For example, if you use an environment variable in an IF/ELSE statement inside the adder1drc.ev runset, the AD4FUL.ev file includes only the commands that are executed based on the value of the runset environment variable.

Example 2-9 AD4FUL.ev File

```
HEADER {
   INLIB  = EX_ADDER_1
   INLIB_PATH = ./              /* For Synopsys internal use only! */
   OUTLIB = EX_ADDER_1
   OUTLIB_PATH = ./             /* For Synopsys internal use only! */
   BLOCK = AD4FUL
   GROUP_DIR =  /remote/wwas1/hercules/venu/HERCULES_DOC/tutorial/
      Getting_Started_Hercules_DRC/addertest1/group
   FORMAT = MILKYWAY
   OUTPUT_FORMAT = MILKYWAY
   LAYOUT_PATH = /remote/wwas1/hercules/venu/HERCULES_DOC/tutorial/
      Getting_Started_Hercules_DRC/addertest1
   OUTPUT_BLOCK = TOP_ERR
   COMPARE_DIR = /remote/wwas1/hercules/venu/HERCULES_DOC/tutorial/
      Getting_Started_Hercules_DRC/addertest1/compare
   RUN_DETAILS_DIR =  /remote/wwas1/hercules/venu/HERCULES_DOC/
      tutorial/Getting_Started_Hercules_DRC/addertest1/run_details
}
TECHNOLOGY_OPTIONS {
   EXPLODE_AREFS = FALSE
   EXPLODE_CELL_SIZE_PERCENT = 70
   EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
   EXPLODE_BIG_SPARSE_CELL = TRUE
   POST_VCELL_EXPLODE_CELL_SIZE <= 10
   BAR_AUTO_EXPLODE=TRUE
   POST_VCELL_EXPLODE_LOW_MEMORY=TRUE
   VIA_AUTO_EXPLODE = TRUE
   SUBLEAF_AUTO_EXPLODE = 6
   CELL_SIZE_AUTO_EXPLODE <= 10
   EXPLODE_DATA_CELL_LIMIT = 4
   POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
   POST_VCELL_EXPLODE_HIER_SPARSE_CELL = TRUE
   EXPLODE_HOLDING_CELL_LIMIT = 1
   EXPLODE_PLACEMENT_LIMIT = 1
VCELL_PASS   {
```

```
      STYLE = PAIRS
      ITERATE_MAX = 15
      ARRAY_ID = TRUE
      EXPLODE_INTO_VCELL = SMART
      MIN_COUNT = 20
      TOP_PERCENT_OF_VALUE = 40
   }
}
EVACCESS_OPTIONS {
   PATH = /remote/wwas1/hercules/venu/HERCULES_DOC/tutorial/
      Getting_Started_Hercules_DRC/addertest1/run_details/evaccess
   LIBRARY = AD4FUL
   CREATE_MSG_VIEW = TRUE
   CREATE_NETLIST_VIEW = TRUE
   CREATE_XREF_VIEW = TRUE
   CREATE_GRAF_VIEW = TRUE
   CREATE_RUNSET_VIEW = TRUE
}
OPTIONS {
   WIDTH = 2
   IUO_SNAP_VALUES = [ 0.01 ]
}
PREPROCESS_OPTIONS {
   CHECK_PATH_90 = FALSE
}
ASSIGN {
   tox    (1)
   poly   (5)
   well   (31)
   psel   (14)
   cont   (6)
   met1   (8)   text(110)
   met2   (10)
   via    (19)} HIERARCHY = ERR_1.HIERARCHY
CREATE_LAYER tox  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=tox
CREATE_LAYER poly {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=poly
CREATE_LAYER well {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=well
CREATE_LAYER psel {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
```

```
} TEMP=psel
CREATE_LAYER cont {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=cont
CREATE_LAYER met1 TEXT=met1.TEXT {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met1 TEXT=met1.TEXT
CREATE_LAYER met2 {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met2
CREATE_LAYER via {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=via
HIERARCHY_CLEANUP HIERARCHY = ERR_2.HIERARCHY
CLEAN_PREPROCESSOR
GRID_CHECK  {
   ASSIGN_LAYERS=TRUE
   tox={ CHECK_45=TRUE }
   poly={ CHECK_45=TRUE }
   well={ CHECK_45=TRUE }
   psel={ CHECK_45=TRUE }
   cont={ CHECK_45=TRUE }
   met1={ CHECK_45=TRUE }
   met2={ CHECK_45=TRUE }
   via={ CHECK_45=TRUE }
}(100)
EXTERNAL met1 {
   SPACING<3 } (101)
```

Essentially, the AD4FUL.ev file is exactly what Hercules is executing while it is running. The AD4FUL.ev file, created during the Hercules run, is very similar to the original adderdrc1.ev runset file with comments removed. The AD4FUL.ev also includes EVACCESS_OPTIONS and TECHNOLOGY_OPTIONS sections with default assignments. Because these defaults are all acceptable for our tutorial, we do not include these sections in our runset.

What's Next?

Thus far you have compared runsets and examined output files to learn about the information they contain. In Chapter 3, "Single Design Error for DRC," we examine a design with one error and introduce you to the different output formats for viewing this error, including Hercules-VUE, a graphical debug tool. All users should proceed to Chapter 3.

3

Single Design Error for DRC

In this chapter we introduce a DRC error into the design of an inverter (INV) in a full adder (ADDER).

We study the differences between the error-free Hercules run from Chapter 2, “An Error-Free Design for DRC,” and the run with errors in this chapter. We also view the graphical output files in Enterprise, the Synopsys layout tool provided with Hercules-VUE for viewing errors.

Summary of Progress to This Point

In our first example you became familiar with Hercules and the files created during a run. You also learned the major components of a simple, yet complete, runset file. Finally, you ran Hercules DRC on an error-free design to examine output files and establish a reference for clean design files. In this step you also verified that your Hercules software installation and licensing were correct.

Learning Objectives for This Chapter

For the single rule error example, a design that contains a single DRC rule violation, do the following:

- Run Hercules on the adder design with an error introduced, and then learn how to interpret the output files to locate errors.
- Examine the graphic output for the Hercules run in Enterprise. This output illustrates how hierarchical checking works and helps you to spot and understand design errors.

Before You Start

Your account should already be set up for this example. If it is not, see Chapter 1, “Installation and Setup.” Otherwise, you will not be able to do this tutorial.

Running Hercules on the Runset File

Be sure that you are in the directory where your adderdrc2.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest2.

Notice that the adderdrc2.ev runset file is almost identical to the adderdrc1.ev file, with two exceptions. Two lines in the file are different from adderdrc1.ev:

- `INLIB = EX_ADDER_2`
- `OUTLIB = EX_ADDER_2_OUT`

The runset file is now referring to our second sample library, EX_ADDER_2, for the input and output libraries.

Example 3-1 Header Section of adderdrc2.ev

```
/* HDRC SAMPLE RUNSET */

/* -------------------------------- Runset Header information */
```

```
HEADER {
   LAYOUT_PATH = ./
   INLIB = EX_ADDER_2
   BLOCK = AD4FUL
   OUTLIB= EX_ADDER_2_OUT
   OUTPUT_BLOCK= TOP_ERR
   GROUP_DIR = group
   FORMAT = MILKYWAY
}
```

Hercules now processes the runset file to perform a check on the block AD4FUL in the EX_ADDER_2 library. The block contains a single error.

Have Hercules process the runset file, adderdrc2.ev. Enter:

hercules adderdrc2.ev

As before, your active window displays the execution process.

Output Results

Running Hercules on the runset file adderdrc2.ev creates the AD4FUL.LAYOUT_ERRORS file in the current directory, with a known error that violates the EXTERNAL spacing check. Notice that the directory structure has not changed. However, the files created by Hercules now contain different information (as emphasized in Example 3-2).

AD4FUL.LAYOUT_ERRORS

Example 3-2 AD4FUL.LAYOUT_ERRORS File (with Errors)

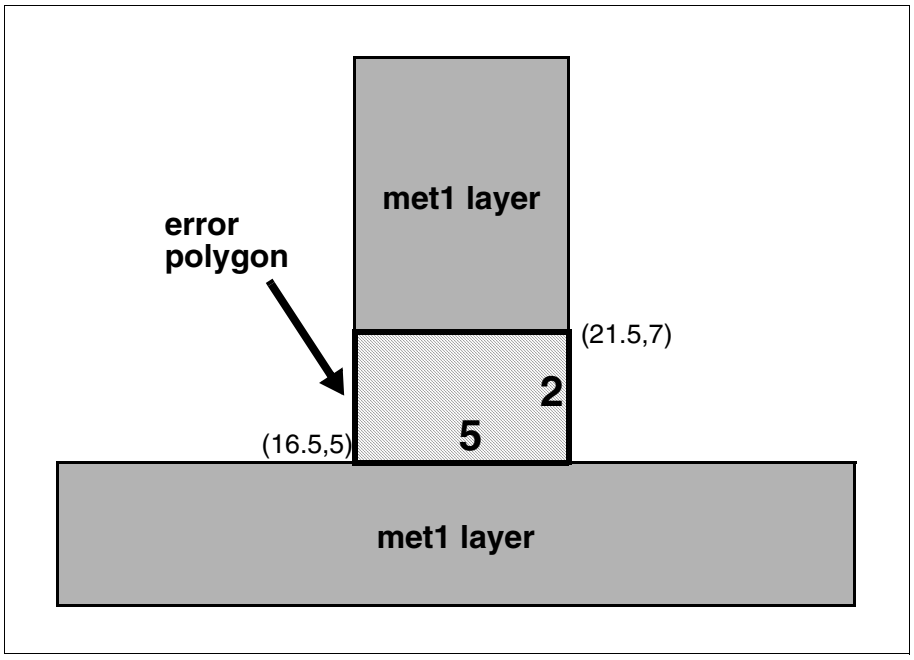
```
                     LAYOUT ERRORS RESULTS: ERRORS

                    ##### ####  ####   ###  ####   ####
                    #     #   # #   # #   # #   # #
                    ####  ####  ####  #   # ####   ###
                    #     #  #  #  #  #   # #  #      #
                    ##### #   # #   #  ###  #   # ####

================================================================
Library name: EX_ADDER_2
Structure name: AD4FUL

                          ERROR SUMMARY

No Comment
 EXTERNAL met1 { } (101) ........................... 1 violation found.

                          ERROR DETAILS
```

```
#####--- ERR_EXTERNAL ----------------------------------

EXTERNAL met1 {
        SPACING<3
        WIDTH=2 } (101)

- - - - -  - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure  ( lower left x, y ) ( upper right x, y )   Distance

- - - - - - -- - - - - - - - - - - - - - - - - - - - - - - - -
INV         (16.500, 5.000)     (21.500, 7.000)       2.000
```

Geometric Representation of Error

The error file shows that an error was generated for the EXTERNAL runset file check, and was written to the AD4FUL.LAYOUT_ERRORS file (Example 3-2) as error polygon coordinates. An error polygon is a set of coordinates that pinpoints the location and magnitude of the violation. In this case, the coordinate values are measured from the origin (0,0) of the cell, the lower left-hand corner of the whole INV cell. The single-error polygon, represented graphically in Figure 3-1, describes an error polygon 5 µm wide and 2 µm high:

Figure 3-1 EXTERNAL Check Error in Cell INV



The error file tells us that we have only a 2-µm spacing between adjacent metal1 objects. The coordinates pinpoint the actual location of the error within the cell INV. Later, we discuss this INV cell with its error as the ERR_INV structure.

AD4FUL.sum

The AD4FUL.sum summary file in the run_details directory provides a quick way to scan through your checks and notice any violations. The summary file looks very similar to the one created for our clean run, except that it warns that an error has been produced. Use your system editor, such as vi (or use the UNIX command more) to view the AD4FUL.sum text file. Near the end of the file you see:

```
EXTERNAL met1 {
        SPACING<3 } (101)
WARNING - 1 spacing violation found.
Check time = 0:00:00  User=0.00 Sys=0.02 Mem=8.279
```

You should locate this error (shown here emphasized) in the AD4FUL.sum file in Example 3-3. Also, notice near the bottom of the file the emphasized section specifying the time it takes to save the error structure.

```
Saving ERR & PERM data time = 0:00:01  User=0.14 Sys=0.03
Mem=9.889
```

Because there were no errors in our first example, this line showed that zero seconds were used to save the error structure. Major sections of the AD4FUL.sum file have been omitted from Example 3-3, because they are identical to those in the AD4FUL.sum for addertest1.

Example 3-3 AD4FUL.sum File (with Errors)

```
Hercules (R) Hierarchical Design Verification, ... DATA OMITTED
  Synopsys.  All rights reserved.

Called as: hercules adderdrc2.ev

...RELEASE VERSION OMITTED
  Synopsys.  All rights reserved.
Running Single-Threaded code

Runset file .....................  adderdrc2.ev
Current Directory ...............  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest2
Hostname ........................  ddthp44
Platform type ...................  HP64_U11
MILKYWAY input library path ......  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest2/
MILKYWAY input file name .........  EX_ADDER_2
MILKYWAY block name ..............  AD4FUL
MILKYWAY output library path .....  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest2/
MILKYWAY output file name ........  EX_ADDER_2_OUT
MILKYWAY output_block name .......  TOP_ERR
Run details .....................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest2/
Group file directory .............  /remote/wwas1/hercules/venu/
```

```
HERCULES_DOC/tutorial/Getting_Started_Hercules_DRC/addertest2/
group
Retain group files ..............  smart
Check reference structures .......  yes

...........OPTIONS SECTION OMITTED...........

EXTERNAL met1 {
   SPACING<3
WARNING - 1 spacing violation found.
   Check time = 0:00:00  User=0.00 Sys=0.02 Mem=8.279

Checks complete.
  Total check time = 0:00:00  User=0.03 Sys=0.03 Mem=8.279

...........DATA OMITTED...........

 Overall ev_engine time = 0:00:30  User=25.66 Sys=0.53 Mem=17.982
```

Running Enterprise and Viewing Results

The following section assumes that you have a copy of Enterprise on your system. Hercules-VUE, the Synopsys graphical debug tool for Hercules, is packaged with a view-only license for Enterprise to help you view your results easily. The installation instructions for Enterprise are included in your Hercules installation. If you are using the Virtuoso Layout Editor from Cadence, refer to Chapter 6, “Hercules Migration With Hercules-VUE,” for details on error viewing. If you are using a different graphical tool, contact Synopsys Technical Support for details on how to import Hercules files into your program.

Note:

Many of you use Hercules-VUE, the Synopsys graphical debug tool designed for viewing errors in Hercules. This section of the tutorial is included to make you familiar with the input design hierarchy and the error hierarchy that Hercules generates.

How to run Enterprise

Make sure you are in the *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest2 directory. Enter:

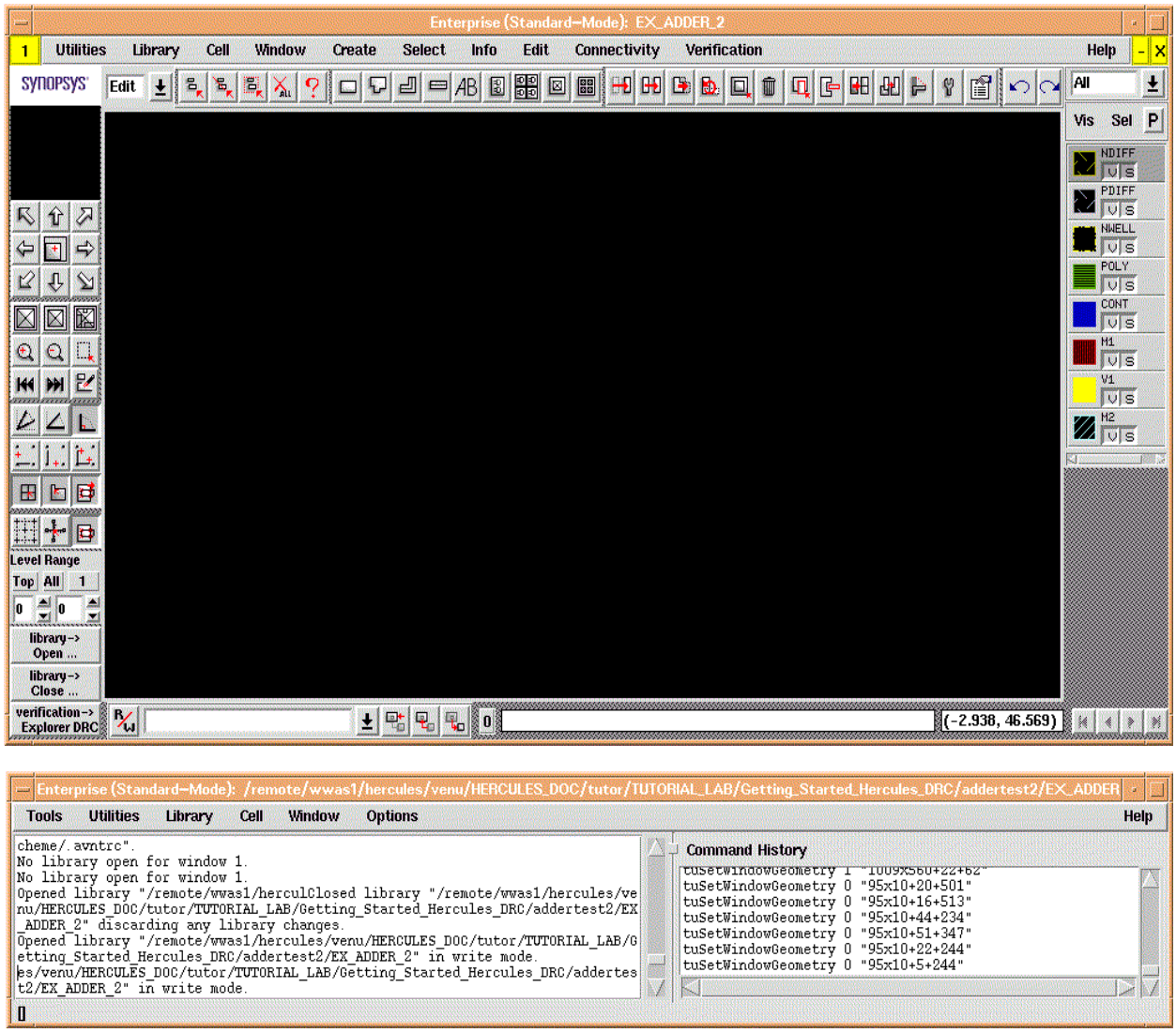
Enterprise &

(The ampersand (&) runs Enterprise as a background process.)

The initial Enterprise Display

You should see the following initial Enterprise display:

Figure 3-2 Opening Enterprise Display (Windows Not to Scale)



Viewing the EX_ADDER_2 Library File Names

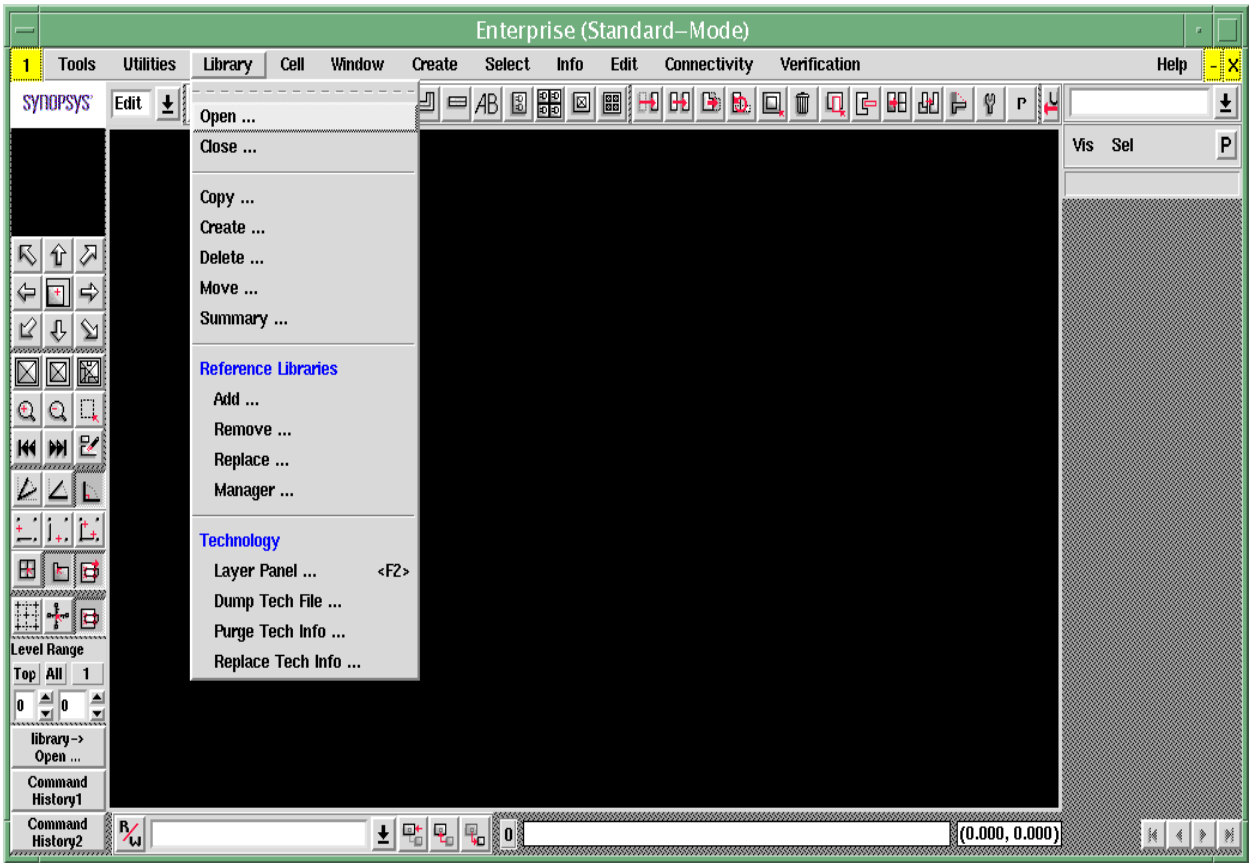
You start by opening a library and its files.

In the graphical window, choose Library > Open. (When you select a command, you see it automatically typed in the command window.) You can also invoke the command by placing your cursor in the command window in the lower left corner of the screen and enter:

gxwOpenLibrary

You see a pop-up window prompting you to enter the library name.

Figure 3-3 Library > Open

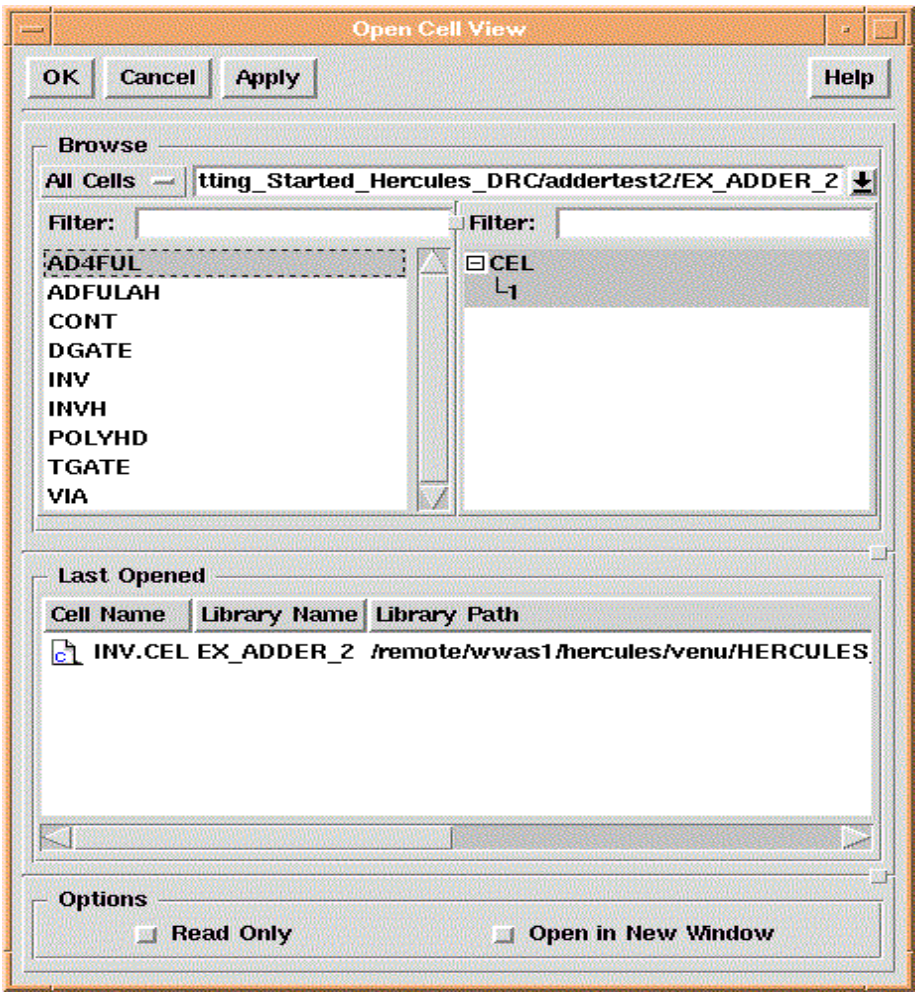


Enter the library name EX_ADDER_2. You can also browse through the directories to select the library.

In the graphical window, choose Cell > Open. You can also invoke the command by placing your cursor in the command window in the lower left corner of the screen by entering:

gxwOpenCellView

Figure 3-4 EX_ADDER Library Structure Names



You see a pop-up window prompting you to enter the cell name. A list of cells for our design appears on the screen, as shown in Figure 3-4. Double-click on cell AD4FUL to open the cell.

EX_ADDER Input and Output Files

The next paragraphs explain the input and output files. Following that explanation, we return to Enterprise to view some cells.

In our runset, all errors are sent to the error hierarchy. The three files prefixed with ERR are called error hierarchy files because they show the three levels of hierarchy: ERR_AD4FUL, ERR_ADFULAH, and ERR_INV (the cell with the error).

Note:
All error hierarchy files begin with ERR as a default, unless you choose a different prefix and set it in the OPTIONS section of the runset file (ERR_PREFIX option).

Locate INV in the hierarchy tree in Figure 2-5 on page 2-23 in Chapter 2, “An Error-Free Design for DRC,” and notice that the cell occurs in ADFULAH and also in the top level, AD4FUL. Recall from the previous section on design structure that the error occurs in INV (ERR_INV), but Hercules outputs all the structures (ERR_ADFULAH and ERR_AD4FUL) that contain a reference to the cell. These ERR_ files appear only when Hercules finds a design error.

TOP_ERR is the holding structure for errors, as defined by OUTPUT_BLOCK in the runset HEADER section, and any other permanent layers that you might create. Permanent layers are explained in Chapter 4, “A Complete Design for DRC.” The TOP_ERR file appears only when Hercules finds a design error.

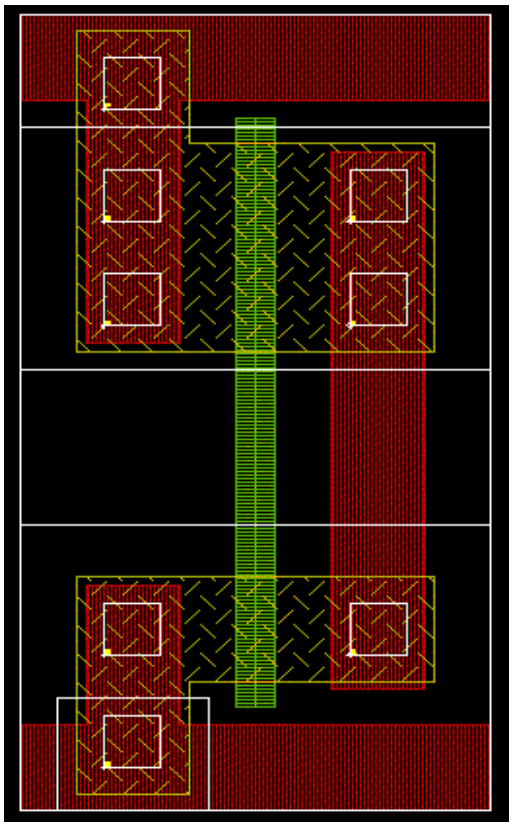
In our layout, you have to fix only the cell INV. Because the design is hierarchical, and because Hercules retains the hierarchy throughout the checking procedure, fixing INV fixes all references to INV throughout the design.

In Enterprise, select Cell > Close to close cell AD4FUL.

Viewing Data in Enterprise

Now look at the design cell that contains the error, INV:

Figure 3-5 INV Cell with Errors

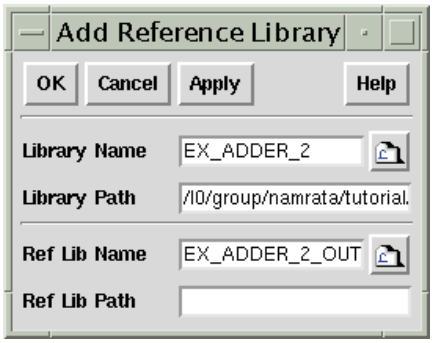


Because you have not looked at the error polygon, you may not be able to spot the area that contains the design rule violation. Also, at this point you are not familiar with the layout, and no dimensioned grid is shown.

Now execute the next series of commands to reference the output library. This allows you to overlay the dimensional error on your original design.

Choose Library > Reference Libraries > Add.

Figure 3-6 Add Reference Library

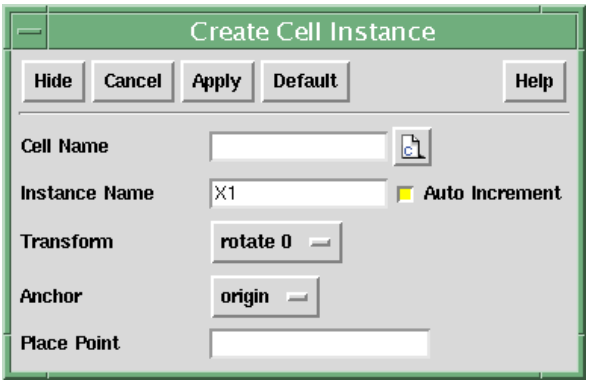


Add the EX_ADDER_2_OUT library as the reference library for the EX_ADDER_2 library as shown in Figure 3-6. Select OK.

Choose Cell > Open. Double-click on cell INV to open.

Choose Create > Cell Instance.

Figure 3-7 Create Cell Instance

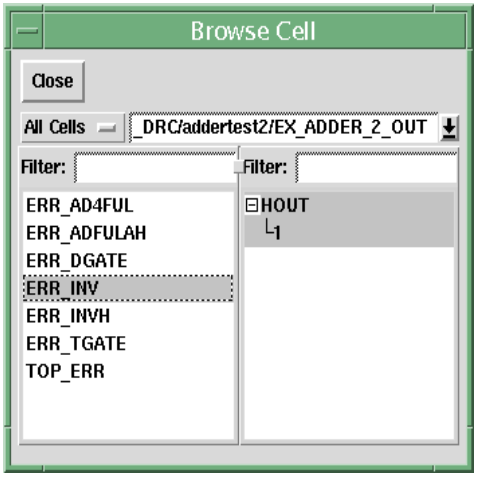


A Create Cell Instance pop-up window appears, as shown in Figure 3-7. Select the browse symbol next to the Cell Name field to browse list of cells.

Select the EX_ADDER_2_OUT library by choosing the down arrow key of the All Cells option in the Browse Cell window (see Figure 3-8).

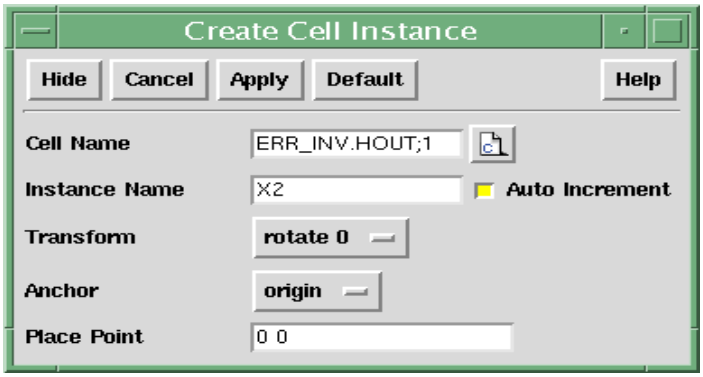
Select cell ERR_INV and select Close. This adds the cell name to the Create Cell Instance window.

Figure 3-8 Browse Cell



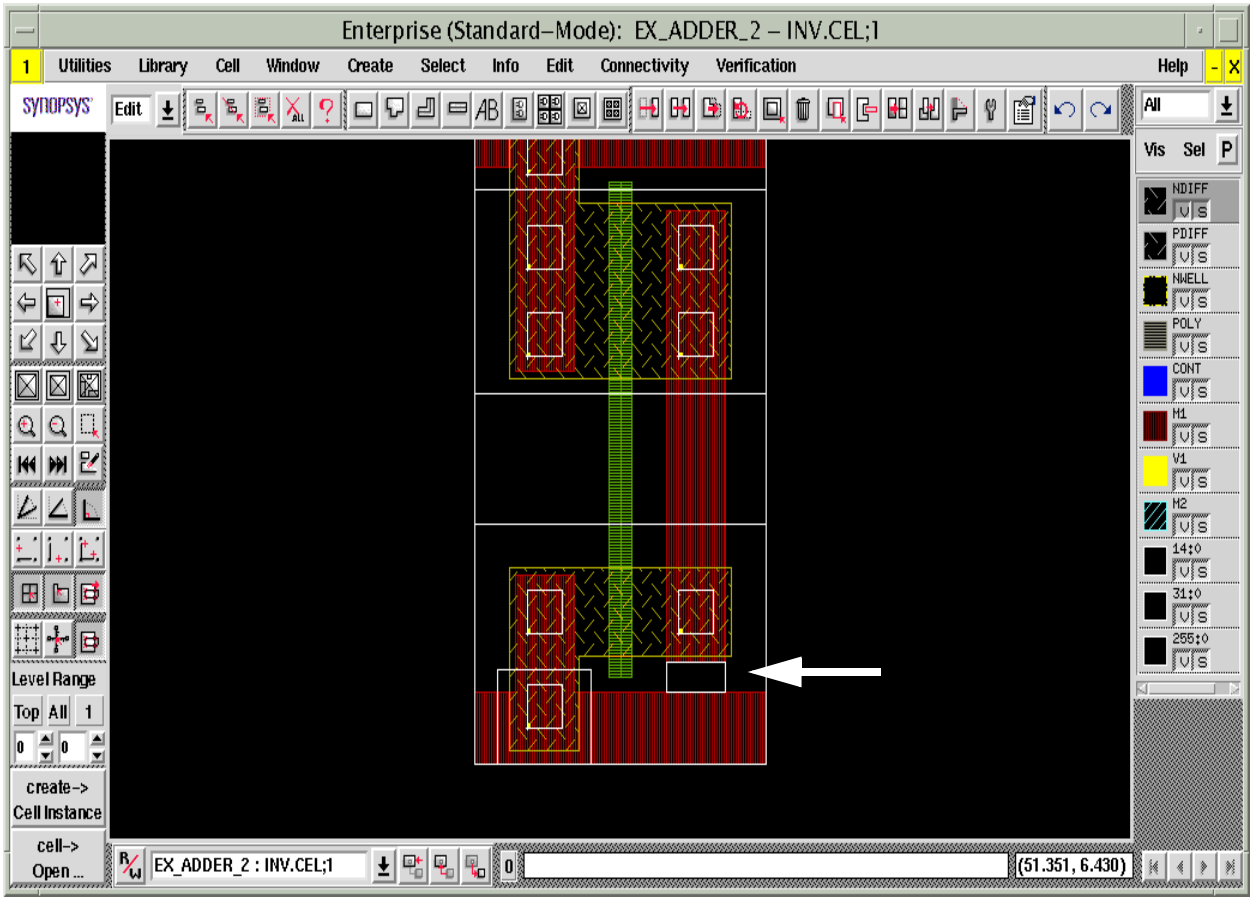
Enter the coordinates (0 0) as shown in Figure 3-9. Click Apply. This creates a cell instance of the master gate_INV from the reference library EX_ADDER_2_OUT at the coordinates (0, 0).

Figure 3-9 Create Cell Instance



You now have the met1 layer 101 overlaid on your original design. You can reference as many of the other layers as you like.

Figure 3-10 Overlay of INV and ERR_INV

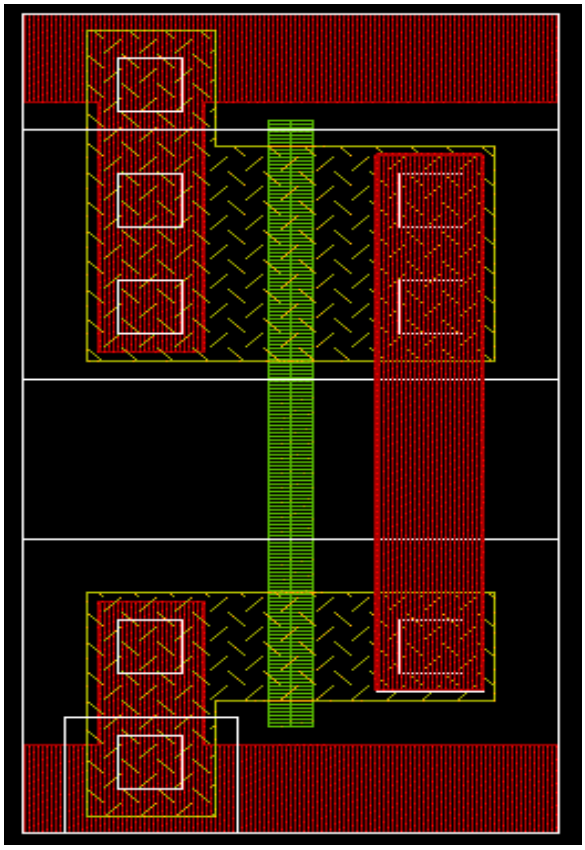


With the error polygon overlaid on the INV structure, you see that the metal1 which connects the drains of the N and P channel devices and makes up the inverter is too long. The error polygon in Figure 3-10 shows that metal1 is only 2 µm away from adjacent metal1, instead of the 3 µm required by the design rule.

If you are the layout person responsible for fixing errors and have a full Enterprise license, the standard methodology at this point is to fix the design error. Enterprise layout edits are beyond the scope of this tutorial; see the *Enterprise User Guide* or the *Enterprise Reference Manual*. When you make your edits, shorten the metal1 strip by 1 µm to allow 3-µm spacing between adjacent metal polygons. Otherwise, use your layout editor of choice to correct the error.

The corrected INV cell looks like Figure 3-11.

Figure 3-11 Corrected INV Cell



The single error polygon indicates that this is the only error for the design. Fixing metal1 cures the problem in this example. In a real design, of course, you have to make sure that any editing does not affect other design rules on the same or other design layers.

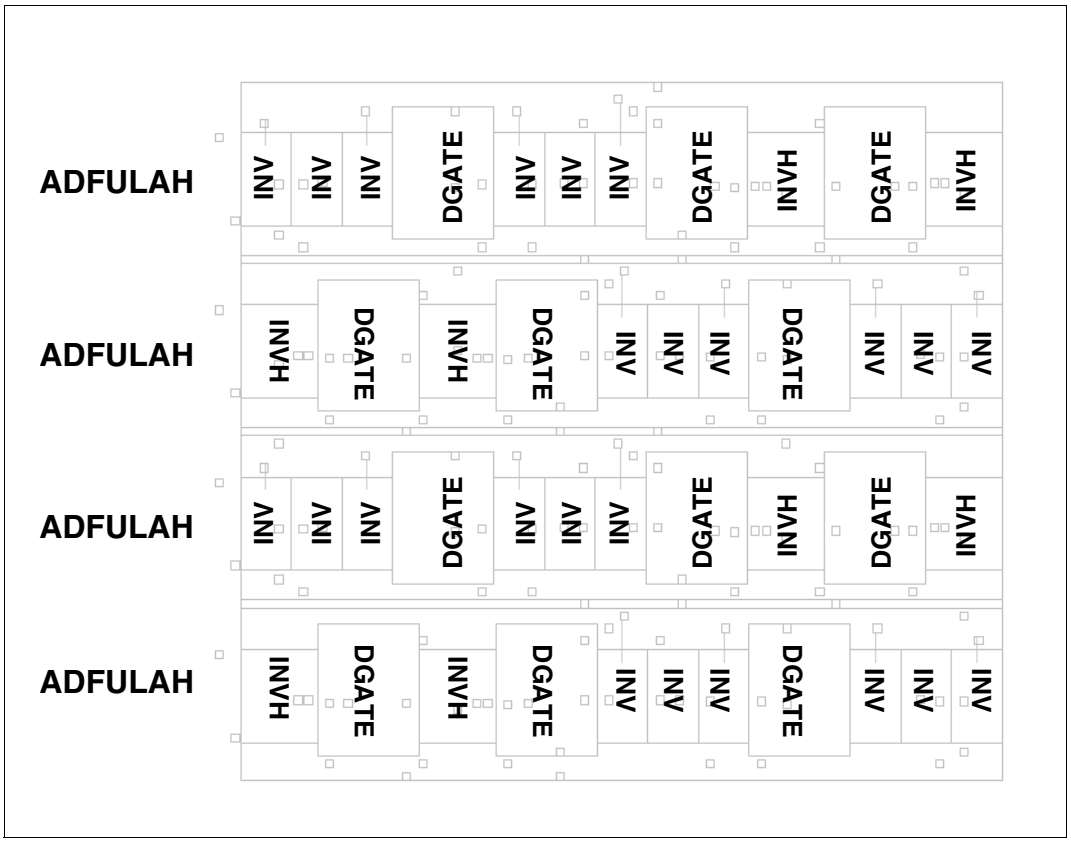
Now, to exit the structure:

Choose Cell > Save in the graphical window to save the cell.

Layout Topology

For our example, we know that fixing the metal1 error fixes the design. But to see how the INV error manifests itself in the rest of the design, look at the layout topology in Figure 3-12.

Figure 3-12 AD4FUL Layout Topology



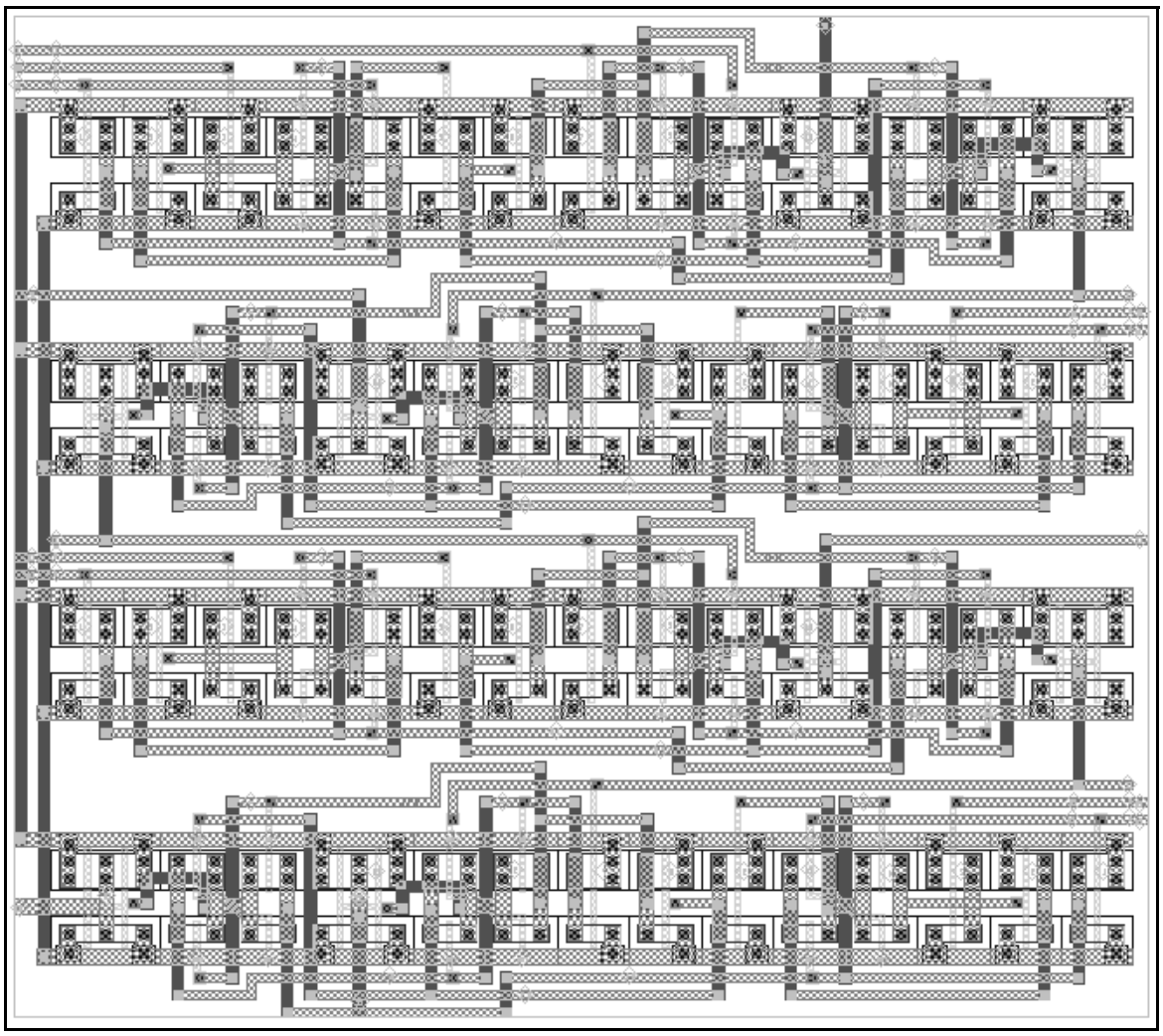
From the hierarchy tree shown in Figure 2-5 on page 2-23 in Chapter 2, “An Error-Free Design for DRC,” you know that INV is placed six times in the ADFULAH structure, and that ADFULAH is placed four times in the top level structure, AD4FUL. Therefore, you have 24 occurrences of INV, and 24 occurrences of the same design rule error in the design. Figure 3-13 helps you anticipate where the 24 error polygons are approximately located.

Open the AD4FUL structure by selecting Cell > Open in the graphics window. Only the top-level cell instances may be visible. In order to display all polygons through all levels of hierarchy, select the Level Range > All button in the button panel to the left of the layout window. You can also enter the following command in the command window to view all levels of hierarchy.

gxwSetViewLevelRange 1 "0-255" "redraw"

The view of the layout shows the actual polygons, complete with all metal interconnections. While there is too much information to be used effectively in this tutorial, you do see the actual layout from the AD4FUL top level.

Figure 3-13 Actual AD4FUL Layout and Interconnect



Choose Cell > Close in the graphical window to close the cell.

Choose Library > Close to close the library.

To quit Enterprise, in the command window enter:

exit

Or, choose Tools > Quit from the command window.

Output Hierarchy Options

The check for the INV example, as well as for all the dimensional checks in this document, uses the error hierarchy for output results. Any error polygons produced appear in a file that is prefixed with ERR_ (unless you specify a different prefix for the error hierarchy with the ERR_PREFIX option).

You can also write your rule check output to separate hierarchy files in addition to, or instead of, the error hierarchy. For instance, you can specify your design rule in the runset file as:

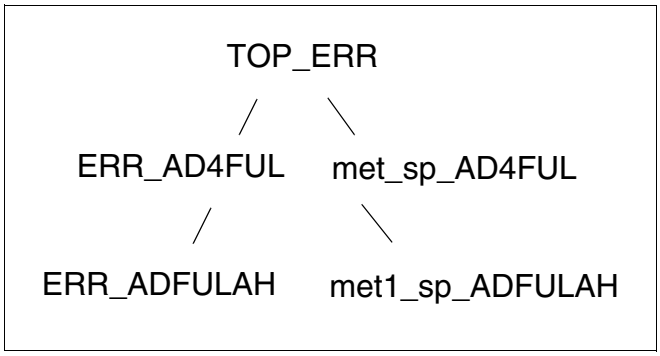
```
EXTERNAL met1 {SPACING<3.0} PERM=met1_sp (101)
```

As indicated by PERM=met1_sp, Hercules output is sent to a permanent set of files prefixed with met1_sp_, with our example producing files named:

met1_sp_INV
met1_sp_ADFULAH
met1_sp_AD4FUL

If you had specified the hierarchy with this PERM setting, the polygons would not appear in the error hierarchy cells prefixed by ERR_, but instead in the met1_sp-prefixed cells that are noted above. Also, the text coordinates of the error polygons would not appear in the AD4FUL.LAYOUT_ERRORS file (or in any other file). The graphical output to the TOP_ERR structure and the error message in the AD4FUL.sum file would still appear. Figure 3-14 shows an example of the hierarchy of the TOP_ERR output block.

Figure 3-14 TOP_ERR Output Block Hierarchy



Another option is to write rule check output to a temporary (TEMP) file. Temporary files are most appropriate when a design rule check creates new information that is temporarily needed for use in another check.

See Writing Output Results to Files in Chapter 4, “A Complete Design for DRC,” for information on how PERM and TEMP files can be used effectively with other types of Hercules runset rules.

Note:

After each command or check, you may specify a layer number indicating that the output of the command goes to the error hierarchy output at the end of the Hercules run. Or, you can specify a PERM or TEMP file. If you specify TEMP, the polygon data written by the command is stored under the ./group directory as a temporary file that Hercules can read when necessary for other commands. For example, a Boolean command may generate TEMP=foo, and then the data written to the temporary file foo might be used in "BOOLEAN foo AND foo1" later in the runset. If you write the data to a PERM file, the data is written to a file under ./group, but it is also written to a graphical file at the end of the Hercules job so you can view the polygon data.

If a command's output is written only to a PERM or TEMP layer and not to the error hierarchy, no data for that error will appear in the *block*.LAYOUT_ERRORS file.

```
EXTERNAL met1 SPACING<3.0 PERM=met1_sp (101)
```

However, a slight modification to the design rule enables the output polygons to be written to the AD4FUL.LAYOUT_ERRORS file as well:

```
EXTERNAL met1 SPACING<3.0 VERBOSE=TRUE} PERM=met1_sp (101)
```

Using the VERBOSE option with any command directed to a PERM or TEMP file enables reporting to the *block*.LAYOUT_ERRORS file (resulting from errors in the file designated by BLOCK in the runset HEADER section) as well as to the specified PERM or TEMP file.

What's Next?

After you fix the error, move on to Chapter 4, "A Complete Design for DRC," to apply the concepts you have just learned to a more challenging example with a multiple-error design. If you are now using Dracula physical verification and trying to transition to Hercules, skip Chapter 4 and go to Chapter 5, "Hercules DRC Migration."

4

A Complete Design for DRC

Our final DRC example presents a multiple-error design and builds from the concepts you have learned thus far.

The activities in this chapter are comparable to the ones in Chapter 3, “Single Design Error for DRC,” but here you create and learn how to correct layout errors similar to those encountered in real designs.

Summary of Progress to This Point

In our first two examples, we detailed all the steps required to set up the software to check a simple design for a single runset rule. Although those examples were very basic, the procedures and files created are nearly the same for runsets of any complexity. We go through a more complex example here, and point out any considerations that should be noted for large runset files or complex designs.

In all cases, the layouts are checked hierarchically. The properties of a hierarchical design allow even very complex chips to be broken down into small modules that can be easily checked with a hierarchical design rule checker like Hercules HDRC.

Learning Objectives for This Chapter

The more realistic runset check example presented in this chapter has a representative sample of checks and procedures that might be encountered in a real design (scaled down to make the Tutorial manageable). With this example you:

- Learn the types of DRC checks available in Hercules, and create a few to place in the runset file.
- Run Hercules with the runset file on a design with several errors.
- Learn how to interpret the errors in the text output files.
- Examine in Enterprise the derived layers created by the runset file.
- Create extra layers to perform other types of checks.
- Examine output files when multiple checks and multiple errors are involved.
- Run Hercules-VUE to see how the error detection process can be made easier.

Before You Start

Before you start this section of the tutorial, make sure that you have completely gone through the installation and setup procedure described in Chapter 1, “Installation and Setup.”

A Summary of Design Rule Checks

Hercules features a wealth of design rule checks that can quickly and efficiently check for any layout spacing, size, relationship, or tolerance. We present a short summary of the types of checks available in this Tutorial, and use some of them as rules for our example. A discussion of all Hercules Data Creation and Dimensional Check operators can be found in the *Hercules Reference Manual*.

Operations Allowed by Hercules

Hercules DRC rules allow you to perform three types of data operations:

- Generation of new layers, which can be verified, saved to your database, or dimensionally checked. The new layers can be formed with Boolean operators, SELECT relationships, SIZE operations, hierarchical interactions, and NEGATE commands.
- Dimensional checks for a single polygon. These include length, width, area, overlap, ratio, and density calculations.
- Dimensional checks between polygon edges or vertices. These include intersecting polygon spacings, enclosure spacings, and external polygon spacings.

There are variations of these operations that are in a special class of their own. Two such examples include the Hercules extensive corner checking ability and its ability to check device widths and lengths for MOS devices and resistors. For details on further specialized checking and data creation, refer to the*Hercules Reference Manual*.

Examining the Runset Rules

As previously noted, we cannot describe each and every Hercules rule in this Tutorial. What we can do, however, is provide a representative sampling of the rules you are most likely to encounter in a real design. We thoroughly explain each rule, and then implement each one in our sample runset. We explore the text and graphical file output that results from violation of these rules, and explain how to search for and correct the errors built into our example. By going through this exercise, you gain a good working knowledge of the methodology needed to understand and verify designs. You can then refer to the *Hercules Reference Manual* for details on the other design rules you encounter or need to implement in your own work.

Note:

For those of you who are familiar with other physical verification tools, but would like to understand the syntax of Hercules and how to use Hercules-VUE, the debug tool, skip ahead to Running Hercules.

This selection of rules is not intended to represent an actual runset that thoroughly checks every polygon rule violation, but rather a reasonable sampling of the more straightforward checks in Hercules. In many cases in an actual runset, several checks might be required to check relationships between polygons on the same or different layers.

The description of each rule is specific to the example in our Tutorial. We describe the rule in terms of the actual layers used in the design. In addition, any options that are mentioned are chosen to check specific relationships in the example. Most of the checks described offer other options that are not employed in this design. If you plan to use any of these rules in your design, be sure to read the *Hercules Reference Manual* for a complete description of all options available for every rule.

Writing Output Results to Files

The results of each of the checks are output to different types of files. Hercules supports three different types of output files:

- Error hierarchy output
- Permanent output
- Temporary output

Error Hierarchy Output

Creating a check with an output layer specified in parentheses, such as,

```
Check {options} (101)
```

tells Hercules to send all output from the check to the error hierarchy on the layer specified, in this case, 101. Checks producing error polygons are usually sent to the error hierarchy.

Permanent Output

Permanent means that the layer is written to the PERM hierarchy and can be used later in the runset.

Creating a check with an output layer specified in the form

```
Check {options} PERM=outlayer1 (101)
```

tells Hercules that a permanent output layer, named outlayer1, is to be created on layer 101. A separate hierarchy of output structures is then created using outlayer1 as the prefix. Specifying a PERM layer also allows you to create a label for the output. This label can be

employed within a subsequent check to use the output layer as an input layer for the new check. The PERM layer information is not output to the error file (*block*.LAYOUT_ERRORS) unless you add `VERBOSE = TRUE` to the command.

Temporary Output

Temporary layers can be used when output created during the check needs to be made available for another check, but does not need to be stored as a permanent result. Temporary files are most often used with Data Creation statements, which we explain in detail in this section. Creating a check with an output layer specified in the form

```
Check {options} TEMP=outlayer2
```

tells Hercules that a temporary output layer, named outlayer2, is to be created. Notice that no layer number is specified. The layer created is available only during the run and is deleted when execution is completed.

There is a way to view temporary files, even though they are normally deleted after the run. By using the GRAPHICS command, temporary files or permanent layers can be written to a separate output database, with a layer number assigned to the TEMP or PERM layer within the GRAPHICS section. For details on the GRAPHICS command, refer to the *Hercules Reference Manual*.

EXTERNAL Checks

The simple runset of our first two examples has already introduced us to EXTERNAL checks. EXTERNAL checks verify distances between polygons, with the measurement specified for polygons on the same layer or between different layers. When checking spacings, the measurement can be for edge-to-edge spacings, corner-to-corner spacings, or corner-to-edge spacings. Spacing checks can also be restricted to polygons that meet specified internal width values.

For our sample design, we have selected three specific EXTERNAL checks:

- metal1 to metal1 spacing
- metal2 to metal2 spacing - LONGEDGE option
- metal1 to metal2 spacing - TOUCH option

metal1 to metal1 Spacing

```
EXTERNAL met1{SPACING<3} (101)
```

This is the familiar check we have already employed. It flags any spacings for metal1 to metal1 distances less than 3 µm, and places all error output in the error hierarchy on layer 101.

metal2 to metal2 spacing - LONGEDGE Option

```
EXTERNAL met2 {SPACING<3.0 LONGEDGE_TO_EDGE<5.0 LONGEDGE>=50} (102)
```

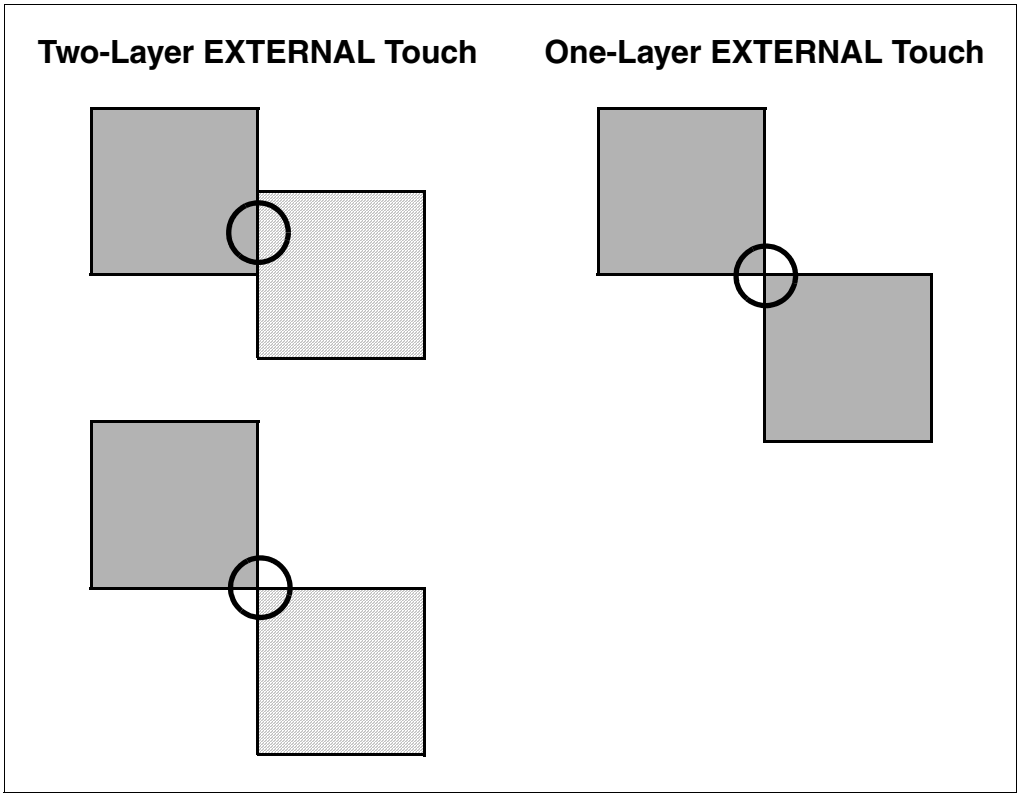
The EXTERNAL check syntax allows two separate checks, one for parallel metal2 runs shorter than 50 µm, and one for those longer than 50 µm. For shorter runs, the command in this form flags metal2 to metal2 spacings less than 3 µm. When two parallel metal2 runs maintain a constant spacing for a length of at least 50 µm, the command imposes a stricter spacing check of a minimum of 5 µm. This type of check is employed in a design when long parallel metal runs need to be farther apart than short runs to meet process specifications.

metal1 to metal2 spacing - TOUCH Option

```
EXTERNAL met1 met2 {SPACING<1.0 TOUCH} (103)
```

This check measures spacing between different metal1 and metal2 layers, and also flags any metal1 polygons that touch metal2 polygons. For a two-layer EXTERNAL check (as in our example), a touch is considered either an edge or point touch. For a one-layer EXTERNAL check, only a point touch is flagged. Refer to the drawings in Figure 4-1.

Figure 4-1 TOUCH Options for EXTERNAL Checks



INTERNAL Checks

INTERNAL checks measure polygon widths; the check is always for polygons on the same layer. As with EXTERNAL checks, the measurement can be for edge-to-edge spacings, corner-to-corner spacings, or corner-to-edge spacings.

For our sample design, we have selected two specific INTERNAL checks:

- poly width - EDGE_45 option
- metal2 width - CORNER option

poly width - EDGE_45 Option

```
INTERNAL poly {SPACING<2.0 EDGE_45<2.5} (104)
```

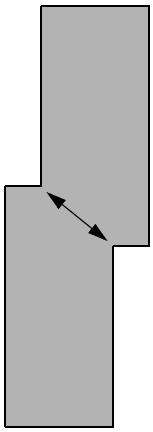
This check measures all poly edges and corners against the 2-µm width value specified. Any poly layer that is thinner than the rule is flagged. The EDGE_45 option applies a 2.5-µm width rule to poly segments set at any angle other than 90 degrees.

metal2 width - CORNER Option

```
INTERNAL met2 {SPACING<4.0 CONVEX_TO_CONVEX<4.5} (105)
```

This check measures all metal2 edges against the 4-µm width value specified. For convex-to-convex corners (as indicated in Figure 4-2 by the arrows), a different value of 4.5 µm is specified. All other corners are checked against the 4.0 µm SPACING value. Hercules can check a variety of corner types; refer to the *Hercules Reference Manual* for a complete discussion.

Figure 4-2 CONVEX-TO-CONVEX Corner Check



CUT Checks

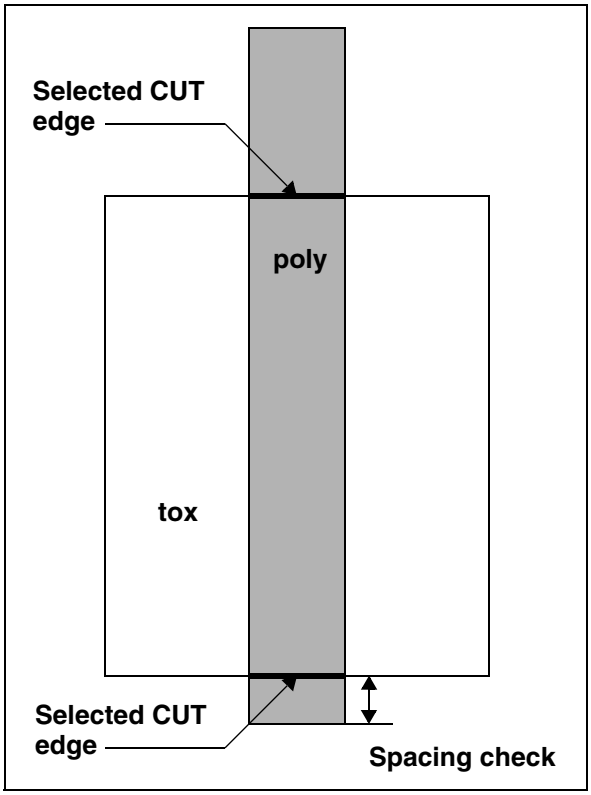
The CUT check selects edges of polygons that intersect by a certain spacing (and, with options, those that TOUCH or are ENCLOSED). For our example, the CUT check provides a way to select a polygon that overlaps portions of another polygon, and must maintain specific distances between polygon edges. We have selected one check.

Cut poly by diffusion - CUT_OUTSIDE Option

```
CUT poly BY tox {CUT_OUTSIDE<1.5} (106)
```

This check measures the overlap of poly against diffusion, to be sure that a MOS gate is properly formed. The CUT check itself selects only edges. The edges selected with the rule are the diffusion edges, which are inside the poly edges. To understand this concept, we look at a simple drawing of a gate formed by poly and diffusion (represented by the layer name tox).

Figure 4-3 Illustration of CUT Check



In Figure 4-3 the heavier, short, horizontal lines are the edges of diffusion, which are inside poly. In other words, poly is cut by diffusion in these two places. The CUT_OUTSIDE option checks that any poly that continues outside the diffusion has an edge parallel to the cut edge, and is outside the cut edge by at least 1.5 µm.

AREA Checks

The AREA check is a simple check that checks for a polygon area, flagging areas that are within the specified range. We have selected a single check:

Via area - RANGE Option

```
AREA via {RANGE = [2.0,2.0]} (107)
```

This check measures areas of via cells, and reports as errors those falling within the specified range In this case, the check flags via polygons with an area of exactly 2 μm^2.

DATA CREATION / INTERNAL Checks

All the checks we defined previously are dimensional checks. They produce errors based on the spacing values in the runset rules. The dimensional check results are placed in the error hierarchy for the design because we have specified the error hierarchy as the output for each of the runset rules.

Data Creation statements can also be included in DRC runsets. These operations either produce data in the form of a polygon shape or manipulate data in some way by moving data to a different hierarchical level. The data can be written to the error hierarchy or to separate TEMP or PERM layers. In our runset we use the Data Creation statements to produce new output polygons written to PERM layers. These layers are then dimensionally checked with INTERNAL checks. Specifically, our example defines MOS gates and then checks those gates for length and width.

BOOLEAN poly AND tox

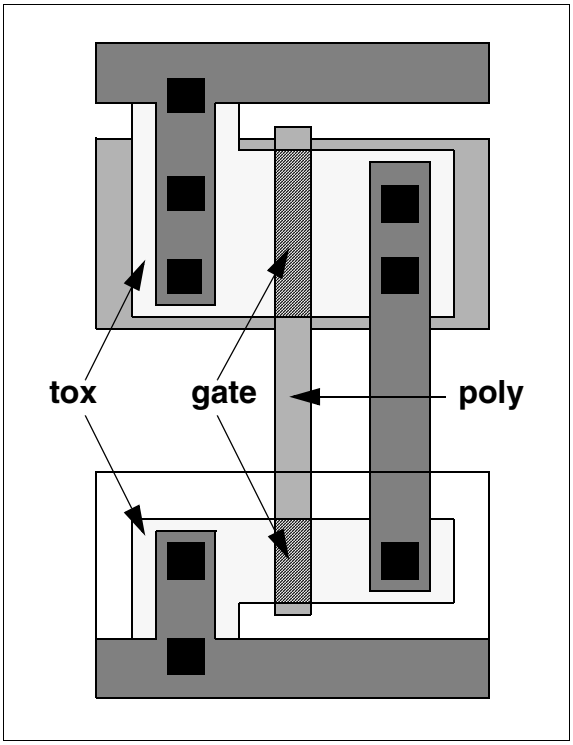
```
BOOLEAN poly AND tox PERM=gate (111)
```

This BOOLEAN operator creates AND, OR, NOT, and XOR relationships between polygons. The PERM statement places the result of the Data Creation operation on a graphical output layer. This step creates a layer from the intersection of poly and tox, which forms a MOS gate structure. A new permanent output layer, gate, is created. (Refer to Figure 4-4.)

Note:

Usually the Data Creation output is written to TEMP layers. We are writing them to PERM layers in our example because later in this chapter we demonstrate how to view the layers we create.

Figure 4-4 Identification of Gate



Now look at Figure 4-5 to see how the specific gates pgate and ngate are formed from psel.

BOOLEAN gate AND psel

```
BOOLEAN gate AND psel PERM=pgate (112)
```

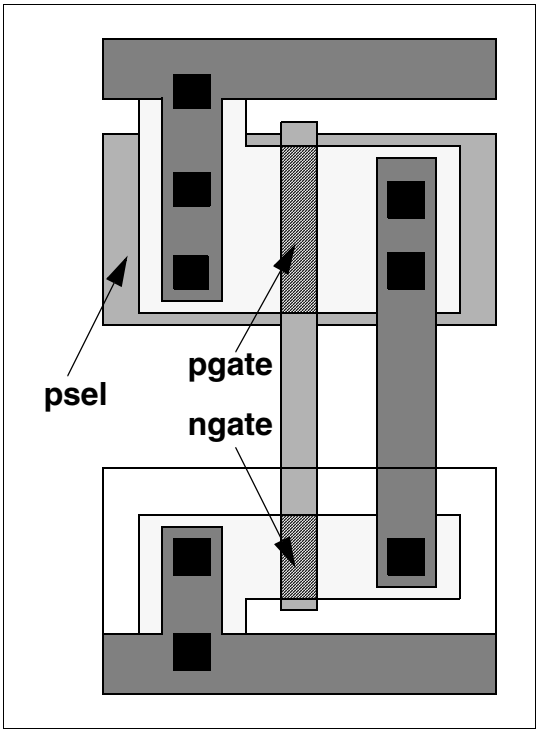
This BOOLEAN operator looks for all gate polygons inside the psel layer and creates a layer from the intersection of gate and psel. A new output layer, pgate, is formed, as shown in Figure 4-5. This layer has the same dimensions as any PMOS gate in the design.

BOOLEAN gate NOT pgate

```
BOOLEAN gate NOT pgate PERM=ngate (113)
```

This BOOLEAN operator looks for all gate polygons that are not pgate and creates a layer by subtracting the pgate layer from the gate layer. A new output layer, ngate, is formed, also shown in Figure 4-5. This layer has the same dimensions as any NMOS gate in the design.

Figure 4-5 Identification of pgate and ngate



INTERNAL pgate DIMENSION Option

```
INTERNAL pgate {DIMENSION = [2.0,12.0]} (114)
```

We have already discussed the INTERNAL check. This particular rule uses the DIMENSION option, which checks polygon widths and lengths against a single or multiple set of values. For our example, the rule checks the pgate layer that was formed by the BOOLEAN operator. The rule states that all pgates must be rectangles of exactly 2 μm by 12 μm. Any polygons falling outside the check limits are flagged on error layer 114.

INTERNAL ngate DIMENSION Option

```
INTERNAL ngate {DIMENSION = [2.0,6.0]} (115)
```

This check performs a DIMENSION measurement on the ngate layer, with error output on layer 115.

DATA CREATION Checks

In this section, we use the Data Creation operators to help perform another task—creating layers that define metal1 to tox contacts. In the design, two types of contacts are present, metal1 to tox and metal1 to poly (refer to Figure 4-6). Because the contacts have different

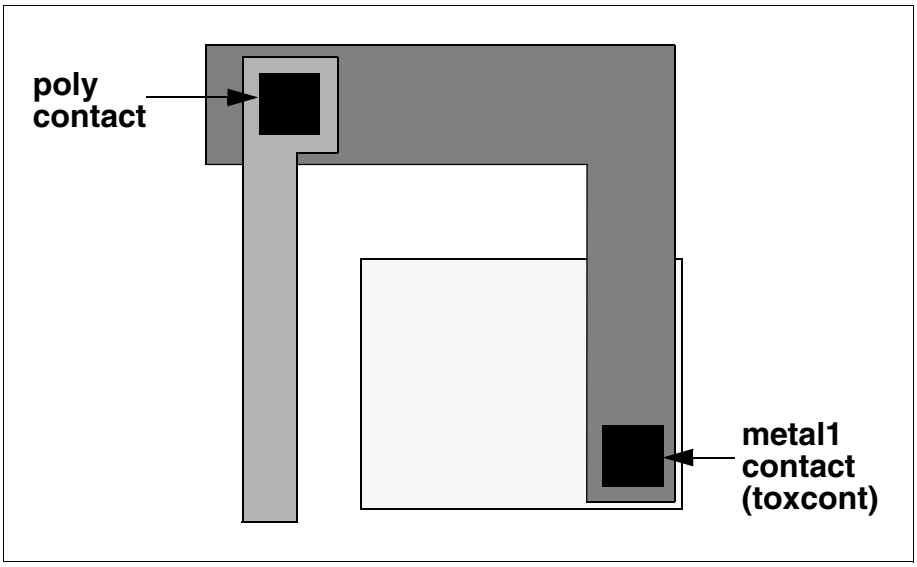
process rule requirements, and therefore different geometries, we need to distinguish them. For our example, we separate all of the metal1 to tox contacts in the hierarchy toxcont. We then perform spacing checks on those contacts within other polygons with the ENCLOSE statement.

BOOLEAN cont NOT poly

```
BOOLEAN cont NOT poly PERM=toxcont (121)
```

This BOOLEAN operator looks for every contact layer that does not have poly underneath. The result is the desired metal1 to tox contact. A permanent output layer, toxcont, is formed that allows you to look at the output, as shown in Figure 4-6. However, because the output layer is not required for viewing, it can be designated a TEMP layer.

Figure 4-6 Metal vs. Poly Contacts



ENCLOSE Check

The ENCLOSE Dimensional Check option checks spacings for polygons nested inside other polygons. A spacing measurement determines precisely where one polygon is situated inside another. Both edges and corners can be checked. ENCLOSE checks can also be qualified based on the characteristics of the polygon. We have two ENCLOSE checks in our runset.

ENCLOSE toxcont by tox

```
ENCLOSE toxcont BY tox {SPACING<1.5} (122)
```

This checks our newly formed toxcont layer to ensure that the contact is inside the tox layer by at least 1.5 µm.

ENCLOSE toxcont by metal1

```
ENCLOSE toxcont BY met1 {SPACING<1.0 TOUCH OVERLAP PARALLEL} (123)
```

This checks our newly formed toxcont layer to ensure that the contact is inside the metal1 layer by at least 1 µm. In addition, the TOUCH and OVERLAP options flag any toxcont layers that are not completely surrounded by metal1. The PARALLEL option specifies that only parallel orthogonal edges are checked.

From Rules to Runsets

The preceding sections explain the seven Dimensional Checks and eight Data Creation operations we incorporate in our runset. The runset uses the same HEADER section as in our previous runset files. We have added `CREATE_VUE_OUTPUT = TRUE` to the OPTIONS section. This tells Hercules to output the necessary data files needed to use Hercules-VUE for graphical debugging. Finally, the checks we described previously are listed in the runset. One exception in the HEADER section is that a separate output library is created to store error polygons and newly created data. Our next step checks this runset against a new design that contains structures flagged by these rules. Example 4-1 is a more realistic runset used for checking.

Example 4-1 A More Realistic DRC Runset

```
 /* HDRC SAMPLE RUNSET */

/* -------------------------------- Runset Header information */
HEADER {
   LAYOUT_PATH = ./
   INLIB = EX_ADDER_3
   BLOCK = AD4FUL
   OUTLIB= EX_ADDER_3_ref
   OUTPUT_BLOCK= TOP_ERR
   GROUP_DIR = group
   FORMAT = MILKYWAY
}

/* -------------------------------- Runset options */
OPTIONS {
   WIDTH = 2
   CREATE_VUE_OUTPUT = TRUE /* added to create the data files for VUE */
}

/* -------------------------------- Data Preprocessing Options */
PREPROCESS_OPTIONS {
        CHECK_PATH_90 = FALSE
}
```

```
/* ---------------------------------- Layer assignments */
ASSIGN {
   tox  (1)
   poly (5)
   well (31)
   psel (14)
   cont (6)
   met1 (8)
   met2 (10)
   via  (19)
}

/* ---------------------------------- SNAP Command */
SNAP {
   ASSIGN_LAYERS = 0.010
   } temp = SNAP_OUT

/* ---------------------------------- GRID checks */
GRID_CHECK {
       ASSIGN_LAYERS = true
       CHECK_45      = true
} (100)

/* ---------------------------------- DRC dimensional checks */
EXTERNAL met1 {SPACING<3.0} (101)
EXTERNAL met2 {SPACING<3.0 LONGEDGE_TO_EDGE<5.0 LONGEDGE>=50} (102)
EXTERNAL met1 met2 {SPACING<1.0 TOUCH} (103)
INTERNAL poly {SPACING<2.0 EDGE_45<2.5} (104)
INTERNAL met2 {SPACING<4.0 CONVEX_TO_CONVEX<4.5} (105)
CUT poly BY tox {CUT_OUTSIDE<1.5} (106)
AREA via {RANGE = [0.0,2.0]} (107)

/* ---------------------------------- DRC data creation for gates */
BOOLEAN poly AND tox PERM=gate (111)
BOOLEAN gate AND psel PERM=pgate (112)
BOOLEAN gate NOT pgate PERM=ngate (113)
INTERNAL pgate {DIMENSION = [2.0,12.0]} (114)
INTERNAL ngate {DIMENSION = [2.0,6.0]} (115)

/* ---------------------------------- DRC data creation for contacts */
BOOLEAN cont NOT poly PERM=toxcont (121)
ENCLOSE toxcont by tox {SPACING<1.5} (122)
ENCLOSE toxcont BY met1 {SPACING<1.0 TOUCH OVERLAP PARALLEL} (123)
```

Running Hercules

Be sure that you are in the directory where your adderdrc3.ev file is located; that is, *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest3.

To run Hercules, use the command:

hercules adderdrc3.ev

As before, your active window displays the execution process.

Output Results

As you can see by the activity during the execution process, our files are much larger and contain more information than in previous examples. We carefully examine the error and sum files, and explain in detail how to correlate the text file results to the graphical files to be reviewed in Enterprise.

Error File

The AD4FUL.LAYOUT_ERRORS error file has a list of runset rule violations. In this test case, most of the selected rules trigger at least one error polygon. By understanding how to interpret the results, you can easily scan the file and quickly locate and fix the errors in the graphical database.

Example 4-2 AD4FUL.LAYOUT_ERRORS File for EX_ADDER_3 Library name: EX_ADDER_3

```
                        LAYOUT ERRORS RESULTS: ERRORS
                    ##### ####  ####   ###  ####   ####
                    #     #   # #   # #   # #   # #
                    ####  ####  ####  #   # ####   ###
                    #     #  #  #  #  #   # #  #      #
                    ##### #   # #   #  ###  #   # ####

=====================================================================

Library name:  EX_ADDER_3
Structure name:  AD4FUL

                          ERROR SUMMARY

No Comment
 EXTERNAL met1 { } (101) ........................... 2 violations found.

No Comment
 EXTERNAL met2 { } (102) ........................... 2 violations found.

No Comment
 EXTERNAL met1 met2 { } (103) ...................... 2 violations found.

No Comment
 INTERNAL poly { } (104) ............................ 1 violation found.
```

```
No Comment
 INTERNAL met2 { } (105) ........................... 2 violations found.

No Comment
 CUT poly BY tox { } (106) ......................... 1 violation found.

No Comment
 INTERNAL pgate { } (114) .......................... 1 violation found.

No Comment
 INTERNAL ngate { } (115) .......................... 1 violation found.

No Comment
 ENCLOSE toxcont BY tox { } (122) .................. 4 violations found.

No Comment
 ENCLOSE toxcont BY met1 { } (123) ................. 4 violations found.

                        ERROR DETAILS

#####--- ERR_EXTERNAL ----------------------------------

EXTERNAL met1 {
        SPACING<3
        WIDTH=2 } (101)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INVH               (16.500, 39.000)    (21.500, 41.000)       2.000
ADFULAH            (207.500, 64.000)   (238.000, 66.000)      2.000

#####--- ERR_EXTERNAL ----------------------------------

EXTERNAL met2 {
        SPACING<3
        LONGEDGE>=50
        LONGEDGE_TO_EDGE<5
        WIDTH=2 } (102)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ADFULAH            (103.500, 17.000)   (104.000, 58.500)      0.500
AD4FUL             (4.500, 60.000)     (8.500, 278.000)       4.000

#####--- ERR_EXTERNAL ----------------------------------

EXTERNAL met1 met2 {
        SPACING<1
        WIDTH=2
        TOUCH=TRUE } (103)
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DGATE              (28.500, 8.000)     (28.500, 18.000)       0.000
DGATE              (28.500, 23.000) (28.500, 38.000)          0.000

#####--- ERR_INTERNAL ----------------------------------

INTERNAL poly {
        SPACING<2
        EDGE_45<2.5
        WIDTH=2 } (104)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DGATE              (21.090, 16.090)    (28.910, 23.910)       1.994

#####--- ERR_INTERNAL ----------------------------------

INTERNAL met2 {
        SPACING<4
        CONVEX_TO_CONVEX<4.5
        WIDTH=2} (105)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ADFULAH            (204.000, 53.000)   (207.000, 60.000)      3.000
AD4FUL             (368.000, 268.000)  (371.000, 269.000)     3.162

#####--- ERR_CUT ---------------------------------

CUT poly BY tox {
        SPACING<1.5
        CUT_OUTSIDE<1.5 } (106)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INV                (11.500, 6.500)     (13.500, 7.500)

#####--- ERR_INTERNAL ----------------------------------

INTERNAL pgate {
        WIDTH=2
        DIMENSIONS = [2,12] } (114)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INVH               (24.500, 26.500)    (27.000, 38.500)       0.000
```

```
#####--- ERR_INTERNAL ----------------------------------

INTERNAL ngate {
        WIDTH=2
        DIMENSIONS = [2,6] } (115)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
TGATE              (11.500, 0.000)     (14.000, 6.000)        0.000

#####--- ERR_ENCLOSE ----------------------------------
ENCLOSE toxcont BY tox {
        SPACING<1.5
        WIDTH=2 } (122)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
TGATE              (21.500, 20.500)    (22.000, 23.500)       0.500
TGATE              (3.000, 1.000)      (4.000, 4.000)         1.000
TGATE              (4.000, 0.000)      (7.000, 1.000)         1.000
TGATE              (3.000, 0.000)      (4.000, 1.000)         1.414

#####--- ERR_ENCLOSE ----------------------------------

ENCLOSE toxcont BY met1 {
        SPACING<1
        WIDTH=2
        PARALLEL=TRUE
        TOUCH=TRUE
        OVERLAP=TRUE } (123)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure          ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INV                (14.500, 9.000)     (16.500, 12.000)       0.000
TGATE              (21.500, 20.500)    (21.500, 23.500)       0.000
TGATE              (3.500, 1.000)      (4.000, 4.000)         0.500
TGATE              (4.000, 0.500)      (7.000, 1.000)         0.500
```

Summary File

Before we begin to explore these errors, we complete our text file examination by looking at the summary file located in the run_details directory. The .sum file appears to repeat the information in the error file, listing each runset rule and summarizing the number of violations for each. However, the .sum file also provides information on the Data Creation layers, which were created by the additional statements in adderdrc3.ev. Combining the information in the two files gives us a fairly complete picture of our design and tells us what we need to observe when we get into Hercules-VUE.

In Example 4-3, notice the emphasized sections.

Example 4-3 AD4FUL.sum File for EX_ADDER_3

```
Hercules (R) Hierarchical Design Verification, ... DATA OMITTED
  Synopsys.  All rights reserved.

Called as: hercules adderdrc3.ev

... RELEASE VERSION OMITTED
  Synopsys.  All rights reserved.

Running Single-Threaded code

Runset file ......................  adderdrc3.ev
Current Directory ................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_DRC/
addertest3
Hostname .........................  ddthp42
Platform type ....................  HP64_U11
MILKYWAY input library path ......  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_DRC/
addertest3/
MILKYWAY input file name .........  EX_ADDER_3
MILKYWAY block name ..............  AD4FUL
MILKYWAY output library path .....  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_DRC/
addertest3/
MILKYWAY output file name ........  EX_ADDER_3_ref
MILKYWAY output_block name .......  TOP_ERR
Run details ......................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_DRC/
addertest3/run_details
Group file directory .............  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_DRC/
addertest3/group
Retain group files ...............  smart
Check reference structures .......  yes

OPTIONS {
   ALL_TEXT_GLOBAL=FALSE
   ASCII_ONLY=FALSE
   ATTACH_TEXT=FALSE
   BOX_CORNER=FALSE
   CHECK_REF_LIB=TRUE
   COUNT_TRAPS=FALSE
   ERR_LIMIT_PER_CHECK = UNLIMITED
   ERR_PREFIX = ERR
   EXPLODE_AREFS=FALSE
   EXPLODE_HOLDING_CELL_LIMIT=0
   EXPLODE_LIMIT=0
   FLAG_ALL_AREF_ERRORS=FALSE
   FLAT_COUNT=FALSE
```

```
    FLAT_ERROR=FALSE
    GENERATE_INSTANCE_NAME=TRUE
    HIERARCHICAL_DELIMITER = \
    IGNORE_CASE=FALSE
    INCREMENTAL_CELLS=FALSE
    INCREMENTAL_CELLS_FILE =
    INSTANCE_PREFIX =
    NET_PREFIX =
    SELECT_CELL_TO_NO_EXPLODE=TRUE
    EQUIV_TO_NO_EXPLODE=TRUE
    SCHEMATIC_TO_NO_EXPLODE=TRUE
    BLACK_BOX_TO_NO_EXPLODE=TRUE
    PROTOTYPE_PLACEMENTS=FALSE
    NO_MERGE=FALSE
    GENERATE_INSTANCE_NAME=TRUE
    PRINT_ERRSUM_FILE=TRUE
    MAXIMUM_CELLNAME_LENGTH=32
    SIZE_ENDPOINTS=FALSE
    SNAP_RES=TRUE
    SQUARE_CORNER=FALSE
    STOP_ON_GROUP_ERROR=TRUE
    TEXT_RECT=0.000
    USE_EXPLODED_TEXT=FALSE
    WIDTH=2.000
    MAGNIFICATION_FACTOR=1
    OUTPUT_MAGNIFICATION_FACTOR=1
    POLYGON_COUNT_IN_ASSIGN = FALSE
    FLAT_POLYGON_COUNT = FALSE
    CREATE_VUE_OUTPUT = TRUE
    REMOVE_DANGLING_PORT = UNTEXTED
}
PREPROCESS_OPTIONS {
    CELL_PROFILE = FALSE
    CELL_PROFILE_CNT=20
    CHECK_PATH_ENDPOINTS = TRUE
    CHECK_PATH_45 = TRUE
    CHECK_PATH_90 = FALSE
    DESIGN_STATS = TRUE
    TREE = TRUE
    CELL_STATS = TRUE
    PRINT_PREPROCESS_FILES = TRUE
}
TECHNOLOGY_OPTIONS {
    BAR_AUTO_EXPLODE = TRUE
    VIA_AUTO_EXPLODE = TRUE
    SUBLEAF_AUTO_EXPLODE = 6
    ALLOW_EXPLODE_WITH_TEXT = TRUE
    POST_VCELL_EXPLODE_CELL_SIZE <= 10
    EXPLODE_CELL_SIZE_PERCENT = 70
    CELL_SIZE_AUTO_EXPLODE <= 10
    EXPLODE_AREFS = FALSE
    EXPLODE_1XN_AREFS = FALSE
    EXPLODE_DATA_CELL_LIMIT = 4
```

```
    POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
    EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
    EXPLODE_BIG_SPARSE_CELL = TRUE
    EXPLODE_HOLDING_CELL_LIMIT = 1
    EXPLODE_PLACEMENT_LIMIT = 1
    POST_VCELL_EXPLODE_HIER_SPARSE_CELL = TRUE
    POST_VCELL_FLATTEN_TRAP_LIMIT != 0
    POST_VCELL_EXPLODE_LOW_MEMORY = TRUE
}
EVACCESS_OPTIONS {
    PATH = /remote/wwas1/hercules/venu/HERCULES_DOC/tutor/
    TUTORIAL_LAB/Getting_Started_Hercules_DRC/addertest3/
    run_details/evaccess
    LIBRARY = AD4FUL
    CREATE_MSG_VIEW = TRUE
    CREATE_NETLIST_VIEW = TRUE
    CREATE_XREF_VIEW = TRUE
    CREATE_GRAF_VIEW = TRUE
}
ASSIGN {
    tox     (1)
    poly    (5)
    well    (31)
    psel    (14)
    cont    (6)
    met1    (8)
    met2    (10)
    via     (19)} HIERARCHY = ERR_1.HIERARCHY

DATATYPE_OFFSET=FALSE. There will be no datatype difference
between FRAM and CEL views.
Input Library Format:  127 char cellnames
Output Library Format:  No output library

Reading hierarchy time = 0:00:00  User=0.04 Sys=0.01 Mem=15.028

Updating hierarchy time = 0:00:00  User=0.00 Sys=0.00 Mem=13.794

Preprocessing group files...
Warning #190: Adding default TECHNOLOGY_OPTIONS.  You can look
in the AD4FUL.tech file to see what options were used.
Preprocess Step 1 : Exploding
  Preprocessing time = 0:00:01  User=0.09 Sys=0.05 Mem=14.459

TECHNOLOGY_OPTIONS {
    VCELL_PASS   {
        STYLE = PAIRS
        ITERATE_MAX = 15
        ARRAY_ID = TRUE
        EXPLODE_INTO_VCELL = SMART
        MIN_COUNT = 20
        TOP_PERCENT_OF_VALUE = 40
    }
```

```
}
Preprocess Step 2 : Vcell_pass Arrays and Pairs Iteration 1
   Pairs time = 0:00:00  User=0.01 Sys=0.00 Mem=6.936

VCELL_PASS 1, no changes.

  Combined VCELL time = 0:00:00  User=0.01 Sys=0.01 Mem=6.936

Preprocess Step 3 : Post-VCell Explodes
  Post VCell time = 0:00:00  User=0.00 Sys=0.00 Mem=6.858

  Determine Region time = 0:00:00  User=0.00 Sys=0.00 Mem=11.239

0 Total self_intersect errors found.
  Create Layer Setup time = 0:00:00  User=0.00 Sys=0.00 Mem=11.239

  Preprocessing time = 0:00:00  User=0.00 Sys=0.00 Mem=11.317

Checking database:

CREATE_LAYER tox  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=tox
6 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.01 Sys=0.01 Mem=9.395

CREATE_LAYER poly  {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=poly
16 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.427

CREATE_LAYER well {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=well
3 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380

CREATE_LAYER psel {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=psel
6 unique polygons written.
```

```
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380

CREATE_LAYER cont {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=cont
38 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.380

CREATE_LAYER met1 {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met1
75 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.427

CREATE_LAYER met2 {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=met2
25 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.427

CREATE_LAYER via {
   SNAP_OPTIONS = {
      SNAP = 0.01
   }
} TEMP=via
53 unique polygons written.
0 Total self_intersect errors found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.395

HIERARCHY_CLEANUP HIERARCHY = ERR_2.HIERARCHY
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.146

REMOVE_HIERARCHY { HIERARCHY = ERR_1.HIERARCHY }
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.130

CLEAN_PREPROCESSOR
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=9.130

Total Create Layer time = 0:00:00  User=0.01 Sys=0.01 Mem=10.130

GRID_CHECK  {
   ASSIGN_LAYERS=TRUE
   tox={ CHECK_45=TRUE }
```

```
   poly={ CHECK_45=TRUE }
   well={ CHECK_45=TRUE }
   psel={ CHECK_45=TRUE }
   cont={ CHECK_45=TRUE }
   met1={ CHECK_45=TRUE }
   met2={ CHECK_45=TRUE }
   via={ CHECK_45=TRUE }
}(100)
No output written.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=9.349

EXTERNAL met1 {
   SPACING<3 } (101)
WARNING - 2 spacing violations found.
   Check time = 0:00:01  User=0.01 Sys=0.04 Mem=8.544

EXTERNAL met2 {
   SPACING<3
   LONGEDGE>=50
   LONGEDGE_TO_EDGE<5 } (102)
WARNING - 2 spacing violations found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.746

EXTERNAL met1 met2 {
   SPACING<1
   TOUCH=TRUE } (103)
WARNING - 2 spacing violations found.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=7.653

INTERNAL poly {
   SPACING<2
   EDGE_45<2 } (104)
WARNING - 1 width violation found.
   Check time = 0:00:00  User=0.01 Sys=0.01 Mem=7.590

INTERNAL met2 {
   SPACING<4
   CONVEX_TO_CONVEX<4 } (105)
WARNING - 2 width violations found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.684

CUT poly BY tox {
   CUT_OUTSIDE<1.5 } (106)
WARNING - 1 cut_outside violation found.
   Check time = 0:00:00  User=0.01 Sys=0.01 Mem=8.887

AREA via {  RANGE = [2,2]} (107)
No output written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.512

BOOLEAN poly AND tox { } PERM=gate(111)
8 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.715
```

```
BOOLEAN gate AND psel { } PERM=pgate(112)
4 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.762

BOOLEAN gate NOT pgate { } PERM=ngate(113)
4 unique polygons written.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=8.746

INTERNAL pgate {
   DIMENSIONS = [2,12] } (114)
WARNING - 1 violation found.
   Check time = 0:00:00  User=0.00 Sys=0.01 Mem=8.575

INTERNAL ngate {
   DIMENSIONS = [2,6] } (115)
WARNING - 1 violation found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.622

BOOLEAN cont NOT poly { } PERM=toxcont(121)
27 unique polygons written.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=8.887

ENCLOSE toxcont BY tox {
   SPACING<1.500 } (122)
WARNING - 4 enclose violations found.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=8.887

ENCLOSE toxcont BY met1 {
   SPACING<1
   PARALLEL=TRUE
   TOUCH=TRUE
   OVERLAP=TRUE } (123)
WARNING - 4 enclose violations found.
   Check time = 0:00:00  User=0.01 Sys=0.01 Mem=8.887

WRITE_MILKYWAY {
      LIBRARY_NAME = EX_ADDER_3_ref
      LIBRARY_PATH =
/remote/wwas1/hercules/venu/HERCULES_DOC/tutor/TUTORIAL_LAB/
Getting_Started_Hercules_DRC/addertest3
      TECHNOLOGY_FILE = EX_ADDER_3.tf
}
Use technology file "EX_ADDER_3.tf" to create milkyway library.
Technology file EX_ADDER_3.tf has been loaded successfully.
   Writing time = 0:00:00  User=0.06 Sys=0.01 Mem=11.434

Checks complete.
   Total check time = 0:00:00  User=0.10 Sys=0.04 Mem=12.011

Technology file hx2mw.tf has been loaded successfully.
   Saving ERR & PERM data time = 0:00:01  User=0.17 Sys=0.09
Mem=11.528
```

```
Technology file hx2mw.tf has been loaded successfully.
========== Cache Usage Status ==========
Initial cache size:              16384K bytes
Largest cache size:              17408K bytes
Current cache size:              16384K bytes

Largest number of cache blocks:     2
Current number of cache blocks:     1
Current number of cache pages:   4088
=======================================

Milkyway db session terminated successfully!

 Overall ev_engine time = 0:00:02  User=0.62 Sys=0.21 Mem=15.028
```

Viewing Data Creation Layers in Enterprise

In this section we look at the Data Creation layers generated by the Boolean checks for EX_ADDER_3. These layers are not error polygons, but are used for checking by the Hercules dimensional operators. We do not look at all of the generated polygons, but a representative sample that should be sufficient for you to use as a guideline to explore as much of the hierarchy tree as you desire. When you begin writing more complex runsets that include complex layer generation for DRC checking, you might want to view your intermediate Data Creation layers to check the accuracy of your DRC runset coding. Enterprise provides a quick and easy way to do this. If you are a runset writer, you should go through this section of the tutorial.

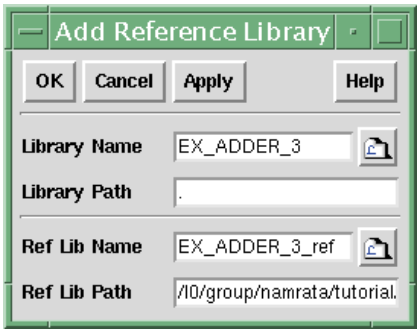
Open Enterprise in the addertest3 directory. Enter the command:

Enterprise &

Select Library > Open to open the input library “EX_ADDER_3”.

Select Library > Reference Libraries > Add.

Figure 4-7 Add Reference Library

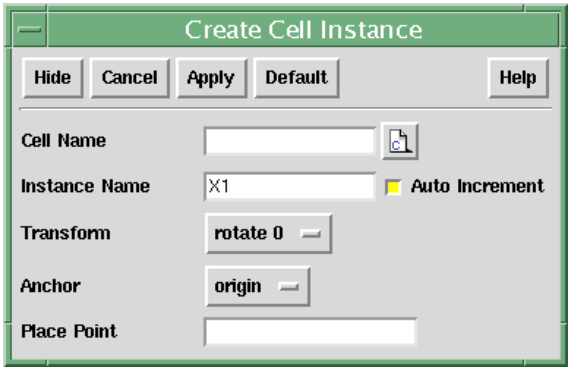


Add the EX_ADDER_3_ref library as the reference library for the EX_ADDER_3 library as shown in Figure 4-7.

Select Cell > Open and choose INV cell. Click OK to open this cell.

Select Create > Cell Instance.

Figure 4-8 Create Cell Instance

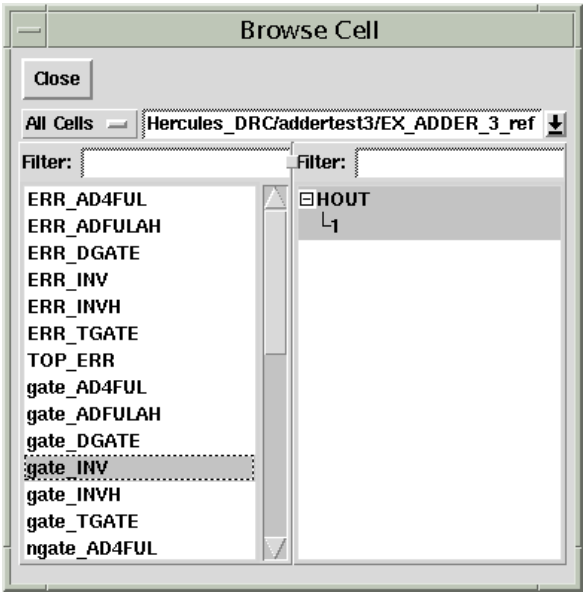


A Create Cell Instance pop-up window appears as shown in Figure 4-8. Selecting the symbol next to the Cell Name field allows you to browse a list of cells.

Select the EX_ADDER_3_ref library by using the down arrow button of the All Cells option in the Browse Cell window (see Figure 4-9).

Select the gate_INV cell and click Close. This adds the cell name to the Create Cell Instance window.

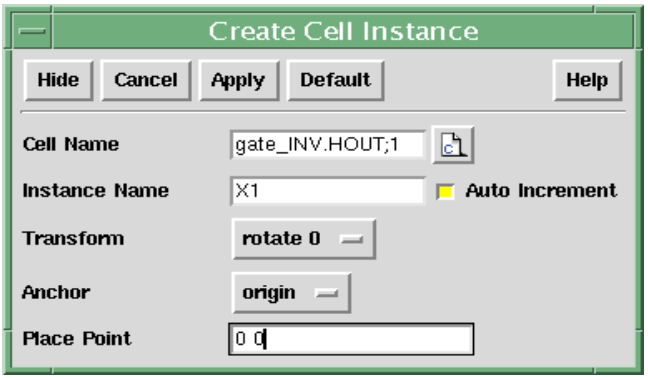
Figure 4-9 Browse Cell



Enter the coordinates (0 0) as shown in Figure 4-10.

Click Apply. This creates a cell instance of the master gate_INV from the reference library EX_ADDER_3_ref at the coordinates (0, 0).

Figure 4-10 Create Cell Instance



You now have the gate layer 111 overlaid on your original design. You can reference as many of the other layers as you like. For example, you can reference pgate_inv. Also, if you would rather look at only INV, replace all instances of INV with DGATE in the previous instructions.

If you would like to view the error polygon structures in Enterprise, you can do so now by opening the TOP_ERR block in the EX_ADDER_3_ref library. You should, however, also complete the next part of the tutorial, which uses Hercules-VUE to view the design errors.

Note:

Once you have finished viewing the error polygon structures, close the existing library and exit the Enterprise session. To quit Enterprise, in the command window enter:

exit

Introducing Hercules-VUE

Hercules-VUE, used in combination with Enterprise, Virtuoso Layout Editor, or IC WorkBench EV Plus, makes it easy to locate a design error. The following sections introduce the fundamentals of running Hercules-VUE and provide basic examples to familiarize you with the program. These examples assume you are using Enterprise as your layout viewing tool. For instructions on using other layout tools, refer to the *Hercules Reference Manual*.

The following instructions for using Hercules-VUE to process runsets assume that you have not run Hercules on adderdrc3.ev in the UNIX shell. If you have run in the UNIX shell, be aware that most of the information in the Hercules-VUE dialog boxes is filled in automatically.

Running Hercules-VUE

Begin by making sure you are in the addertest3 directory.

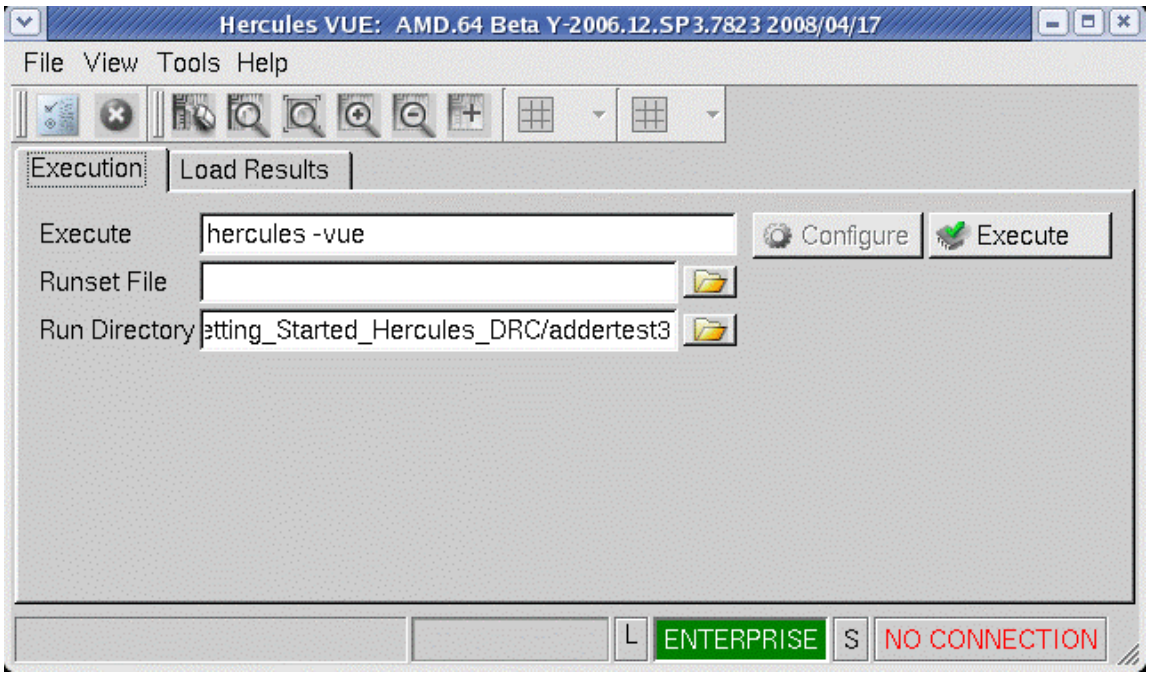
Run Enterprise by using the command:

Enterprise &

From the Verification menu select Hercules > Hercules VUE.

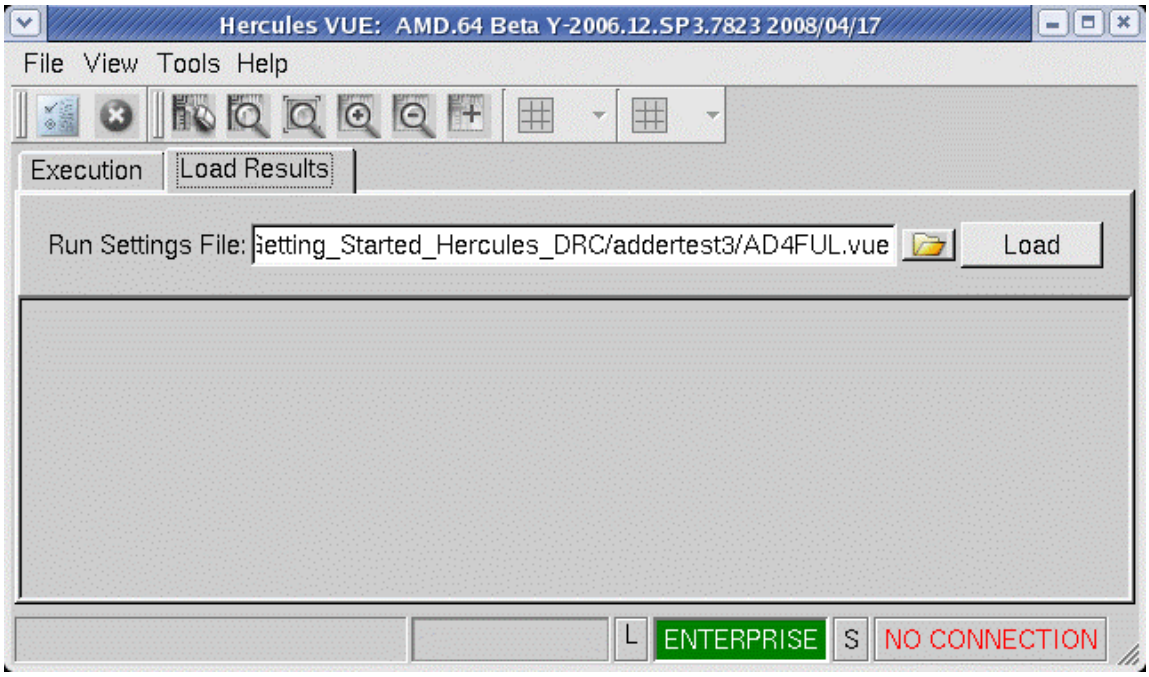
The Hercules-VUE dialog box appears, as shown in Figure 4-11.

Figure 4-11 Initial Hercules-VUE Dialog Box



Since we have already executed Hercules, we will now load the results. Select Load Results tab.

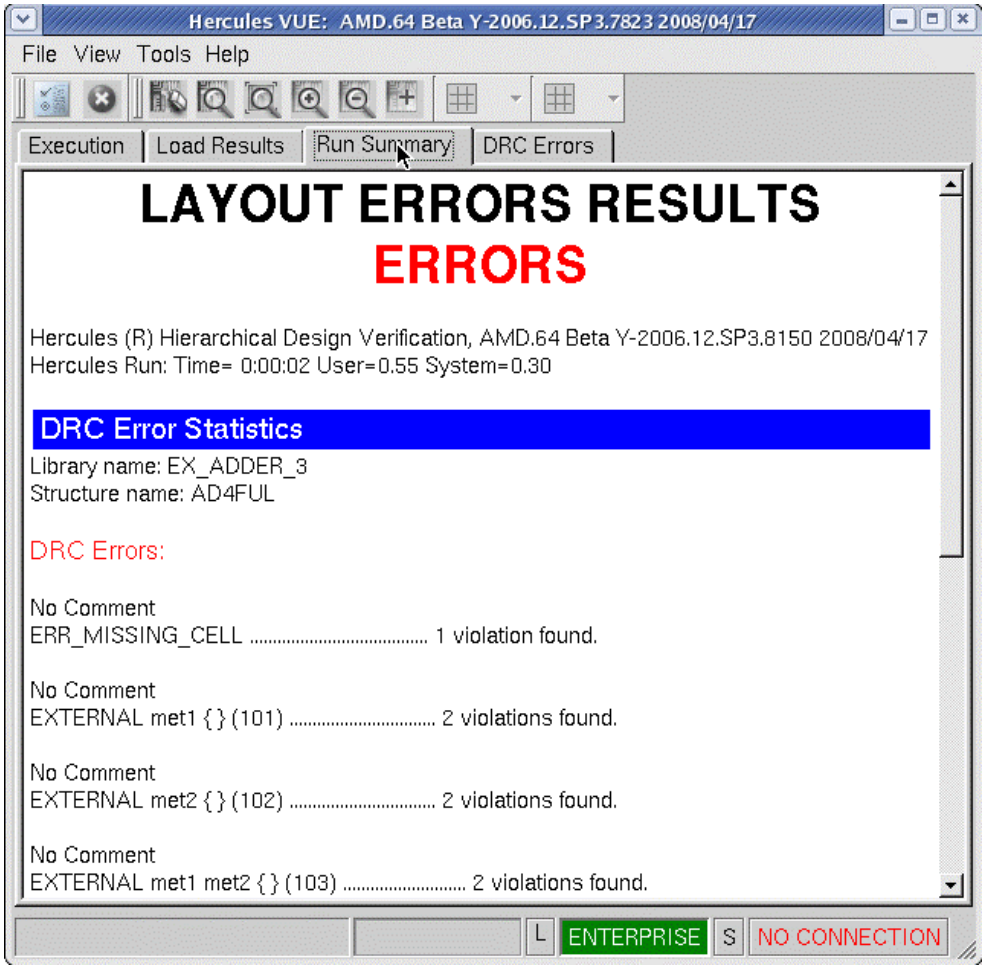
Figure 4-12 Hercules Load Results



Click Load to load the AD4FUL.vue file in the addertest3 directory.

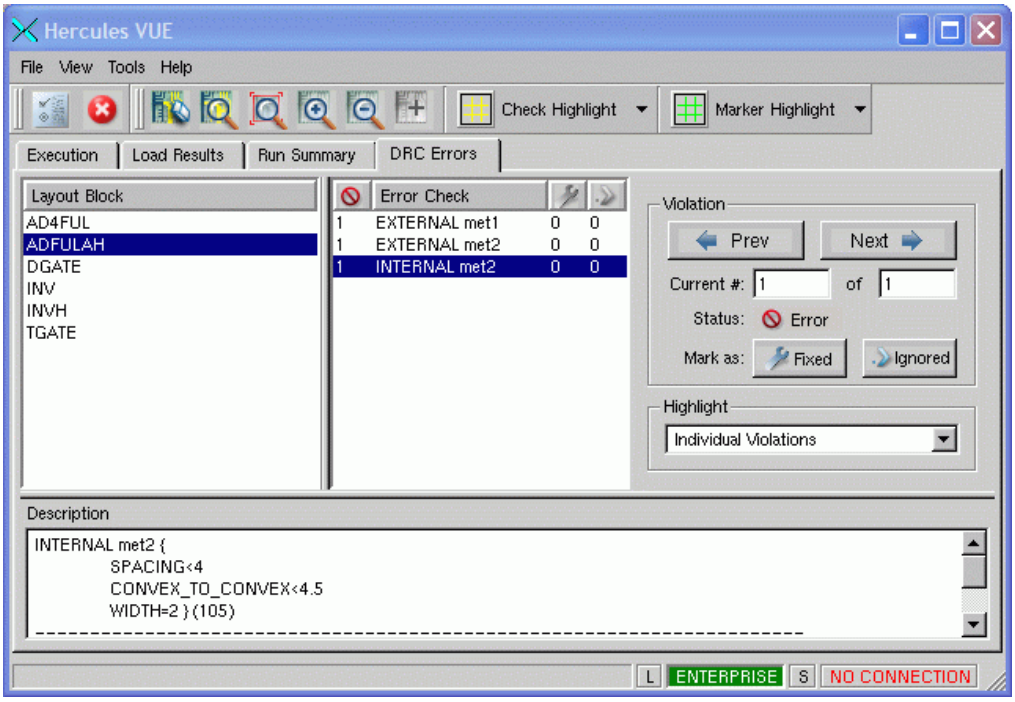
The Run Summary tab appears with overall layout errors as shown in Figure 4-13.

Figure 4-13 Hercules-VUE Main Window after Loading



Next we will load the DRC errors by choosing DRC Errors tab. Select AD4FUL in the Layout Block pane. When the structures are loaded, Enterprise displays the top-level AD4FUL structure.

Figure 4-14 Hercules-VUE Window with EX_ADDER_3 Design Rule Violations



The window contains several sections, including:

- The Layout Block window, which lists the Enterprise library structures.
- The Error Check window, which lists a shortened version of the design rule errors for that structure. The number to the left of the Error Check field indicates how many error polygons are created for each design rule. The Prev and Next buttons allow you to examine each of these error polygons within a design rule violation.
- The Description window, which lists the selected design rule as it appears in the runset file.

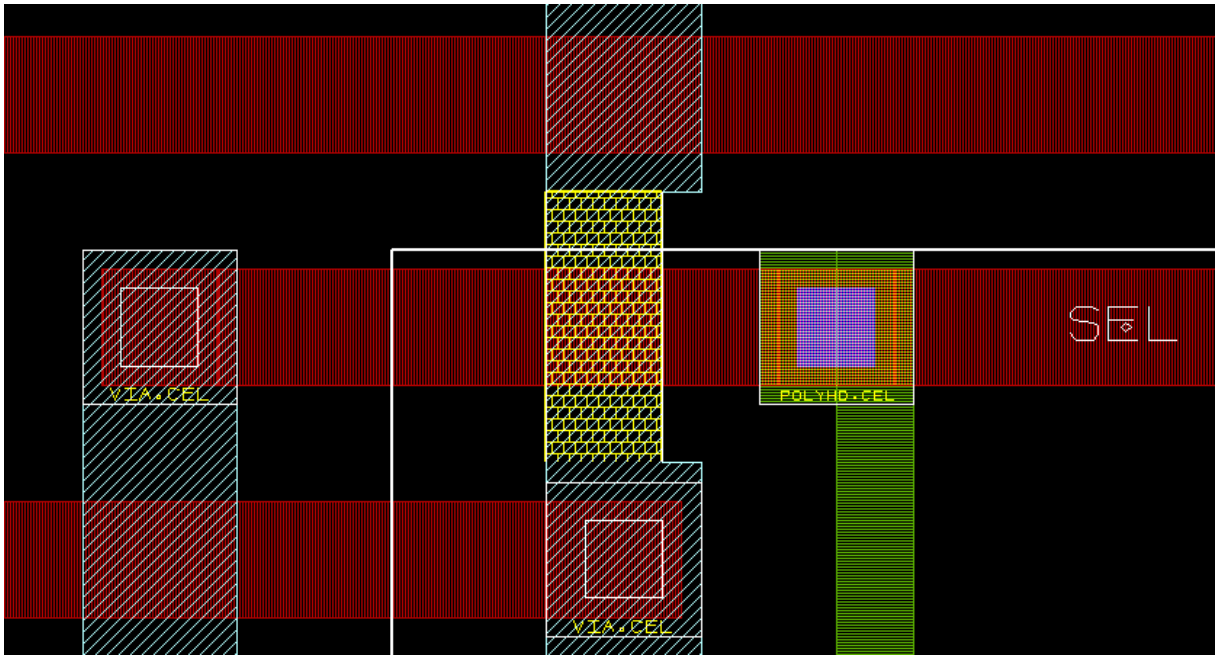
Now, select ADFULAH in the Layout Block window (See Figure 4-14.).

Select the INTERNAL met2 command in the Error Check window.

Viewing Errors Using Hercules-VUE

The rule, indicated in the Error Check window, flags met2 polygons (layer 10) that do not meet the required 4.0 µm minimum width spacing, as shown in Figure 4-15.

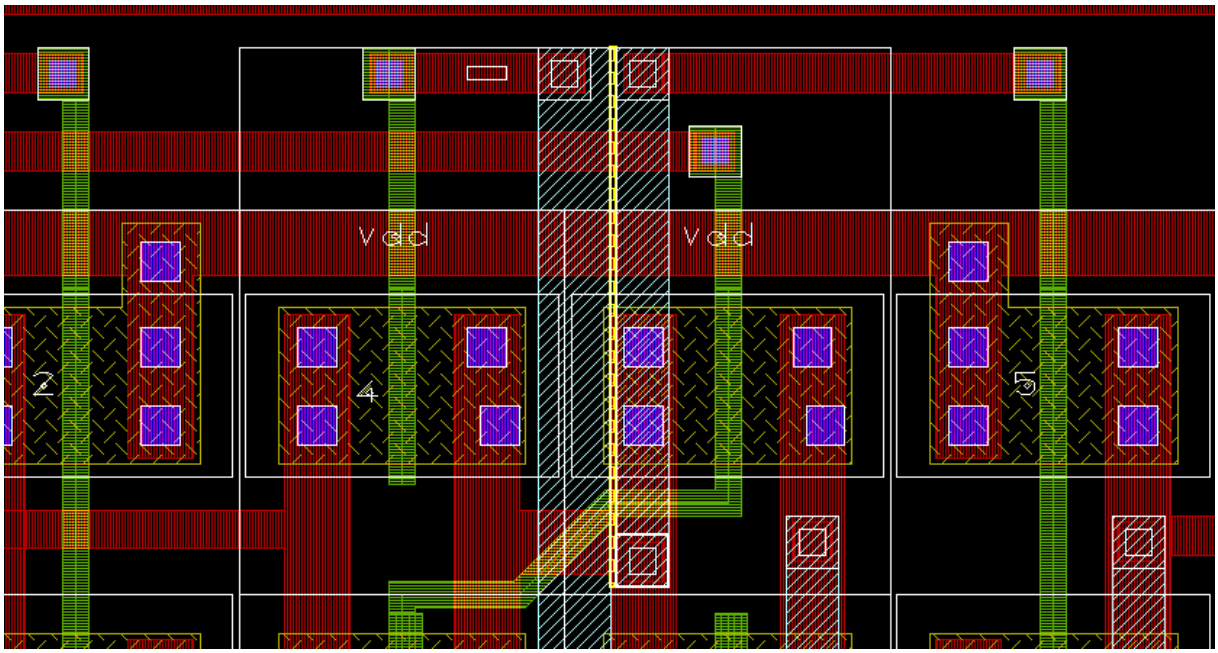
Figure 4-15 ADFULAH INTERNAL Error Displayed in Enterprise Using Hercules-VUE



Keep ADFULAH as your selected Layout Block and select the EXTERNAL met2 rule.

Again, a spacing violation is flagged. This time, met2 must have a distance of 3 µm from the other met2 (on layer 10), as shown in Figure 4-16.

Figure 4-16 ADFULAH EXTERNAL Error Displayed in Enterprise Using Hercules-VUE



Using the Checks Option

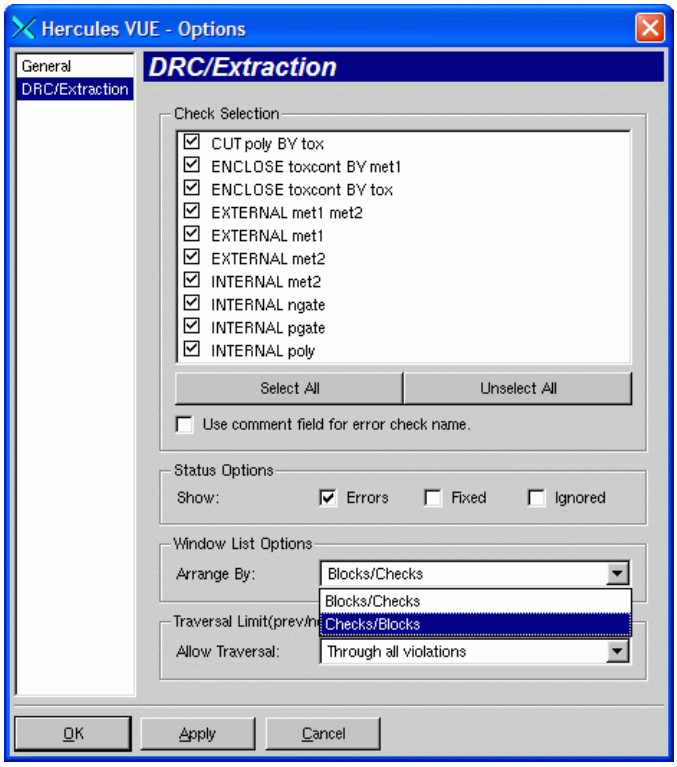
Continue to explore all of the errors in the design. The organization of error polygons is by structure, with a list of error polygons displayed for each structure in the Enterprise library. To organize errors according to design rules, VUE allows you to select a rule and list the structures affected by the rule.

To view errors with this organization, start by choosing Tools > Options menu button along the top of the main window. A new window opens, as shown in Figure 4-17.

Select “DRC/Extraction”. The window lists all design rules that generate error polygons. Initially the window enables the display of all errors.

Under Window List Options, select arrange by Checks/Block from the pull-down menu and click OK.

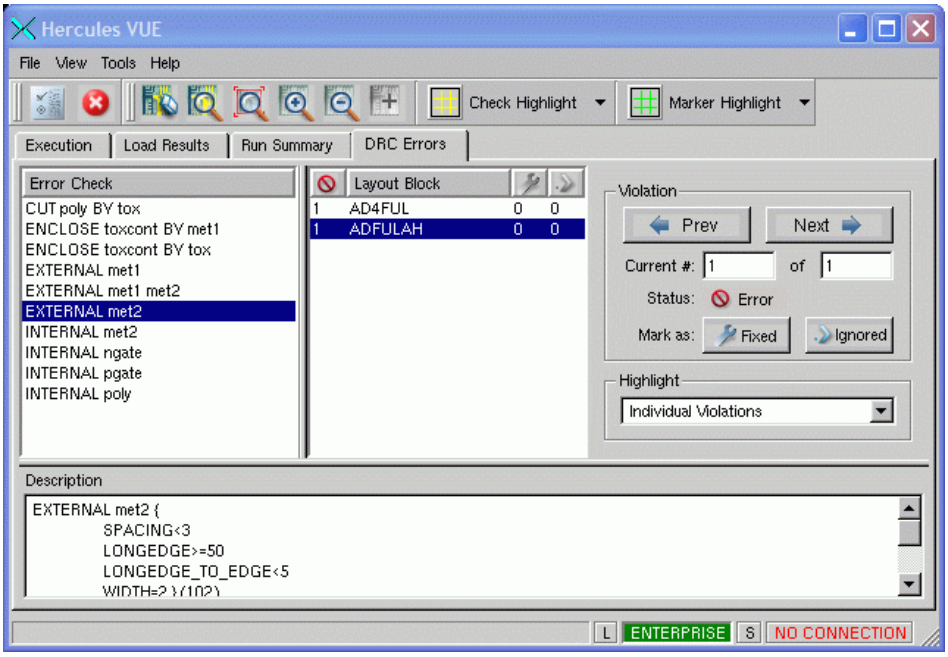
Figure 4-17 Hercules VUE-Options Menu



Select the EXTERNAL met2 rule to enable the display of this error.

Select ADFULAH from the Layout Block window. The main Hercules-VUE window appears as shown in Figure 4-18.

Figure 4-18 Hercules-VUE Main Window with Checks/Blocks Displayed



The window shows only those structures affected by the selected rule. Any single rule or combination of rules can be selected with this method to further aid you in analyzing your design.

As these examples show, Hercules-VUE makes the process of analysis much easier than opening structures and analyzing the error polygons that appear. You can also continue to use Hercules-VUE while editing the design if you have a complete Enterprise editing license or are using Hercules-VUE with another supported editing tool. To correct all of the design errors in this mode, be sure to read the *Hercules VUE User Guide*, Running Hercules Within VUE chapter, which provides a complete list of the features and capabilities.

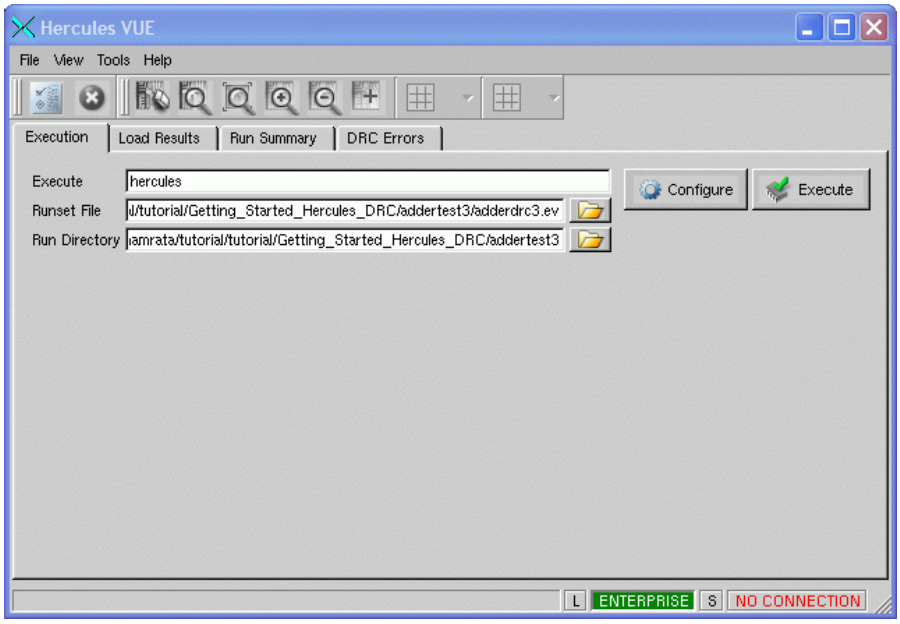
Note:

The chosen runset rules for the tutorial examples flag errors that cannot easily be fixed.

Running Hercules Inside of Hercules-VUE

At the top left of the screen, select the Execution tab. The Execute Hercules dialog box appears, as shown in Figure 4-19.

Figure 4-19 Execute Hercules Dialog Box



Enter the Runset File (adderdrc3.ev), and the path to the runset in their respective fields.

Click the Execute button to run Hercules on adderdrc3.ev. We are rerunning our EX_ADDER_3 example so you can see how to execute Hercules from the Hercules-VUE GUI.

When Hercules is finished, the Load Hercules Data window automatically appears and loads the block name, AD4FUL. The Layout Block window is refreshed. Select File > Exit to exit Hercules-VUE and Tools > Quit to exit Enterprise.

What's Next?

Chapter 5, "Hercules DRC Migration," is for Dracula physical verification users who need to convert Dracula physical verification runsets to Hercules runsets. You learn how to use various translation options in order to become familiar with operational and syntactical differences between the two tools. If you have already completed Chapters 5 and 6, or if you are not a current Dracula physical verification user, continue on to Chapter 7, "Introduction to Hercules HLVS," for an introduction to Hercules LVS.

5

Hercules DRC Migration

In this chapter we convert two simple DRC Dracula physical verification rule files to Hercules runsets, using various translation options to familiarize you with operational and syntactical differences between the two tools. In the next chapter you run Hercules on a design rule check (DRC) database in the Virtuoso Layout Editor environment. The database includes some errors which you view with Hercules-VUE, the graphical debug tool from Synopsys that interfaces to Virtuoso Layout Editor.

Summary of Progress to This Point

In the first two examples using Hercules, we explained in detail all the steps required to set up the software to check a simple design for a single runset rule. Although the examples were very basic, the procedures and files created were nearly the same for runsets of any complexity. Now that you are familiar with Hercules and its syntax, we take you through the process of converting a Dracula physical verification rule file into a Hercules runset.

Learning Objectives for This Chapter

In this part of the tutorial, we try to simulate a real design environment so you have a better understanding of how to apply Hercules to your application. We introduce you to Drac2He, the Dracula physical verification rule file translator. We present two simple examples of Dracula physical verification rule files, which we translate using different Drac2He options. With these examples you learn the following:

- The general syntax conversion that occurs between Dracula physical verification and Hercules
- Several common Drac2He command-line options
- How to run Drac2He with the Dracula physical verification rule file using different command-line options, as well as how to interpret the output files from Drac2He using these different options
- Operational differences between Hercules and the Dracula physical verification, including error hierarchy and SUBSTRATE or BULK processing for DRC

Before You Start

Your account should already be set up for this example. If it is not, see Chapter 1, “Installation and Setup.” Before starting this chapter, you should complete the exercises in Chapter 2, “An Error-Free Design for DRC” and Chapter 3, “Single Design Error for DRC.”

What Is Drac2He?

Hercules contains a Dracula physical verification rule file translator, Drac2He. Eecute the translator from your UNIX command line using your Dracula physical verification DRC, LVS, or ERC rule file as an argument.

Note:

Hercules also has the -dracula command-line option, which allows you to give Hercules a Dracula physical verification rule file as input instead of a Hercules runset, and run Drac2He as part of Hercules. Because this part of the tutorial concentrates on the output from Drac2He, we do not use the -dracula option to run Drac2He inside of Hercules, but instead run Drac2He separately on the command line.

Using the Migration Tutorials

Before you start the Hercules Migration part of the tutorial, review and execute the tutorials in Chapter 2, “An Error-Free Design for DRC” and Chapter 3, “Single Design Error for DRC.” This guarantees that your Hercules installation is correct, and give you some understanding of the Hercules execution and debug process. The earlier tutorials should take you less than 20 minutes to review.

Learning Method: Design Example

Throughout this part of the tutorial, we use a four-bit full adder as the sample design, which is the same example used in the Hercules DRC tutorials in previous chapters. This section of the manual concentrates on the actual conversion from Dracula physical verification to Hercules, assuming that you have a basic understanding of Hercules and Hercules-VUE.

Getting Started with Drac2He

In the first example you take the simple Dracula physical verification rule file located in the *your_path*/tutorial/Getting_Started_Drac2he_DRC/migration1/ directory and translate it to a Hercules runset using three different Drac2he command-line options, -OutType, -rc, and -N. These options demonstrate some general differences between Hercules and Dracula physical verification conventions and formats. The following is a list of the three options and their purposes:

Note:
In “Other Dracula Physical Verification Options” on page 5-4 we summarize the numerous other options.

-OutType

As you learned in the initial Hercules runset example in Chapter 2, “An Error-Free Design for DRC,” you can output PERM layers or error hierarchy or both. When you run Drac2He, the default outputs both PERM layers and error hierarchy. This directly mimics what Dracula physical verification requires as output for all dimensional checks. The -OutType option allows you to specify outputting PERM layers or error hierarchy. We demonstrate turning off the output of the PERM layers and only outputting the error hierarchy. Because the Hercules syntax does not require a PERM layer to be defined, outputting only error hierarchy is the preferred methodology, unless you want to add the layer to your database or view it in a layout editor. For a more detailed explanation of PERM, ERROR, and TEMP layer outputs, refer to Writing Output Results to Files in Chapter 4, “A Complete Design for DRC.”

-rc

Historically, Hercules runsets use uppercase for input layers and lowercase for derived layers. This is not required, but recommended to help make the runset easier to read. By default, Drac2He automatically generates lowercase derived layers and uppercase input layers. To produce a runset that has the identical case as the input runset, you need to specify the -rc command line option.

-N

All of the Dracula physical verification syntax is copied by default into the translated runset and placed in comments. This makes it easy for previous Dracula physical verification users to see the correlation between the Dracula physical verification and Hercules commands and options. In some cases, however, it makes it harder to read the resulting runsets because they are filled with extraneous comments. The -N option allows you to disable the generation of those comments.

Other Dracula Physical Verification Options

The examples in this chapter demonstrate the most commonly used options for DRC translations. The following is a list of the main Dracula physical verification translator options. For a complete explanation of all the options, see *Hercules General Usage Information*.

```
/l0/synopsys/bin/SUN64_57_hercules/drac2he [-BaseCell cellname] [-E
errorfile] [-FE errorfile] [-GateArray] [-IncTechDefs] [-LvlSize]
[-MinCellOverlap number] [-mosFilter] [-N] [-nofixtemp] [-outgds] [-
OutType perm|error] [-Parse] [-rc] [-ShareBaseCell] [-Size number]
[-text_map textfile] [-V] [-Version] commandfile
```

Argument	Description
`-BaseCell cellname`	Specifies gate array base cell. Uses double quotes around the list when specifying more than one.
`-E errorfile`	Writes errors to file.
`-FE errorfile`	Writes only fatal errors to file.
`-GateArray`	Sets up TECHNOLOGY_OPTIONS for gate array design.
`-IncTechDefs`	Includes default TECHNOLOGY_OPTIONS section in runset.

Argument	Description
`-LvlSize`	Sets LEVEL_NON_ORTHOGONAL for all SIZE commands.
`-MinCellOverlap` *number*	Specifies MIN_CELL_OVERLAP percent for sets vcell pass
`-mosFilter`	Turns on filtering for MOS devices. Specifies Hercules MOS filter options. Uses double quotes around the list when listing more than one option.
`-N`	Produces non-commented runset.
`-nofixtemp`	Turns off optimization of Dracula physical verification TEMPORARY-LAYERS.
`-outgds`	Sets output format to GDS.
`-OutType error | perm`	Chooses error hierarchy or PERM output definitions for Dracula physical verification OUTPUT cells.
`-Parse`	Parses only command file.
`-rc`	Retains case for all layers as specified in the Dracula physical verification rule file.
`-ShareBaseCell`	Turns on SHARED_BASE_CELL for gate array designs.
`-Size` *number*	Value by which to increment SIZE.
`-text_map` *textfile*	Uses a TEXT_MAP file for lower-level text.
`-V`	Prints Drac2He version.
`-Version`	Prints product and library version.
commandfile	Dracula physical verification command file.

How to Run Drac2He

Now you run three different Drac2He translations on the first Dracula physical verification rule file example, Migration1. We do not review the results of Migration1 examples until you have completed all of them.

Be sure that you are in the directory where your migration1.drc file is located, *your_path*/hercules-Examples/Getting_Started_Drac2he_DRC/migration1.

For the default example, enter the command:

drac2he migration1.drc > drc1_default.ev

No data is displayed on your screen while the translator runs. When the translation is complete, drc1_default.ev contains your Hercules runset ready for execution. You do not run Hercules at this time. First you run some more example translations.

The following command turns off the generation of the PERM layers because you specify the -OutType as only ERROR:

drac2he -OutType error migration1.drc > drc1_error.ev

The following example retains the Dracula physical verification rule file case and turns off the generation of comments in the Hercules runset:

drac2he -rc -N migration1.drc > drc1_case_nocomments.ev

Translation Results for Migration1 Example

Now look at the resulting Hercules runset files. Example 5-1, Example 5-2, and Example 5-3 show excerpts from the output files and briefly explain the files preceding each figure, to help you identify what is important in the files.

Output from First Dracula Physical Verification Translation

Example 5-1 shows the output from the first Drac2He run. If you completed the tutorial in Chapter 1, “Installation and Setup,” most of the runset sections should look familiar. There are a few items (which are emphasized) that you should be sure to notice.

Example 5-1 Drac2He Output from Migration1 Example: drc1_default.ev

```
/*Converted runset  drac2he: VERSION DATA OMITTED
 * Called as:  drac2he migration1.drc
 */
/*    ;*************************************************************
      ;                  D E S C R I P T I O N   B L O C K
      ;*************************************************************
 */

/*  *DESCRIPTION
   PRIMARY = AD4FUL
   SYSTEM = GDS2
   INDISK = DRAC_example.GDS
   OUTDISK = OUTPUT
   SCALE = 0.001 MICRONS
   RESOLUTION = 0.001 MICRONS
```

```
   MODE = EXEC NO
   PROGRAM-DIR = ~/example/drac/
   KEEPDATA = INQUERY
   LISTERROR = 300
   CHECK-MODE = FLAT
*END  */
HEADER {
   INLIB = DRAC_example.GDS
   OUTLIB = EV_OUT
   BLOCK = AD4FUL
   GROUP_DIR = group
   FORMAT = GDSII
   OUTPUT_FORMAT = LTL
   OUTPUT_LAYOUT_PATH = .
}
OPTIONS {
   IGNORE_CASE=TRUE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.001
   NET_PREFIX = N_
}
EVACCESS_OPTIONS {
   PATH = evaccess
   CREATE_VIEWS = FALSE
}

TEXT_OPTIONS {
   USE_COLON_TEXT=TRUE
   TRUNCATE_FLAG=FALSE
   REMOVE_TEXT_FROM_SHORT=TRUE
   CONNECT_BY_NAME = MIXED_MODE
   ATTACH_TEXT = ALL
}

  /*
    ;*****************************************************************
    ;                   I N P U T - L A Y E R B L O C K
    ;*****************************************************************
  */

/*  BEGIN INPUT LAYER BLOCK  */
/*  *INPUT-LAYER  */
ASSIGN {
   TOX    (1)    /* TOX = 1  */
   POLY   (5)   /* POLY = 5  */
   WELL   (31)  /* WELL = 31  */
   PSEL   (14)  /* PSEL = 14  */
   CONT   (6)   /* CONT = 6  */
   MET1   (8)   text(63)   /* MET1 = 8 TEXT = 63  */
   MET2   (10)  /* MET2 = 10  */
   VIA    (19)   /* VIA = 19  */}

/*  END BLOCK  */
```

```
/*  *END  */

  /*
    ;*****************************************************************
    ;                  O P E R A T I O N    B L O C K
    ;*****************************************************************
  */

/*  BEGIN OPERATION BLOCK  */
/*  *OPERATION  */

/*  ;check contacts are .3x.3  */

/*  WIDTH[L] CONT SELNE 0.3 OUTPUT cont_err1_Out 98   */
INTERNAL CONT {
   COMMENT = "WIDTH[L] CONT SELNE 0.3 OUTPUT CONT_err1 98 "
   VERBOSE=TRUE
   DIMENSIONS = [0.300,0.300] } PERM=cont_err1_Out (98)

/*  ;metal1 spacing not less than 3.0  */

/*  EXT MET1 LT 3 OUTPUT met1_err1_Out 100   */
EXTERNAL MET1 {
   COMMENT = "EXT MET1 LT 3 OUTPUT MET1_err1 100 "
   SPACING<3.000
   VERBOSE=TRUE } PERM=met1_err1_Out (100)

/*  EXT MET1 LT 0.4 OUTPUT e02420_Out 80   */
INTERNAL MET1 {
   POINT_TOUCH=FALSE
   NON_PARALLEL=FALSE
   OUTPUT_EDGES=TRUE
   SEGMENT>10.000} TEMP=met1_edges
EXTERNAL met1_edges {
   COMMENT = "EXT MET1 LT 0.4 &"
   SPACING<0.400
   OUTPUT_EDGES=TRUE } PERM=e02420_Out (80)

/*  LENGTH COMMAND TRANSLATED ABOVE  */
/*  LENGTH MET1 GT 10 OUTPUT e02420_Out 80   */

/*  ;metal 1 to metal 2 spacing not less than 1.0
  */

/*  EXT[T] MET1 MET2 LT 1 OUTPUT met1_err2_Out 101   */
EXTERNAL MET1 MET2 {
   COMMENT = "EXT[T] MET1 MET2 LT 1 OUTPUT MET1_err2 101 "
   SPACING<1.000
   TOUCH=TRUE
   VERBOSE=TRUE } PERM=met1_err2_Out (101)

/*  ;* ------------------------------- DRC data creation for contacts*
  */
```

```
/*  NOT CONT POLY toxcont OUTPUT toxcont_Out 121   */
BOOLEAN CONT NOT POLY {  VERBOSE=TRUE } PERM=toxcont_Out (121)
COPY toxcont_Out { } TEMP=toxcont

/*  ENC toxcont TOX LT 1.5 OUTPUT toxcont_err1_Out 122   */
ENCLOSE toxcont BY TOX {
   COMMENT = "ENC toxcont TOX LT 1.5 OUTPUT TOXCONT_err1 122 "
   SPACING<1.500
   VERBOSE=TRUE } PERM=toxcont_err1_Out (122)

/*  ENC[TO] toxcont MET1 LT 1 OUTPUT toxcont_err2_Out 123   */
ENCLOSE toxcont BY MET1 {
   COMMENT = "ENC[TO] toxcont MET1 LT 1 OUTPUT TOXCONT_err2 123 "
   SPACING<1.000
   TOUCH=TRUE
   OVERLAP=TRUE
   VERBOSE=TRUE } PERM=toxcont_err2_Out (123)
DISCONNECT
```

DRACULA_DEFAULTS, OPTIONS Section

Look at the OPTIONS section in the drc1_default.ev file. The DRACULA_DEFAULTS option has been added and set to TRUE. Hercules has default settings for all runset options as well as all command options. In order to make your Hercules runset output match the output from Dracula physical verification, Hercules has a master option, DRACULA_DEFAULTS, designed to alter the necessary option settings.

OUTPUT Hierarchy

As was mentioned previously, all of the output data from dimensional checks is written to a PERM layer. For example, the first dimensional command, INTERNAL, has the output written to PERM=cont_err1_out (98). The VERBOSE option is also set in the dimensional commands, causing the output of the INTERNAL to be written to the error hierarchy. This is done to match the OUTPUT of Dracula physical verification, but is not necessary. In the next translation of migration1.drc, the OUTPUT is written only to the error hierarchy.

COMMENT Option

Each Hercules dimensional command contains the COMMENT option. This option adds user comments, which appear in the summary file and in the error file. The comments also appear in the Hercules-VUE information window with each command. Drac2He automatically copies the original Dracula physical verification command into the COMMENT option to help former Dracula physical verification users understand the Hercules version of the command they are trying to execute.

COMMENTS in the Hercules Runset

Hercules uses /* */ to designate all comments. When you run Drac2He, all of your Dracula physical verification options, commands, and comments are placed by default in comments in the resulting Hercules runset. This helps to teach you how the new Hercules syntax you are reading matches Dracula physical verification physical verification tool syntax.

CASE of LAYERS

All ASSIGN_LAYERS are in uppercase and all TEMP layers, or derived layers, are in lower case. For example, MET1, CONT, and MET2 are ASSIGN layers. TEMP layers are met1_edges and toxcont.

CONJUNCTIVE and COPY Commands

Look at the INTERNAL MET1/EXTERNAL met1_edges combination for an example of a Dracula physical verification CONJUNCTIVE check and how it is translated. There is also an example of how a COPY command is used if you perform a duplicate command, such as the BOOLEAN AND between CONT and POLY. The translator Drac2He uses the COPY command simply to copy the output of the first command to the output layer of the second command instead of performing the command twice.

Output with PERM Omitted

In Example 5-2, notice the keyword PERM is gone. Now the output from each dimensional command is written only to the error hierarchy. Data Creation commands are still written to TEMP layers, which are deleted after they are used by all of the commands that need them. For a detailed explanation of PERM, ERROR, and TEMP output hierarchy, see the *Hercules Reference Manual*.

Example 5-2 Drac2He Output when OutType Option is Set to ERROR in Migration1: drc1_error.ev

```
/*Converted runset  drac2he: VERSION DATA OMITTED
 * Called as:  drac2he -OutType error migration1.drc
 */

          ................. DATA OMITTED ..................

 /*  WIDTH[L] CONT SELNE 0.3 OUTPUT CONT_err1 98   */
INTERNAL CONT {
   COMMENT = "WIDTH[L] CONT SELNE 0.3 OUTPUT CONT_err1 98 "
   DIMENSIONS = [0.300,0.300] } (1;0)

/*  ;metal1 spacing not less than 3.0  */

/*  EXT MET1 LT 3 OUTPUT MET1_err1 100   */
EXTERNAL MET1 {
   COMMENT = "EXT MET1 LT 3 OUTPUT MET1_err1 100 "
```

```
   SPACING<3.000 } (1;1)

/*  EXT MET1 LT 0.4 OUTPUT E02420 80   */
INTERNAL MET1 {
   POINT_TOUCH=FALSE
   NON_PARALLEL=FALSE
   OUTPUT_EDGES=TRUE
   SEGMENT>10.000} TEMP=met1_edges
EXTERNAL met1_edges {
   COMMENT = "EXT MET1 LT 0.4 &"
   SPACING<0.400
   OUTPUT_EDGES=TRUE } (1;2)

/*  LENGTH COMMAND TRANSLATED ABOVE  */
/*  LENGTH MET1 GT 10 OUTPUT E02420 80   */

          ..................... DATA OMITTED ..................
```

-rc and -N Options

In our final Drac2He output example, Example 5-3, we specify the -rc and -N options. All of the layers are in uppercase, just as they are in the Dracula physical verification rule file. This is because of the -rc (retain case) option. Also, the runset is much shorter because the -N option specified no comments to the output. You still have the original Dracula physical verification syntax in the COMMENT field of each dimensional check for your reference.

Example 5-3 Drac2He Output with -rc and -N Options Specified: drc1_case_nocomment.ev

```
/*Converted runset  drac2he: VERSION DATA OMITTED */

HEADER {
   INLIB = DRAC_example.GDS
   OUTLIB = EV_OUT
   BLOCK = AD4FUL
   GROUP_DIR = group
   FORMAT = GDSII
   OUTPUT_FORMAT = LTL
   OUTPUT_LAYOUT_PATH = .
}

OPTIONS {
   IGNORE_CASE=TRUE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.001
   NET_PREFIX = N_
}

EVACCESS_OPTIONS {
   PATH = evaccess
   CREATE_VIEWS = FALSE
}
```

```
TEXT_OPTIONS {
   USE_COLON_TEXT=TRUE
   TRUNCATE_FLAG=FALSE
   REMOVE_TEXT_FROM_SHORT=TRUE
   CONNECT_BY_NAME = MIXED_MODE
   ATTACH_TEXT = ALL
}

ASSIGN {
   TOX    (1)
   POLY   (5)
   WELL   (31)
   PSEL   (14)
   CONT   (6)
   MET1   (8)   text(63)
   MET2   (10)
   VIA    (19)
}

INTERNAL CONT {
   COMMENT = "WIDTH[L] CONT SELNE 0.3 OUTPUT CONT_err1 98 "
   VERBOSE=TRUE
   DIMENSIONS = [0.300,0.300] } PERM=CONT_err1_Out (98)

EXTERNAL MET1 {
   COMMENT = "EXT MET1 LT 3 OUTPUT MET1_err1 100 "
   SPACING<3.000
   VERBOSE=TRUE } PERM=MET1_err1_Out (100)

INTERNAL MET1 {
   POINT_TOUCH=FALSE
   NON_PARALLEL=FALSE
   OUTPUT_EDGES=TRUE
   SEGMENT>10.000} TEMP=MET1_edges

EXTERNAL MET1_edges {
   COMMENT = "EXT MET1 LT 0.4 &"
   SPACING<0.400
   OUTPUT_EDGES=TRUE } PERM=E02420_Out (80)

EXTERNAL MET1 MET2 {
   COMMENT = "EXT[T] MET1 MET2 LT 1 OUTPUT MET1_err2 101 "
   SPACING<1.000
   TOUCH=TRUE
   VERBOSE=TRUE } PERM=MET1_err2_Out (101)

BOOLEAN CONT NOT POLY { VERBOSE=TRUE } PERM=TOXCONT_Out (121)

COPY TOXCONT_Out { } TEMP=TOXCONT

ENCLOSE TOXCONT BY TOX {
   COMMENT = "ENC TOXCONT TOX LT 1.5 OUTPUT TOXCONT_err1 122 "
```

```
   SPACING<1.500
   VERBOSE=TRUE } PERM=TOXCONT_err1_Out (122)

ENCLOSE TOXCONT BY MET1 {
   COMMENT = "ENC[TO] TOXCONT MET1 LT 1 OUTPUT TOXCONT_err2 123 "
   SPACING<1.000
   TOUCH=TRUE
   OVERLAP=TRUE
   VERBOSE=TRUE } PERM=TOXCONT_err2_Out (123)

DISCONNECT
```

This concludes our first set of simple Drac2He examples. The next section goes into more detail on how substrate processing and Drac2He errors are handled.

Running Drac2He With Warnings and Errors

In the next example you translate a Dracula physical verification rule file that contains SUBSTRATE processing commands, as well as syntax errors.

Go to the directory that contains the second migration test, *your_path*/hercules-Examples/Getting_Started_Drac2he_DRC/migration2. Enter the command:

drac2he migration2.drc > drc2_default.ev

No data is displayed on your screen while the translator runs. When the translation is complete, drc2_default.ev contains your Hercules runset with some translation errors and warnings. You do not run Hercules at this time, but instead run another translation example.

The following command redirects any errors realized during translation to a separate error file called 'error.out', rather than outputting the errors to the translated runset:

drac2he -E error.out migration2.drc > drc2_errorout.ev

Translation Results for Migration2 Example

You now take a look at the resulting runset files and review how warnings and errors are reported during Drac2He translations.

Output of Translated migration2.drc File

Example 5-4 shows the output from the first Drac2He translation of the migration2.drc file. Notice that all ERRORS are located at the top of the runset file. The line containing the error is not output in the runset, except in the ERROR comments. Even though an ERROR

occurs, Drac2He tries to create a working runset by commenting out the syntax it cannot translate. All WARNING messages are placed throughout the runset at the same location as the actual syntax in question. (Our next example, Example 5-5, illustrates how you can choose to output the ERRORS and WARNINGS to a separate file.)

Example 5-4 Drac2He Output with ERRORS and WARNINGS Written to Runset File drc2_default.ev

```
/*parse error*//*PARSE ERROR: Line 121  -> .3*/
/*PARSE ERROR: Line 121  -> .3*/
/*PARSE ERROR: Line 121  -> output*/
/*PARSE ERROR: Line 121  -> co1*/
/*PARSE ERROR: Line 121  -> 54*/
/************************************************
wid[1] con
************************************************/
/*Converted runset  drac2he: VERSION DATA OMITTED
* Called as : drac2he migration2.drc  */

/*  ;***************  DESCRIPTION BLOCK  ******************/

/*  *DESCRIPTION
   PRIMARY = test2
   SYSTEM = GDS2
   INDISK = test2.gdsii
   OUTDISK = test2_out.gds
   SCALE = 0.001 MICRONS
   RESOLUTION = 0.001 MICRONS
   MODE = EXEC NOW
   PROGRAM-DIR = /apps/cadence/dracula/bin/
   FLAG-SELFTOUCH = YES
   FLAG-ACUTEANGLE = YES
   FLAG-NON45 = YES
   FLAG-OFFGRID = YES  0.01
   FLAG-SELFINTERS = YES
   KEEPDATA = INQUERY
   LISTERROR = YES
   CHECK-MODE = FLAT
   TRANSISTOR-NUM = 20000000
*END  */
HEADER {
   INLIB = test2.gdsii
   OUTLIB = EV_OUT
   BLOCK = test2
   GROUP_DIR = group
   FORMAT = GDSII
   OUTPUT_FORMAT = LTL
   OUTPUT_LAYOUT_PATH = .
}

..................... DATA OMITTED .....................

/*
```

```
    ;*************************************************************
*/

/*  BEGIN INPUT LAYER BLOCK  */
/*  *INPUT-LAYER  */
ASSIGN {
   NWELL   (1)  /* NWELL = 1  */
   PFLD   (2)   /* PFLD = 2  */
   PCOMP   (3)  /* PCOMP = 3  */
   NCOMP   (4)  /* NCOMP = 4  */
   DIFF   (5)   /* DIFF = 5  */
   NP   (6)     /* NP = 6  */
   PP   (7)     /* PP = 7  */
   SAL   (8)    /* SAL = 8  */
   POLY   (11)  /* POLY = 11  */
   CON   (12)   /* CON = 12  */
   GUARD   (26) /* GUARD = 26  */}

/*  DRAC2HE WARNING: SUBSTRATE LAYER BEING TRANSLATED WITH
CELL_EXTENT COMMAND. */
/*  SUBSTRATE = bulk 63  */
CELL_EXTENT {
        CELL_LIST = { * }
}TEMP=BULK

/*  END BLOCK  */
/*  *END  */

..................... DATA OMITTED .....................

/*  BEGIN OPERATION BLOCK  */
/*  *OPERATION  */

/*  NOT BULK NWELL pwell  */
BOOLEAN BULK NOT NWELL { } TEMP=pwell

/*  AND DIFF NP ncomp1   */
BOOLEAN DIFF AND NP { } TEMP=ncomp1

/*  AND DIFF PP pcomp1   */
BOOLEAN DIFF AND PP { } TEMP=pcomp1

/*  OR NCOMP ncomp1 ncomps   */
BOOLEAN NCOMP OR ncomp1 { } TEMP=ncomps

/*  OR PCOMP pcomp1 pcomps   */
BOOLEAN PCOMP OR pcomp1 { } TEMP=pcomps

/*  AND ncomps NWELL ntap   */
BOOLEAN ncomps AND NWELL { } TEMP=ntap

/*  AND pcomps NWELL pdif   */
BOOLEAN pcomps AND NWELL { } TEMP=pdif
```

```
/*  AND pcomps pwell ptap  */
BOOLEAN pcomps AND pwell { } TEMP=ptap

/*  AND ncomps pwell ndif  */
BOOLEAN ncomps AND pwell { } TEMP=ndif

 ..................... DATA OMITTED .....................

/*  AND CON SAL sb08 OUTPUT sb08 54   */
BOOLEAN CON AND SAL {  VERBOSE=TRUE } PERM=sb08_Out (54)

/*  ;-------------------------------------------------------
----; CONTACT RULES
-------------------------------------------------------------
  */

/*  END BLOCK  */
/*  *END  */
DISCONNECT
```

error.out

Example 5-5 shows the error.out file you created in your second translation of the migration2.drc runset. Notice that all of your error and warning messages are now in this file. You should take a minute to view the drc2_errorout.ev runset. Notice that there are no errors or warnings in the Hercules runset file.

Finally, be aware that a detailed explanation accompanies each warning on the SUBSTRATE translation. In general, warnings are just informational messages noting an operation difference between Dracula physical verification and Hercules.

Example 5-5 Drac2He Error Output File with -E Option: error.out

```
/*parse error*//*PARSE ERROR: Line 121  -> .3*/
/*PARSE ERROR: Line 121  -> .3*/
/*PARSE ERROR: Line 121  -> output*/
/*PARSE ERROR: Line 121  -> co1*/
/*PARSE ERROR: Line 121  -> 54*/

/* DRAC2HE WARNING File migration2.drc Line 37:
   SUBSTRATE LAYER BEING TRANSLATED WITH CELL_EXTENT COMMAND.*/
SUBSTRATE = bulk 63
```

If you would like to fix the syntax error and re-translate, execute the following steps:

1. Edit the migration2.drc file.
2. Go to line 121, as the ERROR indicates.

3. Change the first command, wid, to width. Leave the rest of the line the same and save the file. Enter:

```
4.    drac2he -E error.out migration2.drc > drc2_errorout.ev
```

The error.out file should no longer contain any errors.

What's Next?

Now that you are familiar with the Drac2He translator, you can move on. In Chapter 6, "Hercules Migration With Hercules-VUE," you run another Drac2He translation and then run Hercules on the resulting file. The Hercules run generates some design rule violations that you view using Hercules-VUE interfaced to Virtuoso Layout Editor.

6

Hercules Migration With Hercules-VUE

In this chapter you convert a simple Dracula physical verification rule set to a Hercules runset using the translation options you learned in Chapter 5. Then you run Hercules on this runset and view the results using Hercules-VUE interfaced to Virtuoso Layout Editor, a Cadence layout tool.

Summary of Progress to This Point

If you are completing the exercises in the recommended order, you should now be familiar with Hercules and its associated input and output files, know how to execute Hercules from a UNIX command line, and be familiar with the Dracula physical verification to Hercules translator, Drac2He. Now that you have all of the basic information, you can move on to a more realistic example in your design environment.

Learning Objectives for This Chapter

In this part of the tutorial, we try to simulate a real design environment so that you have a better understanding of how to apply Hercules to *your* application. You will:

- Convert a Dracula physical verification rule set to a Hercules runset using Drac2He
- Set up the Hercules and Hercules-VUE menus in Virtuoso Layout Editor

- Run Hercules on the output from Drac2He
- Introduce and run Hercules-VUE connected to Virtuoso Layout Editor to show how the error detection process can be made easier

Generating a Runset With Drac2He

In the tutorial for this chapter, you translate a Dracula physical verification rule set by setting the option on the command line to generate the necessary options for Hercules-VUE and Virtuoso Layout Editor. You also run Hercules and then view the ERROR output with Hercules-VUE in the Virtuoso Layout Editor.

Go to the directory that contains the third migration test, *your_path*/hercules-Examples/Getting_Started_Drac2he_DRC/migration3.

This directory contains the necessary technology files for the Virtuoso Layout Editor environment, as well as two versions of the AD4FUL library:

- GDSII: AD4FUL.gds
- Cadence 4.4.x library: ./lib4

In both cases the top block is AD4FUL. Enter the command:

drac2he -E error.out migration3.drc > drc3.ev

Your screen does not display data while the translator runs. When the translation is complete, drc3.ev contains the Hercules runset ready for execution. Before you run Hercules on this runset, we review its contents.

Translation Results for Migration3 Example

Example 6-1 shows the HEADER and OPTIONS sections you should have in the drc3.ev file that Drac2He output. Notice that:

- INLIB is AD4FUL.gds and the FORMAT is GDSII.
- OUTPUT_FORMAT is LTL.

Example 6-1 drc3.ev HEADER and OPTIONS Sections

```
HEADER {
   INLIB = AD4FUL.gds
   OUTLIB = EV_OUT
   BLOCK = AD4FUL
   GROUP_DIR = group
   FORMAT = GDSII
```

```
    OUTPUT_FORMAT = LTL
    OUTPUT_LAYOUT_PATH = .
}
OPTIONS {
    IGNORE_CASE=TRUE
    DRACULA_DEFAULTS=TRUE
    RESOLUTION=0.001
    PRINT_ERRSUM_FILE=TRUE
    NET_PREFIX = N_
}
```

Edit the runset drc3.ev to make following two changes:

- Search for the HEADER section and change OUTPUT_FORMAT from LTL to OUTPUT_FORMAT = MILKYWAY.
- Search for the OPTIONS section and add CREATE_VUE_OUTPUT = TRUE to generate Hercules-VUE output.

Setting Up Hercules in the Virtuoso Layout Editor Environment

The next section of the tutorial shows you how to set up Hercules-VUE and the Virtuoso Layout Editor interface.

First, enter the command:

setenv XPROBE /tmp/xprobe-$user

This sets the XPROBE environment variable. When Hercules-VUE is started, it creates a file at the XPROBE location. Be sure to set XPROBE to a location where you have read/write privileges. You should choose a location on a local disk drive.

Starting Virtuoso Layout Editor

The icfb command starts Virtuoso Layout Editor, version 4.2.2 or higher. See the *Hercules Reference Manual* for earlier versions.

Enter the command:

icfb &

Loading Synopsys SKILL Code

To use VUE with the Cadence tools, you must first load the SkillVueMenu.il file. It is found here:

"$HERCULES_HOME_DIR/etc/VUE/SkillVueMenu.il"

You can load the SkillVueMenu.il file automatically by executing the following command in the Command Interface Window (CIW).

load(strcat(getShellEnvVar("HERCULES_HOME_DIR") "/etc/VUE/SkillVueMenu.il"))

You can add this statement to the .cdsinit file so that the SkillVueMenu.il file is loaded automatically every time you start the Cadence environment.

A successful setup of the Virtuoso Layout Editor environment with the Synopsys tools results in the following message in your icfb window:

```
************************************
* Welcome to Synopsys Hercules VUE Skill Interface.
* Running under Skill version 'SKILL09.10'.
*
************************************
Loading SkillVueManu.il...
Done.
.
Loading SkillVueFilter.il..
Done.

Synopsys Hercules SKILL environment setup is done
/************************************/
```

At this time start a layout window in Virtuoso Layout Editor, begin your Virtuoso Layout Editor session, and then open the AD4FUL library.

Opening Your Layout

From the Tools menu in the Command Interface Window (CIW), open the Library Manager.

If the library is not in the search path, you have to execute the next three steps:

1. Under Library Manager select Edit > Library Path.
2. Under Library Path Editor select Edit > Add Library to add the library path.
3. Use the browser to locate the AD4FUL library and add it to the path.

Now you should have the AD4FUL library in your path and you can continue.

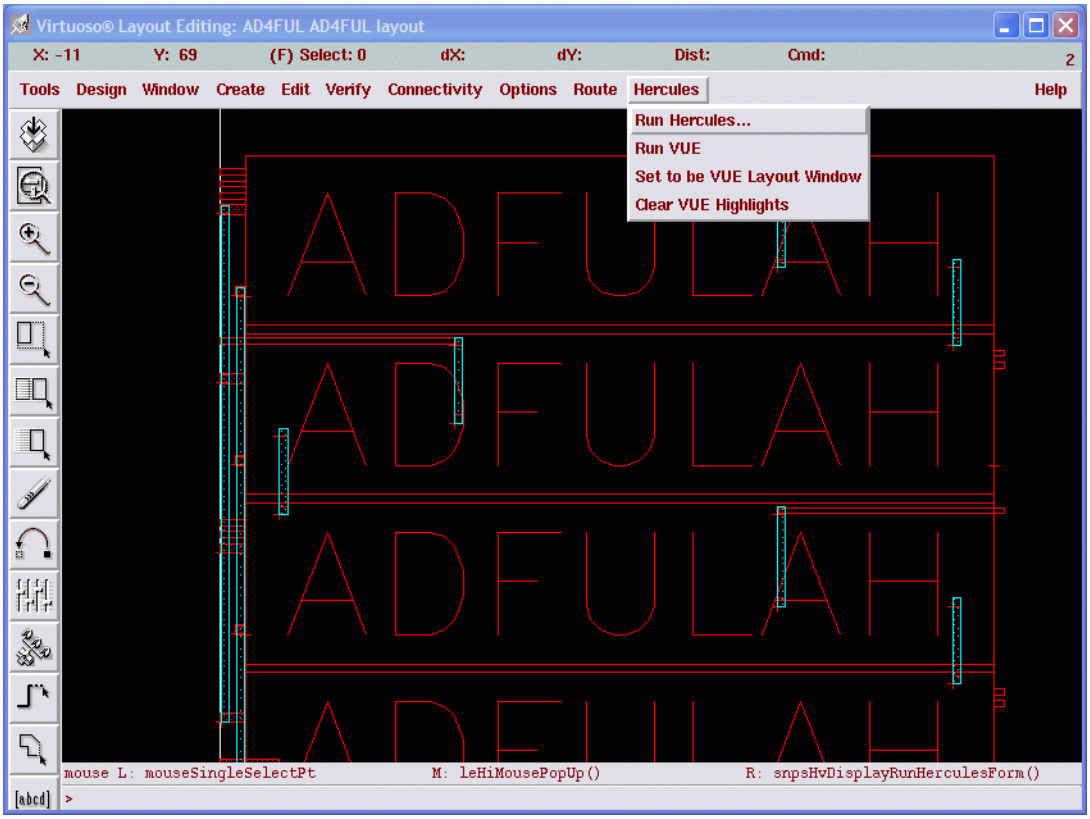
1. Under Library select AD4FUL.
2. Under Cell select AD4FUL.
3. Under View select Layout.

You should have the AD4FUL layout open in Virtuoso Layout Editor. Finally, before you can run Hercules, you need to bring up the Synopsys GUI-based tools.

Running Hercules

Select Hercules > Run Hercules.

Figure 6-1 Hercules Menu

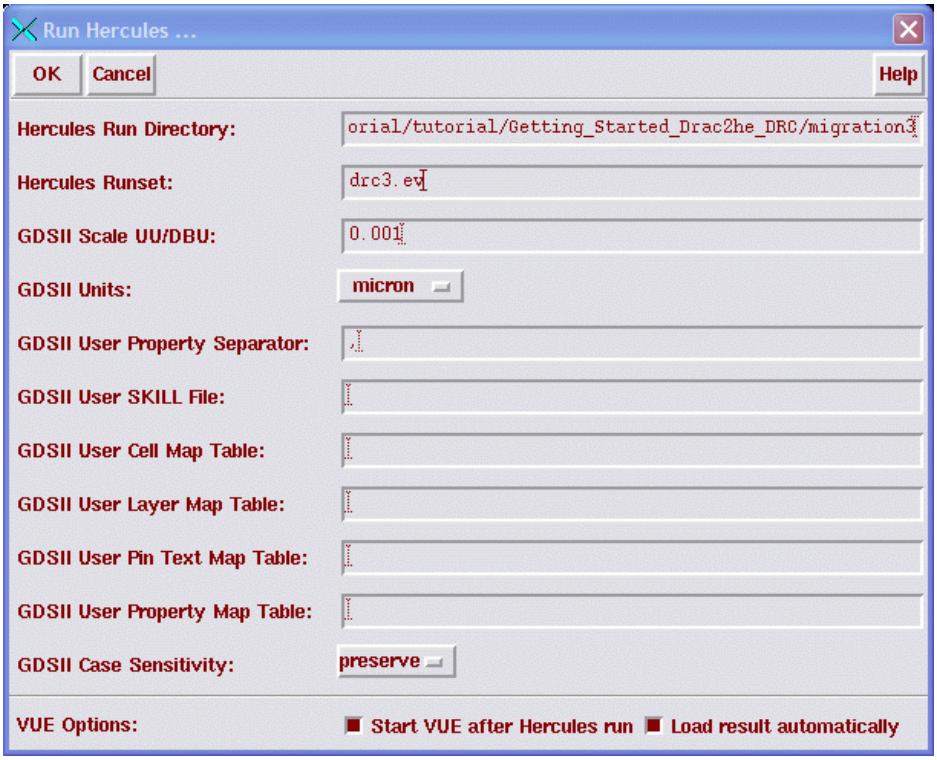


You should be prompted to enter or verify the Hercules run directory. Complete the following steps as shown in Figure 6-2:

1. Verify that the Hercules run directory is *your_path*/hercules-Examples/Getting_Started_Drac2he_DRC/migration3.
2. For Hercules Runset, enter drc3.ev.
3. For GDSII Scale UU/DBU, enter 0.001.
4. Verify that the GDSII unit is set to micron and GDSII case sensitivity is set to preserve.

5. Verify that GDSII User Property Separator is a comma (,)
6. Select OK at the top of the menu.

Figure 6-2 Run Hercules Options



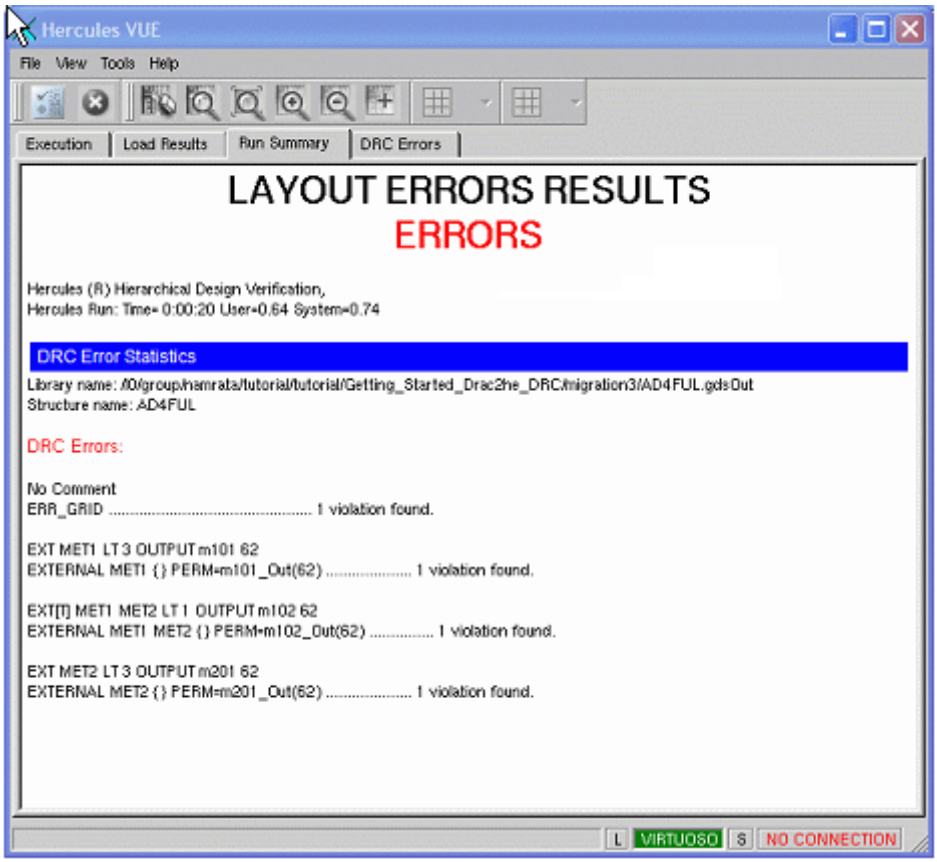
You have now set up your Hercules/Hercules-VUE/Virtuoso Layout Editor environment and executed Hercules on the drc3.ev runset. You will notice that VUE options are set to allow you to start VUE and load results automatically following the Hercules run. If you had any problems or want a more detailed description of the commands available in the SKILL files supplied by Synopsys, see the *Hercules VUE User Guide* for a complete description of the Hercules-VUE/Virtuoso Layout Editor interface.

Opening Hercules-VUE

When the run is complete, Hercules-VUE should be automatically launched and data is loaded if VUE options are selected, as shown in the previous figure. If these options are deselected, select Run VUE from the Hercules menu.

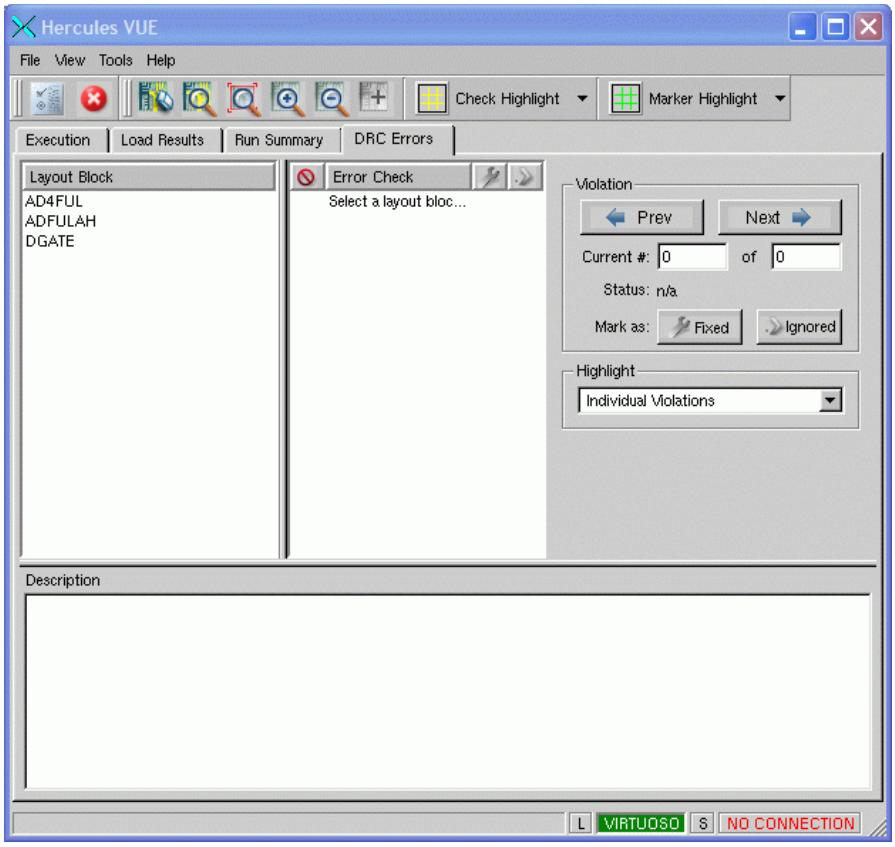
At this time, the Run Summary tab is loaded with overall error statistics.

Figure 6-3 Run Summary Window



Once you examine the run summary, select the DRC Errors tab.

Figure 6-4 DRC Errors Window

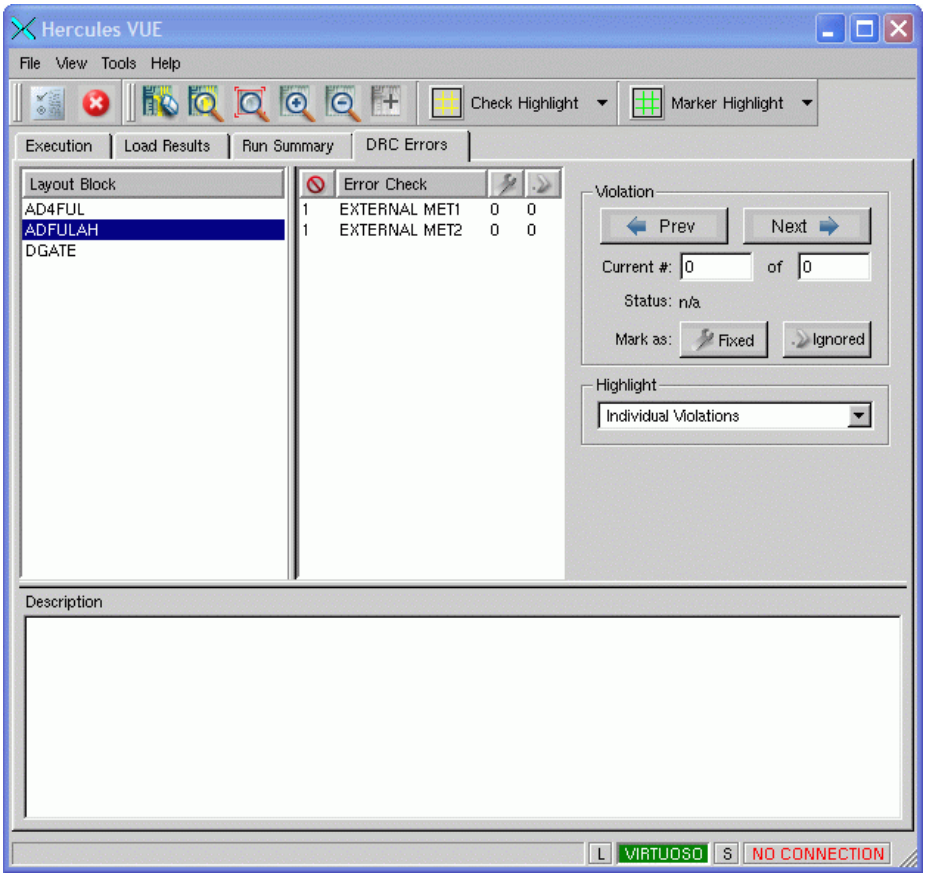


A list of design errors should be loaded in your Hercules-VUE DRC Errors window.

Debugging With Hercules-VUE in the Virtuoso Layout Editor Environment

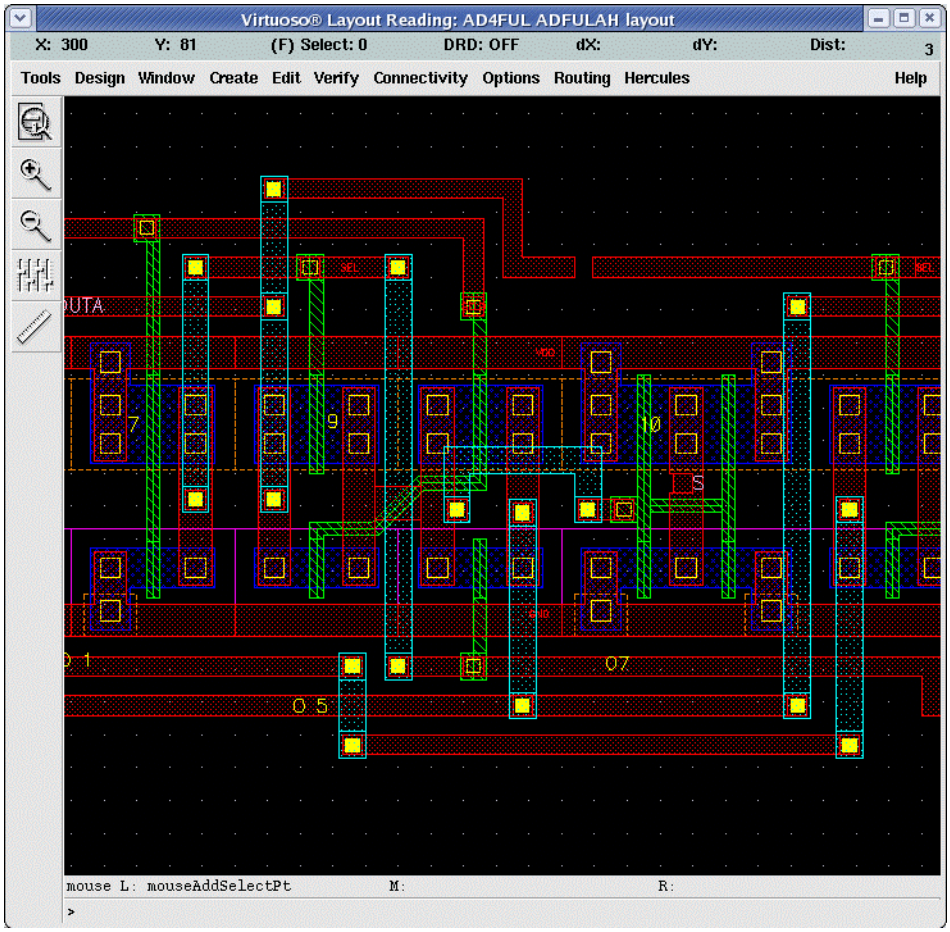
Select the ADFULAH block in the Layout Block Selection pane, as shown in Figure 6-5.

Figure 6-5 Selecting ADFULAH



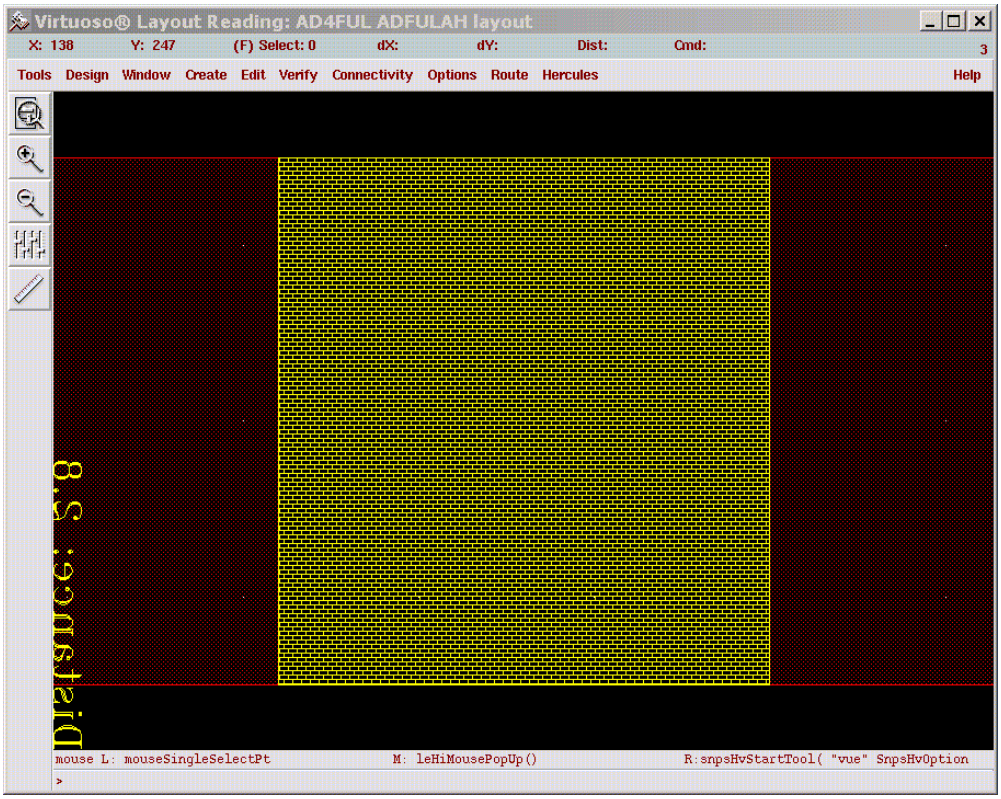
The ADFULAH block is then opened in the Virtuoso Layout Editor editing window, as shown in Figure 6-6.

Figure 6-6 ADFULAH Block Shown in Virtuoso Layout Editor Editing Window



Select the first error in the ADFULAH block, EXTERNAL MET1. The description of the error is displayed in the Description box, while the actual error appears in the Virtuoso Layout Editor editing window (see Figure 6-7).

Figure 6-7 External Met1 Error in Virtuoso Layout Editor Editing Window



To view the next error, EXTERNAL MET2, select it by clicking on the next button in the top right-hand corner of the Hercules-VUE DRC window.

After you have fixed all or some of the errors, you can again select Run Hercules under the Hercules menu to rerun Hercules and verify edits.

What's Next?

You have now completed a Dracula physical verification to Hercules translation, job execution, and debug example. If you plan to write Hercules runsets in the future, or if you want more details on Hercules commands, go back and complete the tutorial in Chapter 4. If you have already completed Chapter 4, "A Complete Design for DRC," or if you are not interested in this aspect of Hercules DRC, continue on to Chapter 7 for an introduction to Hercules LVS.

7

Introduction to Hercules HLVS

In this chapter we discuss the basic principles of Hercules Hierarchical Layout vs. Schematic checking (LVS). We also review the tutorial design used in this section of the document. You learn about the components of a Hercules LVS job and the contents of runset and output files created for an error-free LVS job.

Learning Objectives for This Chapter

- To learn the components of hierarchical LVS.
- To understand the principles of hierarchical LVS.
- To learn the benefits of hierarchical LVS and how to take advantage of these benefits.
- To learn the difficulties presented by hierarchy and how to avoid these difficulties in your designs.
- To run Hercules on an error-free design and examine file output, establishing a reference for clean design files. (This step also verifies that your Hercules software installation and licensing are correct.)

Before You Start

Before you start this tutorial, make sure that you have completely gone through the installation and setup procedures described in Chapter 1, “Installation and Setup,” or that your setup files are in order. See Creating Directories and Getting the Files, in Chapter 1, for directory structure and necessary files. You should also, at the very least, have completed the tutorials in Chapters 2 and 3.

Learning Method: the Design Example

To learn Hercules LVS you use a simple 32-bit microprocessor design that includes an 8Kx32 shared Instruction and Data cache. The complete design is 1.6 million transistors. The technology is a 2-level metal, single poly 1 μm CMOS design. It is a single N well, P+ substrate with digitized N+ and P+ diffusion layers. The separate diffusion layers avoid the need to include an implant layer, thus simplifying the example. The microprocessor is built hierarchically, making it well-suited to take advantage of the Hercules checking capability. This example is used throughout Chapters 7–10. As we go through the LVS tutorial, we introduce more detail about the microprocessor as it relates to running LVS.

As in the HDRC section of this tutorial, the design is configured so that you can use independent runsets on variations of the chip. Each variation is in its own directory to avoid any possible confusion. Because your design copy is in your own account, you are free to experiment with the files. Examine the graphical files not specifically covered in this tutorial, and make changes to see what effect they have on the output. In fact, because this tutorial cannot realistically cover every possible aspect of the design or the design structures, at several points along the way we encourage you to explore on your own, reinforcing the learning experience with independent experimentation.

What Are LVS and Its Components?

LVS verifies the connectivity of a layout netlist against a schematic netlist. The schematic netlist is generated from a Schematic Design tool. For these tutorials, the schematic netlist is considered a required input file. Hercules extracts a hierarchical netlist from the layout that describes the devices and their net connectivity in the layout database. Generating this layout netlist is the first major component of LVS and our tutorial.

An LVS tool reads in the two netlists, converts them to a network representation, and then determines whether the two networks are identical. This comparison of the two netlists is the second major component of LVS and our tutorial. If the two networks are the same, the LVS result is said to be LVS clean. If not, the tool is designed to diagnose and pinpoint the root of the connectivity differences properly and effectively.

What Is Hierarchical LVS?

Hierarchical LVS consists of two phases involving hierarchy:

1. Extraction of a hierarchical layout netlist
2. Comparison of the hierarchical layout netlist to a hierarchical schematic netlist.

(Throughout this chapter we often refer to the comparison phase as COMPARE.)

Before discussing the specifics of the LVS runset, we need to review hierarchical texting. The next three sections discuss hierarchical device extraction, hierarchical texting, and hierarchical netlist comparison. These concepts are very important to understand before you try to analyze a hierarchical LVS operation.

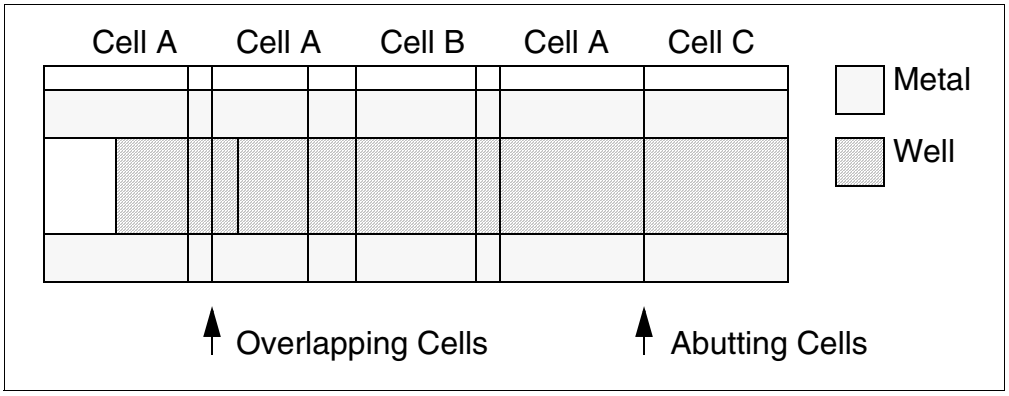
Hierarchical Device Extraction

Hercules performs a hierarchical device extraction. Hercules does not require that all the polygon data or device layers be in a single cell. For example, your poly layer and diffusion layers might be in different cells, but Hercules is still able to recognize their interaction and form the gate, source, and drain layers for a MOSFET device. You should, however, try to lay out or design your devices in such a way that all of the terminal layers are in the same cell, in order to guarantee the best performance from Hercules and the easiest debugging.

Layers That Make Up a Device Should Be in the Same Block

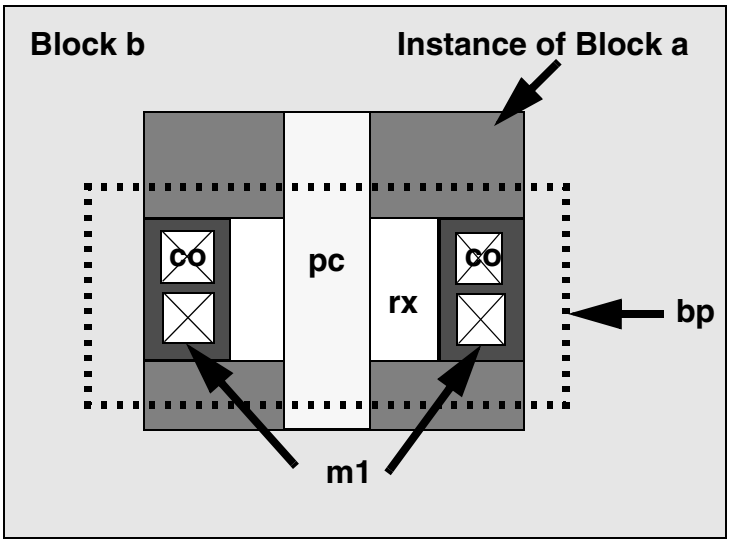
Some interaction of data across the hierarchy is to be expected. For example, power rails and nwells from blocks placed side by side abut each other or slightly overlap and interact during verification. Another normal situation is when metal and via polygons connect blocks together. Figure 7-1 shows an example of these situations. Notice that the polygons either abut or only slightly overlap.

Figure 7-1 Interaction of Data Across the Hierarchy



However, layers in a device should be kept in proper hierarchies. There can be exceptions, such as diffusion-programmable ROMs. Consider Figure 7-2, for example.

Figure 7-2 Structure Crossing Hierarchies in a p-device



Notice that this p-device has an instance of block *a* within block *b*. Block *a*, in turn, has layers *pc*, *rx*, *co*, and *m1*, each layer having one or more geometries. However, *bp* is in block *b*. Because *bp* is not in block *a*, there is some data interaction across the hierarchy. Preferably, *bp* would be a part of block *a*.

Chapter 8, “HLVS Advanced Concepts,” contains details on designing devices for hierarchical extraction. There we examine various situations where designing across the hierarchy is necessary. Chapter 8 outlines how to get the best performance from Hercules,

both in the optimal hierarchical situation, where all of the device layers are contained in a single cell, and in the less-than-optimal situation, where the devices are designed across the hierarchy.

Hierarchical Texting

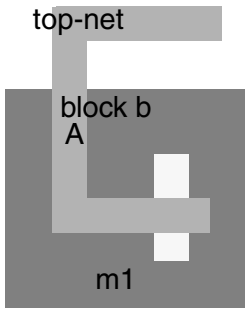
Hercules looks at text hierarchically. Therefore, text placed in lower level cells, or what many refer to as pin text in library cells, can be read by Hercules and attached to the nets in that cell. This is also true for text placed on pins in memory blocks or other blocks (cells) besides the top one. In some cases, text is placed over a layer or net but at a different level of the hierarchy. You must tell Hercules the rules for attaching this type of text by setting the ATTACH_TEXT option in the TEXT_OPTIONS section of the runset. This option is reviewed later in the chapter, when we go through our Hercules LVS runset in detail.

Match Text Appropriately at All Hierarchical Levels

To achieve the best performance and easiest debug from Hercules it is always best to match text appropriately, at all hierarchical levels between the layout and the schematic. However, it is acceptable to have different text at different levels of the hierarchy. Consider the following diagram. Here, we have text A in block b and top-net at the higher level. It is important to match text between schematic and layout at the different levels.

Schematic **Layout**

```
{ block b
  { port A
  { inst m1=n
  {pin A=GATE... }
}
{block topblock
  { inst x1=block b
  { pin top-net=A... }
}
```



Use Different Texting Layers for Different Polygon Layers

In the first example, the ASSIGN section has text placed in one layer, DATA.TEXT. Based on the following TEXT commands, Metal1 is texted first, then Metal2. There is the risk that text intended for Metal2 gets inadvertently associated with Metal1 if the polygons overlap.

In the assign section of your runset, you typically associate a different text layer for each connecting layer.

Example 7-1

```
ASSIGN{
    Metal 1 (8)
    Metal 2 (12)
    DATA(255) TEXT (20)     (unsound practice)

TEXT {Metal1 by DATA.TEXT
      Metal2 by DATA.TEXT}
```

To avoid mistexting, it is preferable to separate out the text as follows:

Example 7-2

```
ASSIGN{
    Metal 1 (8)  TEXT(20)
    Metal 2 (12) TEXT(30)

TEXT {Metal1 by Metal1.TEXT
      Metal2 by Metal2.TEXT}
```

Hierarchical Netlist Comparison

Hierarchical netlist comparison is a bottom-up, divide-and-conquer approach. Hercules uses a list of cell or block names, called the equivalence file, as a guide to decide which cell in the layout it should try to match to a particular cell in the schematic. Example 7-3 shows the form of the equivalence file. See Example 7-8 for an example of an actual equivalence file.

Example 7-3 Form Taken by the Equivalence File

```
EQUIV  A=A { }
EQUIV  B=B { }
EQUIV  C=C { }
EQUIV  I=I { }
EQUIV  J=J { }
EQUIV  F=F { }
EQUIV  G=G { }
EQUIV  K=K { }
EQUIV  L=L { }
```

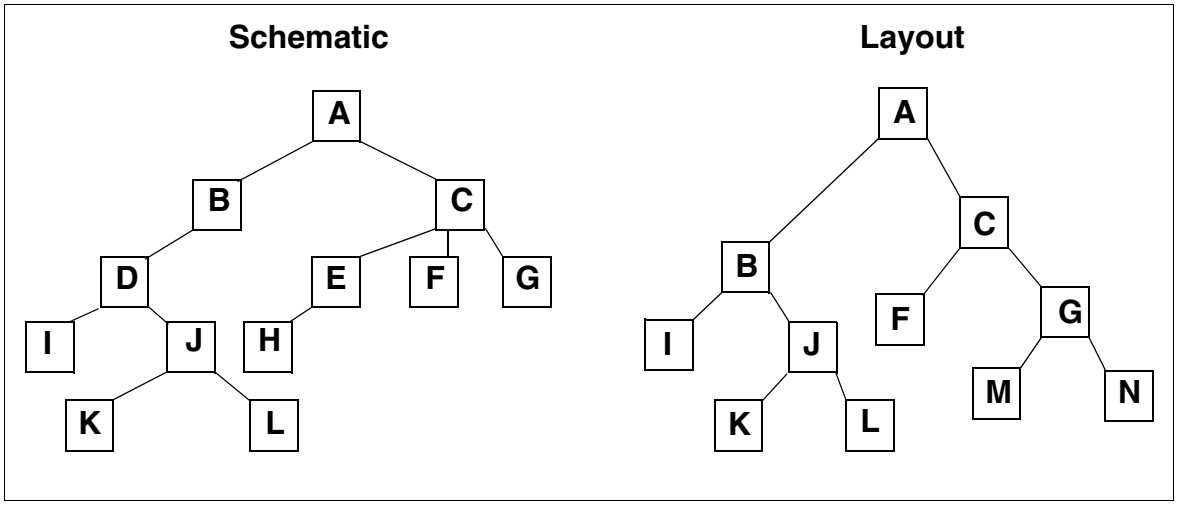
The equivalence file is not a required input file; however, if you do not supply the file, Hercules can be configured to add a preprocessing step to analyze quickly the devices and connections of each cell in the layout and schematic, automatically generating this file. For details on the equivalence file and setting its options and variables, refer to the *Hercules Reference Manual*.

Hercules works its way up the hierarchy, comparing each sub-block listed in the equivalence file. Once a sub-block is compared, Hercules creates a model of the port relationship of the sub-block, to be used at the next level of hierarchy. Once the devices in a sub-block are compared, Hercules never needs to process the information on those devices again. If you

have a very good hierarchical design, when Hercules reaches the top block it probably compares only port relationships between sub-blocks. In other words, instead of comparing five million transistors in the top block, it compares only a few hundred sub-block port relationships.

Figure 7-3 is an example of how hierarchy might look in a schematic and layout netlist. Following Figure 7-3 is list of steps describing how Hercules would process the hierarchy and compare the two netlists. For this example, if the names of two cells are the same, they contain the same logic. Remember, you cannot always assume this fact in all designs.

Figure 7-3 Schematic and Layout Netlist Hierarchies



1. Hercules reads in the schematic and layout netlists and the equivalence file.
2. Hercules determines which sub-blocks in the equivalence file make up the lowest level of hierarchy. In our example, those are I, K, L, F, and G. The lowest level of hierarchy is defined as the bottom of each branch of a tree.
3. A flat representation of the netlists for each of the sub-blocks is generated. In our example, G is the only sub-block that has hierarchy beneath it. Sub-blocks M and N would be flattened into the layout netlist of G before the schematic and layout netlists of G are compared.
4. These flat sub-blocks are compared, and, if they match, a port representation of each is created and saved for use in the next level of hierarchy.
5. Next, Hercules reads in the sub-blocks at the next level. In our example, these are J and C. B is not included in this level because it depends on J, so it must wait until J is compared. B is in the next level of hierarchy.

6. Flat netlists are generated for sub-blocks J and C. In order to do this, port relationships for child cells, generated in step 4, are substituted into the netlists in place of the cell definitions and instances of the child cells.
7. The flat sub-blocks are compared and, if they match, a port representation of each of these, J and C, is created.
8. The same process is repeated for the next level, which contains cell B, and finally the top cell, A.

This example of hierarchy comparison is a very simple one. Many variations of this hierarchy comparison exist. For example, what if cell K in the schematic did not match cell K in the layout? In the following chapters, we go through different flows of hierarchy processing, each of which depends on different ways of telling the Hercules comparison engine to operate. For now, however, our tutorial in this chapter uses a sample flow similar to the one described previously.

Designing to Benefit From Hierarchical LVS

Hierarchical LVS has benefits over flat LVS, during both the extraction and comparison phases. The following is a list of the greatest benefits.

- Shorter runtimes and lower memory usage, because sub-blocks are processed and their information saved, instead of millions of devices being processed at one time.
- Easier debug, because you can debug a mismatched net or device in a sub-block containing hundreds of devices, instead of the top block which can contain millions of devices.
- Smaller netlist generation, because both the layout and schematic netlists are hierarchical.
- In a System-on-Chip (SOC) environment, hierarchical LVS allows you to define different comparison rules for each sub-block, if necessary.
- In memory designs, port swappability can be defined for sub-blocks.

Most of your pre-existing designs benefit from a hierarchical LVS extraction and comparison, but by following some simple design rules for your future designs you can drastically improve your benefits. For example, in the previous discussion on Hierarchical Device Extraction we pointed out that you can improve your runtime and debug ability by designing devices so that a single cell contains all of the device layers. Texting correctly also improves the runtime and debug ability. The next few paragraphs specifically address designing to improve the comparison phase of your LVS job.

Match the Hierarchy Between Schematic and Layout

Hercules runs faster when there are commonalities between the layout and the schematic. This allows Hercules to analyze mismatches between the layout and schematic more efficiently, as shown by Figure 7-4 and Figure 7-5.

Figure 7-4 Recommended Schematic Matches Layout

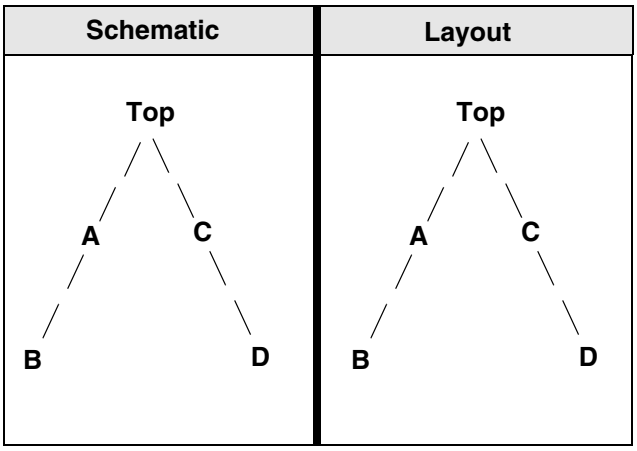
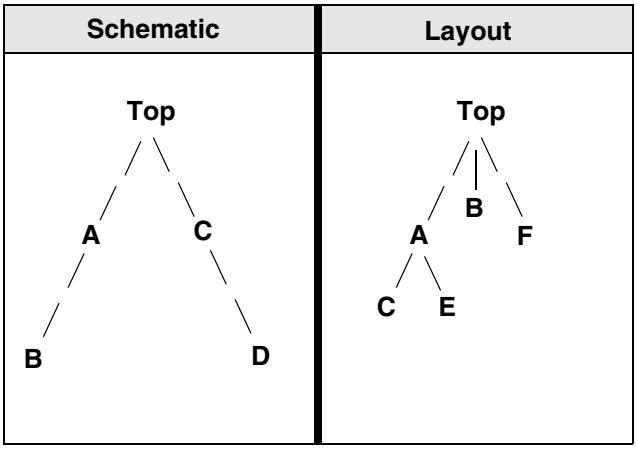


Figure 7-5 Poor Hierarchical Correlation



Notice the major problem with this design: block names, such as the bottom ones, B and D vs. C and E, are not the same, making it difficult to find the levels in the layout that do correspond to those in the schematic.

Defining the Most Beneficial Equivalence Points

It is not the number of equivalence points that determines the best possible LVS runtime and memory usage, but the choice of these equivalence points. For example, if you have 1,000 devices in your design and you create 100 equivalence points, Hercules spends more time partitioning the netlist and creating the port representation for each of these equivalence points than it would just comparing the 1,000 devices flat.

Choose your equivalence points based on easy debug points and natural breaks in the design hierarchy. For example, in a standard cell or ASIC design, the library cells or standard cells are natural equivalence points, because most design flows require the schematic and layout blocks to match at this level.

Macro blocks are also usually required equivalence points in a standard cell design. They should be listed in the equivalence file to improve your runtime, memory usage, and to give you a good intermediate level of hierarchy for debug.

In the case of memory blocks, there should be an equivalence point set up for a register level block (8 or 9 bits), or, if that is not available, a bit cell. Also, if possible, there should be one or two intermediate points, below the top memory block and at the top memory block level.

Match Layout Block Name to Schematic Block Name

Using the same block name in the schematic and layout for logically equivalent blocks simplifies the hierarchical LVS comparison process. For example: for a a given level of hierarchy, both the layout and the schematic have the name buffer, rather than the layout having the name buf_0 and the schematic having the name buffer.

Match Block Port Names for Blocks That Will Be Equivalence Points

Using consistent port names between equivalent schematic and layout blocks also simplifies the hierarchical LVS comparison process. For example: for a block in a given level of hierarchy, both the layout and the schematic have the port names PYI, PYO, YI, and YO, rather than the layout having the port names PY, PYO, Y, and YO, and the schematic having the port names PYI, PYO, YI, and YO.

Difficulties Presented by Hierarchy in LVS

Most difficulties presented by hierarchy in an LVS job are due to the third dimension hierarchy adds to connectivity. This extra dimension involves accounting for cell-to-cell interactions for both the extraction and the comparison phases.

Devices Floating Out of Equivalence Points

The major difficulty hierarchy presents in the extraction stage is the possibility of devices crossing hierarchical boundaries, floating out of the cell where they were originally designed and, in some cases, out of a cell you want to define as an equivalence point. In a flat design, all devices are compared at the top, and therefore all layout devices are netlisted flat. In a hierarchical design, you want to preserve your hierarchical boundaries wherever possible. Thus, if you have device layers that cross cell boundaries and Hercules is forced to netlist the device in a parent cell, you might lose some of your equivalence points. See Guaranteeing Devices Are Netlisted in the Cell Where Designed in Chapter 8, "HLVS Advanced Concepts," for details on preserving equivalence points during hierarchical device extraction.

Port Swappability

The major difficulty hierarchy presents in the comparison stage is determining port swappability of sub-blocks. There are two types of swappable ports:

- Independently swappable ports are logically equivalent ports that can be interchanged without affecting their function within the block. For example, any inputs of an NAND gate can be swapped without changing the function of the gate.
- Dependently swappable ports rely on their functional relationship within the block and, therefore, cannot be separated from one another.

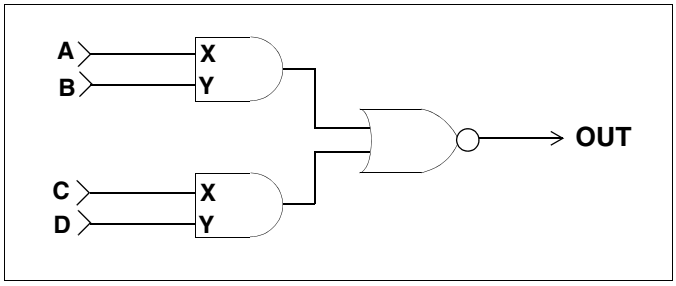
We go through a simple explanation of each in this chapter.

Detecting Swappable Ports

The LVS COMPARE option offers an advanced algorithm for detecting and handling the comparison of swappable or permutable ports. Placing the DETECT_PERMUTABLE_PORTS command in the COMPARE section of the runset allows you to have swappability rules applied and extracted automatically.

In the AND-OR-INVERT cell of Figure 7-6, A cannot be swapped with C unless B is swapped with D, thus demonstrating dependent swappability. In contrast, the independently swappable ports, A and B, are interchangeable, as are C and D.

Figure 7-6 AND-OR-INVERT



Hercules has algorithms built into the code to determine the most complex swappability cases, but, in some flows, the designers might want to restrict what blocks are allowed to have swappable ports. You should set DETECT_PERMUTABLE_PORTS to TRUE in the COMPARE options section, and then set the same option to FALSE in the equivalence file for each block whose swapping you want to restrict.

Equivalence Points That are Resolved in Their Parent

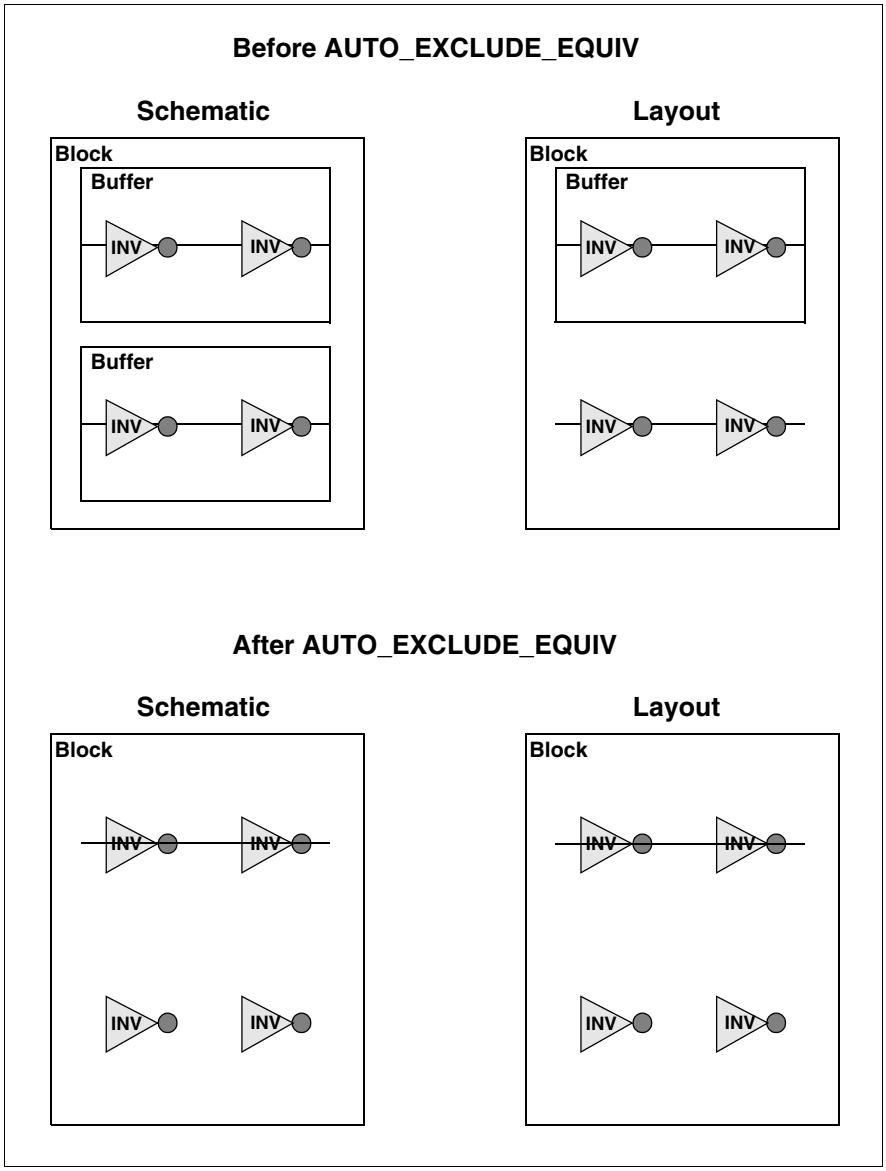
In some cases, especially those created by devices floating out of a cell or by swappability issues, a particular cell might not match between the layout and schematic. However, by simply removing the equivalence point the parent cells will compare. For example, a PMOS device is netlisted in the AND-OR-INVERT functional macro instead of in the AND cell in which it was designed. The AND cells in the layout schematic do not compare, but the AND-OR-INVERT cells do compare. You can turn on EXPLODE_ON_ERROR to tell Hercules simply to explode all blocks that do not compare to their parent, and then compare the parent. You will have an error in your results for the sub-block, but as long as the parent compares, your design is considered clean. If you want to have tight control over which blocks compare, turn this option off, and either require that the layout be redesigned (which might be a bit extreme) or that the equivalence point be removed so that no errors are reported.

Different Number of Instances of a Cell in the Layout and Schematic

One other difficulty presented by hierarchy in the comparison phase occurs in designs where parent blocks contain different numbers of child blocks between the layout and the schematic. For example, the schematic has an array of 100 buffers, but due to space requirements in the layout, only 98 of the buffers are arrayed as cells. The PMOS and NMOS devices, making up the remaining buffer logic, are laid out in the parent as individual devices. Hercules has a COMPARE option, AUTO_EXCLUDE_EQUIV, that is TRUE by default. The child equivalence points would be exploded in this case, because a different

number of instances occurs in the schematic than in the layout. If you set this option to FALSE, Hercules keeps the buffer as an equivalence point and explodes the two extra instances that did not match in the schematic. For an example see Figure 7-7.

Figure 7-7 AUTO_EXCLUDE_EQUIV Options



Overview of Required and Optional Input Files for LVS

Hercules LVS requires two input files, a runset file and a schematic netlist. Hercules LVS also offers two optional input files to help improve the performance of the Hercules LVS run: an equivalence file and an edtext file. Before we execute our first Hercules LVS job, we review the format and content of these four files.

The Runset File

Runset files are ASCII (text) files containing instructions that tell Hercules where to get its files and how to run. Each runset file has sections with variables and commands defining the functions Hercules performs. A Hercules LVS runset can be divided into two parts:

1. The commands for hierarchical netlist extraction.
2. The commands for a hierarchical comparison.

The following describe each section and the variables and commands in a Hercules LVS runset.

Example of an Actual Runset File

Files with the .ev extension are runset files. Example 7-4 shows our first runset example, dac96lvs1.ev. We go over each line so that you understand the basics of runset file creation.

Study this example of an actual runset file, noting the various sections whose labels appear to the right of the series of dashes (----). The short titles are followed by a space and an open brace ({) and appear in the far left margin. They are emphasized for your convenience.

Following Example 7-4 there is a discussion of each of these sections, including a general description of each section and details of the settings for each of them.

Example 7-4 The dac96lvs1.ev Runset File

```
/* HLVS SAMPLE RUNSET */

/* ------------------------- Runset Header information */
HEADER {
   LAYOUT_PATH= ./
   INLIB = dac96
   BLOCK = DAC96
   OUTLIB = dac96_out
   FORMAT = MILKYWAY
   GROUP_DIR = group
   COMPARE_DIR = compare
   EQUIVALENCE = dac96.eqv
   SCHEMATIC = dac96.hrc
```

```
   SCHEMATIC_FORMAT = HERCULES
}

/* ------------------------- Runset options */
OPTIONS {
   CREATE_VUE_OUTPUT = TRUE
   IGNORE_CASE = FALSE
   SCHEMATIC_GLOBAL = {VDD GND VDDIO VSSIO}
   SCHEMATIC_GROUND = {GND VSSIO}
   SCHEMATIC_POWER = {VDD VDDIO}
   LAYOUT_POWER = {VDD VDDIO}
   LAYOUT_GROUND = {GND VSSIO}
   EDTEXT=dac96.text
   NET_PREFIX = XX_
}

/* -------------------------- Data Preprocessing options */
PREPROCESS_OPTIONS {
   CHECK_PATH_90 = false
}

/* -------------------------- Text Processing options */
TEXT_OPTIONS {
   ATTACH_TEXT = ALL
}

/* -------------------------- Layer assignments */
ASSIGN {
   NDIFF    (1)
   PDIFF    (2)
   NWELL    (3)
   POLY     (5) TEXT (25)
   CONT     (6)
   M1       (8) TEXT (28)
   V1       (9)
   M2       (10) TEXT (30)
   PAD      (15) TEXT (35)
   DIFF     (1-2)
   RESP     (50)
}

/*=====================================================*/
/*       LVS NETLIST EXTRACTION SECTION OF RUNSET      */
/*=====================================================*/

/* MOSFET extraction layer generation*/
BOOL PDIFF AND NWELL  temp=pdev
BOOL NDIFF NOT NWELL  temp=ndev

BOOL PDIFF NOT NWELL  temp=subtie
BOOL NDIFF AND NWELL  temp=welltie

BOOL POLY AND ndev  temp=ngate
```

```
BOOL POLY AND pdev  temp=pgate

BOOL ndev NOT ngate  temp=nsd
BOOL pdev NOT pgate  temp=psd

BOOL POLY NOT ngate  temp= temp_field
BOOL temp_field NOT pgate  temp= tmp_field_poly
BOOL tmp_field_poly AND RESP temp = rpoly
BOOL tmp_field_poly NOT rpoly temp = field_poly

/* Resistor Extraction layer generation */
SELECT field_poly touching rpoly temp = res_term

/* Diode Extraction layer generation */
SELECT NDIFF interacting POLY  temp = nodiode
BOOL NDIFF NOT nodiode temp = pos_ndiffdio
BOOL pos_ndiffdio NOT NWELL temp = ndiffdio

/* ------------------------------- Diode Device Definition */
DIODE ndio ndiffdio ndiffdio substrate
  {DIODE_TYPE=NP;} temp = ndiode

/* ------------------------------- Resistor Device Definition */
RES rp rpoly res_term res_term
  {EV_RESVAL = 1.0; } temp = pdevice

/* ------------------------------- NMOS Device Definition */
NMOS n ngate nsd nsd SUBSTRATE
  { MOS_CALC_NRS_NRD = true } temp = ndevice

/* ------------------------------- PMOS Device Definition */
PMOS p pgate psd psd NWELL
  { MOS_CALC_NRS_NRD = true } temp = pdevice

/* ----------------------------- Define Connectivity Rules */
CONNECT {
   ngate pgate BY field_poly
   M2 BY PAD
   M1 M2 BY V1
   M1 ndiffdio res_term field_poly nsd psd welltie subtie BY CONT
   NWELL by welltie
   SUBSTRATE by subtie
}

/* -------------------------------Define Text Attachment Rules */
TEXT {
   M1 BY M1
   M2 BY M2
   field_poly BY POLY
   PAD BY PAD
   NWELL BY "VDD"
   SUBSTRATE BY "GND"
}
```

```
/* ----------------------- Layout Netlist Generation */
NETLIST

/* ------------------------- Output Layer assignment */
GRAPHICS{
   vue_layers   (100)
   NDIFF   (1)
   PDIFF   (2)
   NWELL   (3)
   POLY   (5)
   PAD   (15)
   RESP   (50)
   M1   (8)
   M2   (10)
   CON   (6)
   V1   (9)
}

/*=====================================================*/
/*       LVS COMPARISON SECTION OF RUNSET              */
/*=====================================================*/

/* --------- Define Schematic Diode equal to layout diode */
EQUATE DIODE NDIO = ndio A B {
   check_properties = {area}
   rel_tolerance[area] = {5}
}

/* ---------- Define Schematic Resistor equal to layout resistor */
EQUATE RES  RP1 = rp A B { }

/* ---------- Define Schematic N Mosfet equal to layout N mosfet */
EQUATE NMOS N = n G S D VBB {
   check_properties={l,w}
   rel_tolerance[l]={5}
   rel_tolerance[w]={5}
   filter_options={nmos-3 nmos-8}
}

/* ---------- Define Schematic P Mosfet equal to layout P mosfet */
EQUATE PMOS P = p G S D VBB {
   check_properties={l,w}
   rel_tolerance[l]={5}
   rel_tolerance[w]={5}
   filter_options={pmos-3 pmos-8}
}

/* --------- Layout verse Schematic Global Comparison Options */
COMPARE {
   COMPARE_PROPERTIES = TRUE
   PROPERTY_WARNING = FALSE
   EXPLODE_ON_ERROR = TRUE
```

```
    RETAIN_NEW_DATA = TRUE
    STOP_ON_ERROR = FALSE
    FILTER = TRUE
    MERGE_SERIES = FALSE
    MERGE_PARALLEL = TRUE
    MERGE_PATHS = FALSE
    EQUATE_BY_NET_NAME = TRUE
    AUTO_EXCLUDE_EQUIV = TRUE
    REMOVE_DANGLING_NETS = TRUE
    PUSH_DOWN_PINS = TRUE
    PUSH_DOWN_DEVICES = TRUE
    DETECT_PERMUTABLE_PORTS = TRUE
}
```

General Runset File Sections

The first group of runset sections described are similar to those described in the Hercules DRC error-free example in Chapter 2, “An Error-Free Design for DRC.” These are found in all types of Hercules runs, with different options set depending on the type of Hercules job.

Runset Header Information

The Runset Header Information section is defined by the keyword HEADER, found in the left margin of the file. HEADER contains variables defining where Hercules can find the layout libraries and other input files, as well as where to write intermediate processing files and output files. A description of LAYOUT_PATH, INLIB, BLOCK, OUTLIB, FORMAT, and GROUP_DIR can be found in Chapter 2, “An Error-Free Design for DRC.” These variables are generally set for all types of Hercules jobs. We describe the remaining variables listed in the HEADER section. These four are unique to Hercules LVS jobs:

COMPARE_DIR

This setting tells Hercules where to place all the COMPARE output files it temporarily or permanently creates. The default value for the COMPARE_DIR variable is run_details/compare/. The COMPARE directory contains all of the intermediate files Hercules uses during the actual layout netlist versus schematic netlist comparison. This directory also contains debug information for each cell that Hercules LVS uses as a comparison point. We describe this in more detail when we examine the output files written to this directory.

EQUIVALENCE

This variable specifies the location and name of the equivalence file. The equivalence file is a comparison input file that lists the structures to be compared. It is generally referred to as the EQUIV file. For example, the top block, DAC96, and sub-blocks, such as AND2, NOR2, and IOBUF, would be listed in this file. Specifying the equivalence file is optional. Hercules automatically produces an equivalence file if none is specified here. Later we discuss the advantages and disadvantages to specifying this file.

We also discuss the COMPARE options later in this chapter. Most options that can be specified in the COMPARE section can also be specified in the EQUIV file for individual blocks. For example, in the COMPARE section you can set the FILTER option to TRUE, and in the EQUIV file you can set FILTER to FALSE for CELL A.

SCHEMATIC

This variable is set to the file name, including the path, of the schematic netlist file. This variable is required to execute a Hercules LVS run. If you do not specify this file, or if Hercules cannot find the file you specify, the Hercules job terminates with an error indicating that the schematic file is not specified or that Hercules cannot open the schematic netlist for reading.

SCHEMATIC_FORMAT

This variable is set to the format type of the input schematic netlist. Hercules supports CDL, Hercules, SPICE, EDIF, EDIF3, VERILOG, and SILOS. This variable is required if the input schematic netlist specified is not in the Hercules format. If this format is set to something other than Hercules, Hercules calls NetTran, the netlist translation utility. See the NetTran section later in this chapter.

General Runset Options

The OPTIONS section allows you to specify global assignments for various Hercules processes. You need to specify an option value only when you wish to override the Hercules default. In our example we selected just a few of the many Hercules options you can use to control your LVS run. Refer to Chapter 4 of the *Hercules Reference Manual* for a complete OPTIONS list.

CREATE_VUE_OUTPUT

When CREATE_VUE_OUTPUT is set to TRUE, Hercules generates all of the output data required to run VUE.

IGNORE_CASE

The IGNORE_CASE option applies only to LVS. This includes the way the LVS layout netlist is generated and the case sensitivity of the schematic netlist, equivalence definitions, and equate definitions. In our example we use the default of FALSE so that all cases are preserved and considered when writing and comparing the layout and schematic netlists.

SCHEMATIC_GLOBAL

The SCHEMATIC_GLOBAL option specifies a list of schematic net names that are treated as global during netlist comparison. Typically, you define power and ground nets that are assumed to be globally connected throughout the schematic netlist. These names are used by COMPARE during merging and filtering operations. If no SCHEMATIC_GLOBAL nets are

specified in an LVS runset, Hercules attempts to create them based on global net information in the schematic netlist and typical global net names. Some of the typical names for which Hercules does a string search are:

VDD
VCC
POWER
PWR
VPP
VBB
VEE
VSS
GND
GROUND

SCHEMATIC_GROUND

The SCHEMATIC_GROUND option specifies a list of schematic net names that are considered to be connected to the ground rails in the schematic netlist. These names are used by COMPARE during merging and filtering operations. If no SCHEMATIC_GROUND nets are specified in an LVS runset, Hercules attempts to create them based on global net information in the schematic netlist and typical ground net names (see the list under SCHEMATIC_GLOBALS).

SCHEMATIC_POWER

The SCHEMATIC_POWER option specifies a list of schematic net names that are considered to be connected to the power rails in the schematic netlist. These names are used by COMPARE during merging and filtering operations. If no SCHEMATIC_POWER nets are specified in an LVS runset, Hercules attempts to create them based on global net information in the schematic netlist and typical power net names (see the list under SCHEMATIC_GLOBALS).

LAYOUT_POWER

The LAYOUT_POWER option specifies a list of layout net names that are considered to be connected to the power rails in the layout. These names are used by COMPARE during merging and filtering operations. In our example we have defined our LAYOUT_POWER nets as VDD and VDDIO.

LAYOUT_GROUND

The LAYOUT_GROUND option specifies a list of layout net names that are considered to be connected to the ground rails in the layout. These names are used by COMPARE during merging and filtering operations. In our example we have defined our LAYOUT_GROUND nets as GND and VSSIO.

EDTEXT

The EDTEXT option specifies a path and file name that contains text placement information. The text from this file is placed in the extracted netlist along with any text found in the layout. If there is text in the layout and text in an EDTEXT file located on the same net, the text in the EDTEXT file takes priority over the layout text and is applied to the net. We review the dac96.text file later in this chapter. An EDTEXT file is not required for Hercules LVS.

NET_PREFIX

Hercules generates unique numbers for all nets that are not texted in the layout or in an EDTEXT file. The NET_PREFIX option allows you to specify a string that is added to the front of the numeric net names. This helps you distinguish nets named by Hercules and nets with numeric text. In our example, NET_PREFIX is set to XX_. Another typical NET_PREFIX is NET_.

If a NET_PREFIX is not specified and numeric text is found in the layout, the text is discarded to avoid shorts between the numeric text and a net number Hercules assigns. If a NET_PREFIX is specified and text is found that begins with the NET_PREFIX and contains only numeric text, the text is discarded to avoid shorts between the text and a net number.

Preprocessing Options

The PREPROCESS_OPTIONS section allows you to specify the printing of information to show the Hercules efficiency at processing the design hierarchy, as well as to specify the setting of path grid checking options. You can set options for increased information in the *block*.LAYOUT_ERRORS file and the tree files. You can also set options for path grid checking, to be done when the layers are read during the ASSIGN section. The remainder of the grid checking is done during the GRID_CHECK command, discussed in Chapter 2, “An Error-Free Design for DRC.” You may also want to use the CHECK_PATH_90 option; refer to Chapter 2.

Texting Options

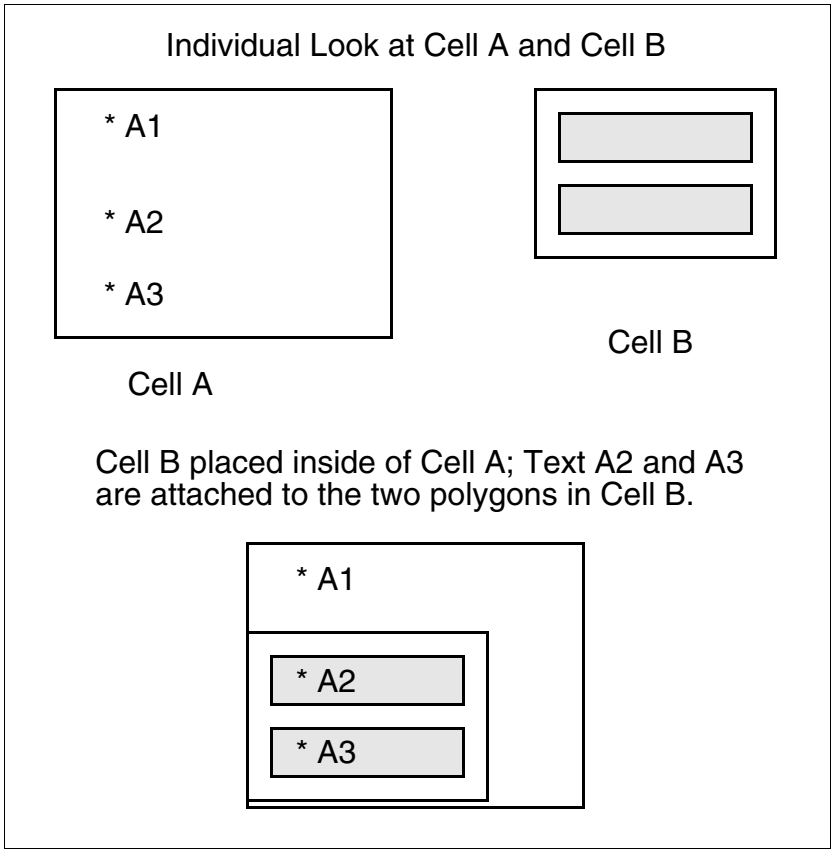
The TEXT_OPTIONS section allows you to define how your text is attached to polygon data, and to replace text strings or text characters, if necessary.

ATTACH_TEXT

The ATTACH_TEXT option defines how the text in the layout is attached to the polygon data across the hierarchy. In our example we define ATTACH_TEXT as ALL. This tells Hercules to create new ports and nets, if necessary, for attaching text. When text from a cell is being applied, if no polygon in the cell interacts with the text origin, this option checks to see if polygons on the lower level cells interact with the text origin. If these polygons exist and do not form a physical port to the cell containing the text, this option creates a port from a

polygon of a lower level cell to the cell containing the text. This new port connects to a newly created net in the cell containing the text. It is named by that text. In Figure 7-8 we give an example of how this might occur in a design.

Figure 7-8 Attaching Text



Layer Assignments

The ASSIGN section assigns symbolic names to the database layers found in the design. Our design uses 11 polygon layers and 4 text layers. All input polygon and text layers must be defined in this section before they are used in the remainder of the runset. In our example, notice that we have merged layers 1 and 2 together and defined a new layer, DIFF. You can read in a single layer and assign it to different names or combine layers, as we have done. We do not actually process the DIFF layer in our example, but have done this just to give an example of syntax.

LVS Netlist Extraction Section

We group the next set of commands together for discussion. They include device layer generation, device definitions, connectivity, texting, layout netlisting, and graphical output. Together, these make up the first phase of the Hercules LVS job, the extraction of a hierarchical layout netlist.

Device Layer Generation

This section of the runset contains the BOOLEAN and SELECT commands used to define the layers that make up the devices which Hercules extracts and netlists. Our example includes MOSFETs, a resistor, and a diode. Other types of devices you might define include bipolar transistors, capacitors, and other generic devices.

Device Definitions

This section of the runset contains the designed device commands for PMOS, NMOS, RES, and DIODE extraction. Hercules extracts and netlists these devices based on the layers defined in the commands and the type of command used. Hercules also supports a CAPACITOR command for capacitor extraction, DEVICE and GENDEV commands for generic device extraction, and an NPN or PNP command for bipolar transistor extraction. A complete definition of each of these commands and their options is in Chapter 5 of the *Hercules Reference Manual*.

In general, each designed device command is followed by a device name, in our example, ndio, rp, n, and p. The next argument is the device layer used to define the body of the device. In the case of the diode and resistor, the device layer should interact with one polygon from each terminal layer. For the MOSFETs, the device layer polygon must edge touch exactly one polygon of each of the terminals, unless the MOS_SINGLE_SD or MOS_MULTITERM_EXTRACT options are set to TRUE. The terminal layers follow the device layer; note that the device layer also serves as the gate terminal layer for the MOSFETs, whereas the device layer is not a terminal layer for the diode or resistor. A final bulk terminal layer may also be optionally specified for each of these devices. Note that in this example, bulk terminal layers are specified for the NMOS and PMOS commands but not for the RES and DIODE commands. Example 7-5 is a general example of syntax for a designed device command. Later in the tutorial we go into more graphical detail on designed device extraction.

Example 7-5

```
NMOS device_name   device_layer  terminal  terminal  bulk
{ options....} temp = output_definition
```

Connectivity Commands and Options

The CONNECT command is used to define electrical connectivity for all layers. Each polygon in a layer that is specified to the left of the BY keyword is connected independently to each polygon with which it interacts in the layer specified to the right of the BY keyword. The default interaction is defined as OVERLAP and TOUCH, but can also be defined exclusively as OVERLAP or TOUCH. For complete details on the CONNECT command, refer to the *Hercules Reference Manual*.

Texting Commands

The TEXT command defines how the text files defined in the ASSIGN section or from text files (for example, EDTEXT) are associated with polygon layers. For example, all text defined for M1 in the ASSIGN section of our example runset should be attached to the M1 layer, and all of the text defined by POLY in the ASSIGN section should be attached to the field_poly layer. You can also text a layer with a string. In our example we text NWELL with the VDD string and SUBSTRATE with the GND string. The coordinates of TEXT placements in the text layer must overlap the polygons in order to be applied.

Layout Netlisting Command

The NETLIST command tells Hercules to generate a layout netlist (*block*.net) that is used by Hercules for the next phase of the LVS job, the comparison of the hierarchical layout netlist to a hierarchical schematic netlist. Up until this point in the netlist extraction phase, all of the device data was stored in an internal database. If you omit the NETLIST command, Hercules is not be able to continue to the next phase of the LVS job.

Graphical Output

The GRAPHICS command contains layer assignments for the special graphic database, which can be created from the extracted database. The database is identical to the internal database from which the netlist was extracted. This database includes any modification caused by TECHNOLOGY_OPTIONS, Boolean commands, or any other commands specified in the runset file. Netlist information is attached to the graphic data in the form of properties. This information includes net number, net text (if any), whether the net is an I/O for the block, instance number, device number, and device type. The NET_PREFIX is added to the net numbers, the instance numbers are prefixed with I, and the device numbers are prefixed with the device name.

The output of the GRAPHICS command is written to the layout database specified by the OUTLIB and OUTPUT_FORMAT options in the HEADER section. Any layer generated in the Hercules runset can be manually specified for output with the GRAPHICS command. Likewise, all layers that participate in CONNECT or device extraction commands may be automatically output by specifying the VUE_LAYERS option inside the GRAPHICS command.

LVS Comparison Section

We group the next set of commands together for discussion. They include the device equate options and compare options. Together they make up the second phase of the Hercules LVS job, the comparison of the hierarchical layout netlist to a hierarchical schematic netlist.

LVS Device Equate Options

The EQUATE command is used to associate schematic primitive devices with their corresponding extracted layout primitive devices. An EQUATE command entry must be included for every non-parasitic extracted device in the design. There are many options associated with the EQUATE command. The ones that we have set in this example are listed here. For a complete list, refer to the *Hercules Reference Manual*. If you do not include EQUATE commands, Hercules attempts to generate the EQUATE commands automatically.

CHECK_PROPERTIES

The CHECK_PROPERTIES option names the properties compared for the equated devices. In our example, we are comparing the l and w properties for the NMOS and PMOS devices, as well as the area property for the DIODE. These properties are compared if the COMPARE_PROPERTIES option is set to TRUE in the COMPARE section of the runset. (We discuss the COMPARE section later.) For a complete list of properties you can compare, see the CHECK_PROPERTIES command in Chapter 5 of the *Hercules Reference Manual*.

REL_TOLERANCE

The REL_TOLERANCE (or TOLERANCE) option allows different relative tolerances to be set for each property. The property name is the same as the properties specified by the CHECK_PROPERTIES option. The value can be either one or two numbers, which are used as percentages. If one value is given, as in our sample runset, it is used as the allowed percentage difference by which the layout property value can vary from the schematic property value. If two values are given, the first is the percentage the layout value can exceed, and the second is the percentage the layout value can be less than the schematic value for the same property. Hercules offers an ABS_TOLERANCE option as well. For more information on tolerances, see the *Hercules Reference Manual*.

FILTER_OPTIONS

The FILTER_OPTIONS control filtering in the schematic and in the layout. There are filtering options available for each device type; they are specified in a table in the *Hercules Reference Manual*, Chapter 5, under the EQUATE command section. In our example, we have set the NMOS-3, PMOS-3, NMOS-8, and PMOS-8 options. NMOS-3 tells Hercules to filter all NMOS devices whose gate pin is connected to ground. PMOS-3 filters all PMOS devices whose gate pin is connected to power. NMOS-8 and PMOS-8 filter all MOSFETs whose gate, source, or drain pin is floating.

LVS Comparison Options

The COMPARE section defines all of the options associated with the actual comparison of the layout and schematic netlists. The following sections describe some of the available options. For a complete list, see the *Hercules Reference Manual*.

COMPARE_PROPERTIES

The COMPARE_PROPERTIES option is TRUE, telling Hercules to check device properties for matching values. A warning is issued when a property is contained in one netlist but not the other. The percent difference allowed between the layout and schematic properties is determined by the REL_TOLERANCE, TOLERANCE, or ABS_TOLERANCE variables in the EQUATE statements, or by the PROPERTY_TOLERANCE variable in the COMPARE options section. Using the PROPERTY_WARNING options in the COMPARE section, you can choose to make mismatched properties a warning or an error. By default, if COMPARE_PROPERTIES is TRUE, an error is generated. If you set PROPERTY_WARNING to TRUE (the default is FALSE), warnings are generated for mismatched properties.

PROPERTY_WARNING

The PROPERTY_WARNING option is set to FALSE. We did not need to include this option, because its default is FALSE, but it is helpful to include it in the runset with the COMPARE_PROPERTIES option. The option reminds you that because COMPARE_PROPERTIES is TRUE and PROPERTY_WARNING is FALSE, all mismatched properties generate an error.

EXPLODE_ON_ERROR

When the EXPLODE_ON_ERROR option is set to TRUE, modules or blocks that have errors are exploded upward into their parent. Remember that we are reviewing the COMPARE options, so the explode is done in the context of the layout and schematic netlists, *not* in the layout extraction. In Chapter 8, “HLVS Advanced Concepts,” we discuss the Hercules process of comparing the individual blocks of the layout and schematic netlists and how Hercules works its way up the hierarchy. This option should be used with caution because, as the blocks with errors are exploded, the netlist becomes flatter and the COMPARE less efficient. For designs with less than 5 million transistors, however, there is probably no need to worry about how flat the design gets. Some users have a design methodology requiring each equivalence point (the blocks you tell Hercules to compare) to match, and the EXPLODE_ON_ERROR option is not allowed in such a flow.

RETAIN_NEW_DATA

As we described in the HEADER section, files resulting from the netlist comparison are written into the COMPARE directory specified by the COMPARE_DIR option. When the RETAIN_NEW_DATA option is set to TRUE, all of these files are retained. We discuss the contents of all of these files later in the chapter. This option is set to FALSE by default to

minimize the amount of disk space used by the Hercules LVS tool. When this option is FALSE, files associated with modules that matched without warnings or errors are deleted after the comparison is complete.

STOP_ON_ERROR

When the STOP_ON_ERROR option is set to TRUE, the comparison process ceases as soon as a block with a compare error is detected. In our example, the STOP_ON_ERROR option is set to FALSE to allow Hercules to continue the netlist comparison even if an error has occurred in a sub-block. This option defaults to TRUE unless EXPLODE_ON_ERROR is TRUE. Because we have already set EXPLODE_ON_ERROR to TRUE, we did not need to specify this option. If you were to set EXPLODE_ON_ERROR to FALSE and STOP_ON_ERROR to FALSE, Hercules would continue its comparison for all levels of the hierarchy, but would not compare modules containing submodules with errors.

FILTER

We have set the FILTER option to TRUE so that devices that are unused in either the layout or schematic are removed before comparison. The types of unused devices filtered out depend on the FILTER_OPTIONS specified in the EQUATE for each device.

MERGE_SERIES

In our example we set the MERGE_SERIES option to FALSE. The majority of design styles, however, require MERGE_SERIES to be TRUE, so we are providing here a complete explanation of how MERGE_SERIES works in Hercules.

Capacitors and resistors in series can be merged into a single device having the same net connections as the terminal pins on the chain.

NMOS and PMOS devices are in a series chain when their drain and source pins are connected consecutively. All devices in the chain must be of the same device type. In addition, the nets connecting the drain/source pins of neighboring devices cannot have any other connections or be ports of the block. A single merged device is created having drain/source pin connections to the terminal drain/source pins of the chain and multiple Gate pin connections corresponding to the number of devices in the chain. The Gate pins on the merged device are automatically designated as interchangeable. As a result, net connections to the Gate pins are allowed to be in any order.

All bulk or substrate pin connections must be tied to the same point in order for devices to be classified as series devices.

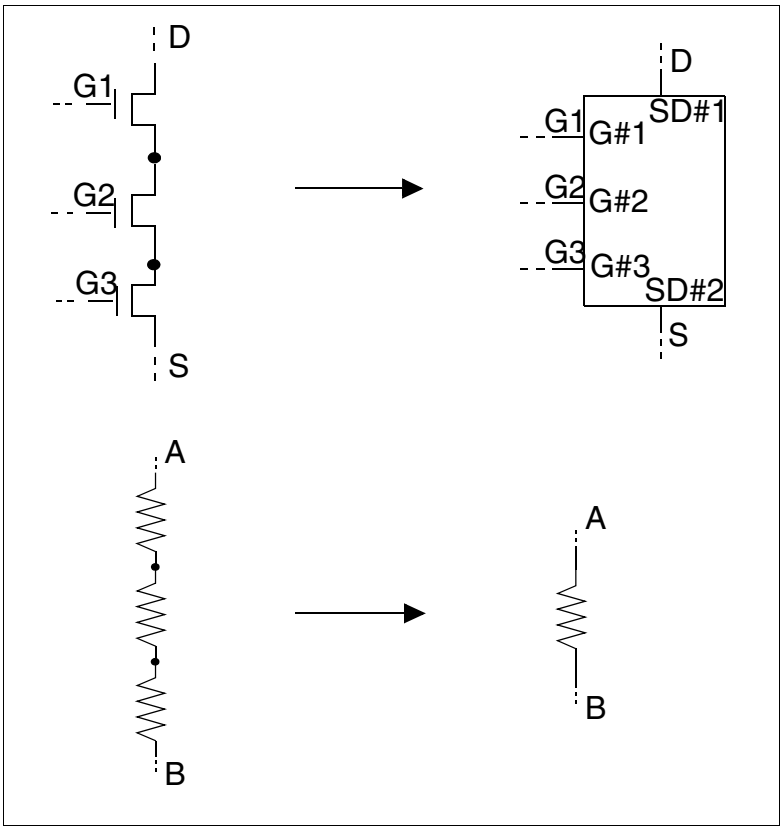
For capacitors and resistors, the merged device type is the same as that of the physical device members. However, a new device type must be created for NMOS and PMOS devices, because the merged device has multiple gate pins. The name for the new device type is the name of the member device type, followed by two dashes (--) and ending with the

number of gate pins in the chain. For example, in a series chain where drain and source pins are connected consecutively, four devices with type name nmos would result in a merged device type of nmos--4.

Newly created series chain devices are assigned instance names of SerChain#1, SerChain#2, and so on.

An example of series merging is shown in Figure 7-9.

Figure 7-9 Series Merging



MERGE_PARALLEL

In our example, we set the MERGE_PARALLEL option to TRUE. Most design methodologies, except for full custom, highly specialized circuits, require MERGE_PARALLEL to be activated. A complete description of how Hercules merges parallel devices follows.

Devices are parallel if every corresponding pin pair between the devices is connected to the same net. Two or more parallel devices are combined into a single merged device with the same device type and net connections.

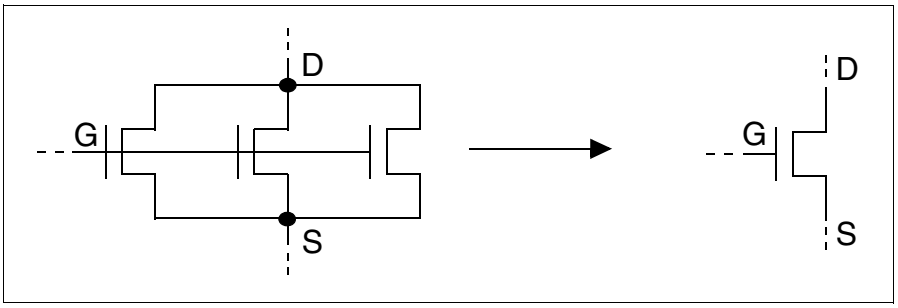
Bulk or substrate pin connections must also correspond in order for devices to be classified as parallel devices.

Parallel merging applies only to devices specified in the EQUATE commands. Higher level blocks that satisfy the criteria for parallel devices are not parallel merged.

Newly created parallel devices are assigned instance names of Parallel#1, Parallel#2, and so on.

An example of parallel merging is shown in Figure 7-10.

Figure 7-10 Parallel Merging



MERGE_PATHS

In our example we set the MERGE_PATHS option to FALSE. Not as commonly used as MERGE_SERIES, MERGE_PATHS is probably the most complicated merging option in Hercules. The following is an explanation of how MERGE_PATHS works in Hercules.

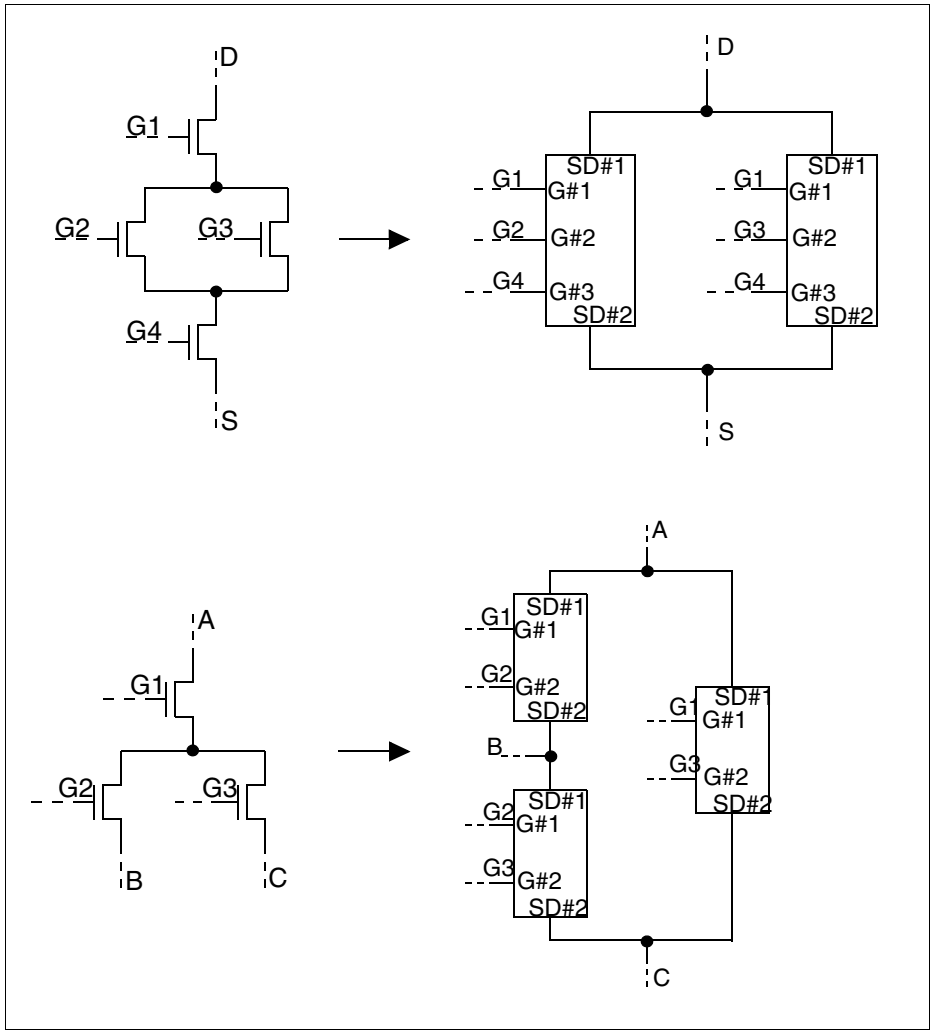
Paths of NMOS or PMOS devices are an extension of series chains. Path merging can be used for more complex structures where groups of devices are stacked. This type of structure is commonly found in AND-OR-INVERT logic. The series components of the stacked structure provide the AND function, while the parallel components result in an OR operation.

While series chains have only two terminating drain/source pins, path structures can have two or more terminating drain/source pins. The stacked structure is expanded into non-duplicate series chains between the terminating drain/source pins. This results in more than one merged device being created for each stacked structure. Be aware that this differs from all other merging operations, which replace multiple physical devices with a single merged device. In addition, a single physical device can be a member of many merged devices created from a path structure.

All bulk or substrate pin connections must be tied to the same point in order for devices to be classified as series devices. Because path merging is an extended version of series chain merging, the naming conventions are the same as described previously.

An example of path merging is shown in Figure 7-11.

Figure 7-11 Path Merging



EQUATE_BY_NET_NAME

When the EQUATE_BY_NET_NAME option is set to TRUE, net names that are the same in the schematic and layout are set equivalent before the comparison begins. However, nets found to have a different number of connections are *not* equated, even though their names match. For designs where texting methodology is known, this can greatly improve the comparison stage of the Hercules LVS run. However, in highly symmetrical designs where equating nets by name is the most beneficial, it can also be the most detrimental. If pins are

swapped so ENA actually equals EN and vice versa, or the circuit has very little uniqueness and the names are incorrect, Hercules might spend unnecessary processing time trying to determine that nets you said are equated are actually not.

AUTO_EXCLUDE_EQUIV

When the AUTO_EXCLUDE_EQUIV option is set to TRUE, Hercules LVS explodes child equivalence cells when comparing their parents if there is a different number of instances between the schematic and the layout. If this option is set to FALSE, the child equivalence cells are maintained, but any extra unmatched instances of the cell in either the layout or schematic netlist are exploded in order to facilitate comparison of the parent block. This option is discussed in the section "Different Number of Instances of a Cell in the Layout and Schematic" on page 7-12 earlier in this chapter. The default value of this option is TRUE.

REMOVE_DANGLING_NETS

The REMOVE_DANGLING_NETS option is set to TRUE in our runset example, so that preprocessing of the input layout netlist finds all cases where a port net in a cell never connects to any extracted device throughout the hierarchy. In our design example we have some gate array logic that assumes unnecessary ports are created when the base layers and programming layers are combined to form a new cell. This is done automatically during the hierarchy preprocessing, as was discussed in Chapter 2, "An Error-Free Design for DRC." Because the same cells are created in a standard cell format, without having the extra ports, we must turn this option on so the port list of the gate array cells and the port list of the standard cells match and both can be equated to a single schematic cell in the schematic netlist.

PUSH_DOWN_PINS

The PUSH_DOWN_PINS option is set to TRUE in our runset example, so both the schematic and layout netlists are preprocessed to determine all cases where two or more pins on a given cell are connected above the hierarchy across all instances of that cell. In such cases, the COMPARE algorithm replaces these pins with a single merged pin as the connection is pushed down into the cell. This resolves cases in which cells are equivalent between the schematic and layout but, for example, two devices are connected inside the cell in the schematic but outside the cell in the layout.

In addition, PUSH_DOWN_PINS propagates power and ground information down the hierarchy to any net that ultimately connects to power and ground across all instances. This power and ground information is particularly useful for merging and filtering operations. Power and ground nets are defined by the LAYOUT_POWER, LAYOUT_GROUND, SCHEMATIC_POWER, and SCHEMATIC_GROUND options in the OPTIONS section of the runset.

PUSH_DOWN_DEVICES

The PUSH_DOWN_DEVICES option is set to TRUE in our runset example, so that both the schematic and layout netlists are preprocessed to determine all cases where devices existing up in the hierarchy of the netlists can be moved down legitimately. This helps preserve lower level equivalence points, particularly in cases where the hierarchical extraction moved devices up the hierarchy. For example, in the layout most of the source/drain region of a MOSFET is in cell B and all of the gate is in cell B. However, there is a portion of the source/drain region that is also in cell A, the parent of B.

When the Boolean operations are performed to generate the MOSFET layers in the layout extraction, and there are instance specific interactions across the hierarchy, the device is generated in cell A (the parent), even though it was intended to be placed in cell B. An instance-specific interaction would be a case where one placement of cell A interacts with cell B differently than the other placements.

A device must have all terminals directly connected to ports on a cell, and this connectivity must be consistent across all instances of that cell. In such a case, those devices are removed from the higher level and placed again inside the original cell.

The default for this option is FALSE. In general, for designs that were not designed with hierarchical verification in mind (in other words, they do not follow the guidelines of good hierarchical design given in Chapter 8), you should set `PUSH_DOWN_DEVICES = TRUE`.

DETECT_PERMUTABLE_PORTS

The DETECT_PERMUTABLE_PORTS option automatically extracts and applies port swappability rules. The default is FALSE, but you should set this option to TRUE to avoid miscompares due to swappability issues. It handles dependent or independent swappability without additional input. The option can be set in the COMPARE section, as we have done here, or in the EQUIV file for individual cells. The DETECT_PERMUTABLE_PORTS option extracts the symmetries for each block, and then the information is applied when determining equivalence for the parent cell of the block. Therefore, when DETECT_PERMUTABLE_PORTS is specified globally, it does not operate on the top-level block. To extract symmetries for the top-level block, you must explicitly invoke DETECT_PERMUTABLE_PORTS in the entry for the top-level block in the EQUIV file.

The Schematic Netlist File

The Schematic Netlist file is a required input file when executing a Hercules LVS job. You must provide a properly formatted schematic netlist so that Hercules has something to compare with the layout netlist it generates in the first phase of the Hercules LVS job. As discussed previously, Hercules supports all of the major formats of schematic netlists. The Hercules executable, however, reads only a Hercules-formatted schematic netlist, so all other formats must be converted to the Hercules format. This conversion can be done

manually by the user or automatically as part of the Hercules LVS job. An example of this format is shown in Example 7-6. The conversion to the Hercules format is done by the utility NetTran (executed as nettran). To avoid too much confusion, our first example starts with a Hercules-formatted netlist. The following is a brief overview of NetTran and its functionality. For more details see the *Hercules Reference Manual*.

NetTran

NetTran is a netlist translation utility. There are two ways to execute NetTran.

Executing NetTran as a UNIX Shell Command

The first way is by executing the nettran command on a UNIX shell command line. The command-line options are:

```
nettran [-V] [-Version] [-build] [-usage] [-cdl file ...]
   [-edif file ...] [-edif3 file ...] [-hercules file ...]
   [-silos file ...] [-sp file ...] [-verilog file ...]
   [-cdl-a] [-cdl-chop] [-cdl-chopDev]
   [-cdl-fopen model_name ...] [-cdl-fshort model_name ...]
   [-cdl-M file] [-cdl-p] [-cdl-P] [-cdl-R] [-cdl-renameDev]
   [-cdl-s] [-cdl-U] [-cdl-x] [-edif-f]
   [-edif-globalNets netName ...] [-edif-p] [-edif-pa]
   [-edif-U] [-sp-fopen model_name ...]
   [-sp-fshort model_name ...] [-sp-finst inst_name ...]
   [-sp-fp delimiter] [-sp-od pattern] [-sp-m number]
   [-sp-S] [-sp-topCell cell] [-sp-topCellPorts portName]
   [-sp-U] [-sp-voltThresh number] [-sp-x]
   [-verilog-b0 netName] [-verilog-b1 netName]
   [-verilog-voltmap filename] [-verilog-R] [-verilog-U]
   [-verilog-busLSB] [-acct] [-acctFile file] [-dup]
   [-equiv file] [-forceGlobalsOn]
   [-herc-globalNets netName ...] [-hier_delimiter pattern]
   [-logFile file] [-mprop] [-msgError NTR-# ...] [-msgId]
   [-msgSuppress model_name ...] [-msgTable]
   [-namePrefix prefix] [-noflatten] [-rootCell cell]
   [-slash] [-undef] [-wireLog file]
   -outName file
```

Argument	Description
`-V`	Print product version.
`-Version`	Print product and library version.
`-build`	Print build information.
`-usage`	Print NetTran usage.

Argument	Description
`-cdl` *file ...*	Input CDL format netlist(s).
`-edif` *file ...*	Input EDIF 2 0 0 format netlists.
`-edif3` *file ...*	Input EDIF 3 0 0 format netlists.
`-hercules` *file ...*	Input Hercules format netlists.
`-silos` *file ...*	Input SILOS format netlists.
`-sp` *file ...*	Input Spice format netlists.
`-verilog` *file ...*	Input Verilog format netlists.
`-cdl-a`	Preserve all passive devices (resistors, capacitors, diodes).
`-cdl-chop`	Remove first character of all instance names.
`-cdl-chopDev`	Remove first character of all device names.
`-cdl-fopen` *model_name...*	Filter-out devices with specific model-name.
`-cdl-fshort` *model_name ...*	Filter-out R&C devices with specific model-name, the two pins will be shorted.
`-cdl-M` *file*	CDL map file.
`-cdl-p`	Translate instance parameter to prop if the model has no instances.
`-cdl-P`	Multiply scale_factor by param_global.
`-cdl-R`	Include unused parameters in the Hercules netlist.
`-cdl-renameDev`	Prefix device names with type to resolve identical name conflict.
`-cdl-s`	Don't treat slash as delimiter.
`-cdl-U`	Convert cdl input to uppercase.
`-cdl-x`	Trim prefix x of instance name.
`-edif-f`	Force EDIF primitives in Hercules netlist.

Argument	Description
`-edif-globalNets` *netName* ...	List of EDIF global nets.
`-edif-p`	Convert EDIF properties.
`-edif-pa`	Convert all EDIF properties.
`-edif-U`	Convert edif input to uppercase.
`-sp-fopen` *model_name* ...	Filter-out devices with specific model-name.
`-sp-fshort` *model_name* ...	Filter-out R&C devices with specific model-name, the two pins will be shorted.
`-sp-finst` *inst_name* ...	To filter out devices with specific inst_name (accept wildcard '*').
`-sp-fp` *delimiter*	Filters out parasitic resistors.
`-sp-od` *pattern*	Define the order of the output ports from Verilog to Spice.
`-sp-m` *number*	Device property multiplication factor.
`-sp-S`	Set '.options scale=1u'
`-sp-topCell` *cell*	Set name of top cell.
`-sp-topCellPorts` *portName*	Add ports to Spice/CDL topcell ports list. Must use option -sp-topCell to specify topcell.
`-sp-U`	Convert spice input to uppercase.
`-sp-voltThresh` *number*	Voltage source threshold value for shorts.
`-sp-x`	Trim prefix x of instance name.
`-verilog-b0` *netName*	Verilog global ground net.
`-verilog-b1` *netName*	Verilog global power net.
`-verilog-voltmap` *filename*	Verilog global net mapping file.
`-verilog-R`	Retain backlashes on verilog net names.
`-verilog-U`	Convert verilog input to uppercase.

Argument	Description
`-verilog-busLSB`	Verilog bus starts with least significant bit (0) first.
`-acct`	Write accounting file nettran.acct.
`-acctFile` *`file`*	Write accounting file *file*.
`-dup`	Choose the nearest one of duplicate cells.
`-equiv` *`file`*	Output skeletal equivalence filename.
`-forceGlobalsOn`	Forced global ports connect to global nets.
`-herc-globalNets` *`netName ...`*	List of Hercules global nets.
`-logFile` *`file`*	NetTran summary log filename. (default is nettran.log)
`-hier_delimiter` *`pattern`*	User-defined hierarchical delimiter to separate cell instances in an instance path when the -mprop option is used.
`-mprop`	Expand M property in Hercules netlist.
`-msgError` *`NTR-# ...`*	List of messages to upgrade to error.
`-msgId`	Include message ID in message.
`-msgSuppress` *`NTR-# ...`*	List of messages to suppress reporting.
`-msgTable`	Print message table and exit.
`-namePrefix` *`prefix`*	Prefix for names used in output netlist (default is NT_).
`-noflatten`	Do not flatten cell to instantiate parameters
`-rootCell` *`cell`*	Set root cell for netlist output.
`-slash`	Use slash (/) as the hierarchical delimiter.
`-undef`	Write undefined cells to the file undef.log.
`-wireLog` *`file`*	Dissolved Nets log filename.
`-outName` *`file`*	Output netlist name.

Executing NetTran by Specifying a SCHEMATIC_FORMAT

The second way to execute NetTran is by specifying a SCHEMATIC_FORMAT other than Hercules in the Hercules runset. You then use the NETTRAN_OPTIONS variable (under the OPTIONS section in the Hercules runset) to specify the NetTran options necessary to translate your netlist.

An example of a Hercules-formatted schematic netlist as translated by NetTran is shown in Example 7-6.

Example 7-6 Hercules-Formatted Schematic Netlist

```
{NETLIST DATBIDIR
{VERSION 1 0 0}

{PROP_DEFAULT}

{CELL INVA
  {PORT IN OUT}
  {INST $1I1=N
    {PROP L=1}{PROP W=13}
    {PIN OUT=D IN=G GND=S GND=VBB}}
  {INST $1I2=P
    {PROP L=1}{PROP W=20.5}
    {PIN VDD=D IN=G OUT=S VDD=VBB}}
}

{CELL DATBIDIR
  {PORT DIN DIO DOUT EN}
  {INST $1I10=INVA
    {PIN DIN=IN $1N15=OUT}}
  {INST $1I11=INVA
    {PIN DIO=IN $1N34=OUT}}
  {INST $1I3=N
    {PROP L=1}{PROP W=13}
    {PIN DIO=D $1N23=G GND=S GND=VBB}}
  {INST $1I4=P
    {PROP L=1}{PROP W=20.5}
    {PIN VDD=D $1N19=G DIO=S VDD=VBB}}
  {INST $1I49=INVA
    {PIN $1N15=IN $1N17=OUT}}
  {INST $1I5=NAND2
    {PIN $1N19=QN $1N17=A $1N66=B}}
  {INST $1I50=INVA
    {PIN $1N74=IN $1N66=OUT}}
  {INST $1I51=INVA
    {PIN $1N34=IN DOUT=OUT}}
  {INST $1I53=INVA
    {PIN $1N34=IN DOUT=OUT}}
  {INST $1I55=NOR2
    {PIN $1N23=QN $1N74=A $1N17=B}}
  {INST $1I61=INVA
    {PIN EN=IN $1N74=OUT}}
```

```
}

{CELL NAND2
  {PORT QN A B}
  {INST $1I1=N
    {PROP L=1}{PROP W=13}
    {PIN $1N20=D A=G GND=S GND=VBB}}
  {INST $1I25=N
    {PROP L=1}{PROP W=13}
    {PIN QN=D B=G $1N20=S GND=VBB}}
  {INST $1I3=P
    {PROP L=1}{PROP W=20.5}
    {PIN VDD=D B=G QN=S VDD=VBB}}
  {INST $1I4=P
    {PROP L=1}{PROP W=20.5}
    {PIN VDD=D A=G QN=S VDD=VBB}}
}

{CELL NOR2
  {PORT QN A B}
  {INST $1I1=N
    {PROP L=1}{PROP W=13}
    {PIN QN=D A=G GND=S GND=VBB}}
  {INST $1I2=N
    {PROP L=1}{PROP W=13}
    {PIN QN=D B=G GND=S GND=VBB}}
  {INST $1I3=P
    {PROP L=1}{PROP W=20.5}
    {PIN $1N5=D B=G QN=S VDD=VBB}}
  {INST $1I4=P
    {PROP L=1}{PROP W=20.5}
    {PIN VDD=D A=G $1N5=S VDD=VBB}}
}

}
```

The EDTEXT File

The Edtext file, optional for Hercules LVS, contains a list of text placements to be included in the extracted layout netlist. The text is specified for a specific structure and (x, y) coordinate.

The file is organized by block name and lists a set of text placements for each block. The block name is specified using the STRUCTURE keyword. Beneath the STRUCTURE line, each text placement is listed in a separate line containing five fields: text string, layer number, data type, x-coordinate, and y-coordinate. Comments enclosed by /* */ are allowed in this file, but nested comments are not. Blank lines are allowed.

Example 7-7 EDTEXT File Syntax

```
STRUCTURE lowdec8
```

```
VDD 28 0 2.5 46.5
VDD 28 0 2.5 132.5

STRUCTURE meddec8
VDD 28 0 2.5 -590.5
VDD 28 0 2.5 -418.5
VDD 28 0 2.5 -246.5
VDD 28 0 2.5 -74.5
VDD 28 0 2.5 528.5
VDD 28 0 2.5 356.5
VDD 28 0 2.5 700.5
VDD 28 0 2.5 183.5

.................. DATA OMITTED ..................

STRUCTURE ce64kd
BL0 30 0 2.5 5.5
BLN0 30 0 10.5 5.5
BL128 30 0 1666.5 5.5
BLN128 30 0 1673.5 5.5
WL255 25 0 13 3.5
WL127 25 0 13 2755.5
VDD 28 0 2.5 21.5
VDD 28 0 2.5 2773.5
```

The layer number and data type must be the same as those in a corresponding ASSIGN section TEXT entry. The x- and y-coordinate values should be the absolute local coordinates within the named structure where the text is placed. The path to the Edtext input file must be specified in the runset file OPTIONS section with the EDTEXT option. An EDTEXT text placement command at the exact coordinates of an existing layout text placement overrides the layout text placement.

The Equivalence File

The Equivalence file, optional for Hercules LVS, contains a list of EQUIV or BLACK_BOX command entries that associate schematic block (cell) names and layout block (cell) names. These entries determine which schematic block to compare to which layout block. (BLACK_BOX is discussed later.)

Comments enclosed by /* */ are allowed in this file, but nested comments are not. Blank lines are allowed.

Hercules reads the equivalence file entry during the hierarchical schematic to layout netlist comparison. If there are specific COMPARE options that apply only to specific cells, you can turn these on or off in the equivalence file.

This file is optional. Hercules LVS automatically creates this file if you do not supply it. Example 7-8 shows a sample of the equivalence file syntax.

Example 7-8 Equivalence File Example

```
/* Equivalence File Example for DAC96 */
/* EQUIV schematic_cell = layout_cell
        { options }
*/
EQUIV COREHI=corehi {}
EQUIV CORELOW=corelow {}
EQUIV CS_ADD1=cs_add {}
EQUIV IBUF=IPAD {}
EQUIV ADD4=add4 {}
EQUIV OBUF=OPAD {}
EQUIV IOBUF=IOBUF {}
EQUIV INVA = inva {}
EQUIV NOR2 = nor2 {}

.................. DATA OMITTED ...................

EQUIV DOUT8=dout8 {}
EQUIV RAM64K=ram64k {}
EQUIV RAM128K=ram128k {}
```

Running Hercules LVS

Now that we have explained the contents of the runset file, schematic netlist, edtext file, and equivalence file, you need to run the runset as a check against the design example, dac96.

How to Run Hercules LVS

Be sure that you are in the directory where your dac96lvs1.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs1/. Enter the command:

hercules dac96lvs1.ev

Your active window displays the execution process. While the screen contents might scroll quickly (freeze the screen with Control-s and restart with Control-q), you should be able to notice any specific actions or warnings that appear. All of this information appears in a file for your examination. The sample runset file should take only a few minutes to run.

Run File Results

Now that you have run the dac96lvs1.ev file, take a look in the dac96lvs1 directory and see the files Hercules has added. You find some files that always appear, and others that appear depending on the statements in your runset file.

To get a directory listing similar to the one in Example 7-9, enter the command:

ls -R

Example 7-9 Directory Listing After Running Hercules

```
DAC96.LAYOUT_ERRORS    dac96.eqv      dac96lvs1.ev
DAC96.LVS_ERRORS       dac96.gds      dac96lvs1_gds.ev
DAC96.RESULTS          dac96.hrc      group
DAC96.net              dac96.text
DAC96.vue
compare                dac96_out      run_details
dac96                  dac96_out.tf

compare:
DAC96          buf4x4         mux          or3_ga
Device         buf4x8         mux16        or3b
IOBUF          corehi         mux4         or3c
IPAD          corelow         nand2        pc_256
/

.......... SOME FILES IN THIS DIRECTORY OMITTED ..........

bsel             invc             nor3c            wdec256
bsel8            latr             or2              xfer
bsel_a1          lowdec8          or2_ga
bsel_a2          meddec8          or2b
buf4x            membidir         or2c

compare/DAC96:
lay.DAC96   sch.DAC95    sum.DAC95.DAC96

compare/IOBUF:
lay.IOBUF   sch.IOBUF    sum.IOBUF.IOBUF

compare/IPAD:
lay.IPAD    sch.IBUF     sum.IBUF.IPAD

.......... SUBDIRECTORIES OF compare/ OMITTED ..........

compare/xfer:
lay.xfer    sch.XFER     sum.XFER.xfer

run_details:
DAC96.acct      DAC96.sum      DAC96.tree3   expandDEV.ev
DAC96.bbox      DAC96.tech     DAC96.vcell   lvsflow.
DAC96.blackbox  DAC96.tree0    equiv.run     vue_info
DAC96_lvs.log   DAC96.tree1    evaccess

run_details/evaccess:
DAC96         DAC96.errstr  DAC96.msg     VA.libs
DAC96.errbin  DAC96.ev

run_details/evaccess/DAC96:
#1  #2  #3  cellTOC  viewTOC
```

```
.
run_details/lvsflow:
lay.tree  sch.tree

.................. DATA OMITTED ...................
```

What If Your Output Is Not Correct?

If your directory does not match the one shown in Example 7-9, refer to this list of possible problems and solutions. If you have a problem that is not listed here, contact Synopsys support.

Problem: You get a message saying you do not have permission to write.

Solution: Check your licensing rights.

Problem: You get a message saying that the disk is full.

Solution: Check your disk space. Hercules requires 70 MB of disk space.

Problem: You get an error writing a file, but your disk is not full.

Solution: The UNIX variable Descriptors need to be set to 256 to allow Hercules to read and write more than the default of 64 files. To fix this problem, execute the following command in your UNIX shell:

descriptors limit 256

Overview of Hercules LVS

Hercules LVS runs in two phases, as described earlier in this chapter. For a better idea of how these phases are executed, we first look at the DAC96.RESULTS file. This file details the following information:

- version of Hercules
- summary of the class of checks (for example, DRC or LVS)
- overall runtime for your entire Hercules job

In Example 7-10, notice the two major phases of the Hercules LVS job (device extraction, comparison), as well as the minor phase (netlist file generation).

Example 7-10 DAC96.RESULTS File

```
Hercules (R) Hierarchical Design Verification, DATA OMITTED
  Synopsys.  All rights reserved.

Called as: hercules dac96lvs1.ev

- Parsing runset "dac96lvs1.ev" ... DONE

- Checking input conditions ... DONE

The EV_ENGINE EXECUTABLE runs the Device Extraction phase of LVS

- Performing DRC checks and/or device extraction ... DONE
    No layout errors

The EV_NETLIST executable is run during the netlist file generation

- Outputting Hercules netlist ... DONE

The LSH executable is run during the comparison phase of LVS

- Performing layout vs schematic comparison ... DONE
    No LVS errors

Creating EVaccess data ... DONE

Hercules Run: Time=  0:02:24 User=93.71 System=13.18
Hercules is done.
```

Each step in the Hercules LVS run represents a separate executable, which was called by the Hercules executable. For DRC or device extraction, Hercules calls the ev_engine executable. For netlisting, Hercules calls ev_netlist. For comparing the netlists, Hercules calls lsh. During the Hercules LVS job it is important to be aware of these stages, so that if there is an error or problem with your job you know which error file to look in for the details of the error. Also, each of these stages can be run independently. We go through examples of running the individual stages later in this chapter and in Chapter 8, “HLVS Advanced Concepts.”

In the next two sections of our Tutorial we discuss output files associated with the Hercules LVS device extraction and netlisting phases of the job, and then the Hercules LVS comparison phase of the job. Keep in mind that we review all of these files to make you aware of the information available to you as a user. In the next chapter we show you user interfaces that take you through these files automatically, so you do not have to memorize all of this information.

LVS Device Extraction Output Files

The first group of output files we review are those associated with the device extraction from the layout and generating the layout netlist. The executable running during this phase is ev_engine, the same executable used for Hercules DRC. Because of this, most of the output files are the same as those we explained in Chapter 2. Review that chapter if you have not already done so. We only briefly describe the files that are the same as those in Chapter 2, "An Error-Free Design for DRC." We go into detail for the files that are unique to LVS Device Extraction or have unique entries for LVS Device Extraction or have unique entries for LVS Device Extraction.

DAC96.LAYOUT_ERRORS

The design block that was run generates this error file during the LVS Extraction stage. In this example, we specified DAC96 in the HEADER section of the dac96lvs1 runset file. We have no errors, so the DAC96.LAYOUT_ERRORS file contents are minimal, containing only some repeated runset information (shown in Example 7-11). This file contains all texting errors, including shorts, opens, and unused text. This file also contains any device extraction error messages. For example, if, during the extraction, Hercules could find only the gate layer of a PMOS device, this file would show an error of missing terminals for that device. We review these types of errors in the Chapter 9, "Hercules HLVS Debugging" example.

Example 7-11 DAC96.LAYOUT_ERRORS File (No Errors)

```
                      LAYOUT ERRORS RESULTS: CLEAN

                       #### #     #####   ##   #   #
                      #     #     #      #  #  ##  #
                      #     #     ####  ###### # # #
                      #     #     #     #    # #  ##
                       #### ##### ##### #    # #   #

============================================================

Library name:  dac96
Structure name:  DAC96

                           ERROR SUMMARY

                           ERROR DETAILS
```

DAC96.acct

The design block that was run generates this accounting file, listing all devices successfully extracted during the Hercules job. The *block*.acct file is located in the run_details directory. Notice, in Example 7-12, that the devices are listed hierarchically or under each sub-block where they were found. At the end of the file is a summary of all of the devices found in the design. Each total listed under the sub-blocks has

- a cell level count, totaling the devices extracted from that block
- a flat count, totaling the devices extracted from the block, plus all of the devices extracted in each sub-block placed under the hierarchy of the block.

For example, AND3 might have a placement of NAND3 and INVA, each containing designed devices. The count reported in the AND3 cell level is 0, because there are no devices in AND3. The count reported for the flat count is the combined total of devices in NAND3 and INVA.

The final information in this file is the number of filtered devices. By default, Hercules tries to filter MOS devices during the device extraction phase based on the options in the EQUATE commands. This option is called MOS_FILTER and is set in the PMOS or NMOS commands. The default for this option is TRUE if filtering is specified in the EQUATE and COMPARE sections of the runset. The accounting file lists how many devices were filtered from each block. One of our filtering options was to filter all devices that contain a floating pin. During this device extraction phase, Hercules was able to filter 148 NMOS devices and 4 PMOS devices from the layout before it generated a layout netlist. The main advantage to this step is seen in gate array or ROM style devices. In many cases, thousands of floating devices are extracted from unprogrammed regions. By filtering at this stage of the job, Hercules saves time and disk space by not netlisting them, and also saves time during the comparison stage because it does not have to read in all of these devices and operate on them. For more information on the MOS_FILTER option, see the *Hercules Reference Manual*.

Example 7-12 DAC96.acct File

```
cellpwr
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
cellpwr2
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
PADVIA
```

```
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0

..................... DATA OMITTED .....................
ram64k
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 264881
      PMOS[p] = 133601
ram128k
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 530282
      PMOS[p] = 267722
corelow
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 5   Filtered:  NMOS-8 = 4
      PMOS[p] = 2   Filtered:  PMOS-8 = 4
   Flat Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 533255   Filtered:  NMOS-8 = 4
      PMOS[p] = 270644   Filtered:  PMOS-8 = 4
corehi
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0
   Flat Device Counts
      DIODE[ndio] = 0
```

```
      RES[rp] = 0
      NMOS[n] = 532648
      PMOS[p] = 270088
DAC96
   Cell Level Device Counts
      DIODE[ndio] = 0
      RES[rp] = 0
      NMOS[n] = 0
      PMOS[p] = 0

********  DEVICE EXTRACTION STATS    TOPBLOCK = DAC96  *******
      DIODE[ndio] = 144
      RES[rp] = 144
      NMOS[n] = 1068927   Filtered:  NMOS-8 = 148
      PMOS[p] = 546060   Filtered:  PMOS-8 = 4
```

DAC96.sum

The summary file located in the run_details directory also gets its name from the block we specified (DAC96). The summary file contains information about the steps Hercules executed during the device extraction and netlisting phase of the LVS job. It also shows the resources used in each step. The summary file repeats much of the information that appeared in the runset file, plus a full list of options in the various runset sections, complete with user and default settings.

Many of the sections are the same as the ones reviewed in Chapter 2, “An Error-Free Design for DRC.” Only the sections unique to a Hercules LVS job are explained in detail. For more detail on the other sections, refer to Chapter 2.

Example 7-13 DAC96.sum File (No Errors)

```
Hercules (R) Hierarchical Design Verification, DATA OMITTED
 Synopsys. All rights reserved.

The information here shows the source of Hercules, followed by
the command you typed to process the runset, "hercules
dac96lvs1.ev."

Called as: hercules dac96lvs1.ev

DATE OMITTED

The following line shows the version of EV_Engine used to run the
Device Extraction. This is the same executable used for DRC. All
polygon processing is done with EV_Engine. The naming convention
for each release of this executable is as follows:
year.quarter.compile#. If there is a patch release, the naming
convention changes to: year.quarter.patch#.compile#

EV_Engine (R) Hierarchical Design Rule Checker, DATA OMITTED
  Synopsys, Inc.  All rights reserved.
```

```
Running Multi-Threaded code with 2 thread(s)

Notice that the following information shows the names of the input
and output files as well as the sources and destinations of each.

Runset file ......................  dac96lvs1.ev
Current Directory ................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_LVS/
dac96lvs1
Hostname .........................  sunserv1
Platform type ....................  SUN64_58
MILKYWAY input library path ......  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_LVS/
dac96lvs1/
MILKYWAY input file name .........  dac96
MILKYWAY block name ..............  DAC96
MILKYWAY output library path .....  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_LVS/
dac96lvs1/
MILKYWAY output file name ........  dac96_out
MILKYWAY output_block name .......  EV_OUTPUT
Run details ......................  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_LVS/
dac96lvs1/run_details
Group file directory .............  /remote/wwas1/hercules/venu/
HERCULES_DOC/tutor/TUTORIAL_LAB/Getting_Started_Hercules_LVS/
dac96lvs1/group
Retain group files ...............  smart
Check reference structures .......  yes

The OPTIONS section lists the settings you used to analyze and
 display data, including the default setting your runset did not
 alter. Notice that the EDTEXT file path is listed here.

OPTIONS {
   ALL_TEXT_GLOBAL=FALSE
   ASCII_ONLY=FALSE
   ATTACH_TEXT=ALL
   BOX_CORNER=FALSE
   CHECK_REF_LIB=TRUE
   COUNT_TRAPS=FALSE
   EDTEXT=/remote/wwas1/hercules/venu/HERCULES_DOC/tutor/
TUTORIAL_LAB/Getting_Started_Hercules_LVS/dac96lvs1/dac96.text
   ERR_LIMIT_PER_CHECK = UNLIMITED
   ERR_PREFIX = ERR
   EXPLODE_AREFS=FALSE
   EXPLODE_HOLDING_CELL_LIMIT=0
   EXPLODE_LIMIT=0
   FLAG_ALL_AREF_ERRORS=FALSE
   FLAT_COUNT=FALSE
   FLAT_ERROR=FALSE
   GENERATE_INSTANCE_NAME=TRUE
   HIERARCHICAL_DELIMITER = \
```

```
    IGNORE_CASE=FALSE
    INCREMENTAL_CELLS=FALSE
    INCREMENTAL_CELLS_FILE =
    INSTANCE_PREFIX =
    LAYOUT_GROUND = { GND VSSIO }
    LAYOUT_POWER = { VDD VDDIO }
    NET_PREFIX = XX_
    SELECT_CELL_TO_NO_EXPLODE=TRUE
    EQUIV_TO_NO_EXPLODE=TRUE
    SCHEMATIC_TO_NO_EXPLODE=TRUE
    BLACK_BOX_TO_NO_EXPLODE=TRUE
    PROTOTYPE_PLACEMENTS=FALSE
    NO_MERGE=FALSE
    GENERATE_INSTANCE_NAME=TRUE
    PRINT_ERRSUM_FILE=TRUE
    MAXIMUM_CELLNAME_LENGTH=32
    SCHEMATIC_GLOBAL = { VDD GND VDDIO VSSIO }
    SCHEMATIC_GROUND = { GND VSSIO }
    SCHEMATIC_POWER = { VDD VDDIO }
    SIZE_ENDPOINTS=FALSE
    SNAP_RES=TRUE
    SQUARE_CORNER=FALSE
    STOP_ON_GROUP_ERROR=TRUE
    TEXT_RECT=0
    USE_EXPLODED_TEXT=FALSE
    WIDTH=0
    MAGNIFICATION_FACTOR=1
    OUTPUT_MAGNIFICATION_FACTOR=1
    POLYGON_COUNT_IN_ASSIGN = FALSE
    FLAT_POLYGON_COUNT = FALSE
    CREATE_VUE_OUTPUT = TRUE
    REMOVE_DANGLING_PORT = UNTEXTED
}
```

The PREPROCESS OPTIONS allow you to specify printing information on Hercules tool efficiency while processing the design hierarchy, as well as to set options for path grid checking. This section lists all available options.

```
PREPROCESS_OPTIONS {
    CELL_PROFILE = FALSE
    CELL_PROFILE_CNT=20
    CHECK_PATH_ENDPOINTS = TRUE
    CHECK_PATH_45 = TRUE
    CHECK_PATH_90 = FALSE
    DESIGN_STATS = TRUE
    TREE = TRUE
    CELL_STATS = TRUE
    PRINT_PREPROCESS_FILES = TRUE
}
```

The TECHNOLOGY OPTIONS are the default hierarchy optimization options performed while processing the layout. We did not change any of

```
these options in our runset.

TECHNOLOGY_OPTIONS {
   BAR_AUTO_EXPLODE = TRUE
   VIA_AUTO_EXPLODE = TRUE
   SUBLEAF_AUTO_EXPLODE = 6
   ALLOW_EXPLODE_WITH_TEXT = TRUE
   POST_VCELL_EXPLODE_CELL_SIZE <= 10
   EXPLODE_CELL_SIZE_PERCENT = 70
   CELL_SIZE_AUTO_EXPLODE <= 10
   EXPLODE_AREFS = FALSE
   EXPLODE_1XN_AREFS = FALSE
   EXPLODE_DATA_CELL_LIMIT = 4
   POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
   EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
   EXPLODE_BIG_SPARSE_CELL = TRUE
   EXPLODE_HOLDING_CELL_LIMIT = 1
   EXPLODE_PLACEMENT_LIMIT = 1
   POST_VCELL_FLATTEN_TRAP_LIMIT != 0
   POST_VCELL_EXPLODE_LOW_MEMORY = TRUE
}

EVACCESS_OPTIONS {
   PATH = /remote/wwas1/hercules/venu/HERCULES_DOC/tutor/
TUTORIAL_LAB/Getting_Started_Hercules_LVS/dac96lvs1/
run_details/evaccess
   LIBRARY = DAC96
   CREATE_MSG_VIEW = TRUE
   CREATE_NETLIST_VIEW = TRUE
   CREATE_XREF_VIEW = TRUE
   CREATE_GRAF_VIEW = TRUE
}
ASSIGN {
   NDIFF    (1)
   PDIFF    (2)
   NWELL    (3)
   POLY     (5)    text(25)
   CONT     (6)
   M1       (8)    text(28)
   V1       (9)
   M2       (10)   text(30)
   PAD      (15)   text(35)
   DIFF     (1-2)
   RESP     (50)}  HIERARCHY = ERR_1.HIERARCHY
TEXT_OPTIONS {
   ATTACH_TEXT = ALL
}
DATATYPE_OFFSET=FALSE. There will be no datatype difference
between FRAM and CEL views.
Input Library Format:  127 char cellnames
Output Library Format:  No output library

  Reading hierarchy time = 0:00:01  User=0.69 Sys=0.39 Mem=31.754
```

```
  Updating hierarchy time = 0:00:00  User=0.00 Sys=0.00 Mem=20.934

Preprocessing group files...

NOTE: If no options are specified, Hercules uses the default
values for the TECHNOLOGY options, as explained previously.

If you supply an equivalence file during a Hercules LVS run, the
TECHNOLOGY_OPTIONS section automatically places all layout cells
listed as equivalence points in a NO_EXPLODE list. This guarantees
that all of the layout cells you want to have as equivalence
points are not exploded or flattened due to cell size and that
they will appear in your layout netlist.

Preprocess Step 1 : Exploding
  EQUIV_TO_NO_EXPLODE changed 84 cells to NO_EXPLODE
  Exploding time = 0:00:05  User=2.45 Sys=0.24 Mem=22.044

TECHNOLOGY_OPTIONS {
   VCELL_PASS   {
      STYLE = PAIRS
      ITERATE_MAX = 15
      ARRAY_ID = TRUE
      EXPLODE_INTO_VCELL = FALSE
      MIN_COUNT = 20
      TOP_PERCENT_OF_VALUE = 40
   }
}
Preprocess Step 2 : Vcell_pass Arrays and Pairs Iteration 1
   Pairs time = 0:00:00  User=0.00 Sys=0.00 Mem=10.998

VCELL_PASS 1, no changes.

  Combined VCELL time = 0:00:01  User=0.11 Sys=0.05 Mem=11.138

Preprocess Step 3 : Post-VCell Explodes
  Post VCell time = 0:00:00  User=0.00 Sys=0.00 Mem=9.967

  Determine Region time = 0:00:00  User=0.00 Sys=0.00 Mem=16.380

0 Total self_intersect errors found.
  Create Layer Setup time = 0:00:00  User=0.00 Sys=0.01
Mem=18.380

  Preprocessing time = 0:00:00  User=0.02 Sys=0.01 Mem=18.380

Checking database:

The following are the commands that generate the layers used for
the designed device extraction and connectivity. A processing
wall time, CPU time, system time (I/O or other non-CPU events),
and memory usage is given for each command. Also, the "unique
```

```
polygons written" number gives you a hierarchical, not a flat,
count. If you encounter a command that takes a long time to run
relative to the rest of the commands in your runset and has a
relatively high unique polygon count, these two things combined
could indicate a poor hierarchical design or poor hierarchy
processing for your design.
......................... DATA OMITTED .........................

BOOLEAN PDIFF AND NWELL { } TEMP=pdev
38 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.02 Mem=18.754

BOOLEAN NDIFF NOT NWELL { } TEMP=ndev
44 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.01 Mem=18.910

BOOLEAN PDIFF NOT NWELL { } TEMP=subtie
32 unique polygons written.
   Check time = 0:00:00  User=0.02 Sys=0.02 Mem=18.972

BOOLEAN NDIFF AND NWELL { } TEMP=welltie
47 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.01 Mem=18.894

BOOLEAN POLY AND ndev { } TEMP=ngate
156 unique polygons written.
   Check time = 0:00:00  User=0.04 Sys=0.02 Mem=19.941

BOOLEAN POLY AND pdev { } TEMP=pgate
143 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.01 Mem=19.988

BOOLEAN ndev NOT ngate { } TEMP=nsd
210 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.00 Mem=21.144

BOOLEAN pdev NOT pgate { } TEMP=psd
186 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.00 Mem=21.159

BOOLEAN POLY NOT ngate { } TEMP=temp_field
335 unique polygons written.
   Check time = 0:00:00  User=0.05 Sys=0.01 Mem=21.159

BOOLEAN temp_field NOT pgate { } TEMP=tmp_field_poly
426 unique polygons written.
   Check time = 0:00:00  User=0.05 Sys=0.01 Mem=20.191

BOOLEAN tmp_field_poly AND RESP { } TEMP=rpoly
1 unique polygon written.
   Check time = 0:00:00  User=0.01 Sys=0.03 Mem=19.050

BOOLEAN tmp_field_poly NOT rpoly { } TEMP=field_poly
```

```
427 unique polygons written.
   Check time = 0:00:00  User=0.02 Sys=0.01 Mem=19.159

SELECT field_poly TOUCHING rpoly {}TEMP=res_term
2 unique polygons written.
   Check time = 0:00:00  User=0.09 Sys=0.03 Mem=22.362

SELECT NDIFF INTERACT POLY {}TEMP=nodiode
45 unique polygons written.
   Check time = 0:00:00  User=0.07 Sys=0.03 Mem=23.456

BOOLEAN NDIFF NOT nodiode { } TEMP=pos_ndiffdio
35 unique polygons written.
   Check time = 0:00:00  User=0.03 Sys=0.01 Mem=20.315

BOOLEAN pos_ndiffdio NOT NWELL { } TEMP=ndiffdio
1 unique polygon written.
   Check time = 0:00:00  User=0.04 Sys=0.00 Mem=20.347

PREPROCESS_DEVICE DIODE {
   ndiffdio {LAYERTYPE= DEVICE, TERM(1)}TEMP=__PD_ndio_ndiffdio
   SUBSTRATE {LAYERTYPE= TERM(2)}
}
1 unique polygon written.
1 unique polygon written
Pulled 0 polygons
Pulled 0 polygons
   Check time = 0:00:00  User=0.01 Sys=0.01 Mem=22.284

PREPROCESS_DEVICE RES {
   rpoly {LAYERTYPE= DEVICE}TEMP=__PD_rp_rpoly
   res_term {LAYERTYPE= TERM(1), TERM(2)}TEMP=__PD_rp_res_term
}
1 unique polygon written.
2 unique polygons written.
Pulled 0 polygons
Pulled 0 polygons
   Check time = 0:00:00  User=0.02 Sys=0.01 Mem=22.331

PREPROCESS_DEVICE NMOS {
   ngate {LAYERTYPE= DEVICE}TEMP=__PD_n_ngate
   nsd {LAYERTYPE= TERM(1), TERM(2)}TEMP=__PD_n_nsd
   SUBSTRATE {LAYERTYPE= TERM(3)}
}
156 unique polygons written.
210 unique polygons written.
Pulled 23 polygons
Pulled 36 polygons
   Check time = 0:00:00  User=0.06 Sys=0.04 Mem=27.611

PREPROCESS_DEVICE PMOS {
   pgate {LAYERTYPE= DEVICE}TEMP=__PD_p_pgate
   psd {LAYERTYPE= TERM(1), TERM(2)}TEMP=__PD_p_psd
```

```
   NWELL {LAYERTYPE= TERM(3)}TEMP=__PD_p_NWELL
}
143 unique polygons written.
186 unique polygons written.
36 unique polygons written.
Pulled 26 polygons
Pulled 39 polygons
   Check time = 0:00:00  User=0.07 Sys=0.06 Mem=25.736

MERGE_DEVICE_LAYER {
   LAYERLIST = { ndiffdio __PD_ndio_ndiffdio }
}TEMP=ndiffdio
1 unique polygon written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=15.549

MERGE_DEVICE_LAYER {
   LAYERLIST = { rpoly __PD_rp_rpoly }
}TEMP=rpoly
1 unique polygon written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=15.534

MERGE_DEVICE_LAYER {
   LAYERLIST = { res_term __PD_rp_res_term }
}TEMP=res_term
2 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=15.549

MERGE_DEVICE_LAYER {
   LAYERLIST = { ngate __PD_n_ngate }
}TEMP=ngate
200 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=18.549

MERGE_DEVICE_LAYER {
   LAYERLIST = { nsd __PD_n_nsd }
}TEMP=nsd
322 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=18.565

MERGE_DEVICE_LAYER {
   LAYERLIST = { pgate __PD_p_pgate }
}TEMP=pgate
190 unique polygons written.
   Check time = 0:00:00  User=0.00 Sys=0.00 Mem=17.503

MERGE_DEVICE_LAYER {
   LAYERLIST = { psd __PD_p_psd }
}TEMP=psd
304 unique polygons written.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=17.518

MERGE_DEVICE_LAYER {
   LAYERLIST = { NWELL __PD_p_NWELL }
```

```
}TEMP=NWELL
83 unique polygons written.
   Check time = 0:00:00  User=0.01 Sys=0.00 Mem=17.456

The following is the DIODE extraction command. The number of
"unique" devices extracted is a hierarchical count. For a flat
count, you will need to look in the DAC96.acct file. If there
were any extraction errors, you would get a message below the
DIODE command in this summary file. The rest of the device
extraction commands follow.

DIODE ndio ndiffdio ndiffdio SUBSTRATE {
   BODY_SELECT_LAYER = { __PD_ndio_ndiffdio }
   DIODE_TYPE=NP;}
   DIODE_HIERARCHICAL=FALSE;} TEMP=ndiode
Extracted 1 unique 2-terminal device.
   Check time = 0:00:01  User=0.26 Sys=0.14 Mem=24.487

RES rp rpoly res_term res_term {
   BODY_SELECT_LAYER = { __PD_rp_rpoly }
   RES_HIERARCHICAL=FALSE;
   EV_RESVAL = 1;
} TEMP=pdevice
Extracted 1 unique 2-terminal device.
   Check time = 0:00:01  User=0.24 Sys=0.14 Mem=21.518

Notice that even though this design has over 1.5 million devices,
Hercules has to extract only 148 NMOS and 109 PMOS devices. This
shows that the design has very good hierarchy.

NMOS n ngate nsd nsd SUBSTRATE {
   FILTER_OPTIONS = { NMOS-3, NMOS-8 };
   BODY_SELECT_LAYER = { __PD_n_ngate }
   MOS_HIERARCHICAL=FALSE;
   MOS_CALC_NRS_NRD=TRUE;} TEMP=ndevice
Extracted 148 unique 4-terminal devices.
   Check time = 0:00:01  User=0.54 Sys=0.21 Mem=29.814

PMOS p pgate psd psd NWELL {
   FILTER_OPTIONS = { PMOS-3, PMOS-8 };
   BODY_SELECT_LAYER = { __PD_p_pgate }
   MOS_HIERARCHICAL=FALSE;
   MOS_CALC_NRS_NRD=TRUE;} TEMP=pdevice
Extracted 109 unique 4-terminal devices.
   Check time = 0:00:01  User=0.48 Sys=0.24 Mem=31.923

The next two sections show the CONNECT and TEXT commands that
were executed, with runtime and memory usage.

CONNECT {
   ngate pgate BY [ OVERLAP TOUCH ] field_poly
   M2 BY [ OVERLAP TOUCH ] PAD
   M1 M2 BY [ OVERLAP TOUCH ] V1
```

```
   M1 ndiffdio res_term field_poly nsd psd welltie subtie BY
      [ OVERLAP TOUCH ] CONT
   NWELL BY [ OVERLAP TOUCH ] welltie
   SUBSTRATE BY [ OVERLAP TOUCH ] subtie
}  CONNECT_DB = __CONNECT_DB_Dkhniq__0001
   Check time = 0:00:02  User=1.85 Sys=0.07 Mem=40.219

TEXT {
   M1 BY M1.TEXT
   M2 BY M2.TEXT
   field_poly BY POLY.TEXT
   PAD BY PAD.TEXT
   NWELL BY "VDD"
   SUBSTRATE BY "GND"
}
    Check time = 0:00:00  User=0.03 Sys=0.04 Mem=28.799

    Check time = 0:00:00  User=0.03 Sys=0.02 Mem=28.799

    Check time = 0:00:00  User=0.01 Sys=0.02 Mem=25.767

    Check time = 0:00:00  User=0.01 Sys=0.01 Mem=24.767

    Check time = 0:00:00  User=0.01 Sys=0.00 Mem=23.799

    Check time = 0:00:00  User=0.00 Sys=0.00 Mem=23.799

   Total Text time = 0:00:00  User=0.18 Sys=0.17 Mem=33.783

CONNECT_DEVICES
Connecting devices ...
   Device Connect time = 0:00:01  User=0.02 Sys=0.03 Mem=35.578

PROCESS_TEXT_OPENS is a command that is run by default when the
NETLIST or GRAPHICS command appears in the runset. It is run prior
to these commands and checks for text opens and shorts. It also
merges nets with identical text.

PROCESS_TEXT_OPENS {}
Completed opens check. Now merging/renaming nets based on text.
   Check time = 0:00:01  User=0.31 Sys=0.01 Mem=23.440

The GRAPHICS command writes the device layer and connectivity
layer data to an output database with property information. This
is used by VUE for debug, or it can be read into any layout editor
for manual debug. We specified VUE_LAYERS (100). Hercules
automatically determines all of the device and connectivity layers
necessary to create a complete database, starts numbering the
layers with 100, and outputs them. The summary is where you need
to look to see which layers are associated with each output layer
number. VUE does this layer-to-number association automatically
for you; if you are doing it manually in a layout editor, you
will need to refer to this summary file.
```

```
GRAPHICS_NETLIST {
   VUE_LAYERS (100)

   SUBSTRATE       (100)
   subtie          (101)
   welltie         (102)
   field_poly      (103)
   psd             (104)
   pgate           (105)
   ndevice         (106)
   nsd             (107)
   ngate           (108)
   pdevice         (109)
   res_term        (110)
   rpoly           (111)
   ndiode          (112)
   ndiffdio        (113)
   NDIFF           (1)
   PDIFF           (2)
   NWELL           (3)
   POLY            (5)
   PAD             (15)
   RESP            (50)
   M1              (8)
   M2              (10)
   CONT            (6)
   V1              (9)}
GRAPHICS_PROPERTY {
   NET_NAME (1)
   INSTANCE_NAME (4)
}
Writing Graphics Netlist

The following is a summary of the time and memory it took to write
the graphics netlist. You should write it to a local disk if this
database is large.

Check time = 0:00:22  User=5.13 Sys=6.77 Mem=38.044

The Hercules ev_netlist executable must read the connectivity
database that is created below. If this database is created
successfully, but for some reason your NETLIST command does not
complete, you can rerun the following Hercules command to generate
your <block>.net file: >hercules -N dac96lvs1.ev

Creating netlisting connect database ...
   Check time = 0:00:02  User=0.84 Sys=0.60 Mem=45.032

Generating BLOCK.acct file ...
   Check time = 0:00:00  User=0.01 Sys=0.02 Mem=35.798

Checks complete.
```

```
  Total check time = 0:00:37  User=11.93 Sys=9.87 Mem=45.032

  Saving ERR & PERM data time = 0:00:00  User=0.00 Sys=0.00
Mem=30.414
```

Here is the total time and the maximum memory used for the Device Extraction phase of the run, including checks, preprocessing, and outputting data. This time is only for the EV_Engine executable.

```
Overall ev_engine time = 0:01:12  User=39.47 Sys=10.94 Mem=45.032
```

The next section of the summary file gives information on the EV_NETLIST executable that Hercules calls. This step generates the DAC96.net layout netlist file.

```
EV_NETLIST (R) Stand-alone netlist generator, DATA OMITTED

  Synopsys, Inc.  All rights reserved.

 Generating NETLIST file "DAC96.net".
```

Once the netlist is written to an output file, you get a summary of the total runtime and memory usage for this executable.

```
 Netlisting time = 0:00:14  User=12.07 Sys=0.45 Mem=0.000
```

DAC96.net

At the end of the Device Extraction phase and netlisting, Hercules generates an ASCII text file of the layout netlist. This is always the top block name followed by a .net extension. In our design it is DAC96.net. This netlist is a Hercules-formatted netlist representing the graphical input data based on the device definition and connectivity specified in the Hercules runset.

Tree Files and Technology Option Files

The remainder of the files generated during the Device Extraction phase of the Hercules LVS job are DAC96.tree0, DAC96.tree1, DAC96.tree3, DAC96.tech, and DAC96.vcell. These are all similar to the files detailed in Chapter 2. Refer to that explanation of these files if you have not already done so.

LVS Comparison Output Files

The second group of output files we review are those associated with the comparison of the DAC96.net layout netlist file and the dac96.hrc schematic netlist file. The executable running during this phase is lsh. Because this is the first time we have presented them in the tutorial, we go into detail on these files.

DAC96.LVS_ERRORS

The DAC96.LVS_ERRORS file, shown in Example 7-14, is a summary of major problems found in the Hercules LVS job. This Hercules run passed at the top level [DAC95 == DAC96], however, there is a problem with one equivalence error.

Example 7-14 DAC96.LVS_ERRORS File

```
+-------------------------------------------------------------+
Hercules LVS Comparison Report
+-------------------------------------------------------------+
COMPARE (R) Hierarchical Layout Vs. Schematic
    DATA OMITTED
Copyright (C) Synopsys, Inc. All rights reserved.

---------------------------------------------------------------
LVS error file     = DAC96.LVS_ERRORS
Schematic netlist  = dac96.hrc
Layout netlist     = DAC96.net
Equivalence file   = dac96.eqv
Runset file        = dac96lvs1.ev
Working directory  =
/remote/us54h1/marable/tutorial_2006.12/tutorial
Getting_Started_Hercules_LVS/dac96lvs1
Compare directory  = compare
Compare start time = 2006-11-22 11:13:47

---------------------------------------------------------------
Top block compare result: PASS

                      #####   ##   ##### #####
                      #   #  #  #  #     #
                      ##### ###### ##### #####
                      #     #    #     #     #
                      #     #    # ##### #####

                          [DAC95 == DAC96]

---------------------------------------------------------------
Comparison summary

   84 successful equivalencies
*  0 failed equivalencies

Schematic and layout agree at all equivalent points.

End of LVS comparison report
```

DAC96_lvs.log

The DAC96_lvs.log file provides information on runtime and performance. This file contains the Hercules version, environment, and option settings to document the conditions of the run. It also provides information on each stage of the processing:

- Preprocessing Stage: Equate and equivalence point information
- Compare Stage: Device flattening, merging, and filtering; ERROR listing; and detailed Runtime information

The DAC96_lvs.log file includes a comparison summary with the PASS/FAIL logo, and results categorized by pass/fail equivalent points and error/ warning types. It also contains the detailed runtime information including wall-time information.

Example 7-15 DAC96_lvs.log - Comparison Summary File

```
The following information shows the names of the input files and their
sources. It also shows all of the default COMPARE option settings, as
well as the settings from your runset.

HLVS (R) Hierarchical Layout Versus Schematic, DATA OMITTED
  Synopsys.  All rights reserved.

Call as: lsh -s /remote//marable/tutorial_2006.12/tutorial//
Getting_Started_Hercules_LVS
dac96lvs1/dac96.hrc -Q -b DAC96 -ev-off dac96lvs1.ev

WARNING:  CREATE_VUE_OUTPUT is set to TRUE, therefore WRITE_NETLISTS
is set to TRUE.

Hercules LVS compare start time    : 2007-05-17 18:55:40

+-------------------------------------------------------------------+
-------------------------Environment Status-------------------------
+-------------------------------------------------------------------+

Hostname            = luau
Platform type       = AMD64_L24
Runset file         = dac96lvs1.ev
Working directory   = /remote/us54h1/marable/tutorial_2006.12/tutorial/
Getting_Started_Hercules_LVS/dac96lvs1
Top block           = DAC96
Layout netlist      = DAC96.net
Schematic netlist   = dac96.hrc
Equivalence_file    = dac96.eqv
Compare directory   = compare

+-------------------------------------------------------------------+
---------------------------Options Listing-----------------------------
------------------* = different from Hercules default-----------------
+-------------------------------------------------------------------+

 == Compare options ==

 add_width_for_parallel_merge   = FALSE
 all_ports_texted               = FALSE
 auto_exclude_equiv             = TRUE
 combine_output_files           = FALSE
```

```
*compare_properties             = TRUE
delete_lay                      = [none]
delete_sch                      = [none]
*detect_permutable_ports        = TRUE
equate_by_device_name           = FALSE
*equate_by_net_name             = TRUE
equiv_by_name                   = FALSE
*explode_on_error               = TRUE
*filter                         = TRUE
find_additional_equivs
    ignore_case                 = FALSE
    name_full                   = FALSE
    name_substring              = FALSE
    name_prefix                 = FALSE
    statistical                 = FALSE
    single_device_cell          = FALSE
    filter_multi_equivs         = FALSE
    full_equiv                  = FALSE
    duplicate_equivs            = FALSE
find_duplicate_equivs           = FALSE
generate_black_box              = [none]
html_output                     = FALSE
match_by_property               = FALSE
matched_ports_continue          = FALSE
memory_array_comparison         = TRUE
merge_all_caps                  = FALSE
merge_all_ind                   = FALSE
merge_all_res                   = FALSE
merge_net_range                 = 100 1000
* merge_parallel                = TRUE
merge_paths                     = FALSE
merge_paths_device_limit        = 0
merge_series                    = FALSE
merge_series_gates              = FALSE
net_print_limit                 = 10
one_connection_warning          = FALSE
optional_pins                   = TRUE
parallel_merge_ratio            = FALSE
port_swap_on_top_block          = FALSE
print_ignore_equiv              = [none]
property_errors_continue        = FALSE
property_tolerance              = 0.10
property_warning                = FALSE
* push_down_devices             = TRUE
* push_down_pins                = TRUE
* remove_dangling_nets            = TRUE
require_texted_ports_match      = FALSE
require_texted_nets_match       = FALSE
restrict_merge_by_length        = FALSE
restrict_merge_by_width         = FALSE
restrict_merge_series           = FALSE
restrict_parallel_merging       = FALSE
restrict_series_merging         = FALSE
```

```
 * retain_new_data              = TRUE
 retain_previous_data           = FALSE
 schematic_vs_schematic         = FALSE
 short_equivalent_nodes         = FALSE
 static_equated_nets            = TRUE
 * stop_on_error                = FALSE
 stop_on_no_explode_error       = FALSE
 text_resolves_port_swap        = TRUE
 tolerance_device_count         = [none]
 tolerance_net_count            = [none]
 tolerance_type                 = RELATIVE
 use_total_width                = FALSE
 write_netlists               = TRUE
 zero_connection_warning        = FALSE

 == Evaccess options ==

 create_msg_view                = FALSE
 create_netlist_view            = FALSE
 create_xref_view               = FALSE

 == General options ==

 * create_vue_output            = TRUE
 ignore_case                    = FALSE
 layout_global                  = [none]
 * layout_ground                = VSSIO GND
 * layout_power                 = VDDIO VDD
 lvs_report_level               = prop-2 xpin-0 dpin-1
 lvs_report_old_format          = FALSE
 lvs_report_record_limit        = 50
 message_enable                 = [none]
 message_error                  = [none]
 message_identifiers            = FALSE
 message_suppress               = [none]
 print_precision                 = 4
 * schematic_global              = VSSIO VDDIO GND VDD
 * schematic_ground              = VSSIO GND
 * schematic_power               = VDDIO VDD
```

The following functions are preprocessing and formatting performed by the Hercules lsh executable, before the comparison of the individual cells.

```
+-------------------------------------------------------------------+
|                          Preprocessing Stage                      |
+-------------------------------------------------------------------+
Purging Compare Directory ... OK

Reading schematic netlist ...
     Generating schematic index file ...
Reading schematic time = 0:00:00 User=0.03 Sys=0.00 Mem=16.553
Reading layout netlist ...
```

```
     Generating layout index file ...
Reading layout time = 0:00:00 User=0.03 Sys=0.02 Mem=22.756

Processing schematic netlist ...
     Propagating schematic globals ...
     Propagating schematic globals time = 0:00:00 User=0.01 Sys=0.00
Mem=24.818
Processing schematic time = 0:00:00 User=0.01 Sys=0.00 Mem=25.826

Processing layout netlist ...
Processing layout time = 0:00:00 User=0.01 Sys=0.00 Mem=26.849

Creating tree files ...
     Schematic netlist top block: DAC95     Unique cell count: 74
     Layout netlist top block: DAC96        Unique cell count: 109
Creating tree files time = 0:00:00 User=0.00 Sys=0.00 Mem=26.795

Removing dangling nets and pushing down connected pins ...
Dangling nets, push down pins time = 0:00:00 User=0.03 Sys=0.00
Mem=29.912

Pushing down devices ...
Push down device time = 0:00:00 User=0.01 Sys=0.00 Mem=34.013

User-provided equivalence point file: 'dac96.eqv'.

Writing generated runset information ...
Writing generated runset time = 0:00:00 User=0.00 Sys=0.00 Mem=30.904

Memory array comparison time = 0:00:00 User=0.00 Sys=0.00 Mem=30.904

Removing empty schematic cells:
 Cell 'CE8X8'
 Cell 'CE32X32'
 Cell 'MEMORY'
 Cell 'CE64K'
 Cell 'CE16K'
 Cell 'CE2X2'
 Cell 'CE4X4'
 Cell 'CE64X64'
 Cell 'CE16X16'

Removing empty layout cells:
 Cell 'VSSCRPAD'
 Cell 'ce16x16_2'
 Cell 'ce16x16_3'
 Cell 'VDDIOPAD'
 Cell 'ce16k_3'
 Cell 'cell_hlding'
 Cell 'ce64kd'
 Cell 'ce2x2_2'
 Cell 'ce8x8_2'
 Cell 'ce4x4_2'
```

```
 Cell 'ce2x2_3'
 Cell 'ce8x8_3'
 Cell 'ce4x4_3'
 Cell 'VDDCRPAD'
 Cell 'VSSIOPAD'
 Cell 'ce32x32_2'
 Cell 'cell_no_ovlp'
 Cell 'ce64x64_2'
 Cell 'ce32x32_3'
 Cell 'ce64x64_3'

Preprocessing schematic data ... OK

Preprocessing layout data ... OK

Preprocessing stage finished successfully.
Preprocessing stage time = 0:00:00 User=0.17 Sys=0.03 Mem=34.013
```

Once all of the layout and schematic is read and preprocessed, Hercules starts at the lowest level cell and begins to compare blocks. Each time Hercules advances up a level in the hierarchy a new LEVEL title will appear.

```
+-------------------------------------------------------------------+
|                           Comparison Stage                        |
+-------------------------------------------------------------------+

+-----------------------------------------------------------+
|                           Level 10                        |
+-----------------------------------------------------------+
Equivalence point: [BSEL, bsel_a2] Level = 10

 Flattening schematic netlist ...
     4 dangling nets are found.
     total device and net count: 2 6
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=34.278

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=34.262

 Flattening layout netlist ...
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=34.294

 Merging and filtering layout devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=34.262

 Comparing circuit logic ...

     Matching unique nets/devices ...
          Checking swappable ports ...
          Number of swappable ports = 4
     Matching 100% elements time = 0:00:01 User=0.00 Sys=0.00 Mem=35.309

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=36.317
```

```
Result: PASS [BSEL, bsel_a2]
 Summary file: compare/bsel_a2/sum.BSEL.bsel_a2

Elapsed time = 0:00:01 User=0.02 Sys=0.00 Mem=36.294

------------------------------------------------------------

Equivalence point: [BSEL, bsel] Level = 10
 Flattening schematic netlist ...
     4 dangling nets are found.
     total device and net count: 2 6
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Flattening layout netlist ...
     4 dangling nets are found.
     total device and net count: 2 6
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309

 Merging and filtering layout devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Comparing circuit logic ...

     Matching unique nets/devices ...
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0. Mem=35.247

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=36.333

Result: PASS [BSEL, bsel]
 Summary file: compare/bsel/sum.BSEL.bsel

Elapsed time = 0:00:00 User=0.00 Sys=0.00 Mem=36.278

------------------------------------------------------------

Equivalence point: [BSEL, bsel_a1] Level = 10

 Flattening schematic netlist ...
     4 dangling nets are found.
     total device and net count: 2 6
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Flattening layout netlist ...
     4 dangling nets are found.
     total device and net count: 2 6
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309
```

```
 Merging and filtering layout devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Comparing circuit logic ...

     Matching unique nets/devices ...
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0.00 Mem=35.247

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=36.333

Result: PASS [BSEL, bsel_a1]
 Summary file: compare/bsel_a1/sum.BSEL.bsel_a1

Elapsed time = 0:00:00 User=0.01 Sys=0.00 Mem=36.278

+-------------------------------------------------------------+
|                                                             |
|Level 10:                                                    |
|                                                             |
|    Passed equivalence point(s):                             |
|        BSEL == bsel_a2                                      |
|        BSEL == bsel                                         |
|        BSEL == bsel_a1                                      |
|                                                             |
|    Level 10 time = 0:00:01 User=0.03 Sys=0.00 Mem=36.294    |
|                                                             |
+-------------------------------------------------------------+
```

A summary of each layout and schematic block is written to this file, including the elapsed compare time, CPU (user) time and total memory used to compare the blocks. Later in this chapter we will review what the partitioned schematic and partitioned layout look like.

```
+-------------------------------------------------------------+
|                           Level 9                           |
+-------------------------------------------------------------+

Equivalence point: [SENSEAMP, senseamp] Level = 9

 Flattening schematic netlist ...
     4 dangling nets are found.
     total device and net count: 6 10
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Flattening layout netlist ...
     total device and net count: 20 10
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309
```

```
 Merging and filtering layout devices ...
     6 composite parallel devices are created.
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Comparing circuit logic ...

     Matching unique nets/devices ...
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0.00 Mem=35.231

 Comparing device properties ...
 Comparing properties time = 0:00:00 User=0.00 Sys=0.00 Mem=35.247

Result: PASS [SENSEAMP, senseamp]
 Summary file: compare/senseamp/sum.SENSEAMP.senseamp

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=35.333

Elapsed time = 0:00:00 User=0.00 Sys=0.00 Mem=36.262
-------------------------------------------------------------

Equivalence point: [BSEL_8, bsel8] Level = 9

 Flattening schematic netlist ...
     24 dangling nets are found.
     total device and net count: 8 27
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=35.309

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=35.278

 Flattening layout netlist ...
     24 dangling nets are found.
     total device and net count: 8 27
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=36.309

 Merging and filtering layout devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=36.278

 Comparing circuit logic ...

     Matching unique nets/devices ...
          Checking swappable ports ...
          Number of swappable ports = 24
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0.00 Mem=38.294

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=36.333

Result: PASS [BSEL_8, bsel8]
 Summary file: compare/bsel8/sum.BSEL_8.bsel8

Elapsed time = 0:00:00 User=0.01 Sys=0.00 Mem=38.294
```

After Hercules has completed all of the blocks at a given "Level," a

summary of the equivalent and nonequivalent blocks is given. In our example all blocks compare. The equal signs (==) always indicate a successful comparison, as defined by your COMPARE option settings.

```
+-----------------------------------------------------------+
|                                                           |
|Level 9:                                                   |
|                                                           |
|    Passed equivalence point(s):                           |
|        SENSEAMP == senseamp                               |
|        BSEL_8 == bsel8                                    |
|                                                           |
|    Level 9 time = 0:00:00 User=0.01 Sys=0.00 Mem=38.294   |
|                                                           |
+-----------------------------------------------------------+

.........................DATA OMITTED.............................
```

For the IOBUF block, a WARNING is reported. This block will still remain as a compared block and will not be exploded into its parent. Only blocks with ERRORS are exploded. You can control whether certain comparison violations are WARNINGS or upgrade them to ERRORS, causing the block to miscompare and be exploded into the parent. This skill is covered later
in the tutorial.

```
Equivalence point: [IOBUF, IOBUF] Level = 2

 Flattening schematic netlist ...
     2 dangling nets are found.
     total device and net count: 15 18
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=77.932

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=77.901

 Flattening layout netlist ...
     2 dangling nets are found.
     total device and net count: 55 18
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=77.932

 Merging and filtering layout devices ...
     8 composite parallel devices are created.
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=77.901

 Comparing circuit logic ...

     Matching unique nets/devices ...
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0.00 Mem=77.854

 Comparing device properties ...
 Comparing properties time = 0:00:00 User=0.00 Sys=0.00 Mem=77.869

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=48.285
```

```
Result: PASS with WARNING [IOBUF, IOBUF]
     WARNING: Device type does not have check_properties in EQUATE
statement.
     Summary file: compare/IOBUF/sum.IOBUF

Elapsed time = 0:00:00 User=0.01 Sys=0.00 Mem=78.885

.......................DATA OMITTED.............................

When you reach "LEVEL 0," you are at the top block. In our example, the
top block, DAC96 (layout) and DAC95(schematic), compares with WARNINGS.
+-----------------------------------------------------------+
|                           Level 0                         |
+-----------------------------------------------------------+

Equivalence point: [DAC95, DAC96] Level = 0

 Flattening schematic netlist ...
     144 dangling nets are found.
     total device and net count: 146 267
 Flattening time = 0:00:00 User=0.01 Sys=0.00 Mem=80.916

 Merging and filtering schematic devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=78.885

 Flattening layout netlist ...
     144 dangling nets are found.
     total device and net count: 146 267
 Flattening time = 0:00:00 User=0.00 Sys=0.00 Mem=80.916

 Merging and filtering layout devices ...
 Merging and filtering time = 0:00:00 User=0.00 Sys=0.00 Mem=78.885

 Comparing circuit logic ...

     Matching unique nets/devices ...
     Matching 100% elements time = 0:00:00 User=0.00 Sys=0.00 Mem=78.854

 Writing time = 0:00:00 User=0.00 Sys=0.00 Mem=48.285

Result: PASS [DAC95, DAC96]
 Summary file: compare/DAC96/sum.DAC95.DAC96

Elapsed time = 0:00:00 User=0.02 Sys=0.00 Mem=80.916

+-----------------------------------------------------------+
|                                                           |
|Level 0:                                                   |
|                                                           |
|     Passed equivalence point(s):                          |
|         DAC95 == DAC96                                    |
|                                                           |
```

```
|    Level 0 time = 0:00:00 User=0.02 Sys=0.00 Mem=80.916    |
|                                                            |
+------------------------------------------------------------+
```

The last section of the summary file is a summary of blocks that compared with WARNINGS and without WARNINGS, and blocks that did not compare.

```
+-------------------------------------------------------------------------+
|                        Hercules LVS Compare Summary                     |
+-------------------------------------------------------------------------+

Final comparison result: PASS

 Categorized by pass or fail:

 84 passed equivalence point(s):
     BSEL == bsel_a2  (level 10)
     BSEL == bsel  (level 10)
     BSEL == bsel_a1  (level 10)
     SENSEAMP == senseamp  (level 9)
     BSEL_8 == bsel8  (level 9)
     NAND3 == nand3b  (level 8)
     NAND3 == nand3  (level 8)
     NAND2 == nand2b  (level 8)
     NAND2 == nand2  (level 8)
     NOR2 == nor2b  (level 8)
     NOR2 == nor2  (level 8)
     INVA == invb  (level 8)
     INVA == inva  (level 8)
     SAMP1 == samp1_a1  (level 8)
     SAMP1 == samp1  (level 8)
     XFER == xfer  (level 7)
     SAMP4 == samp4  (level 7)
     DATBIDIR == datbidir  (level 7)
     MEMBIDIR == membidir  (level 7)
     PRECHRG == prechrg  (level 7)
     AND3 == and3b  (level 7)
     AND3 == and3  (level 7)
     BUF4X == buf4x  (level 6)
     LOWDEC_8 == lowdec8  (level 6)
     PC_8 == pc_8_l_endcell  (level 6)
     PC_8 == pc_8  (level 6)
     PC_8 == pc_8_r_endcell  (level 6)
     DOUT1 == dout1  (level 6)
     LATR == latr  (level 6)
     DFFR == dffr  (level 5)
     DOUT4 == dout4  (level 5)
     PC_64 == pc_64_l_endcell  (level 5)
     PC_64 == pc_64  (level 5)
     PC_64 == pc_64_r_endcell  (level 5)
     MEDDEC_8 == meddec8  (level 5)
     ADR2DEC4 == adr2dec4  (level 5)
     BUF4X4 == buf4x4  (level 5)
```

```
NOR3 == nor3b  (level 5)
NOR2B == nor2c  (level 5)
NOR2B == nor2_gacell  (level 5)
NAND3B == nand3c  (level 5)
NAND3B == nand3_gacell  (level 5)
NOR3B == nor3c  (level 5)
NOR3B == nor3_gacell  (level 5)
INVB == invc  (level 5)
INVB == inv_gacell  (level 5)
NAND2B == nand2c  (level 5)
NAND2B == nand2_gacell  (level 5)
AND2B == and2c  (level 4)
AND2B == and2_ga  (level 4)
OR3B == or3c  (level 4)
OR3B == or3_ga  (level 4)
AND3B == and3c  (level 4)
AND3B == and3_ga  (level 4)
OR2B == or2c  (level 4)
OR2B == or2_ga  (level 4)
OR3 == or3b  (level 4)
OR2 == or2b  (level 4)
OR2 == or2  (level 4)
AND2 == and2b  (level 4)
MUX == mux  (level 4)
BUF4X8 == buf4x8  (level 4)
ADR3DEC8 == adr3dec8  (level 4)
WDEC_256 == wdec256  (level 4)
PC_256 == pc_256  (level 4)
DOUT8 == dout8  (level 4)
DFFR4 == dffr4  (level 4)
DFFR16 == dffr16  (level 3)
RAM64K == ram64k  (level 3)
MUX_4 == mux4  (level 3)
CS_ADD1 == cs_add  (level 3)
CS_ADDB == cs_add_ovlp  (level 3)
CS_ADDB == cs_add_ga  (level 3)
ADD4B == add4_ga  (level 2)
ADD4B == add4_ovlp  (level 2)
ADD4 == add4  (level 2)
MUX_16 == mux16  (level 2)
RAM128K == ram128k  (level 2)
IOBUF == IOBUF  (level 2)
IBUF == IPAD  (level 1)
COREHI == corehi  (level 1)
CORELOW == corelow  (level 1)
OBUF == OPAD  (level 1)
DAC95 == DAC96  (level 0)
```

Next is a list of the number of successful equivalences and equivalence errors. Because all of our blocks compared, we have 0 equivalence errors. It is possible to have a successful comparison with some equivalence errors. A sub-block might not compare, but if the parent of that block compares when the sub-block is exploded, your top block might still

```
compare. This information should be used simply to judge how much
hierarchy
you were able to maintain during your comparison.

 Categorized by message types:

 1 WARNING: Device type does not have check_properties in EQUATE
statement.
     [IOBUF, IOBUF]

 1 WARNING: Text mismatch.
     [COREHI, corehi]

84 equivalence points checked:
     84 passed
     0 failed
Top equivalence point [DAC95, DAC96] passed.

The total compare time (wall time), CPU time (user), and total memory
are summarized here.

Hercules LVS compare end time      : 2007-05-17 18:55:48
Total runtime for Hercules LVS compare = 0:00:08 User=1.18 Sys=0.57
Mem=85.002
```

Compare Directory Structure

As described earlier in the chapter, Hercules works from the lowest level equivalence points up through the hierarchy, comparing blocks. In our example, a block is either compared successfully and the port connections are saved to be used in the parent block comparison, or the block is compared unsuccessfully and the block is exploded in the context of the schematic and layout netlists. In the latter case, all devices in the block are moved up into the parent for comparison as long as EXPLODE_ON_ERROR=TRUE is specified.

The compare directory structure contains a subdirectory for each block listed in the equivalence file. In that subdirectory is a summary file for the block as well as the layout and schematic netlists that Hercules compared at that point in the hierarchy. Remember, one of the advantages of a hierarchical verification tool is easy debugging at the cell level, where your error is located, eliminating the need to debug the entire design. As an example, we look at one of the sub-blocks in the middle of the hierarchy.

We are reviewing these files to make you aware of the information they contain. Obviously, if you have many errors, it is cumbersome to work your way through all of these files with no direction. You can easily walk through these files using direction from the HTML interface and Hercules-VUE. We go through those exercises after we have completely gone through the basics of a Hercules hierarchical comparison.

./compare/membidir/lay.membidir

This file is the cell-level Hercules-formatted layout netlist for the equivalence point MEMBIDIR, generated by the Hercules tool during the LVS job. Notice in Example 7-16 that it is a flat netlist, where all blocks that successfully compared at a lower level are now only black-box instances.

Example 7-16 lay.membidir: A Cell-Level Layout Netlist

```
{NETLIST membidir
{VERSION 1 0 0}

{CELL membidir
    {PORT DIO BL DO VDD GND RD BLB }
    {INST inva_40=inva {PROP n="inva" x=30.500 y=-1.000 }
       {PIN RD=A XX_5=Z VDD=VDD GND=GND }}
    {INST inva_48=inva {PROP n="inva" x=73.000 y=-1.000 }
       {PIN DIO=A XX_31=Z VDD=VDD GND=GND }}
    {INST M4=p {PROP n="p" x=66.000 y=34.750 Length=1 Width=20.5 }
       {PIN XX_4=GATE DIO=SRC VDD=DRN VDD=BULK }}
    {INST inva_47=inva {PROP n="inva" x=85.000 y=-1.000 }
       {PIN XX_31=A XX_13=Z VDD=VDD GND=GND }}
    {INST M3=n {PROP n="n" x=130.500 y=15.000 Length=1 Width=10 }
}
       {PIN XX_5=GATE BLB=SRC XX_9=DRN GND=BULK }}
    {INST inva_46=inva {PROP n="inva" x=97.000 y=-1.000 }
       {PIN XX_31=A XX_29=Z VDD=VDD GND=GND }}
    {INST M2=n {PROP n="n" x=130.500 y=32.000 Length=1 Width=10 }
}
       {PIN XX_5=GATE XX_13=SRC BL=DRN GND=BULK }}
    {INST inva_45=inva {PROP n="inva" x=109.000 y=-1.000 }
       {PIN XX_29=A XX_9=Z VDD=VDD GND=GND }}
    {INST M1=n {PROP n="n" x=66.000 y=9.500 Length=1 Width=13 }
       {PIN A=GATE DIO=SRC GND=DRN GND=BULK }}
    {INST inva_44=inva {PROP n="inva" x=-23.500 y=-1.000 }
       {PIN DO=A XX_24=Z VDD=VDD GND=GND }}
    {INST inva_43=inva {PROP n="inva" x=-11.500 y=-1.000 }
       {PIN XX_24=A XX_25=Z VDD=VDD GND=GND }}
    {INST nor2_52=nor2 {PROP n="nor2" x=42.500 y=-1.000 }
       {PIN XX_5=A A=QN XX_25=B VDD=VDD GND=GND }}
    {INST inva_41=inva {PROP n="inva" x=30.500 y=-1.000 }
       {PIN XX_5=A XX_22=Z VDD=VDD GND= }}
    {INST nand2_42=nand2 {PROP n="nand2" x=0.500 y=-1.000 }
       {PIN XX_25=A GND=GND XX_4=QN XX_22=B VDD=VDD }}
}
}
```

./compare/membidir/sch.MEMBIDIR

This file is the cell-level Hercules-formatted schematic netlist for the equivalence point MEMBIDIR, generated by the Hercules tool during the LVS job. Notice in Example 7-17 that it is a flat netlist, where all blocks that successfully compared at a lower level are now only black-box instances.

Example 7-17 sch.MEMBIDIR: A Cell-Level Schematic Netlist

```
{NETLIST MEMBIDIR
{VERSION 1 0 0}

{CELL MEMBIDIR
    {PORT DIO BL DO GND VDD RD BLB }
    {INST $1I2=N {PROP n="N" Length=1 Width=10 }
       {PIN $1N43=G $1N47=S BLB=D GND=VBB }}
    {INST $1I52=INVA {PROP n="INVA" }
       {PIN $1N54=IN $1N47=OUT VDD=VDD GND=GND }}
    {INST $1I11=INVA {PROP n="INVA" }
       {PIN DIO=IN $1N34=OUT VDD=VDD GND=GND }}
    {INST $1I51=INVA {PROP n="INVA" }
       {PIN $1N34=IN $1N54=OUT VDD=VDD GND=GND }}
    {INST $1I10=INVA {PROP n="INVA" }
       {PIN DO=IN $1N15=OUT VDD=VDD GND=GND }}
    {INST $1I50=INVA {PROP n="INVA" }
       {PIN $1N43=IN $1N66=OUT VDD=VDD GND=GND }}
    {INST $1I5=NAND2 {PROP n="NAND2" }
       {PIN $1N17=A GND=GND $1N19=QN $1N66=B VDD=VDD }}
    {INST $1I49=INVA {PROP n="INVA" }
       {PIN $1N15=IN $1N17=OUT VDD=VDD GND=GND }}
    {INST $1I61=INVA {PROP n="INVA" }
       {PIN RD=IN $1N43=OUT VDD=VDD GND=GND }}
    {INST $1I4=P {PROP n="P" Length=1 Width=20.5 }
       {PIN $1N19=G DIO=S VDD=D VDD=VBB }}
    {INST $1I55=NOR2 {PROP n="NOR2" }
       {PIN $1N43=A $1N23=QN $1N17=B VDD=VDD GND=GND }}
    {INST $1I3=N {PROP n="N" Length=1 Width=13 }
       {PIN $1N23=G GND=S DIO=D GND=VBB }}
    {INST $1I53=INVA {PROP n="INVA" }
       {PIN $1N34=IN $1N45=OUT VDD=VDD GND=GND }}
}
}
```

./compare/membidir/sum.MEMBIDIR.membidir

Similar to the summary files we have already discussed, this summary file contains detailed information on the processes performed during the comparison between the layout netlist for MEMBIDIR and the schematic netlist for MEMBIDIR. These files, referred to as sum.*block*.*block* files, are the main source of debug information for determining problems in the comparison phase of your LVS job. In the summary files that follow, emphasized comments explain the information found in this file.

Example 7-18 sum.MEMBIDIR.membidir

```
+-------------------------------------------------------------+
|              Hercules LVS Comparison Report                 |

|              Schematic cell name = MEMBIDIR                 |
|              Layout cell name    = membidir                 |
+-------------------------------------------------------------+
```

```
COMPARE (R) Hierarchical Layout Vs. Schematic
        Version DATA OMITTED
Copyright (C) Synopsys, Inc. All rights reserved.

+-------------------------------------------------------------+
|                    Comparison Result                        |
+-------------------------------------------------------------+

                   #####    ##    #####  #####
                   #    #  #  #   #      #
                   #####  ######  ####   ####
                   #      #    #      #      #
                   #      #    # #####  #####

                   [MEMBIDIR == membidir]

ERROR SUMMARY

           0 unmatched schematic device
           0 unmatched schematic net
           0 unmatched layout device
           0 unmatched layout net

           14 matched devices
           17 matched nets
Post-Compare Netlist Statistics summarize the number of matching devices
and nets in a cell and indicate which types matched.

Post-Compare summary (* = unmatched devices or nets):

    Matched      Schematic    Layout          Instance types
                 Unmatched    Unmatched       [schematic, layout]
    ---------    ---------    ---------       ---------
    8            0            0               [INVA, inva]
    3            0            0               [N, n]
    1            0            0               [NAND2, nand2]
    1            0            0               [NOR2, nor2]
    1            0            0               [P, p]
    ---------    ---------    ---------
    14           0            0               Total Devices

    17           0            0               Total Nets

+-------------------------------------------------------------+
|                    Detailed Information                       |
+-------------------------------------------------------------+

+-------------------------------------------------------------+
|                    Comparison Information                     |
+-------------------------------------------------------------+

Device properties are compared because option COMPARE_PROPERTIES is TRUE.
```

```
+-------------------------------------------------------------+
|              Cross-Referencing Information                  |
+-------------------------------------------------------------+
```

The Port Cross-Reference table table shows all equated ports in the schematic and layout, as well as their pin class. The table gives you information on how this sub-block will be viewed when its parent is compared. For example, in the case of MEMBIDIR, a device is created with 7 ports that are all unique in class; therefore, none are swappable.

Port Cross-Reference Table:
    S: Schematic generated port
    L: Layout generated port

Generated port	Port class	Schematic port	Layout port
	7	BL	BL
	6	BLB	BLB
	3	DIO	DIO
	4	DO	DO
	2	GND	GND
	5	RD	RD
	1	VDD	VDD

```
+-------------------------------------------------------------+
|                    Statistics Report                        |
+-------------------------------------------------------------+
```

The next set of steps outlines the schematic netlist processing that takes place on the specific block that is running. All merging of devices and filtering of unused devices is done, and then a table of the results is shown. The table also shows whether any of the devices in this block were put there when the PUSH_DOWN_DEVICES preprocessing was done on the layout netlist. In this example, we have 14 total layout devices and 17 layout nets before and after merging. Notice that previously compared blocks, such as INVA, are considered devices. Once a sub-block is compared successfully, the comparison algorithm treats it in the same way as a transistor or other designed device; it is simply a device with pins.

Layout netlist statistics after filtering and merging:

Initial	PushDown	Filter	Parallel	Path/Ser	Final	Device type
8	0	0	0	0	8	inva
3	0	0	0	0	3	n
1	0	0	0	0	1	nand2
1	0	0	0	0	1	nor2
1	0	0	0	0	1	p
14	0	0	0	0	14	Total Devices

```
    Initial  PushDown  Dangle  0 Connect Path/Ser  Final Net type
    -------  --------  ------  --------  --------  ----- -----
    17       0         0       0         0         17    Total Nets
```

The Post-Merge table gives you a side-by-side listing of the schematic and layout device counts and net counts for the cell. In this example, because there were no errors, the device and net count are identical.

```
Post merge netlist statistics: (* = different count):

    Schematic       Layout          Device type [schematic, layout]
    ---------       ------          -------------------------------
    8               8               [INVA, inva]
    3               3               [N, n]
    1               1               [NAND2, nand2]
    1               1               [NOR2, nor2]
    1               1               [P, p]
    ---------       ------
    14              14              Total Devices

    17              17              Total Nets
```

The following information shows the default settings of the COMPARE options, as well as the settings from your runset and equivalence files. If you set an option in the equivalence file that was specific to a particular block, that option setting is shown in the summary file corresponding to that block.

```
+-------------------------------------------------------------+
|                      Compare Options                        |

|      * = Different from Hercules default setting            |
| @ = Different setting between runset and equivalence point  |
+-------------------------------------------------------------+

     == Compare options ==

 add_width_for_parallel_merge   = FALSE
 all_ports_texted               = FALSE
 auto_exclude_equiv             = TRUE
 combine_output_files           = FALSE
 *compare_properties            = TRUE
 delete_lay                     = [none]
 delete_sch                     = [none]
 *detect_permutable_ports       = TRUE
 equate_by_device_name          = FALSE
 *equate_by_net_name            = TRUE
 equiv_by_name                  = FALSE
 *explode_on_error              = TRUE
 *filter                        = TRUE
 find_additional_equivs
     ignore_case                = FALSE
```

```
    name_full                   = FALSE
    name_substring              = FALSE
    name_prefix                 = FALSE
    statistical                 = FALSE
    single_device_cell          = FALSE
    filter_multi_equivs         = FALSE
    full_equiv                  = FALSE
    duplicate_equivs            = FALSE
find_duplicate_equivs           = FALSE
generate_black_box              = [none]
html_output                     = FALSE
match_by_property               = FALSE
matched_ports_continue          = FALSE
memory_array_comparison         = TRUE
merge_all_caps                  = FALSE
merge_all_ind                   = FALSE
merge_all_res                   = FALSE
merge_net_range                 = 100 1000
* merge_parallel                = TRUE
merge_paths                     = FALSE
merge_paths_device_limit        = 0
merge_series                    = FALSE
merge_series_gates              = FALSE
net_print_limit                 = 10
one_connection_warning          = FALSE
optional_pins                   = TRUE
parallel_merge_ratio            = FALSE
port_swap_on_top_block          = FALSE
print_ignore_equiv              = [none]
property_errors_continue        = FALSE
property_tolerance              = 0.1000
property_warning                = FALSE
* push_down_devices             = TRUE
* push_down_pins                = TRUE
* remove_dangling_nets            = TRUE
require_texted_ports_match      = FALSE
require_texted_nets_match       = FALSE
restrict_merge_by_length        = FALSE
restrict_merge_by_width         = FALSE
restrict_merge_series           = FALSE
restrict_parallel_merging       = FALSE
restrict_series_merging         = FALSE
* retain_new_data               = TRUE
retain_previous_data            = FALSE
schematic_vs_schematic          = FALSE
short_equivalent_nodes          = FALSE
static_equated_nets             = TRUE
* stop_on_error                 = FALSE
stop_on_no_explode_error        = FALSE
text_resolves_port_swap         = TRUE
tolerance_device_count          = [none]
tolerance_net_count             = [none]
tolerance_type                  = RELATIVE
```

```
use_total_width                = FALSE
write_netlists                = TRUE
zero_connection_warning        = FALSE

== Evaccess options ==

create_msg_view                = FALSE
create_netlist_view            = FALSE
create_xref_view               = FALSE

== General options ==

* create_vue_output            = TRUE
ignore_case                    = FALSE
layout_global                  = [none]
* layout_ground                = VSSIO GND
* layout_power                 = VDDIO VDD
lvs_report_level               = [none]
lvs_report_old_format          = FALSE
lvs_report_record_limit        = 50
message_enable                 = [none]
message_error                  = [none]
message_identifiers            = FALSE
message_suppress               = [none]
print_precision                 = 4
* schematic_global              = VSSIO VDDIO GND VDD
* schematic_ground              = VSSIO GND
* schematic_power               = VDDIO VDD
```

Notice that the sections containing reports on unmatched devices and nets and matched devices connected to unmatched nets are not included in this sum.*block*.*block* file because there are no errors.

./lvsflow Directory Structure

When the comparison phase of a Hercules LVS job must generate certain files not supplied by the user, the lvsflow is used as a working directory for those files. The lvsflow directory is located in the run_details directory. In the example we are now reviewing, all necessary files and options are specified. The optional files and options include an equivalence file, the EQUATE commands, the SCHEMATIC_GLOBALS, and the COMPARE section of the runset. There are more details about this in Chapter 5 of the *Hercules LVS User Guide*.

If all of the optional information is supplied directly within the runset, the lvsflow directory contains only tree structures of the layout netlist and the schematic netlist in the lay.tree and sch.tree files, respectively. These files also include a flat count of all of the cells in the netlist. Sections of these two files are shown in Example 7-19 and Example 7-20.

Example 7-19 ./run_details/lvsflow/lay.tree - Tree Structure For Layout Netlist

```
DAC96 Level=0 Count=1
   VSSCRPAD Level=1 Count=2
   corelow Level=1 Count=1
```

```
      ram128k Level=2 Count=1
         ram64k Level=3 Count=2
            buf4x Level=4 Count=3
               inva Level=5 Count=5
            buf4x4 Level=4 Count=1
               buf4x Level=5 Count=4
                     inva Level=6 Count=5
            pc_256 Level=4 Count=1
               pc_64 Level=5 Count=2
               pc_8 Level=6 Count=8
                  prechrg Level=7 Count=8
                     prechrg Level=7 Count=8
               pc_64_r_endcell Level=5 Count=1
                     pc_8 Level=6 Count=7
                        prechrg Level=7 Count=8
                     pc_8_r_endcell Level=6 Count=1
                        prechrg Level=7 Count=8
..........................  DATA OMITTED  .......................

            VDDIOPAD Level=2 Count=1
            VSSCRPAD Level=2 Count=1
          OPAD8 Level=1 Count=1
            OPAD Level=2 Count=8
              IOBUF Level=3 Count=1
                inva Level=4 Count=5
                p60 Level=4 Count=8

      CELL STATISTICS

TOP CELL = DAC96
   Ports                = 0

CELL = datbidir
   Flat Placements      = 32

CELL = samp1_a1
   Flat Placements      = 32

CELL = adr3dec8
   Flat Placements      = 12

CELL = cell_no_ovlp
   Flat Placements      = 260096

..........................  DATA OMITTED  .......................

CELL = nand2b
   Flat Placements      = 48

CELL = nand2c
   Flat Placements      = 24

CELL = pc_64_l_endcell
```

```
   Flat Placements      = 4
```

Example 7-20 ./run_details/lvsflow/sch.tree - Tree Structure For Schematic Netlist

```
DAC95 Level=0 Count=1
   IBUF Level=1 Count=88
      IOBUF Level=2 Count=1
         INVA Level=3 Count=5
   OBUF Level=1 Count=56
      IOBUF Level=2 Count=1
         INVA Level=3 Count=5
   DAC_CORE Level=1 Count=1
      CORELOW Level=2 Count=1
         DFFR16 Level=3 Count=6
            DFFR4 Level=4 Count=4
               INVA Level=5 Count=1
..........................  DATA OMITTED  ........................
                     NOR2 Level=7 Count=1
                  OR3 Level=6 Count=1
                     INVA Level=7 Count=1
                     NOR3 Level=7 Count=1
                  NOR2 Level=6 Count=2
         MUX_16 Level=3 Count=3
            MUX_4 Level=4 Count=4
               MUX Level=5 Count=4
                  INVA Level=6 Count=3
                  XFER Level=6 Count=2
               INVA Level=5 Count=2

      CELL STATISTICS

TOP CELL = DAC95
   Ports                = 4

CELL = DFFR4
   Flat Placements      = 68

CELL = XFER
   Flat Placements      = 1312

..........................  DATA OMITTED  ........................

CELL = NOR2
   Flat Placements      = 128

CELL = NOR3
   Flat Placements      = 16

CELL = PC_256
   Flat Placements      = 4
```

Progress Review of Hercules LVS

Thus far we have reviewed the basics of hierarchical LVS in Hercules, including the input and output files. Many concepts and files were presented in this chapter, because it is important to comprehend the hierarchical concepts and know the resources available for deciphering a Hercules LVS job. It is not necessary, however, to memorize all the available options and debug information in all of the files. Most of what was reviewed in this chapter can be automated, including the debug. We describe how to automate Hercules later in the tutorial, but we want you to be aware of the power of the tool, should you want to customize it to take advantage of its many features.

What's Next?

Now that you have reviewed the options and flow of the Hercules LVS tool, you are ready to continue on to Chapter 8, "HLVS Advanced Concepts." You learn how to complete a strict Hercules comparison, including downgrading default error messages and upgrading default warning messages. In Chapter 8 you are also introduced to the Hercules HTML interface for debugging your LVS errors.

8

HLVS Advanced Concepts

In this chapter we discuss, in detail, the concepts used in a strict Hercules LVS comparison, as well as the Hercules LVS advanced options and commands required to make the flow work. We run two examples with a few errors in order to better illustrate the advanced features of Hercules that are used in this flow.

Learning Objectives for This Chapter

- To learn a strict LVS flow, where all inputs are defined, port text matching is required, and all equivalence points must match
- To run an example of a strict LVS flow with errors
- To learn the Hercules-VUE GUI for debugging the LVS results
- To become acquainted with the standard flow for debugging an LVS job

Before You Start

Before you start this tutorial, make sure that you have completely gone through the installation and setup procedures described in Chapter 1, “Installation and Setup.”

General Requirements of a Strict LVS Flow Comparison

Strict LVS flow comparison requires that all equivalence points match, so building the equivalence file takes extra care. Remember that only a very small percentage of Hercules users require an LVS flow as strict as the one described here. The example in this chapter is meant to teach you some of the more complex options available with the Hercules LVS tool. The next chapter is a better example of a typical mainstream LVS flow. Below is a list of the general requirements that must be followed to generate a successful comparison in our strict LVS flow example.

- Compare the top-block layout ports with the top-block schematic ports
- Require that all ports be texted and that the layout and schematic port text matches in all blocks
- Resolve all symmetrical ports with text, so Hercules does not need to resolve the symmetries
- Match all equivalence points
- Extract and netlist all devices in the cell that contains the device layer, for example the GATE layer of a MOSFET

Before we execute our Hercules job, we explain how to achieve the requirements listed above, using excerpts from a Hercules runset as our examples. That Hercules runset, dac96lvs2.ev, is also be used for our first example in this chapter.

Comparing Top-Block Ports

By default, no ports are netlisted in the top block of a Hercules layout netlist. You must identify for Hercules the polygons that represent the pins, or top-block ports. Hercules has a command, CREATE_PORTS, that is used to designate these ports, especially for the top block, so that the ports are netlisted. The COMPARE process can then check that ports in the layout netlist match the schematic netlist ports.

To designate these ports, define a marker layer in your design. In our design this layer is PAD. The marker layer, or PAD layer, is processed against a connected layer, and the interaction between the marker polygon and the connect layer polygon creates a new port. We use the PAD, layer 15, as the marker layer and the connected layer. In the following example, you notice that PAD is listed in our CONNECT command, so it qualifies as a connected layer.

Finally, in order to allow matching of layout port names with the schematic port names, we also make sure that all of the PAD polygons are texted to match the schematic port text.

Example 8-1 shows the command from the dac96lvs2.ev runset we use to generate our ports for comparison.

Note:
The PAD layer is not located in the top block, DAC96, so we must add the FLATTEN command to flatten all of the PAD polygons to the top block.

Example 8-1 CREATE_PORTS Syntax from Hercules Runset Example dac96lvs2.ev

```
FLATTEN PAD {cell_list = {*}} temp = PADTOP

/* -------------------------------- Define Connectivity Rules */
CONNECT {
   ngate pgate BY field_poly
   M2 BY PADTOP
   M1 M2 BY V1
   M1 ndiffdio res_term field_poly nsd psd welltie subtie BY CONT
   NWELL by welltie
   SUBSTRATE by subtie

}

/* -------------------------------- Define Layout Netlist Ports */

CREATE_PORTS {
              top_cell_only = TRUE
              PADTOP BY PADTOP
             }

/* -------------------------------- Define Text Attachment Rules */
TEXT {
   M1 BY M1
   M2 BY M2
   field_poly BY POLY
   PADTOP BY PAD
   NWELL BY "VDD"
   SUBSTRATE BY "GND"
}
```

Requiring Ports to Match by Name

Once you have generated layout and schematic ports in the top block, you must tell Hercules that these ports need to match by name. The Hercules comparison algorithm has an option, REQUIRE_TEXTED_PORTS_MATCH, which you can set to TRUE. This option is valid in the COMPARE section of the Hercules runset, and you can also set it for a specific block (cell) in the equivalence file. In the strictest methodologies this option is set in the COMPARE section and applies to all texted ports in the design.

In our example we first set this option for only the top block by placing it in the equivalence file. This also serves as an example of setting an option for a specific block instead of globally. We then do a second Hercules run where we set the option globally in the COMPARE section of the runset.

The syntax of an equivalence file is:

```
EQUIV schematic_name=layout_name {option=true/false option=true/false ...
}
```

Requiring All Ports to be Texted

The final step to matching all schematic and layout ports is to tell Hercules that all ports are required to be texted. The Hercules comparison algorithm has an option, ALL_PORTS_TEXTED, which you can set to TRUE to generate a warning or error if all ports are not texted. Similar to the REQUIRE_TEXTED_PORTS_MATCH command, in the strictest methodologies this option is set in the COMPARE section and applies to all texted ports in the design.

In our example we first set this option for only the top block by placing it in the equivalence file. We then do a second Hercules run where we set the option globally in the COMPARE section of the runset.

Example 8-2 shows a sample of the equivalence file used in our first example of the chapter, dac96.eqv2. It contains the options requiring that all ports be texted in the top block, and that all of the texted ports match by name.

Example 8-2 Equivalence File for dac96.eqv2

```
EQUIV DAC95=DAC96 {
   require_texted_ports_match = true
   all_ports_texted = true}
EQUIV COREHI=corehi {}
EQUIV CORELOW=corelow {}
EQUIV CS_ADD1=cs_add {}
EQUIV IBUF=IPAD {}
EQUIV ADD4=add4 {}
EQUIV OBUF=OPAD {}

..... DATA OMITTED .....

EQUIV BUF4X4=buf4x4 {}
EQUIV ADR3DEC8=adr3dec8 {}
EQUIV DFFR4=dffr4 {}
EQUIV SAMP4=samp4 {}
EQUIV ADD4B=add4_ovlp {}
EQUIV ADD4B=add4_ga {}
EQUIV MUX_16=mux16 {}
EQUIV PC_256=pc_256 {}
EQUIV BUF4X8=buf4x8 {}
```

```
EQUIV MEDDEC_8=meddec8 {}
EQUIV DOUT1=dout1 {}
EQUIV DFFR16=dffr16 {}
EQUIV WDEC_256=wdec256 {}
EQUIV DOUT4=dout4 {}
EQUIV DOUT8=dout8 {}
EQUIV RAM64K=ram64k {}
EQUIV RAM128K=ram128k {}
```

Symmetry: Independent and Dependent Swappability

In Chapter 7, “Introduction to Hercules HLVS,” we briefly discussed symmetrical nets. In a strict LVS flow, the designer requires that no ports can be swapped. As a result, the DETECT_PERMUTABLE_PORTS option in the COMPARE section is set to FALSE. This guarantees that even the strictest timing constraints are upheld. For example, in a large DRAM block many of the control pins are swappable. Timing of these control signals is often critical; they might be hooked up in the schematic to reflect the tightest timing constraints. Prohibiting port-swapping guarantees not only the logic, but the critical timing connections on these swappable ports as well. Prohibition of port-swapping occurs only in the strictest custom design environments.

Guaranteeing Equivalence Point Matching

As described in Chapter 7, “Introduction to Hercules HLVS,” the equivalence file is an optional input file. In many flows you do not specify this file, but instead allow Hercules to generate it automatically. In these automatic flows, the equivalence file is used to generate as much hierarchy as possible for easy debug. In stricter LVS flows, however, the designers require that certain blocks of a design match, and guarantee they match in the final design when they are used as equivalence points. EXPLODE_ON_ERROR is set to FALSE to make sure that if the blocks do not match, the comparison fails.

Various reasons exist for the requirement that certain blocks match in a final LVS run. We discuss two of them.

Reuse of IP Blocks and Macros

Many design methodologies require that you be able to reuse IP blocks, or smaller macros, and suggest that you guarantee that these be maintained as equivalence points. This is so that both the schematic and layout blocks can be reused without generating new timing models or other post-layout data. In designs with very tight timing or a lot of custom design areas, it is often easier for a designer to add an engineering change order by grabbing whatever gates are closest to the edit location and not actually including them inside the macro that was changed in the schematic. This results in a macro in the schematic that matches to a macro *plus* some extra logic in the layout, making it difficult to reuse the layout

of the macro in another design. It also makes it impossible to build a reusable timing model for the macro, because there is no equivalence point between the layout and schematic, and therefore no common cell that could be replaced with a timing model.

Debugging Large Designs

A very common and important reason for requiring that all equivalence points match is to guarantee the fastest debugging, or turn-around time, for design changes. The runtime of Hercules can be greatly improved by using a good equivalence file. Below is a list of guidelines to follow when creating the methodology for generating your equivalence file. If the schematic and layout designers follow this methodology, then, as changes are made to a design at the end of the design cycle, the Hercules LVS job runs very quickly and it is easy to debug any problems that occur because of the changes.

Guidelines for a Good Equivalence File

- All library cells should be equivalence points.
- All IP blocks or macros should be equivalence points.
- In most cases, function blocks should not be equivalence points. For example, if you have the hierarchy ALU (macro), ADDER32 (function), ADDER4 (library), you should include ALU and ADDER4 as equivalence points, but exclude ADDER32.
- In a memory (such as RAM, ROM, DRAM, FLASH) include either a bit cell or 8/9 bit cell, an intermediate block (preferably one that is texted correctly), and the top block. For example, RAM512K, RAM32K, and RAM8bit.

As stated above, following these guidelines serves two purposes:

- Hercules LVS optimally processes these types of cells and hierarchy
- Such an equivalence file has a good set of debug points, not too many devices in each block, and not too many actual blocks to debug.

The two extremes of less-than-optimal performance are:

- *Too many equivalence points*—In this scenario Hercules spends extra time processing each equivalence point and the hierarchical interactions between them. Also, you are overwhelmed with all of the debug points.
- *Not enough equivalence points*—In this scenario Hercules spends too much time processing large blocks of transistors, instead of processing smaller blocks and then simply looking at their hierarchical interactions. Also, you end up debugging blocks with tens of thousands of transistors.

Of course, in some cases you might want to add more equivalence points than is optimal, so as to help reduce your debugging time. Adding a few minutes to an LVS run is minor compared to saving hours of debug time, because you have reduced the size and scope of the debug problem.

Guaranteeing Devices Are Netlisted in the Cell Where Designed

The concept of requiring equivalence points to match is considered a more advanced feature of Hercules, and for this reason we present it in this chapter of the guide. This requirement is almost always part of a strict Hercules comparison, but it is also used in any Hercules LVS when the output is going into StarRC, and this includes many less strict LVS flows.

The primary reason users require that all devices be netlisted in the cell where they are designed is that they are using the results of a Hercules LVS job as input for a StarRC extraction job and they need all of their equivalence points to match. StarRC allows you to input a library of timing models for each of the library cells, or macro blocks, in the equivalence file. If, for some reason, an equivalence point fails although the entire design compares at the top, you no longer have this block, or equivalence point, as a reference for inputting the timing model.

When requiring blocks to compare for a Hercules-to-StarRC flow, you generally do not care why the equivalence point failed. You might, for example, not have restrictions on devices being designed across the hierarchy, nor have restrictions on port text matching, such as might exist in a strict Hercules comparison. In such a case you simply need to make sure that the hierarchy of equivalence points matches the library and macro cells that have generated timing models. In most cases, these equivalence points are lost due to hierarchical interaction with devices that cause them to be generated one level above the cell in which they were originally designed.

To help guarantee that devices are placed in the original cells in which they were designed, even in designs where there is a lot of hierarchical interaction between devices, Hercules has two command options, MOS_REFERENCE_LAYER for the extraction phase of LVS, and PUSH_DOWN_DEVICES for the comparison stage.

MOS_REFERENCE_LAYER

In the extraction stage MOS_REFERENCE_LAYER is the option for the PMOS and NMOS extraction commands. A similar option exists for each designed device type. (For example, there is also a RES_REFERENCE_LAYER command.) Even though there might be hierarchical interaction for some of the device layers, such as source drain overlap, the MOS_REFERENCE_LAYER option tells the Hercules extraction code to place the MOSFET in the cell that has this reference layer. The problem of having the hierarchical interactions move the device into a parent cell is thus avoided, as is the problem of the Hercules code being too aggressive and pushing the device into a child cell.

Note:

Having the device in the parent of the intended cell, or a child of the intended cell, happens only if this is a logically and physically equivalent scenario. Hercules does not change the actual design, just the format of the layout netlist hierarchy.

Figure 8-1 and Figure 8-2 illustrate a situation in which Hercules puts the device into a child cell (Cell B). Adding the MOS_REFERENCE_LAYER option to the PMOS and NMOS commands corrects the problem.

Figure 8-1 Effect of Placing Device in Child Cell: Schematic

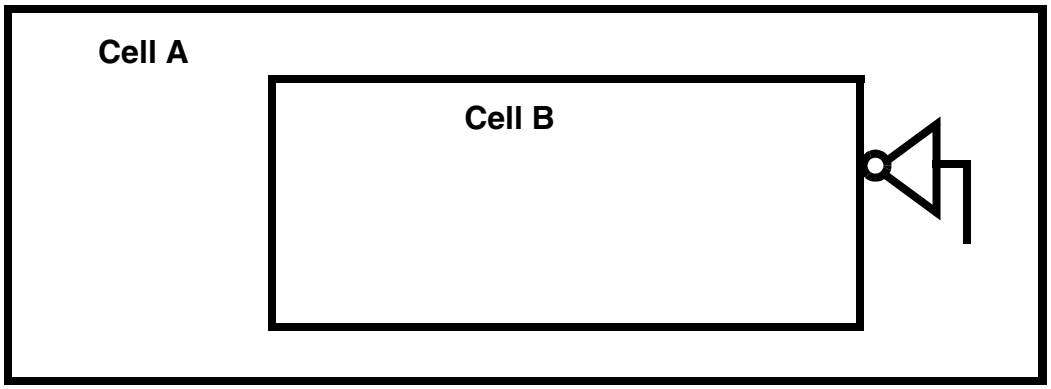
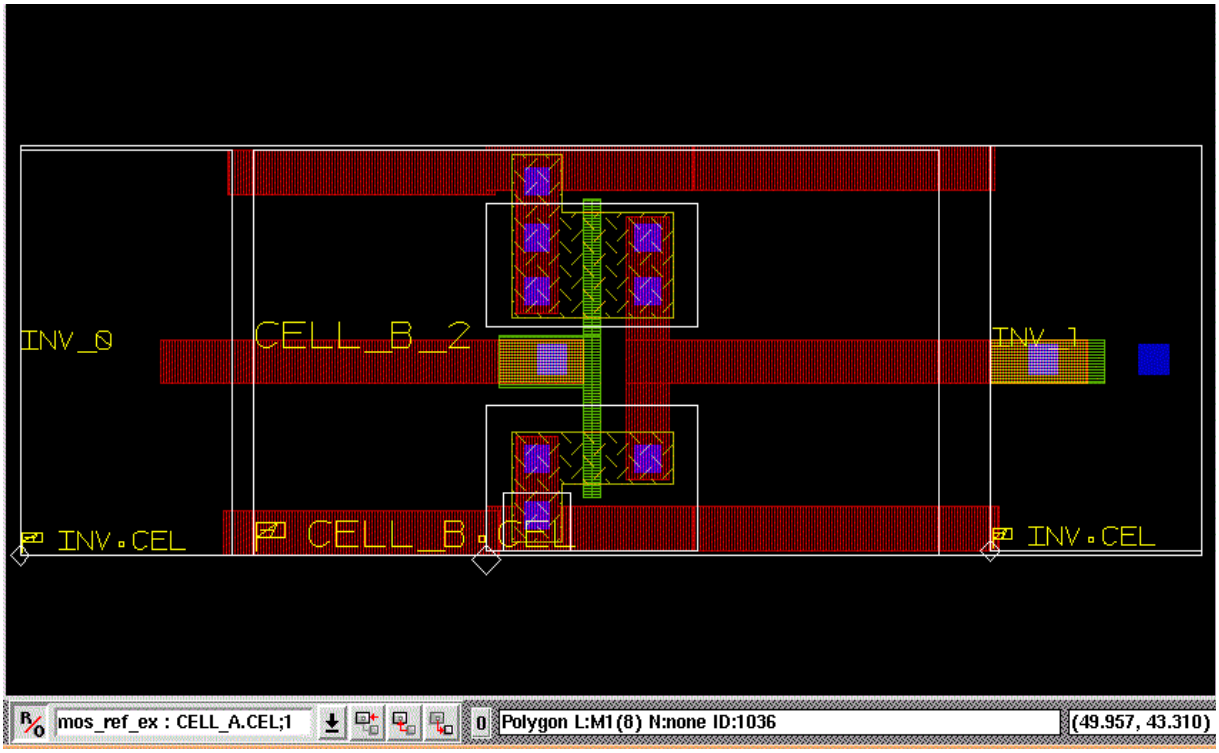


Figure 8-2 Effect of Placing Device in Child Cell: Layout Viewed with Enterprise



The child (Cell B) compares alone. When the parent (Cell A) is run, the INVERTER from the parent gets pushed down into the child, so the child does not compare. Unlike the schematic netlist, the layout netlist shows INV as flat (that is, not an instance). Example 8-3 shows the schematic netlist, Cell_A.sch, for Cell A. Notice that CELL_B, in the schematic, has two instances of INV, and CELL_A has a flat inverter, made from an instance of a p device and an instance of an n device.

Example 8-3 Schematic Netlist for Cell A

```
{NETLIST CELL_A
{VERSION 1 0 0}
/* Level 2 */
{CELL INV
   {PORT 1 2 3 9}
   {INST M1=p {PROP x=12.5 y=32.5 l=2.000 w=12.000}
      {PIN 2=GATE 3=SRC 1=DRN 1=BULK }}
   {INST M2=n {PROP x=12.5 y=10.5 l=2.000 w=6.000}
      {PIN 2=GATE 3=SRC 9=DRN 9=BULK }}
}
/* Level 1 */
{CELL CELL_B
   {PORT 1 4 5 6 7}
   {INST 11=INV {PROP x=-27.5 y=0.5 angle=0 reflection=0}
      {PIN 1=1 8=2 7=3 4=9}}
```

```
   {INST 12=INV {PROP x=28.5 y=0.5 angle=0 reflection=0}
      {PIN 1=1 9=2 6=3 5=9}}
}

/* Level 0 */

{CELL CELL_A
   {INST M1=p {PROP x=22 y=33 l=2.000 w=12.000}
      {PIN 2=GATE 3=SRC 1=DRN 1=BULK }}
   {INST M2=n {PROP x=22 y=11 l=2.000 w=6.000}
      {PIN 2=GATE 3=SRC 7=DRN 7=BULK }}
   {INST 0=INV {PROP x=-45.5 y=0 angle=0 reflection=0}
      {PIN 1=1 13=2 2=3 7=9}}
   {INST 1=INV {PROP x=69 y=0.5 angle=0 reflection=0}
      {PIN 1=1 3=2 6=3 7=9}}
   {INST 2=CELL_B {PROP x=9.5 y=-0.5 angle=0 reflection=0}
      {PIN 1=1 7=4 7=5 3=6 2=7}}
}
}
```

The CELL_A.net layout netlist (shown in Example 8-4) reflects the problem of extracting and netlisting the MOSFETs that make up the INVERTER. We ran Hercules using the mos_ref_err.ev runset to generate the CELL_A.net layout netlist below. Notice that there are two instances of INV in CELL_B, just like in the schematic, but that there is also a flat inverter, made from one instance of a p device and one instance of an n device. Also notice that CELL_A is missing the flat inverter that was in CELL_A in the schematic.

Example 8-4 Cell_A.net Layout Netlist

```
{NETLIST CELL_A
{VERSION 1 0 0}

/* Level 2 */

{CELL INV
   {PORT 1 2 3 9}
   {INST M1=p {PROP x=12.5 y=32.5 l=2.000 w=12.000}
      {PIN 2=GATE 3=SRC 1=DRN 1=BULK }}
   {INST M2=n {PROP x=12.5 y=10.5 l=2.000 w=6.000}
      {PIN 2=GATE 3=SRC 9=DRN 9=BULK }}
}

/* Level 1 */

{CELL CELL_B
   {PORT 1 2 3 6 7 8 9 10 11 12 13 14}
   {INST M1=p {PROP x=12.5 y=33.5 l=2.000 w=12.000}
      {PIN 2=GATE 12=SRC 13=DRN 1=BULK }}
   {INST M2=n {PROP x=12.5 y=11.5 l=2.000 w=6.000}
      {PIN 3=GATE 10=SRC 11=DRN 14=BULK }}
   {INST 11=INV {PROP x=-27.5 y=0.5 angle=0 reflection=0}
      {PIN 1=1 15=2 9=3 6=9}}
```

```
   {INST 12=INV {PROP x=28.5 y=0.5 angle=0 reflection=0}
      {PIN 1=1 16=2 8=3 7=9}}
}

/* Level 0 */

{CELL CELL_A
   {PORT}
   {INST 0=INV {PROP x=-45.5 y=0 angle=0 reflection=0}
      {PIN 1=1 13=2 2=3 7=9}}
   {INST 1=INV {PROP x=69 y=0.5 angle=0 reflection=0}
      {PIN 1=1 3=2 6=3 7=9}}
   {INST 2=CELL_B {PROP x=9.5 y=-0.5 angle=0 reflection=0}
      {PIN 1=1 2=2 2=3 7=6 7=7 3=8 2=9 3=10 7=11 3=12 1=13 7=14}}
}
}
```

Include MOS_REFERENCE_LAYER variables in the NMOS and PMOS commands in the runset, as in the following:

```
PMOS p pgate psd psd substrate
{mos_reference_layer = "poly";} TEMP=pgates
NMOS n ngate nsd nsd well
{mos_reference_layer = "poly";}  TEMP=ngates
```

As an exercise to see varying results, run Hercules on mos_ref_err.ev to generate the miscompare of Cell B. Cell A, however, does match. The mos_ref_fixed.ev runset generates the correct compare of Cell B, with Cell A still matching. You find these runsets under the Getting_Started_Hercules_LVS directory.

PUSH_DOWN_DEVICES

In the comparison stage, PUSH_DOWN_DEVICES is the option used to guarantee equivalence points. It is added to the COMPARE section of your runset and set to TRUE. PUSH_DOWN_DEVICES is the initial recommended solution to the problem of hierarchical interactions causing equivalence points to miscompare. PUSH_DOWN_DEVICES identifies all devices that meet the following two criteria:

1. All of the device terminals are directly connected to ports of one cell.
2. The connectivity of these ports is consistent across all instances of that cell.

If these two criteria are met, the Hercules comparison removes those devices from the higher level and places them in the cell to which the ports are connected. There is a detailed description entitled PUSH_DOWN_DEVICES in Chapter 7, “Introduction to Hercules HLVS.”

Setting up Error and Warning Messages

Warning and error messages are generated during both the extraction and comparison phases of a Hercules LVS job. All LVS extraction ERRORs appear in the *block*.LAYOUT_ERRORS file. The most common of these errors include text opens, text shorts, and device extraction errors. The comparison phase of LVS runs even if there are errors in the *block*.LAYOUT_ERRORS file, so you must make sure this file is okay before you begin to debug the comparison phase of your LVS run.

All comparison error and warning messages appear in the sum.*block*.*block* files and the *block*.cmperr file. Remember, the primary files for reviewing these errors and warnings are in the sum.*block*.*block* files of each block. Hercules has default settings for what it reports as errors and warnings. A table of these defaults is located in the *Hercules Reference Manual*.

Hercules also allows you to change what is reported as an error or a warning. In certain cases you might wish to reduce the size of output files by switching off particular types of messages. Normally, these messages are warnings that are understood by the user, or data that is not generally referenced. In other instances, you might want to turn on other messages to produce more verbose output.

Note:

A message might be more than one line long, particularly for tables that list data unique to a particular block.

The MESSAGE_SUPPRESS command in the OPTIONS section can be used to disable the printing of certain message types in COMPARE. The MESSAGE_ENABLE command in the OPTIONS section is used to enable the printing of certain other messages. The MESSAGE_IDENTIFIERS option is set to display the message identifiers in the listing file. The MESSAGE_ERROR option is set to upgrade a message to a severity level of ERROR.

Example 8-5

```
OPTIONS {
   message_suppress = { CMP-39 CMP-45 } /* Do not print these messages.
*/
   message_enable = { CMP-60, CMP-61 } /* Do print these messages */
   message_identifiers = on /* Prints message IDs in output files. */
   message_error = { CMP-52 } /* Upgrade message(s) level to error. */
}
```

In our first tutorial example you see a sample sum.*block*.*block* file, which includes the MESSAGE_IDENTIFIERS option. These are CMP-XX numbers beside each information output in the file and are controllable with the options listed above.

Hercules Examples

Now that we have explained the different options we set in our examples, here is a brief summary of each example in this chapter and the learning objectives for each example.

Example 1: Run Hercules on a strict comparison runset, where CREATE_PORTS is used in the device extraction and layout netlisting. All top ports are required to be texted and the text must match between the layout and schematic. We run this example twice, once with the standard warning message, and the second time with the CMP-XX error numbers displayed and the warning message upgraded to an error.

Learning Objectives:

- To become familiar with setting options in the equivalence file, as opposed to the COMPARE section
- To practice using Hercules-VUE to search through sum.*block*.*block* files, and the *block*.LVS_ERRORS file
- To see an example of strict port comparison

Example 2: Run Hercules on a strict comparison runset similar to the first example, but set the options in the runset during the COMPARE section. Also, a new warning is generated because a layout port net is equated to a net that is not a port in the schematic.

Learning Objective:

- To review some of the CMP-XX error numbers relating to port matching

Running Hercules LVS for Example 1

Go to the directory where your dac96lvs2.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs2/. Enter the command:

hercules dac96lvs2.ev

Your active window displays the execution process. While the screen contents might scroll quickly (freeze the screen with Control-s and restart with Control-q), you should be able to notice any specific actions or warnings that appear. All of this information appears in a file for your examination. The sample runset file should only take a few minutes to run.

Runfile Results for Example 1

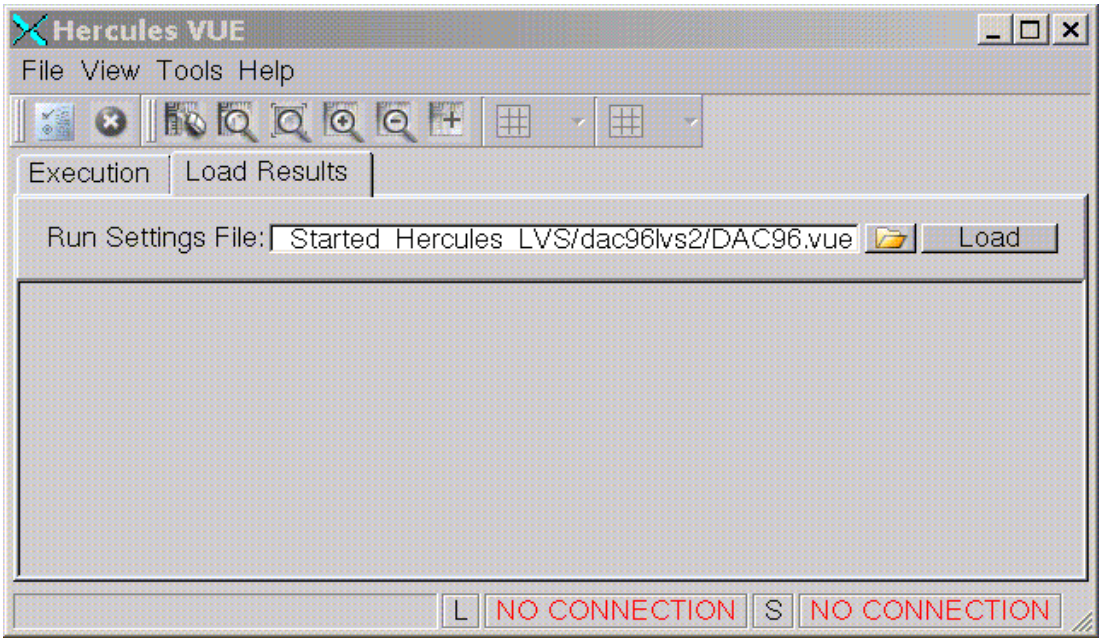
Notice that this runset is the same as the one in Chapter 7, except for the addition of the options described above.

Once Hercules run is completed, enter the command:

vue &

Since we have already executed Hercules, we now load the results in VUE. Select the Load Results tab as shown in Figure 8-3.

Figure 8-3 Loading Results in VUE

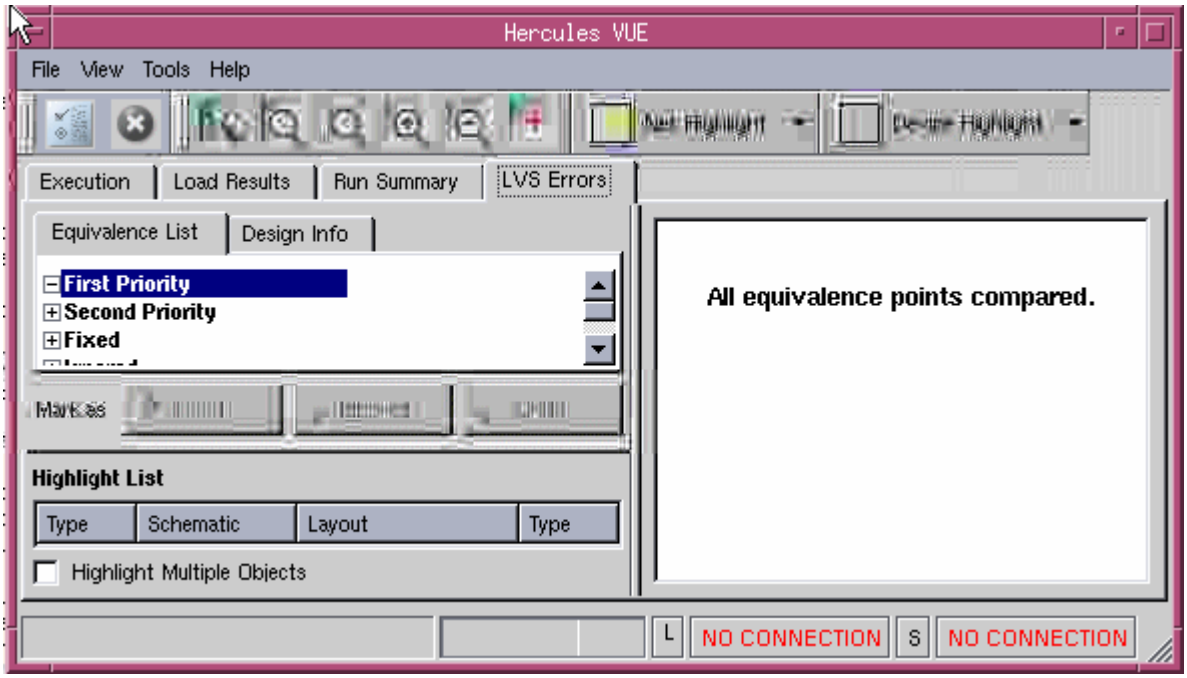


Verify that you have the *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs2/DAC96.vue file as a Run Settings File. Click Load.

A message in the Run Summary tab indicates that your top block compares clean. The LVS Extraction phase of the job ran without errors. Select the LVS Errors tab. As seen in Figure 8-4, there are no first priority equivalence errors.

To view the DAC96_lvs.log file, click Design Info.

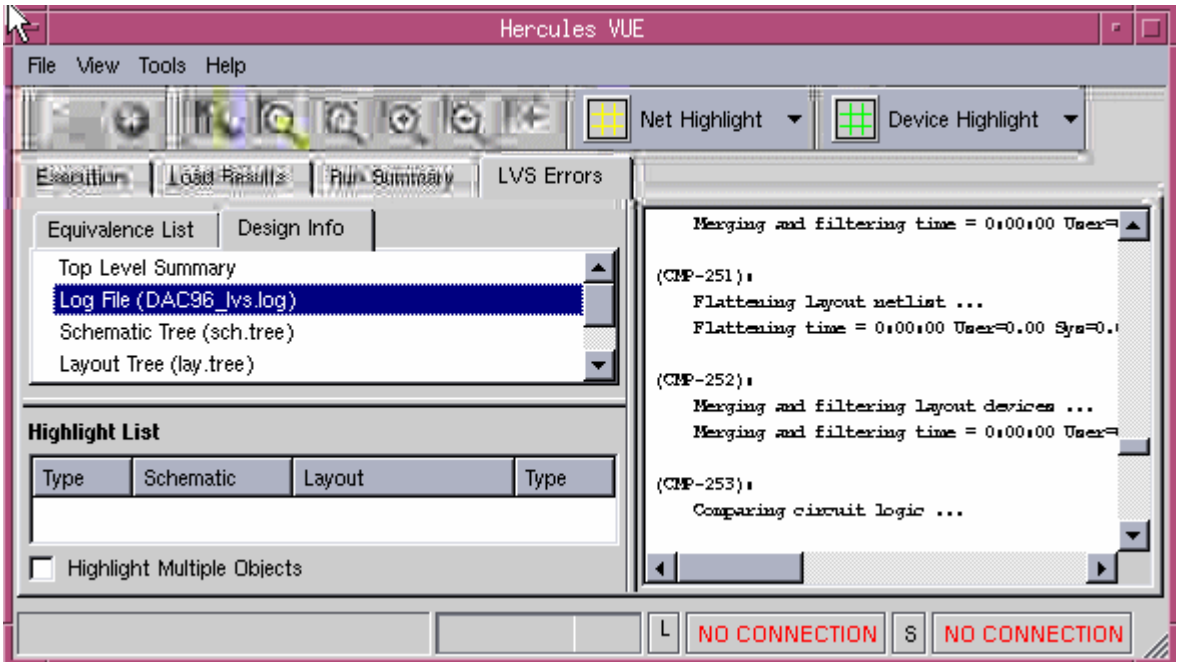
Figure 8-4 LVS Errors Tab: Viewing the Design Information



Note:
Remember, you can resize any of the panes by dragging the frame edge with the mouse.

Click Log File (block_lvs.log), as shown in Figure 8-5 to load the DAC96_lvs.log file. Scroll through the file until you get near the bottom.

Figure 8-5 Bottom Part of DAC96_lvs.log File



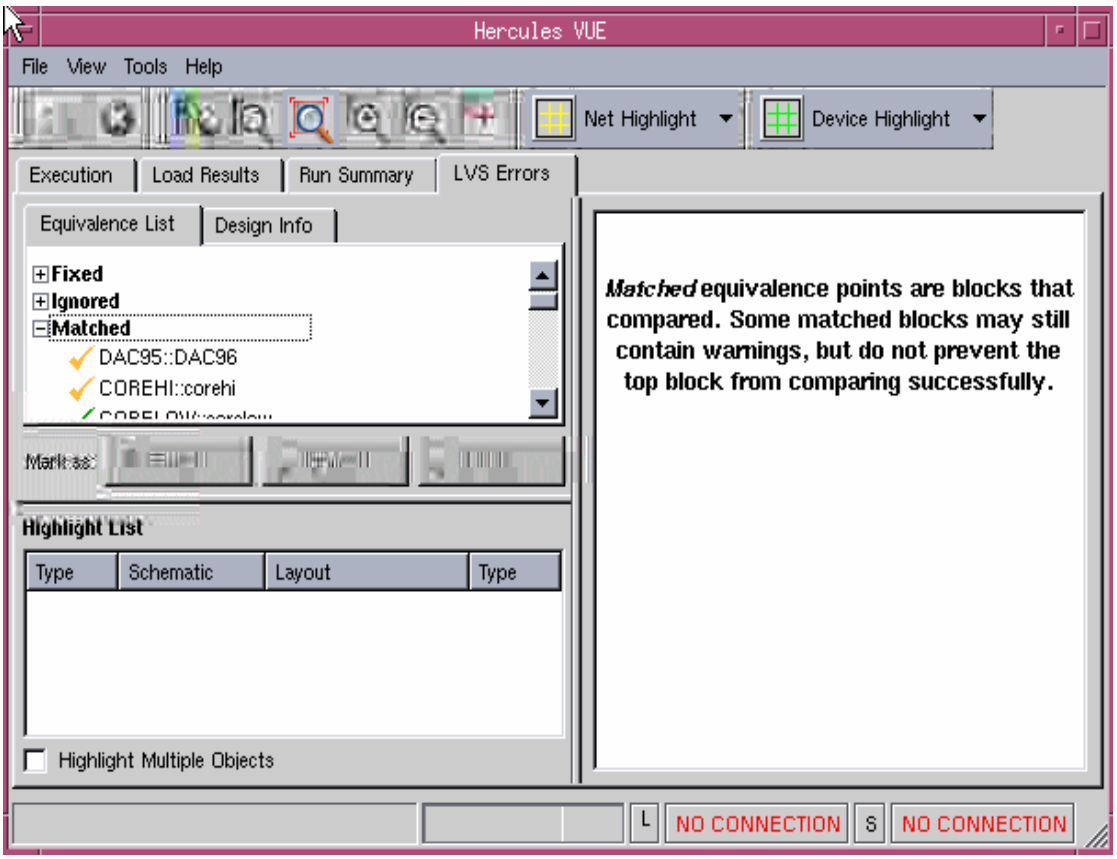
The CMP-XX messages are numbered, so you have the option of customizing the looks of this file by suppressing the printing information for each. Notice the WARNING messages and the links to each of the blocks in the file.

Note:

There are categories of warnings listed here for COREHI. Look at the summary of the COREHI block file after reviewing DAC96.

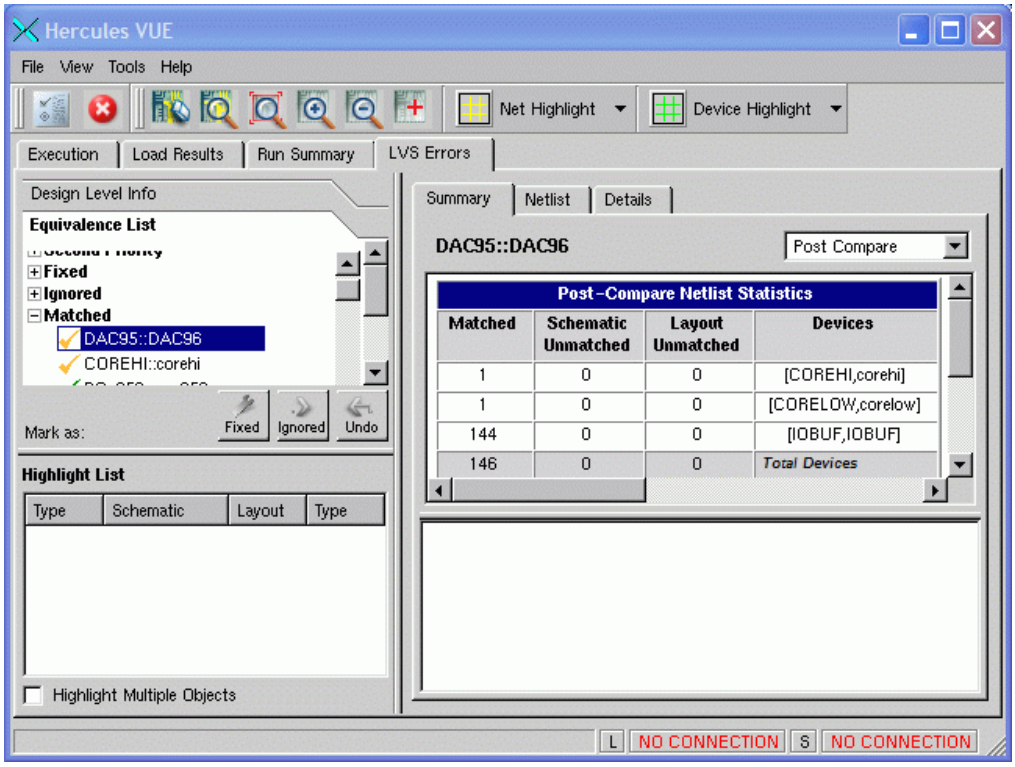
To view the summary of DAC96, click Equivalence List. Expand the Matched equivalence list by clicking on the + sign, as shown in Figure 8-6.

Figure 8-6 Summary of the Matched Block



Now select the DAC96 file from the list in the left frame. You should see a view similar to that shown in Figure 8-7.

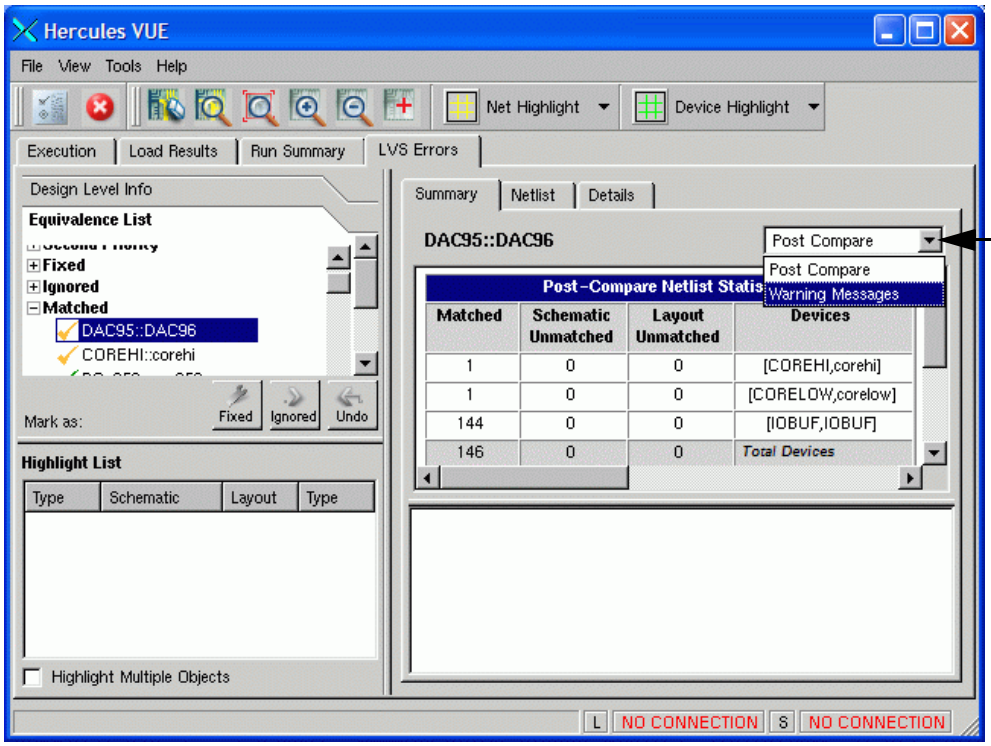
Figure 8-7 Summary for Top Block DAC96



You are now viewing the summary of the top block, DAC96. This is the block where we set the ALL_PORTS_TEXTED and REQUIRE_TEXTED_PORTS_MATCH options. As a result of those settings, we expect to see WARNING messages because that is the default type of output message if there are problems introduced with these option requirements.

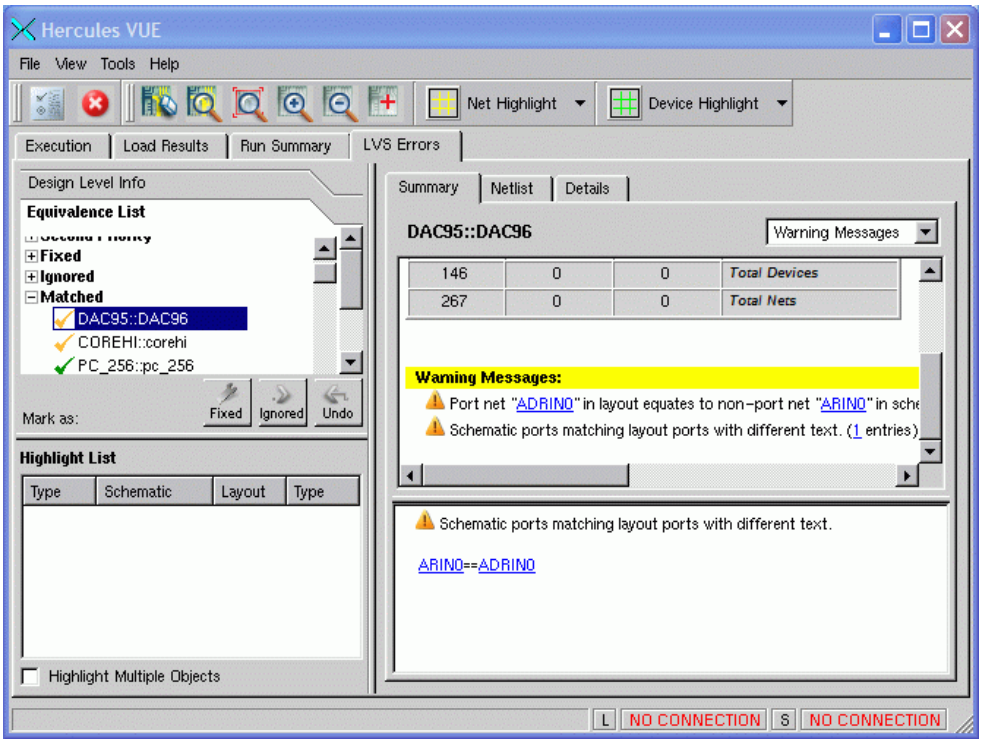
Click the pull-down button of the Summary tab (see the arrow in Figure 8-8) to load the warning messages. (Remember, you can resize the frames.)

Figure 8-8 DAC96 Warning Messages



Click the “Schematic ports matching layout ports with different text” warning message to load the details in the bottom panel (see Figure 8-9).

Figure 8-9 Viewing the Details of the WARNING Messages

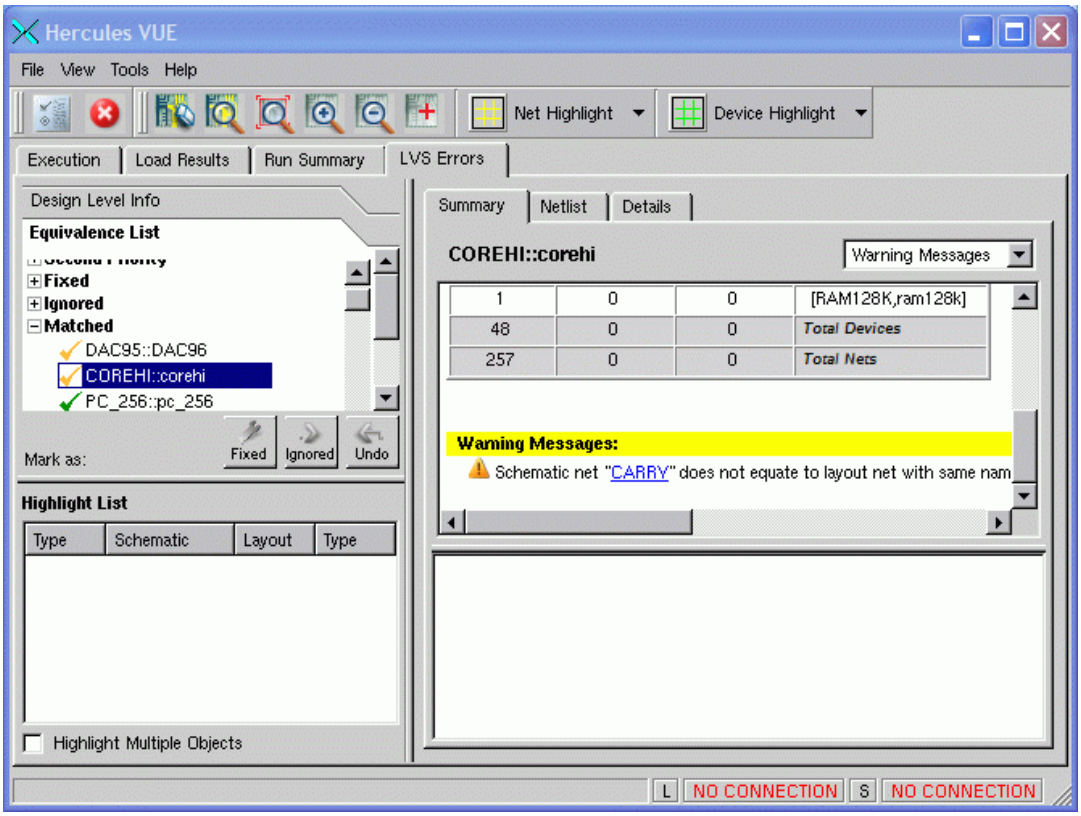


You can now upgrade the WARNING message to an ERROR. You are now going to learn to suppress warning messages before you rerun Hercules.

Select COREHI, the block having warning messages as pointed out in the note following the discussion of the DAC96_lvs.log file.

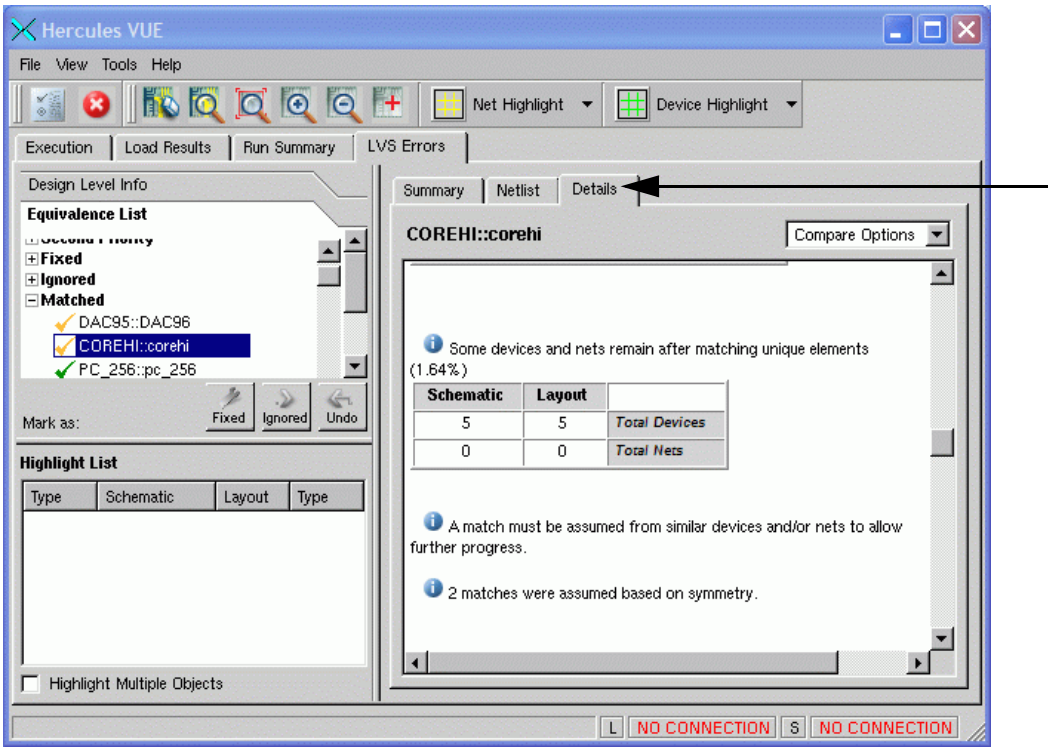
Scroll down to the warning messages section. The warning for schematic net CARRY does not equate to the layout net with the same name that is listed here (see Figure 8-10). VUE considers this a warning message that most people want to pay attention to.

Figure 8-10 Warning Messages of Block COREHI



Remaining warning messages are listed in the Details tab. Click on the Details tab. In this window, details of the sum.*block*.*block* file are loaded. Scroll down to the warning messages near the end (see Figure 8-11).

Figure 8-11 COREHI Details with Two Warnings

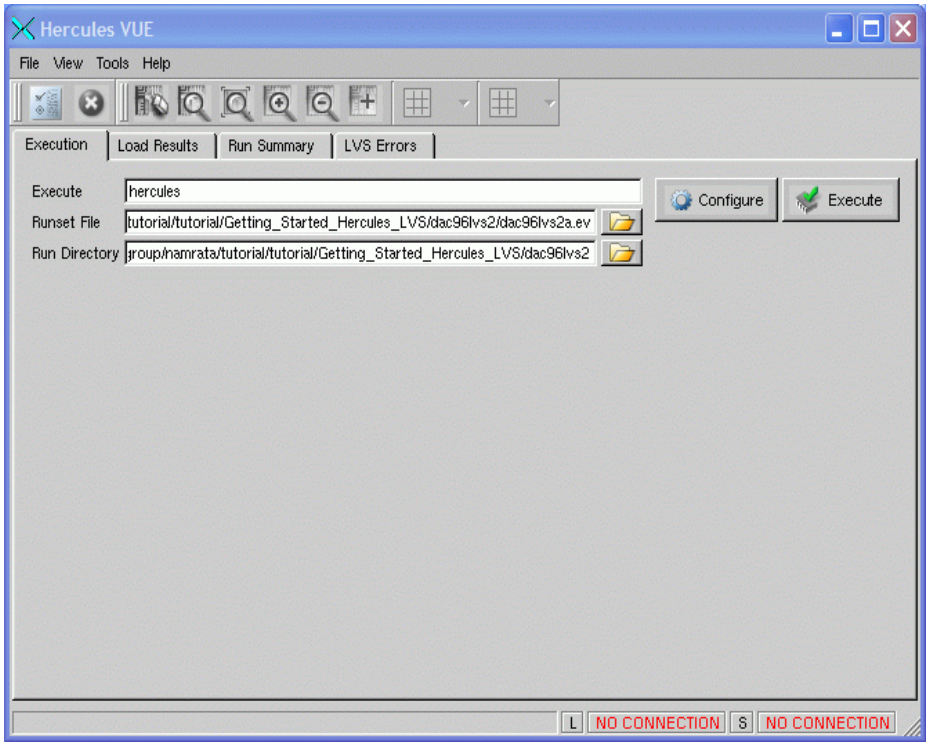


These warnings are generally okay, therefore we are going to suppress them in our example.

Edit the dac96lvs2.ev file to include message_suppress = {CMP-8 CMP-27} and message_error = {CMP-119}. Or, you can use the dac96lvs2a.ev runset that already has the changes.

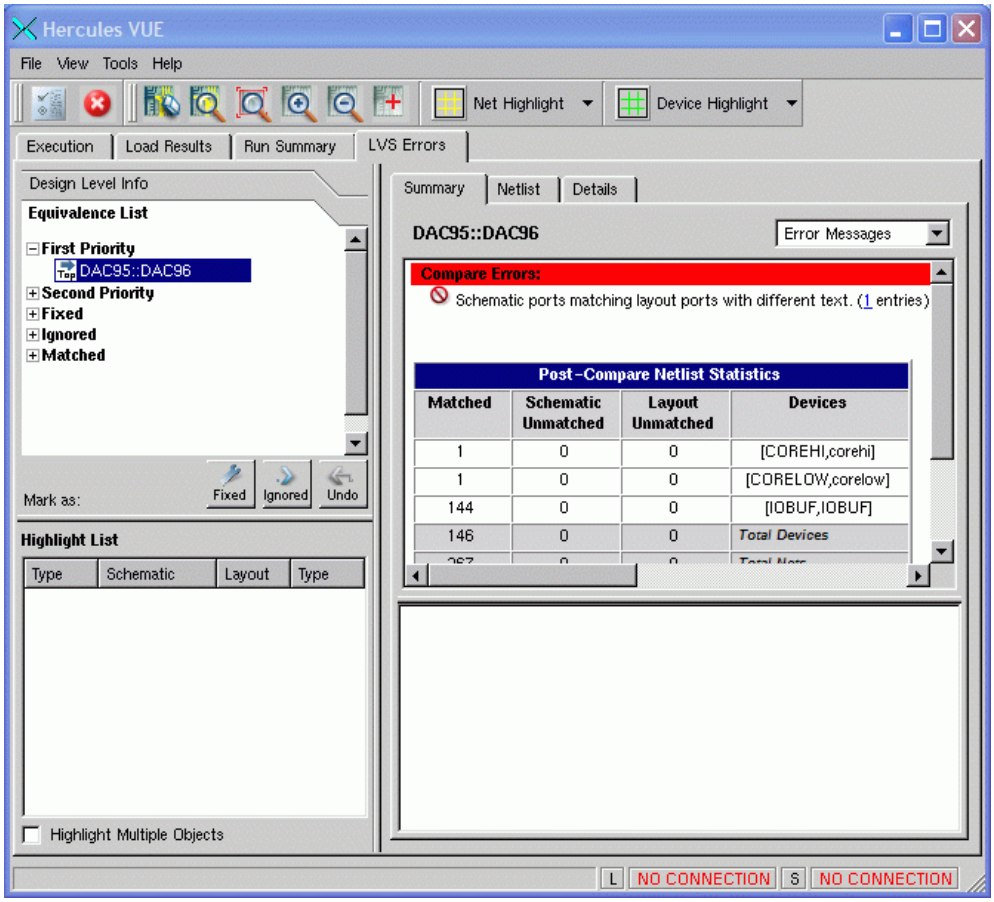
To execute Hercules from VUE, select the Execution tab and select runset file dac96lvs2a.ev, as shown in Figure 8-12. Click Execute.

Figure 8-12 Executing Hercules



Once the Hercules run is completed, data is automatically loaded in VUE and the Run Summary tab shows an unsuccessful compare. View errors by selecting the first priority block DAC96 in LVS Errors window, as shown in Figure 8-13.

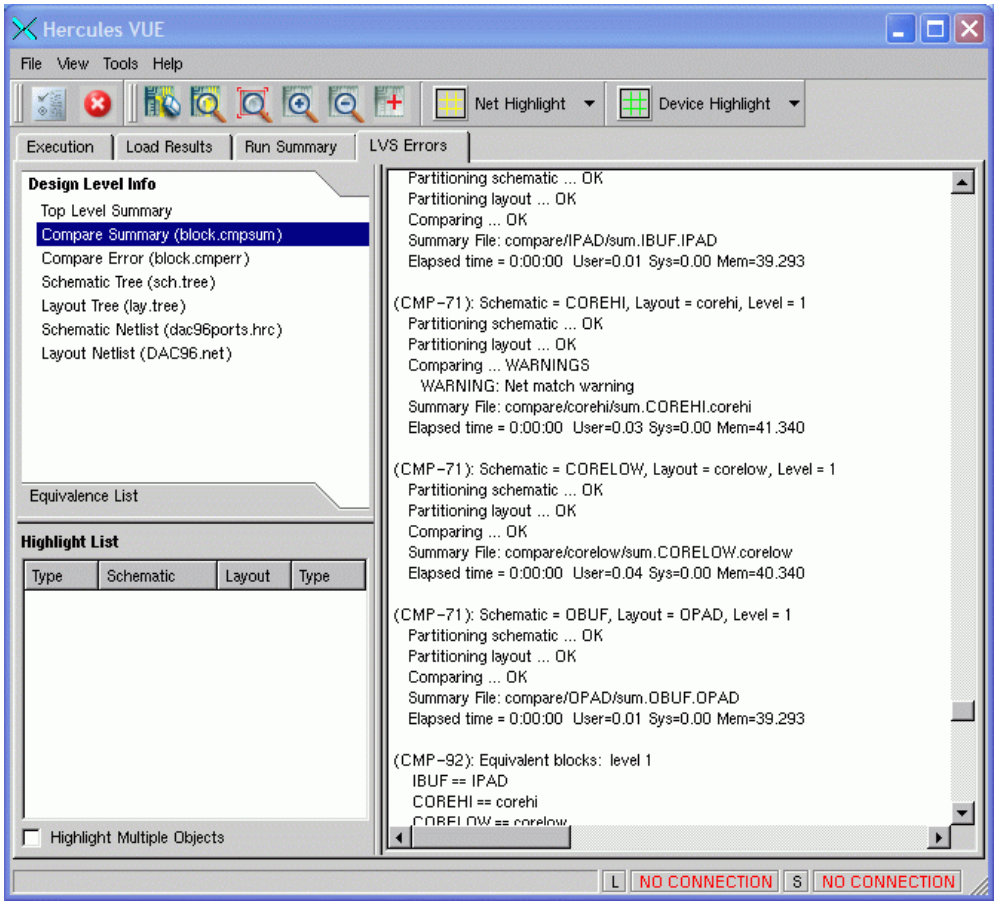
Figure 8-13 Unsuccessful Compare Error



Because we upgraded the warning to an error, the COMPARE is no longer clean, but has an error and fails, as shown in Figure 8-13.

We can also go back and look at the warnings we suppressed. Click Design Level Info and then Compare Summary. Scroll to the same place we were at earlier, with COREHI and DAC96, to get the view shown in Figure 8-14.

Figure 8-14 COREHI with Only One Warning



For COREHI, now there is only one type of warning instead of two. As a practice exercise, go into the summary of COREHI and verify that the warning is no longer there, either. You can suppress many of these in order to change the format and reduce the size of the file. You can also add more informational messages that are suppressed by default. For a complete list of messages based on the files in which they are printed (block_lvs.log or sum.block.block) or suppressed, see the *Hercules Reference Manual*.

Quit VUE by selecting File > Exit.

Running Hercules LVS for Example 2

Go to the directory where your dac96lvs3.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs3/. Enter the following command:

```
hercules dac96lvs3.ev
```

The sample runset file should only take a few minutes to run. Your active window displays the execution process. While the screen contents might scroll quickly (freeze the screen with Control-s and restart with Control-q), you should be able to notice any specific actions or warnings that appear. All of this information appears in a summary file for your examination.

Runfile Results for Example 2

First, examine the runset in Example 8-6 to see that we have upgraded WARNINGs to MESSAGE_ ERRORs for CMP-42 and CMP-43. We have also moved the ALL_PORTS_TEXTED = TRUE and REQUIRE_TEXTED_PORTS_MATCH commands out of the equivalence file and put them in the COMPARE section of the runset, so that they apply to all equivalence points, not just the top block, DAC96.

Example 8-6 Portion of dac96lvs3.ev Runset

```
/* ------------------------ Runset options */
OPTIONS {
        MESSAGE_ERROR = {CMP-42 CMP-43}
        MESSAGE_IDENTIFIERS = ON
        CREATE_VUE_OUTPUT = TRUE
        IGNORE_CASE = FALSE
        SCHEMATIC_GLOBAL = {VDD GND VDDIO VSSIO}
        SCHEMATIC_GROUND = {GND VSSIO}
        SCHEMATIC_POWER = {VDD VDDIO}
        LAYOUT_POWER = {VDD VDDIO}
        LAYOUT_GROUND = {GND VSSIO}
        EDTEXT=dac96.text
        NET_PREFIX = XX_

.......................... DATA OMITTED .......................

COMPARE {
        COMPARE_PROPERTIES = TRUE
        PROPERTY_WARNING = FALSE
        EXPLODE_ON_ERROR = FALSE
        RETAIN_NEW_DATA = TRUE
        STOP_ON_ERROR = FALSE
        FILTER = TRUE
        MERGE_SERIES = FALSE
        MERGE_PARALLEL = TRUE
        MERGE_PATHS = FALSE
        EQUATE_BY_NET_NAME = TRUE
        AUTO_EXCLUDE_EQUIV = TRUE
        REMOVE_DANGLING_NETS = TRUE
        PUSH_DOWN_PINS = TRUE
        PUSH_DOWN_DEVICES = TRUE
        DETECT_PERMUTABLE_PORTS = FALSE
        ALL_PORTS_TEXTED = TRUE
        REQUIRE_TEXTED_PORTS_MATCH = TRUE
```

Once the Hercules run is completed, enter the following command:

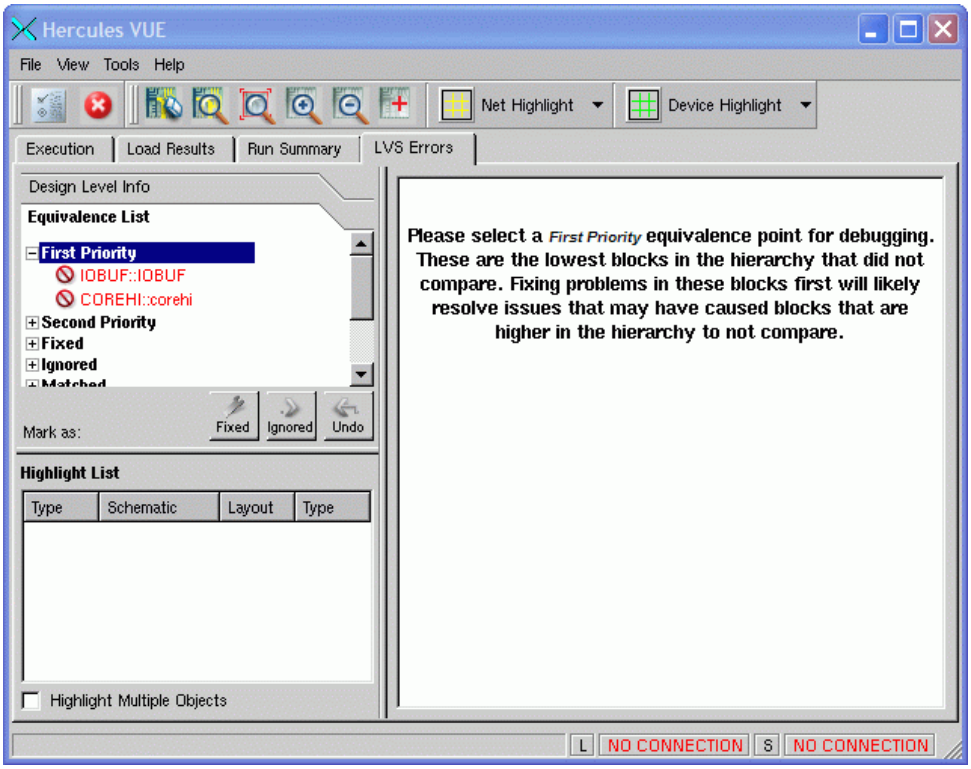
vue &

Since we have already executed Hercules, we load the results in VUE. Select Load Results and click Load the *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs3/DAC96.vue file.

The Run Summary File indicates that top block DAC96 failed compare. As you scroll down this window, it tell you that there are two equivalence errors.

Click the LVS Errors tab to examine first priority equivalence errors. You should see the following:

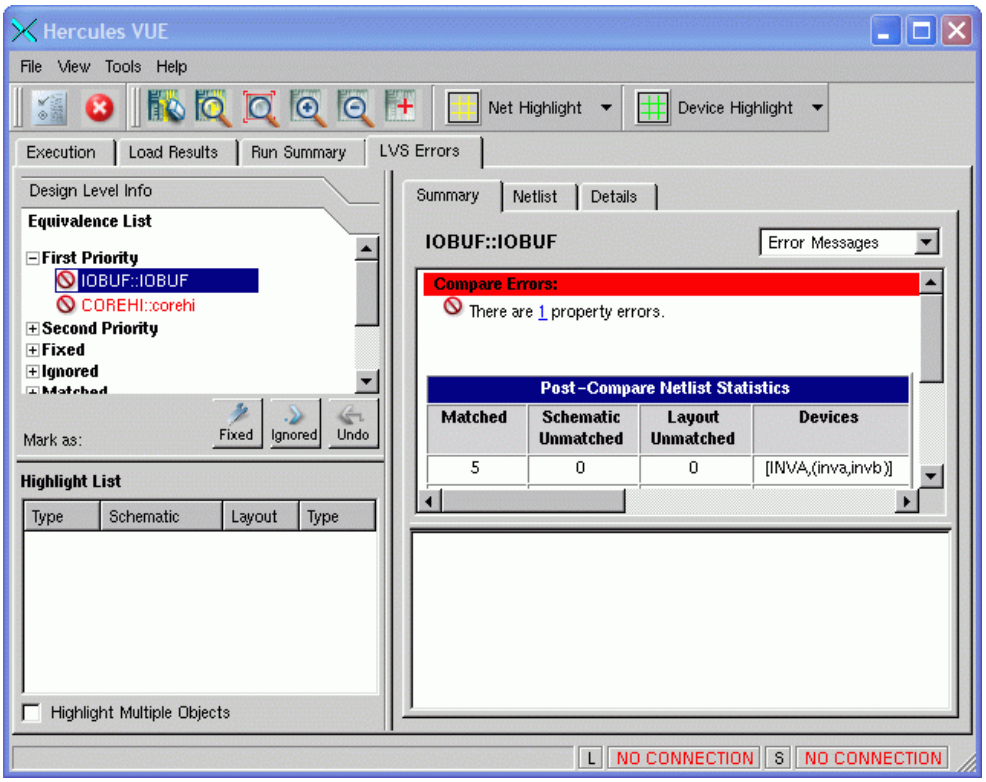
Figure 8-15 LVS Errors



You have two blocks with errors, IOBUF and corehi.

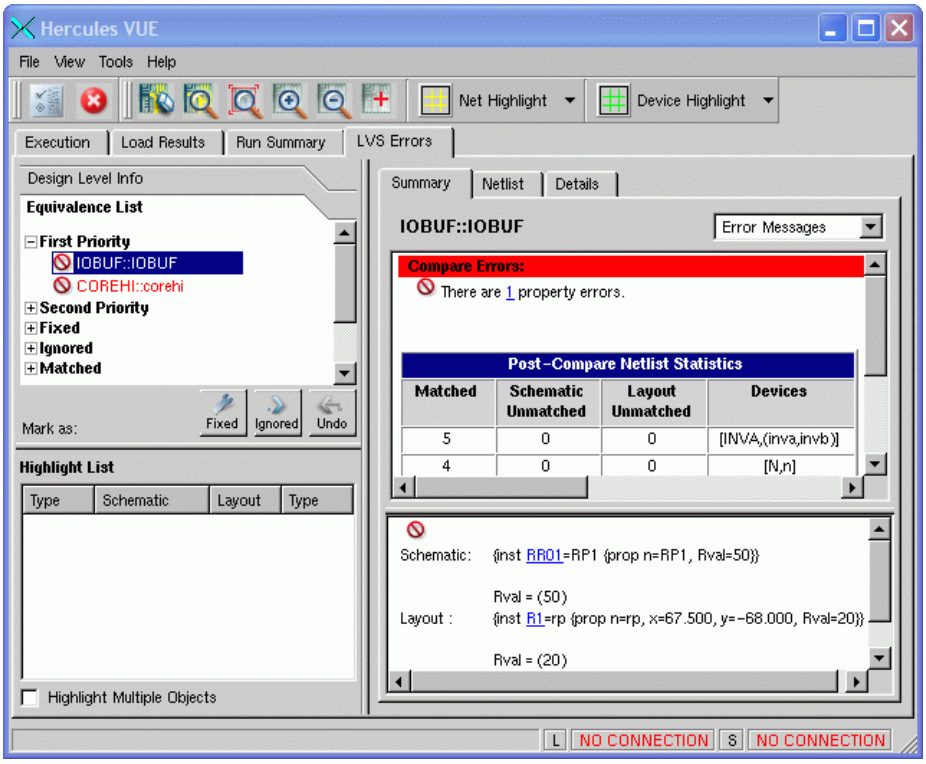
First, open IOBUF by choosing IOBUF in the left window. See Figure 8-16.

Figure 8-16 First Priority Block IOBUF



The compare errors tell you that there is one property error. Click this error to see details on property mismatches for resistor device, as seen in Figure 8-17.

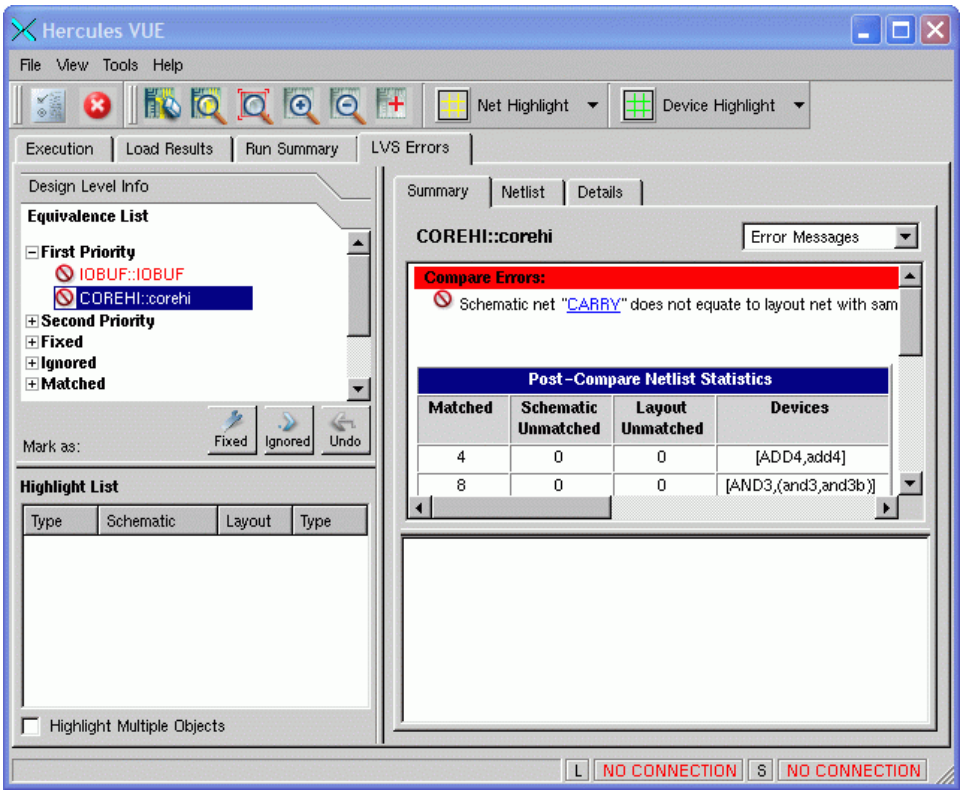
Figure 8-17 Property Mismatch Error



In many cases, the properties in the schematic do not match the properties in the layout netlist. Also, in most cases, if you are debugging the design to try to get it to compare, you are not at liberty to fix these property errors. Hercules has a PROPERTY_WARNING option in the COMPARE section which causes the TRUE setting to report property errors as warnings. You should generate warnings for the property mismatches until you complete all of the other debugging of the design.

Now, look at corehi by choosing corehi in the left panel of the VUE window. The Summary tab loads Compare Error message, as shown in Figure 8-18.

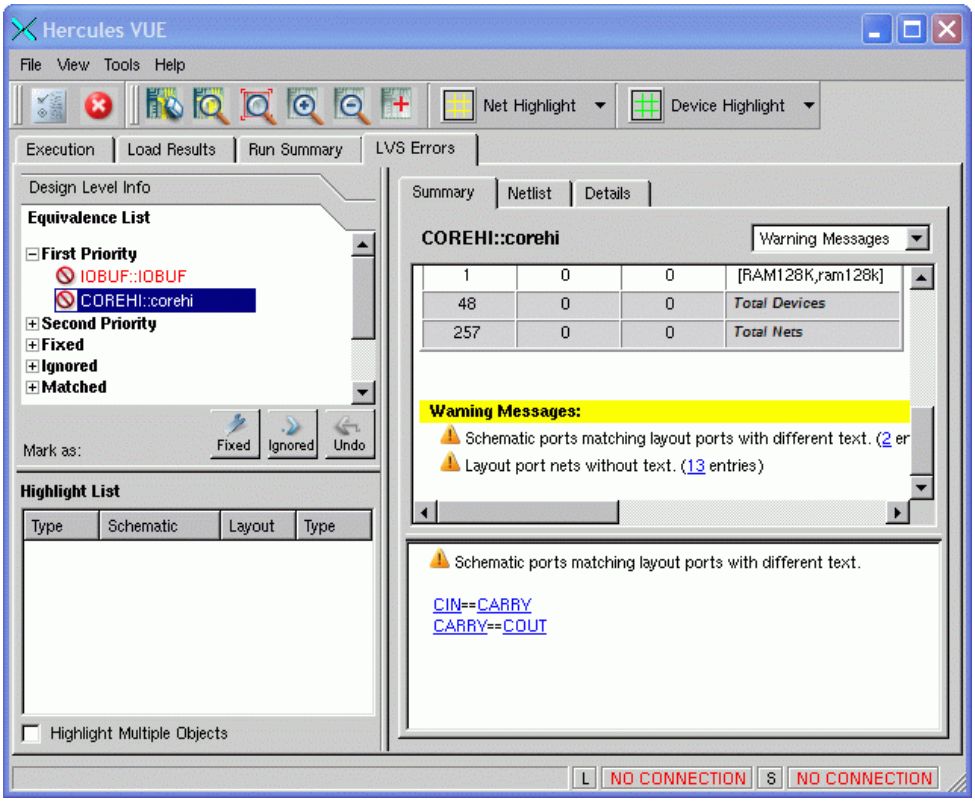
Figure 8-18 Compare Error Message for Block COREHI



Scroll down the Summary tab to view the warning messages. Details of the first warning message (see Figure 8-19) tells you that CARRY = COUT. This causes a compare error since we upgraded this warning message to an error. Also, notice another warning message about “Layout port nets without text.”

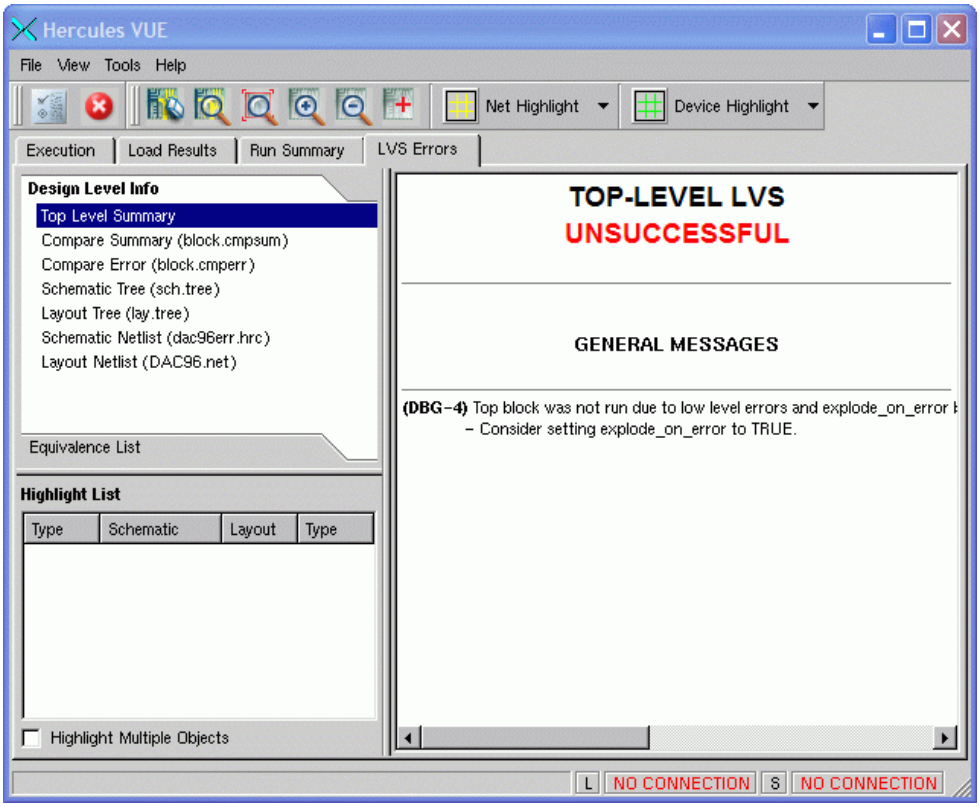
Remember, we required that all ports be texted in the COMPARE section. You should upgrade this message to an error for the strict comparison flow.

Figure 8-19 Warning Messages



Click Design Level Info and select Top Level Summary. Notice (as shown in Figure 8-20) that this block was not run, due to dependency errors. Because corehi and IOBUF did not compare, and they are sub-blocks of DAC96, DAC96 was not compared.

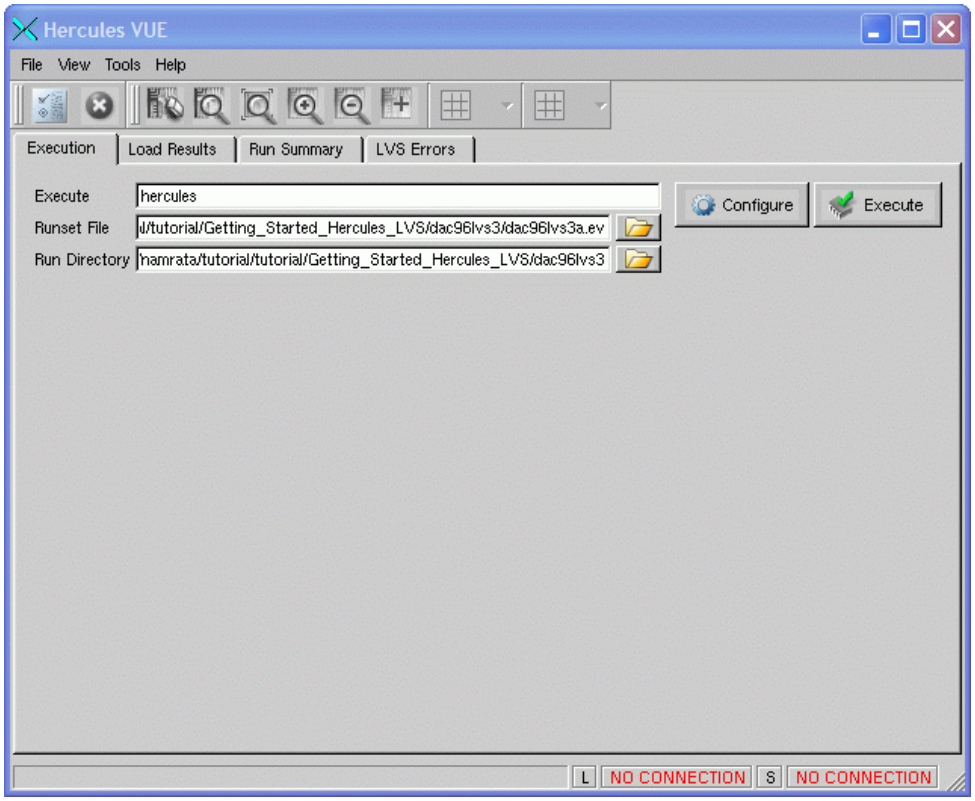
Figure 8-20 DAC96 (Top Level) Summary



Now rerun this example using the dac96lvs3a.ev runset, which contains an upgrade for the CMP-124 WARNING to ERROR, and PROPERTY_WARNING=TRUE. Again, you have to rerun only the comparison phase.

To rerun, select the Execution tab and select the dac96lvs3a.ev runset file. Click Execute.

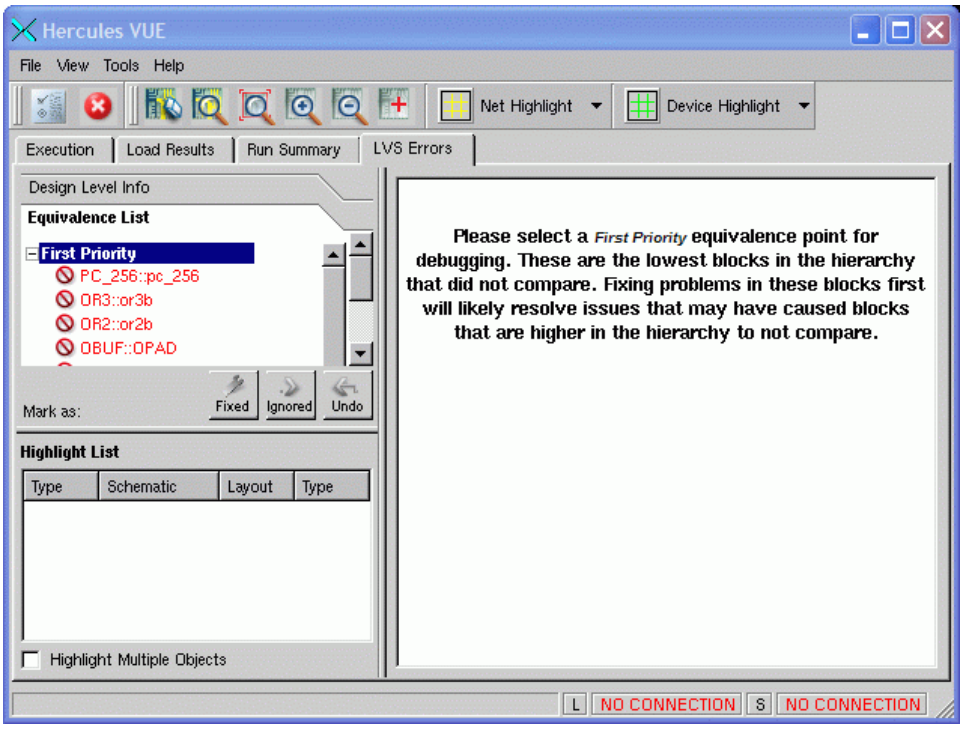
Figure 8-21 Execute Hercules on daclvs3a.ev



When the Hercules run is completed, data is automatically loaded. The run summary shows overall compare results.

Select the LVS Errors tab and notice that IOBUF is no longer listed as a cell with an error. However, there are quite a few blocks that were added to the list of blocks with errors due to the upgrade of CMP-124, as shown in Figure 8-22.

Figure 8-22 Errors Added to CMP-124



Take a few minutes to browse through some of these blocks and the different sections of the files available with Hercules-VUE. We go into more detail on this interface in Chapter 9, “Hercules HLVS Debugging.” This chapter shows examples of the Schematic Unmatched, Layout Unmatched, and Matched Devs Connected to Unmatched Nets sections of the files, and how the interface helps debug these types of problems.

Once completed, select File > Exit to quit VUE.

EQUATE, EQUIV, and COMPARE—Which Setting Takes Priority?

As you have seen in our two examples, you can set options in the COMPARE section of the runset, in the equivalence file, or in both. Many of these options can also be set in the EQUATE commands. The FILTER option for example, could go in all three places. The following rules govern option settings:

Rule 1: Options set in EQUATE specify which devices are eligible for consideration if a given function is performed. In the following example, filter_options means that device n should be considered if the filtering functionality is performed. It says nothing about whether that operation will actually be performed.

```
EQUATE NMOS N = n G S D VBB {
```

```
   filter_options = {nmos-3 nmos-8}
```

Rule 2: Options set in COMPARE globally control functionality. In the following example, `filter = true` tells Hercules to perform the filtering operation. The COMPARE filtering operates only on devices where the filter_options are set in the EQUATE command for that device.

```
COMPARE {
   filter = true }
```

Rule 3: Options set in the equivalence file override the global setting and allow for local control. In the following example, the filtering operation is not performed for the inv block.

```
equiv inv = inv {
   filter = false }
```

What's Next?

Now that you are familiar with running a Hercules LVS job and the basics of the Hercules-VUE, you should continue on to Chapter 9, "Hercules HLVS Debugging," which gives a detailed example of debugging a Hercules LVS job. You learn more about how to use Hercules-VUE to debug specific device and net mismatch errors.

9

Hercules HLVS Debugging

In this chapter you go through a complete Hercules LVS debugging example. You learn how to use Hercules VUE, a GUI-based debugging tool.

Learning Objectives for This Chapter

- To have a quick checklist of things to look for when your LVS job fails
- To run an example of a standard LVS comparison with errors
- To learn the Hercules VUE debugging tool

Before You Start

Before you start this tutorial, make sure you have *completely* gone through the installation and setup procedure described in Chapter 1, “Installation and Setup.” See the Creating Directories and Getting the Files section in Chapter 1, for directory structure and necessary files. You should also, at the very least, have completed the tutorials in Chapter 2, 3, 7, and 8.

Quick Checklist for LVS Debug

Now that you have reviewed the output files available to you for LVS debug, here is a checklist for quickly debugging your LVS results without having to look through all the output files manually. You should refer to this generic list whenever debugging an LVS job.

1. Check for texting or device extraction errors.
2. If you have major texting errors, review your TEXT_OPTIONS, ASSIGN, and TEXT sections to make sure you have the correct text layers attached to the correct polygon layers.
3. Debug all device extraction errors.
4. Rerun your Hercules LVS job.
5. Open the Compare_Results.html file in your browser and review the equivalence points listed in the main LVSDEBUG window.
6. Fix COMPARE errors and rerun Hercules.

A specific example of LVS debug follows the checklist details below, but it does not have examples of all of the potential problems listed below, and therefore is not as generic.

LVS Extraction Debug

Steps 1 through 4 of the quick checklist debug the extraction phase of your LVS job.

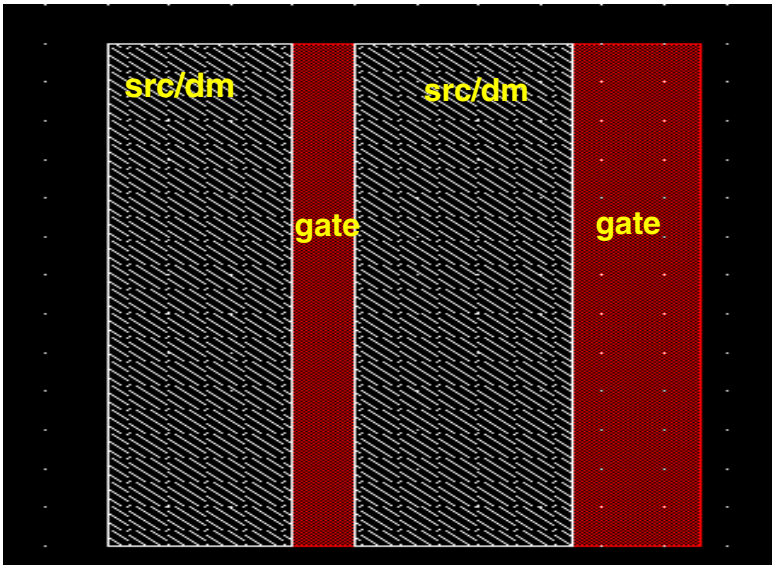
Step 1: Check for Texting or Device Extraction Errors

You should check the *block*.LAYOUT_ERRORS file (using vi or other editors) to make sure you do not have any texting or device extraction errors. Below are some of the more common problems you might encounter during this step and an example of each situation.

Missing Terminals - Device Extraction Errors

In this example we have built an NMOS device that has only one terminal. The NMOS command option, MOS_SINGLE_SD, is set to NORMAL. This tells Hercules to generate an extraction error if any NMOS device does not have two terminals. Figure 9-1 shows the layout of the extracted NMOS with the error and Example 9-1 shows the actual error in the *block*.LAYOUT_ERRORS file.

Figure 9-1 Layout of NMOS Defined with Missing Terminals



Example 9-1 NMOS Defined with Missing Terminals in block.LAYOUT_ERRORS File

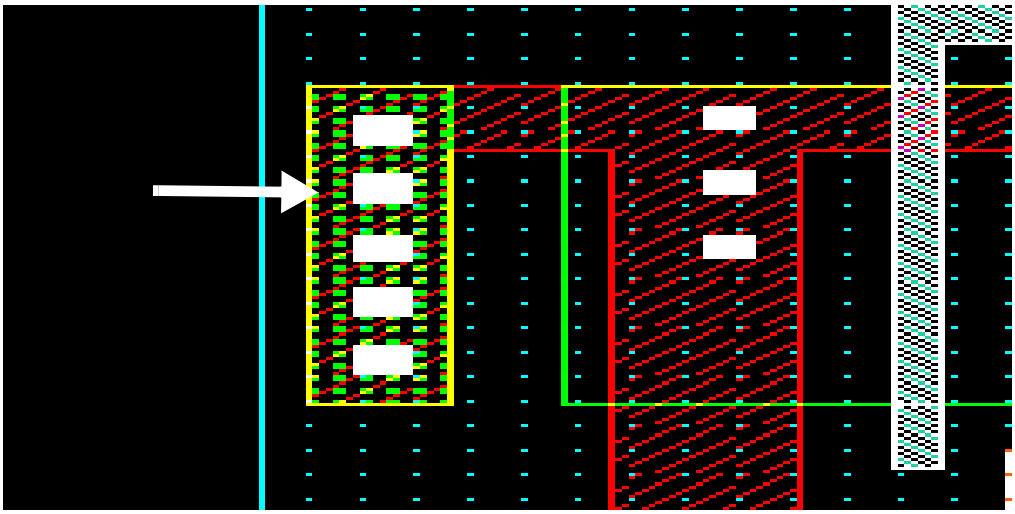
```
Library name:  dac96
Structure name:  inva

#####--- ERR_DEVICE ----------------------------------

NMOS n ngate nsd nsd SUBSTRATE {
   MOS_PRINT_XY_POSITION=FALSE;
   MOS_PRINT_STATS=FALSE;
   MOS_SAVE_ALL_PROPS=FALSE;
   MOS_REFERENCE_LAYER="POLY";
   MOS_HIERARCHICAL=TRUE;
   MOS_CALC_NRS_NRD=TRUE;
   MOS_MULTITERM_EXTRACT=FALSE;
   MOS_NODE_BASE_EXTRACT=TRUE;
   MOS_LEVEL_SD=FALSE;
   MOS_COMBINE_SOURCE=TRUE;
   MOS_SINGLE_SD=NORMAL;} TEMP=ndevice

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure     Error Type       Layer   Value   (position x, y)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inva        MISSING_TERMINALS  nsd     1       (11.000, 10.500)
```

Too Many Terminals - Device Extraction Errors

For our example we built a diode, but, instead of using the diffusion layer as both of the terminals, we used the diffusion-derived layer as one and the diocont layer as the other. Figure 9-2 shows a picture of the device and Example 9-2 shows the ERROR in the *block*.LAYOUT_ERRORS file. Notice that the value in the *block*.LAYOUT_ERRORS file is 2.

This value is always one greater than the number of terminals you are allowed to have for a specific layer. In the case of this example, you should have 1 polygon for the ndiffdio layer, and 1 polygon for the diocont layer.

Figure 9-2 Layout of Diode Defined with Too Many Terminals



Example 9-2 Example of Diode Defined with Too Many Terminals in block.LAYOUT_ERRORS File

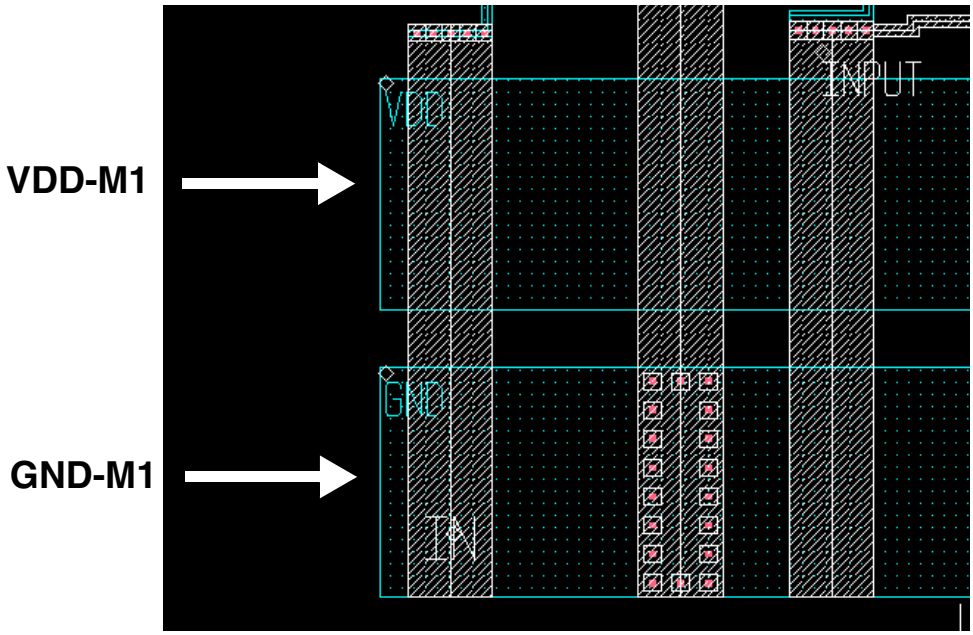
```
Library name:  dac96.db
Structure name:  IOBUF

#####--- ERR_DEVICE ----------------------------------

DIODE ndio ndiffdio diocont substrate {
   DIODE_PRINT_XY_POSITION=FALSE;
   DIODE_PRINT_STATS=FALSE;
   DIODE_SAVE_ALL_PROPS=FALSE;
   DIODE_TYPE=NP;
   DIODE_HIERARCHICAL=TRUE;
   DIODE_RECOGNITION_LAYER_USED=TRUE;} TEMP=ndiode

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure   Error Type       Layer   Value    ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IOBUF    TOO_MANY_TERMINALS  diffcont  2     (-88.000, 112.500)
```

To fix this error, use the ndiffdio layer for both terminals. The DIODE is formed from the N material in the terminals and the P material in the substrate.

Unused Text

If you define a text layer in the ASSIGN section and use it in the TEXT section, Hercules generates an unused text error for each piece of that text that is not attached to a polygon. Attached is defined by the value of the ATTACH_TEXT variable in the TEXT_OPTIONS section, and by whether or not the text string overlaps the layer with which it was associated in the TEXT section. Below is an example of an unused text error. In Figure 9-3 you see that the text strings are overlapping M2, not M1. The TEXT on layer 30 should be attached to M2 instead of M1 to avoid these errors.

Figure 9-3 M1 and M2 Text in IOBUF



Example 9-3 Unused Text ERROR in block.LAYOUT_ERRORS File

```
Library name:  dac96.db
Structure name:  IOBUF

#####--- ERR_TEXT_UNUSED ----------------------------------

Reports unused text of M1.txt file after all text assignments.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure         Unused Text      l;dt    ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IOBUF             GND             30;0    (-89.500, -303.500)
IOBUF             VDD             30;0    (-89.500, -253.300)
```

Text Opens

Whenever you have two unconnected nets in a cell with the same text string, Hercules generates a text open error in the *block*.LAYOUT_ERRORS file. Example 9-4 shows an example of a text open error message in the *block*.LAYOUT_ERRORS file.

There are special TEXT_OPTIONS such as USE_COLON_TEXT and USE_SEMI_COLON_TEXT to suppress certain text open errors in library (or standard) cells. See the *Hercules Reference Manual* for details on these options.

Example 9-4 Text Open ERROR in block.LAYOUT_ERRORS File

```
Library name:  dac96
Structure name:  buf4x

#####--- ERR_TEXT_OPEN ---------------------------------

Nets in named structures are connected by text.

Connect and Text commands used for opens processing:

CONNECT {
   ngate pgate BY [ TOUCH OVERLAP ] field_poly
   M2 BY [ TOUCH OVERLAP ] PADTOP
   M1 M2 BY [ TOUCH OVERLAP ] V1
   M1 ndiffdio res_term field_poly nsd psd welltie subtie BY [ TOUCH
 OVERLAP ] CONT
   NWELL BY [ TOUCH OVERLAP ] welltie
   SUBSTRATE BY [ TOUCH OVERLAP ] subtie
}

TEXT {
   M1 BY M1.text
   M2 BY M2.text
   field_poly BY POLY.text
   PADTOP BY PAD.text
   NWELL BY "VDD"
   SUBSTRATE BY "GND"
}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Parent Struct  Base  l;dt Parent Inst origin  Base  Text From  Path
   Text        Text         (x, y) (x, y)     Net ID           or Info
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
buf4x          IN    30;0   (3.000, 56.000)   XX_5   LAYER     Signal
                     30;0   (57.000, 54.500)          LAYER     Signal
```

Text Shorts

Whenever you have two different text strings attached to the same electrically connected net, Hercules generates a text short error in the *block*.LAYOUT_ERRORS file. Example 9-5 shows how that error appears in the file. Later we show you how to use the TEXT_OPTION FIND_SHORTEST_PATH_BETWEEN_TEXT_SHORTS and Hercules VUE to trace these text shorts.

Example 9-5 Text Short ERROR in block.LAYOUT_ERRORS File

```
Library name:  dac96.db
Structure name:  IOBUF

#####--- ERR_TEXT_SHORT ----------------------------------

Text for net in named structures are shorted.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure   Net ID   Used Text    l;dt    (position x, y)   Text From
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inva         XX_1     *     A     30;0    (3.000, 2.000)      LAYER
                            Z     30;0    (9.000, 2.000)      LAYER
```

Step 2: Review TEXT_OPTIONS, ASSIGN, and TEXT Sections

If you have major texting errors, review your TEXT_OPTIONS, ASSIGN, and TEXT sections in the runset to make sure you have the correct text layers attached to the correct polygon layers. You might also want to open your input database in a layout editor and look at the relationship of the text to the polygon data.

Step 3: Debug All Device Extraction Errors

Debug all device extraction errors using the Hercules VUE graphical debug tool, or open your input layout database in a layout editor of your choice, and view the errors using the coordinates given in the *block*.LAYOUT_ERRORS file.

Step 4: Rerun Your Hercules LVS Job

Once you have fixed all the device extraction and texting errors, rerun your Hercules LVS job.

LVS Comparison Debug

Steps 5 and 6 of the quick checklist debug the comparison phase of your LVS job.

Step 5: Review the Equivalence Points

Open the Compare_Results.html file in your browser and review the equivalence points listed in the main LVSDEBUG window. When you click on each of these links, you jump directly to the error in the sum.*block.block* file. The following sections discuss common errors found on the first run of LVS COMPARE on a design.

Filtering Options Missing

If COMPARE determines that you have extra devices in the layout or schematic that match one of the available Hercules filter options, it generates a message. This message informs you of the filtering option that applies to the extra devices and recommends that you set that option to help your design compare.

Merging Options Missing

If COMPARE determines that you have extra devices in the layout or schematic that can be merged in series or parallel to help your design match, it generates a message. This message informs you of the merging option that applies to the extra devices and recommends that you set that option to TRUE to help your design compare.

POWER/GROUND Shorts

In many cases a text short error seen in the *block*.LAYOUT_ERRORS file also results in POWER/GROUND shorts seen in the sum.*block.block* files. Whenever a short is found, a table is generated in the sum.*block.block* file showing the possible shorted nets in the layout (or schematic) and how they correspond to nets in the schematic (or layout). Example 9-6 shows one of these tables.

Example 9-6 Diagnostic OPEN/SHORT Table in sum.block.block File

```
Diagnostic analysis recognizes the following correspondence
between unmatched nets in the schematic and layout. These might
indicate the source of shorts or opens:

Schematic     Layout        Net Name
Connections   Connections
-----------   -----------   ----------------------
              9             GND
4                           GND
5                           $1N7
```

LAYOUT POWER or LAYOUT GROUND Definitions Missing

When the LAYOUT_POWER or LAYOUT_GROUND definitions are missing, Hercules generates a warning telling you that these nets are not defined, but Hercules tries to generate these net names automatically based on string matching. Chapter 7, “Introduction

to Hercules HLVS," provides further detail on string matching. If the list of LAYOUT_POWER or LAYOUT_GROUND nets is incorrect, filtering and merging might not occur correctly. Example 9-7 illustrates this warning.

Example 9-7 Warning: LAYOUT_POWER or LAYOUT_GROUND Definitions Missing

```
HLVS (R) Hierarchical Layout Versus Schematic, Release DATA OMITTED
  Copyright. Synopsys.  All rights reserved.

   ** Environment Status **

runset              = test.ev
root                = DAC96

........................... DATA OMITTED .......................

stop_on_no_explode_error   = FALSE  static_equated_nets      = TRUE
text_resolves_port_swap    = TRUE   use_total_width          = FALSE
write_netlists             = TRUE   zero_connection_warning  = FALSE

Purging Compare Directory ... OK

Reading schematic netlist ... OK
Reading layout netlist ... OK

WARNING: No layout power/ground nets were found. Generating them
 automatically:  VDDIO VDD GND VSSIO
... Processing schematic netlist ... OK
Propagating schematic globals ... OK
Processing layout netlist ... OK

........................... DATA OMITTED .......................
```

Schematic Globals Missing

Hercules automatically tries to generate schematic globals when they are not defined in your runset. As described in Chapter 7, "Introduction to Hercules HLVS," a search of the schematic netlist generates this list, but Hercules might miss less commonly defined schematic globals. If this section is missing from your runset, double check that Hercules found all the necessary schematic globals, or false opens might exist in the comparison output.

Step 6: Fix COMPARE Errors and Rerun Hercules

Once you have reviewed and fixed all of these major problems, rerun Hercules LVS on your design and refer to the following detailed instructions for debugging Hercules LVS error output.

Running Hercules LVS With Errors

The next group of commands explains how to run a Hercules LVS job with errors. The first section takes you through the checklist steps listed above, showing how to fix some of the global errors. The second section has you rerun Hercules LVS and goes into detail on debugging unmatched nets and devices.

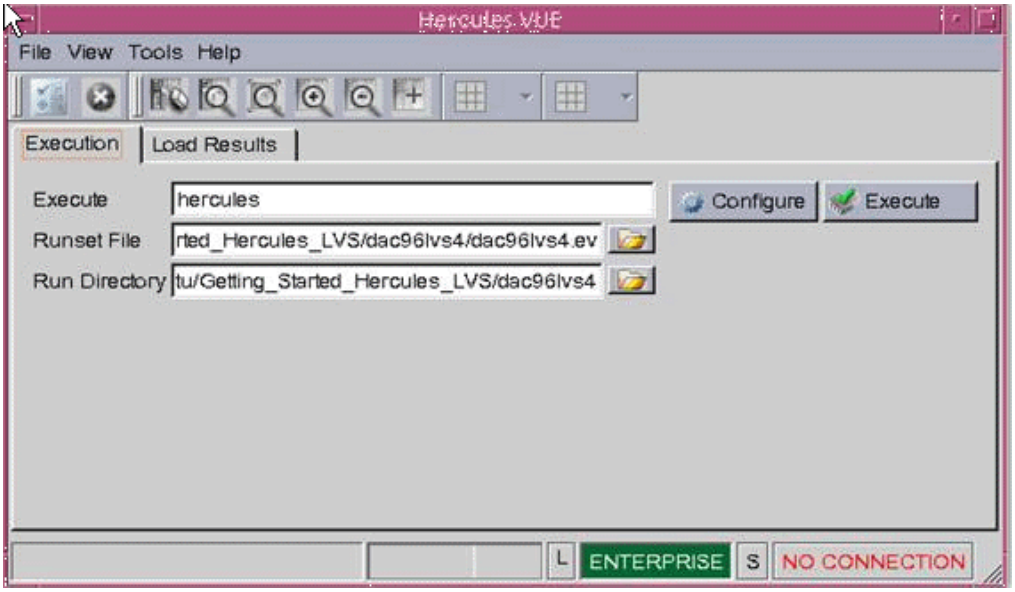
Be sure that you are in the directory where your dac96lvs4.ev file is located, *your_path*/hercules-Examples/Getting_Started_Hercules_LVS/dac96lvs4/. Enter:

```
Enterprise &
```

```
Select Verification > Hercules VUE
```

You should now have the Hercules VUE window on your screen (see Figure 9-4).

Figure 9-4 Hercules VUE

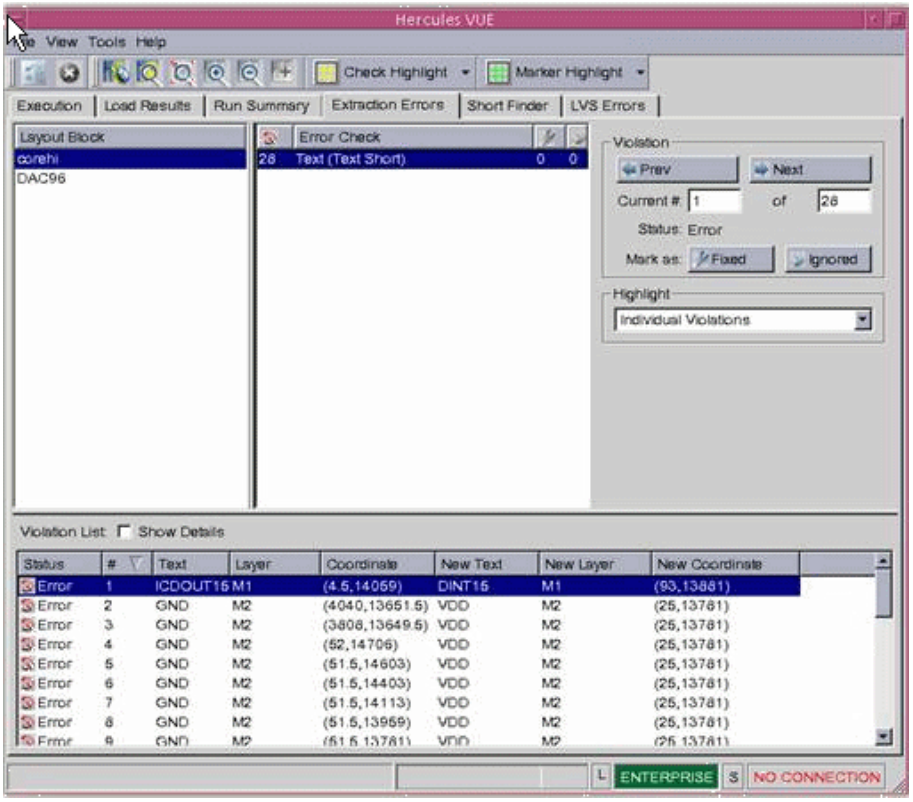


Select Runset File, dac96lvs4.ev, and click Execute.

The sample runset file should take only a few minutes to run. Your active window displays the execution process. Once the Hercules run has completed, results will be loaded automatically.

At the end of the run, the LVS Debug Compare Results are loaded in the Run Summary tab. Select Extraction Errors to see the contents of the DAC96.LAYOUT_ERRORS file (see Figure 9-5).

Figure 9-5 Extraction Errors Tab



Debugging Your Hercules LVS Run

When your Hercules LVS job is complete, follow the steps above in the Quick Checklist for LVS Debug. As you go through the checklist for this example, you are introduced to major topics of interest. The STEP headings continue sequentially through this part of the tutorial.

Step 1: Check for Texting or Device Extraction Errors.

Review the Extraction Errors tab in Figure 9-5.

Based on the DRC Extraction results shown, your first step should be to locate the GND/VDD shorts, and the DINT15/ICDOUT15 short. We now go through the steps for finding these shorts. For now, you can minimize this screen.

Short Finding in Hercules

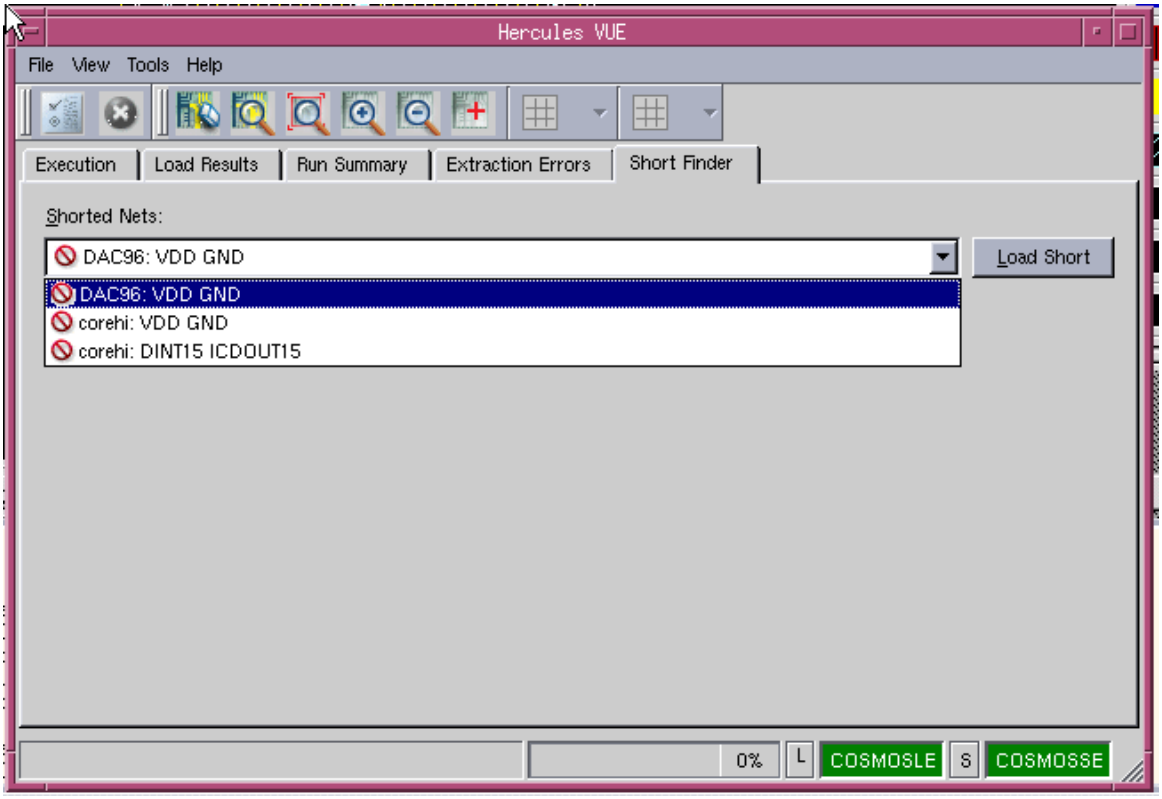
The cause for a text short might not always be obvious. Hercules has two methods for locating shorts in your layout. The first method is to use a feature within Hercules VUE. It is hierarchical Short Finder, run from Hercules VUE interface. The hierarchical Short Finder is available as a feature within the VUEGUI. All text shorts identified by the Hercules tool are available from pull-down list in the Short Finder GUI. After selecting and loading a short from the list, the user is prompted to answer questions about each highlighted section of the net. Based on the user's responses, the highlighted polygons are either eliminated from consideration (pruned) or identified as the root cause of the short. To use the hierarchical Short Finder, the option for creating VUE output data, OPTIONS { CREATE_VUE_OUTPUT = TRUE }, must be set. To generate the data needed for the hierarchical Short Finder, TEXT_OPTIONS { EXTRACT_SHORT_DATA = TRUE }, must be set.

Below we go through the exercise of finding these shorts. We use the hierarchical Short Finder of Hercules VUE for the VDD/GND signal net short and the DINT15/ICDOUT15 short in the block corehi. The VDD/GND short in corehi is the same as the one in DAC96; thus, once we fix the shorts in corehi our block.LAYOUT_ERRORS file will be clean.

Short Finding in Hercules VUE

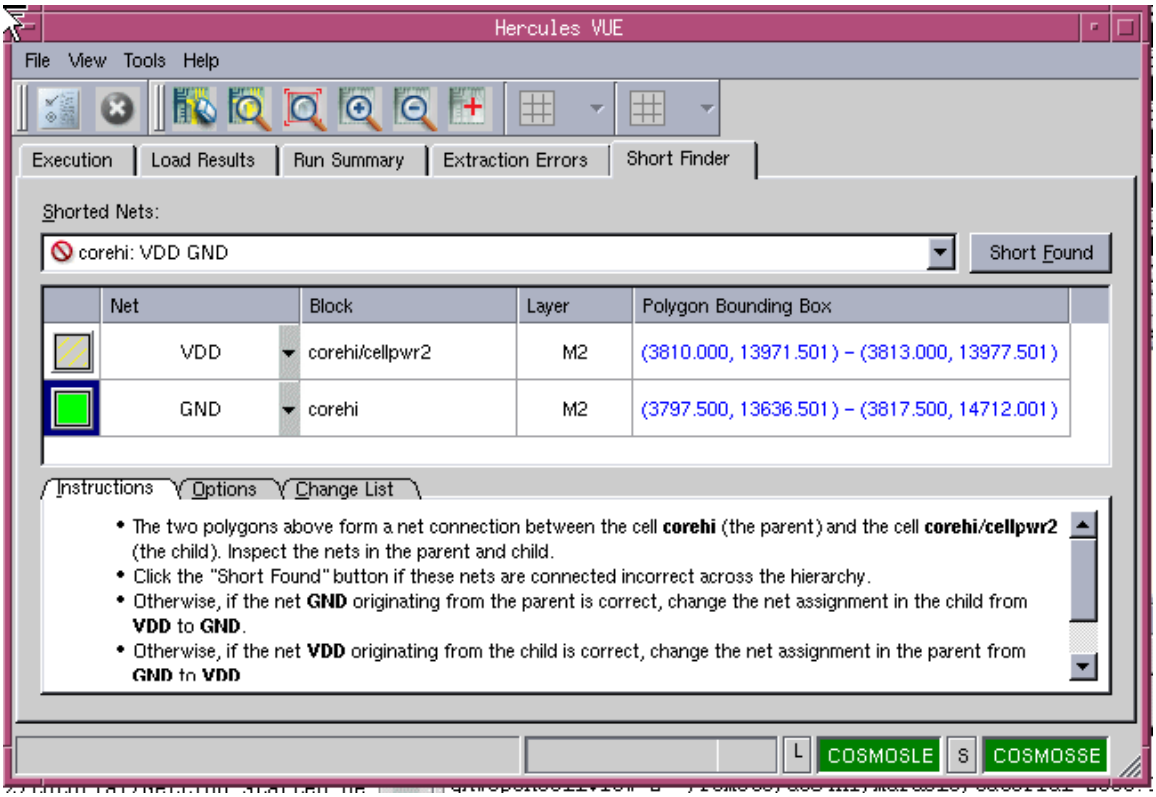
Select Short Finder next to Extraction Errors tab.

Figure 9-6 Hercules VUE: Loading Short Data



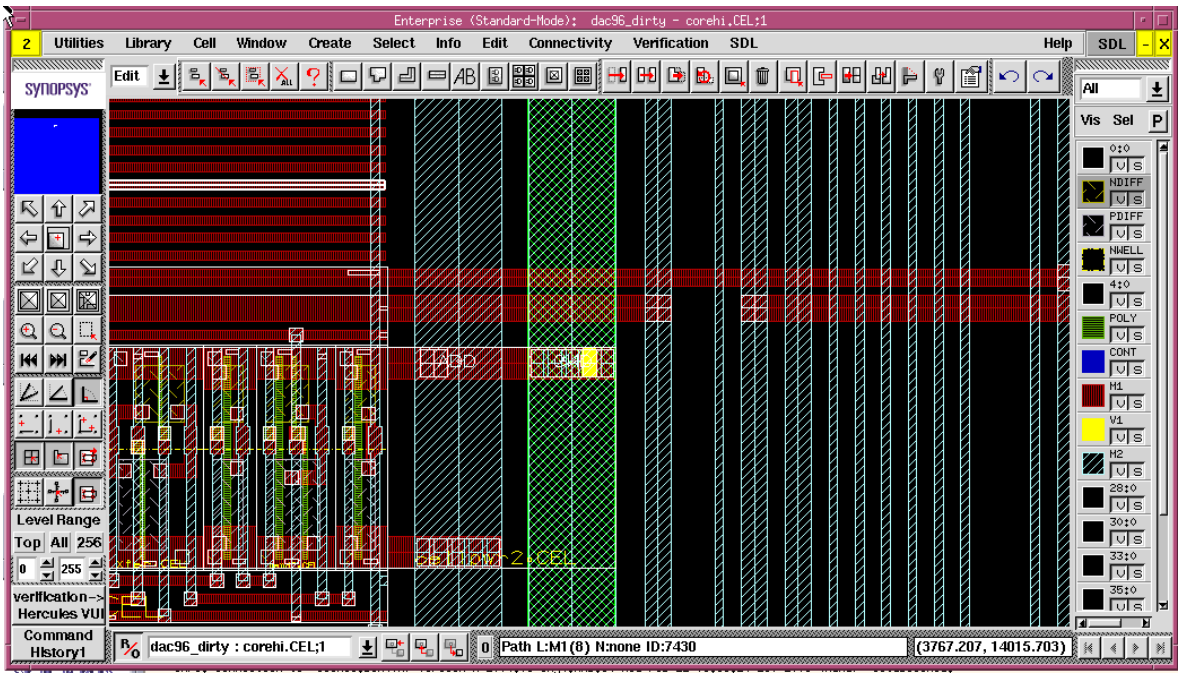
As shown in Figure 9-6, select the VDD/GND short in block corehi and click LoadShort. Details on this short are loaded in the text box as shown in Figure 9-7.

Figure 9-7 VUE Loading VDD/GND Short in block corehi



Analysis of the shorting path through hierarchical cell path is displayed. Corresponding polygons are highlighted in Enterprise, as shown in Figure 9-8

Figure 9-8 Loading VDD/GND Short in Enterprise



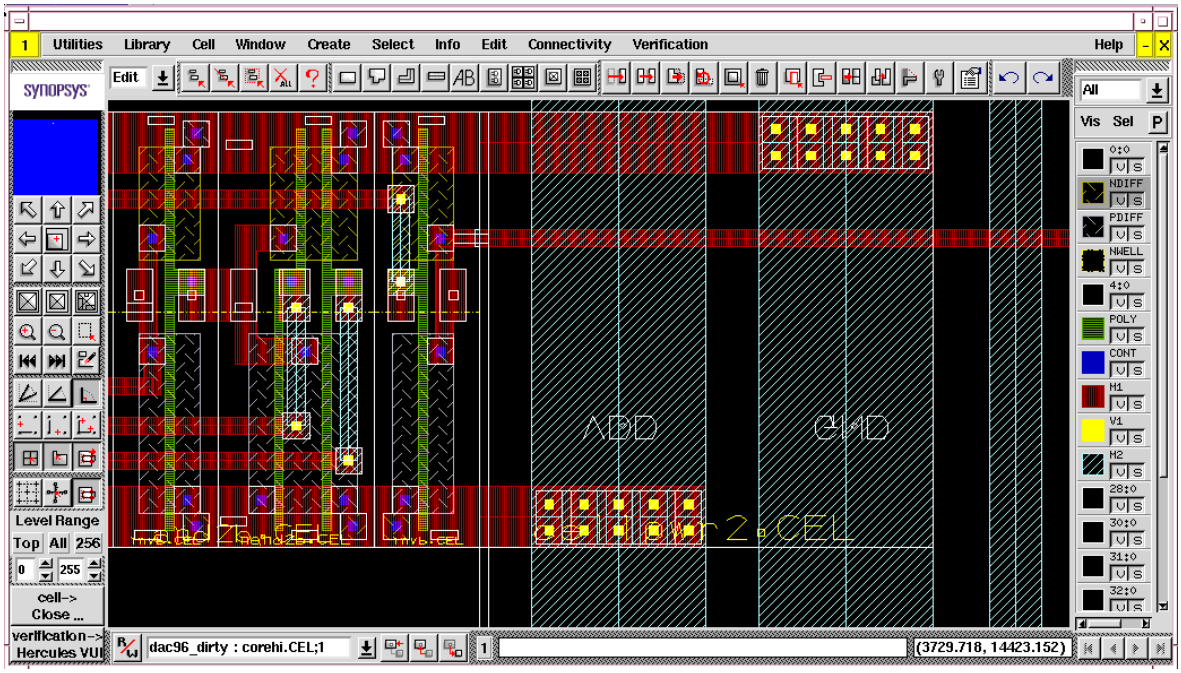
Correcting the VDD/GND Short

If you are using a fully-licensed Enterprise session, you can fix the short at this time. If you do not have a fully-licensed Enterprise session, you can fix short in your own layout editor. We have provided you with an updated layout database for the later steps in the tutorial, so it is not necessary that you fix the error to continue the tutorial. The following is the suggested sequence for deleting the vias.

While in the Enterprise layout window, select Edit > Delete.

Use the left mouse button to select one of the vias to delete. Repeat these steps for the other three vias.

Figure 9-9 Fixed VDD/GND Short in Enterprise



Debugging DINT15/ICDOUT15 Short in Block corehi

Figure 9-10 shows the corresponding Hercules VUE window.

Figure 9-10 Loading DINT15/ICDOUT15 Short in block corehi

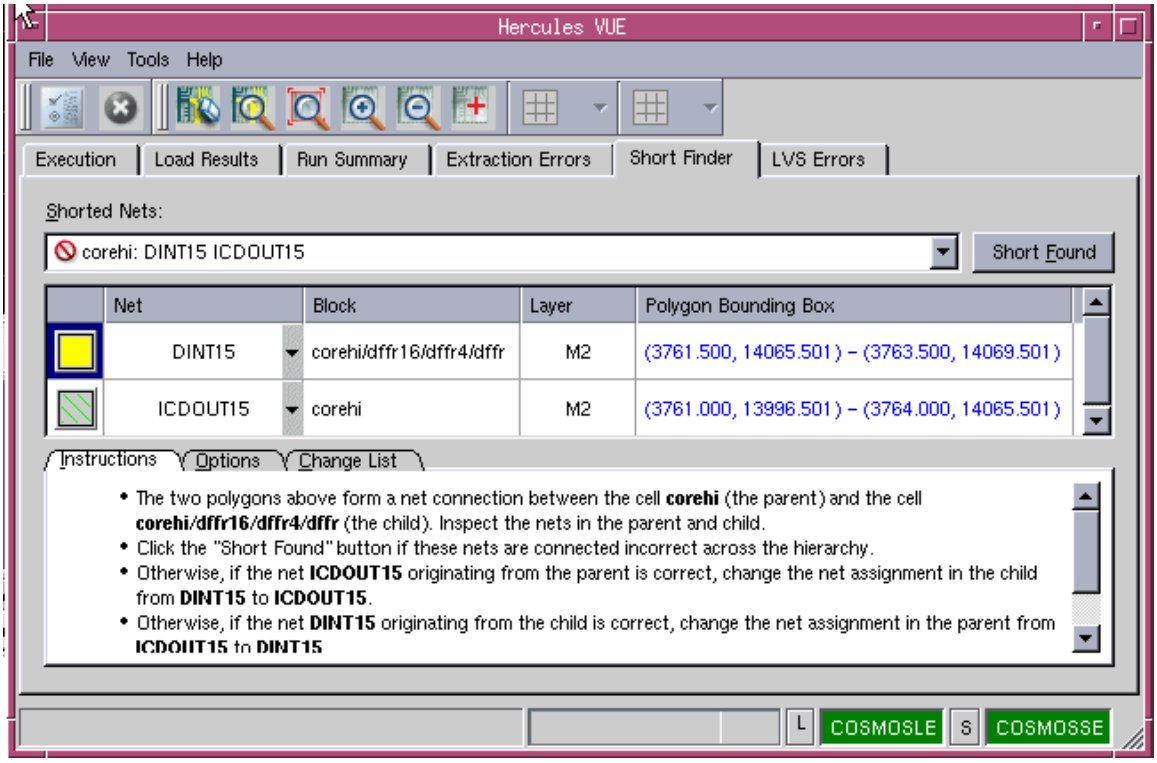
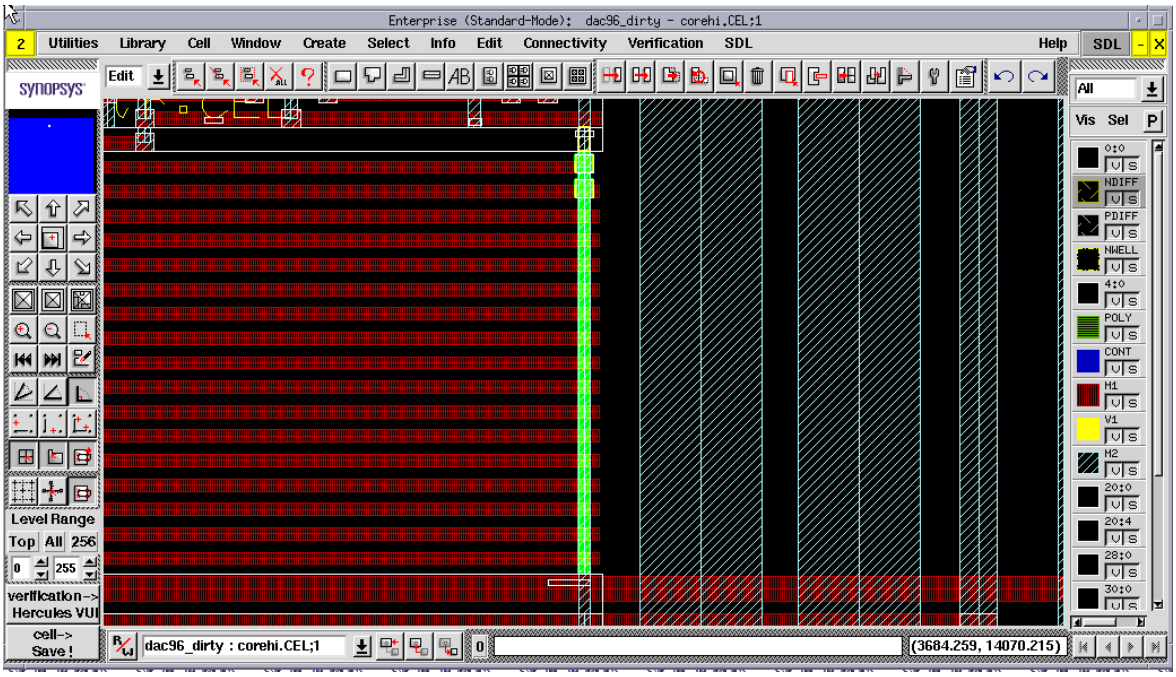


Figure 9-11 Highlight in Enterprise



The following commands show you how to better view the short by turning on only the metal and via layers, and also by zooming in on the path.

Turn off all visible layers except for M1, V1, and M2 (layers 8, 9, and 10, respectively):

Redraw the Enterprise window.

Use the left mouse button to create a window in the Enterprise display window that you want to zoom in on. Zoom in on a location similar to the one shown in You may use the gxwSetView option in Enterprise to zoom to location (3720 14117 3775 14025).

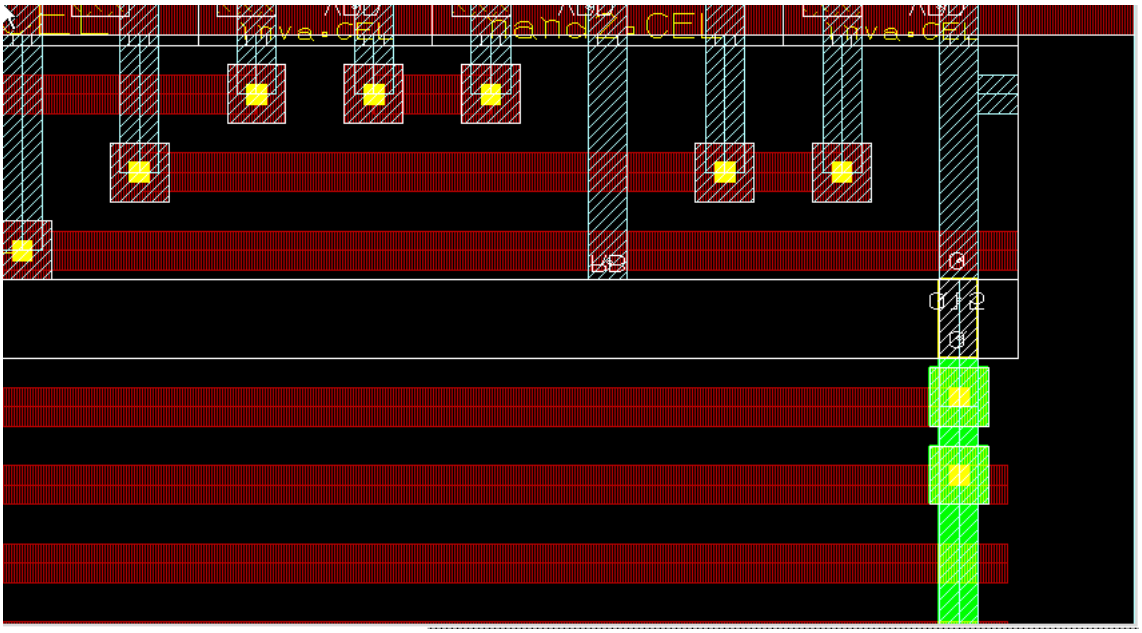
```
gxwSetView 2 "g" { 3720 14117 3775 14025 }
```

Note:

2 in this example is the window number found in a yellow square in the top left corner of the layout window.

Zoom in further in this area. You can see the short between the two nets in Figure 9-12.

Figure 9-12 View of DINT15/ICDOUT15 Short



Fixing the Short Between DINT15 and ICDOUT15

If you are using a fully-licensed Enterprise session, you can fix the short at this time. If you do not have a fully-licensed Enterprise session, you can fix the short in your own layout editor. We have provided you with an updated layout database for the later steps in this tutorial, so it is not necessary that you fix the error to continue the tutorial.

To fix the error, execute the following command in the Enterprise window:

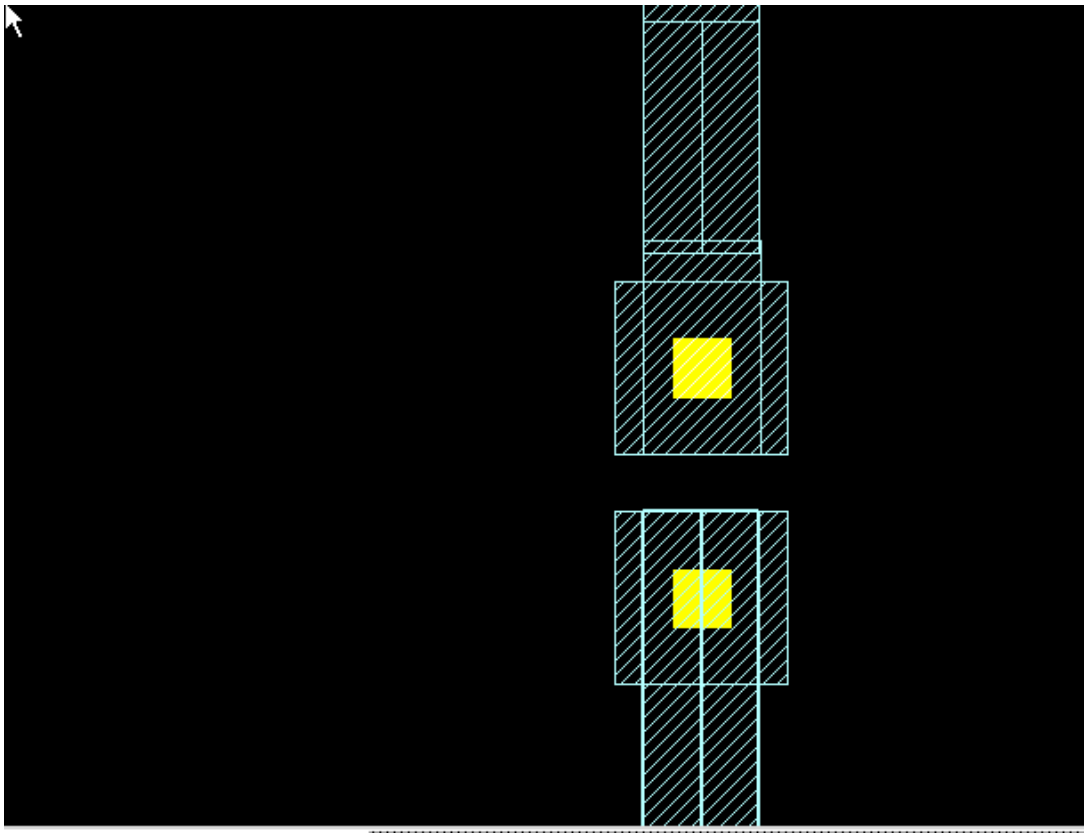
s

Using your left mouse button, select the lower part of the metal 2 line (layer 10) that crosses the top via.

Next, select the top edge of that polygon and slide it down so it is even with the horizontal metal 1 (layer 8) polygon.

Once you have corrected the short, save the design in the layout editor.

Figure 9-13 View of Fixed Short



At this time we have corrected all of the problems in the DAC96.LAYOUT_ERRORS file, and thus completed STEP 1.

Running Hercules After All *block*.LAYOUT_ERRORS Errors Are Fixed

We rerun Hercules on a database that has all of the text shorts fixed. Even if you have been correcting your design as we have explained, use the dac96lvs4b.ev runset for the next series of commands. The dac96lvs4b.ev runset points to a database where the shorts are fixed, guaranteeing that your output match the remainder of this tutorial.

(Remember we are working according to the Quick Checklist for LVS Debug at the beginning of the chapter.)

Step 2: Review TEXT_OPTIONS, ASSIGN, and TEXT Sections

We reviewed the text errors during STEP 1 and there are no texting errors aside from the text shorts we just fixed.

Step 3: Debug All Device Extraction Errors

We reviewed the DAC96.LAYOUT_ERRORS file and there are no device extraction errors in this design.

Step 4: Rerun Your Hercules LVS Job

Now that we have completed fixing all of the problems found in the *DAC96.LAYOUT_ERRORS* file, we rerun Hercules on our updated database.

To execute Hercules again, select the Execution tab from the Hercules VUE window. Select Runset file daclvs4b.ev and click Execute.

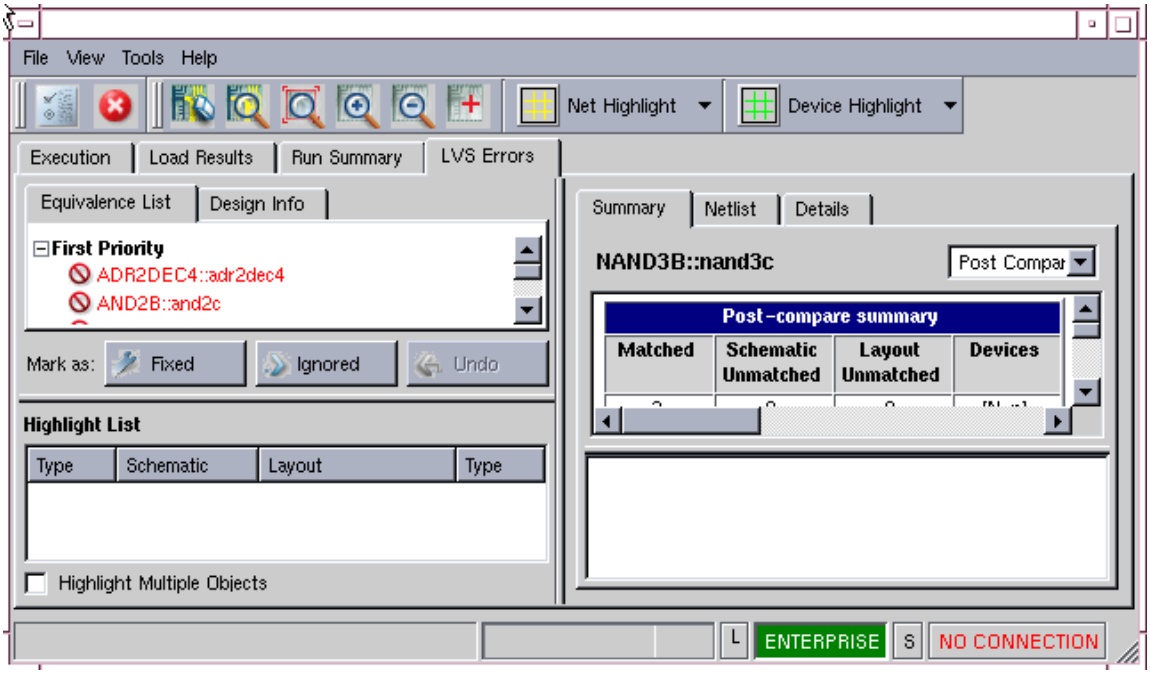
Once the Hercules run is completed, data will be automatically loaded.

Step 5: Review the Run Summary

The next step is to review the run summary to determine overall LVS comparison errors. Run summary highlights that comparison completed with 54 successful equivalencies has 30 equivalence errors.

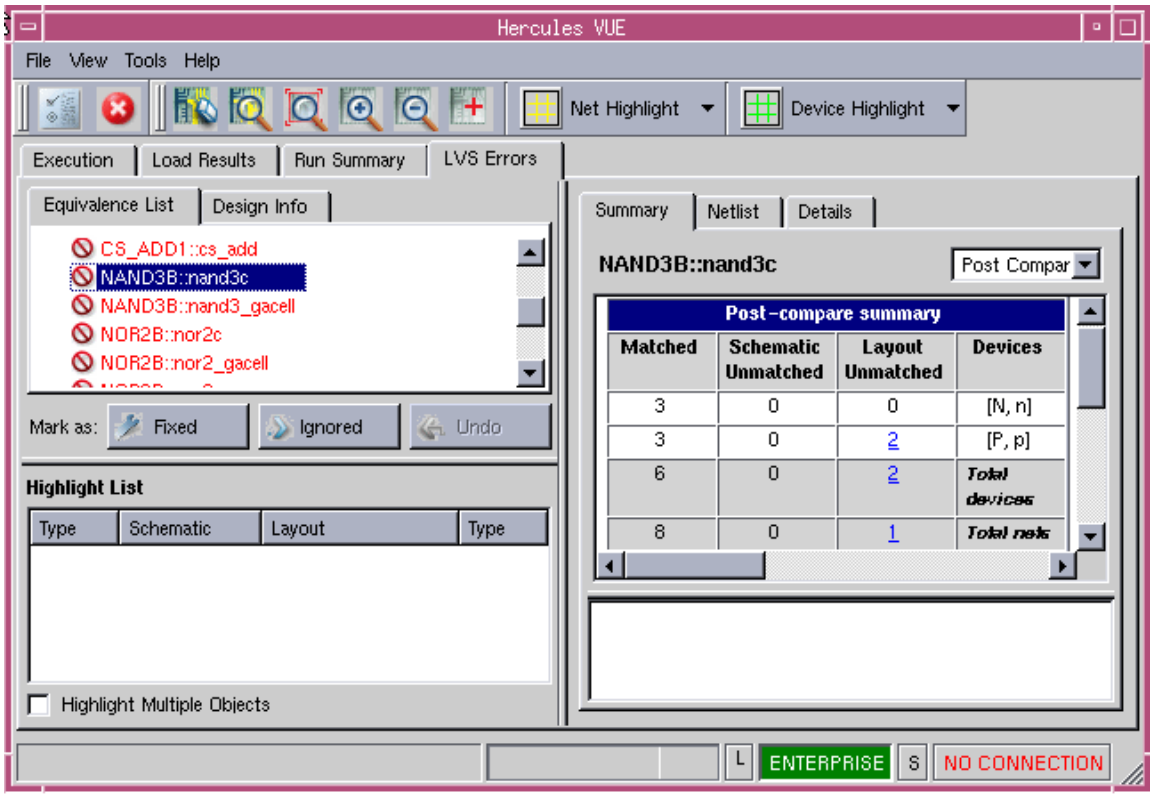
Now select the LVS Errors tab. message area. Figure 9-14 shows what you should see in your window.

Figure 9-14 LVS Errors Tab



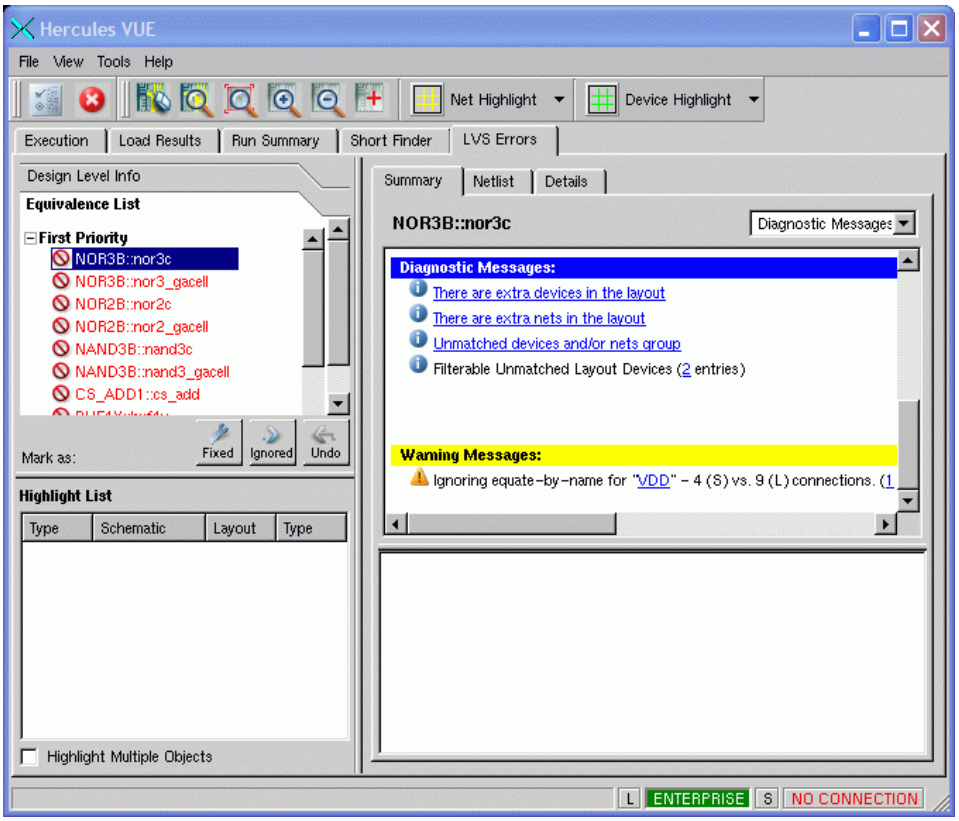
Select the red highlighted section of this message, NAND3B::nand3c. This opens a block that demonstrates the error. Figure 9-15 shows the updated screen after this operation.

Figure 9-15 VUE Window After Selecting NAND3B::nand3c



The Summary tab loads the Compare Errors with details of the unmatched nets/device in the layout/schematic. Scroll down to view additional summaries or use the pull-down tab to jump to the sections. Select Diagnostic Messages to get to the Filterable Unmatched Layout Devices. If you select 2 Filterable Unmatched Layout Devices (shown in Figure 9-16), you notice that all the suggested options for PMOS devices are listed in a table in the bottom VUE window.

Figure 9-16 Filterable Unmatched Layout Devices



If you look in the dac96lvs4b.ev runset (shown in Example 9-8), you see that there are filter options for NMOS (NMOS-3 and NMOS-8), but not PMOS. The table in your VUE window suggests that PMOS-3, PMOS-16, and PMOS-21 are required for the PMOS devices. Before adding or removing any filter options, you should always check with your design rules to make sure a particular filter option is allowed, or if the design needs to be modified to fix your compare problems.

If look at dac96lvs4c.ev, you see that we have added the PMOS-3 and PMOS-8 filter options to the EQUATE commands. PMOS-8 is not listed in the table as an applicable Filter Option and might not be necessary, but we added it to be consistent with the NMOS filter options. Now that we have corrected all of the global problems listed in the LVS Errors file, we rerun Hercules.

Note:

You can also consider the blocks with shorts and opens listed in the LVS ERRORS window as a global type of error. In most cases, however, if you have a major filtering problem, you should rerun Hercules as soon as it is fixed.

Example 9-8 EQUATE Commands in dac96lvs4b.ev Runset

```
/*=======================================================*/
/*       LVS COMPARISON SECTION OF RUNSET                */
/*=======================================================*/

/* ----------------------- Define Schematic Diode equal to layout diode
*/
EQUATE DIODE NDIO = ndio A B {
   check_properties = {area}
   rel_tolerance[area] = {5}
}

/* ----------------- Define Schematic Resistor equal to layout resistor
*/
EQUATE RES  RP1 = rp A B { }

/* ------------------ Define Schematic N Mosfet equal to layout N mosfet
*/
EQUATE NMOS N = n G S D VBB {
   check_properties={l,w}
   rel_tolerance[l]={5}
   rel_tolerance[w]={5}
   filter_options={nmos-3 nmos-8}
}

/* ------------------ Define Schematic P Mosfet equal to layout P mosfet
*/
EQUATE PMOS P = p G S D VBB {
   check_properties={l,w}
   rel_tolerance[l]={5}
   rel_tolerance[w]={5}
}
```

Using Hercules VUE to Debug Specific COMPARE Errors

Now that we have completed debugging and correcting all the LVS extract errors, reviewed for any texting problems, and corrected all global LVS compare errors, we are ready to execute Hercules again and debug specific net and device mismatch errors from the comparison phase.

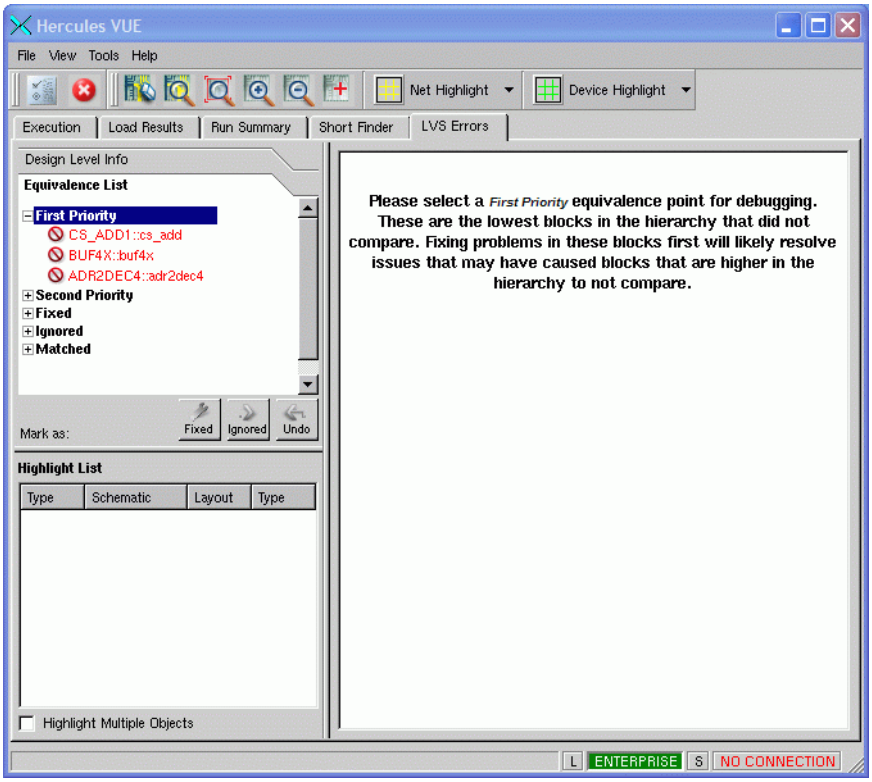
STEP 6: Fix COMPARE Errors and Rerun Hercules

Make sure you are still in the directory where your dac96lvs4c.ev file is located, *your_path*/hercules-Examples/dac96lvs4/.

To execute Hercules again, select the Execution tab from the Hercules VUE window. Select Runset file daclvs4c.ev”and click Execute.

Once the Hercules run is completed, data will be automatically loaded. Review the Run Summary tab and select LVS Errors. Figure 9-17shows the results of your latest Hercules results.

Figure 9-17 LVS Errors in VUE After Filtering Is Corrected



Where to Start Debugging LVS Errors

We now learn the process for LVS debug using a hierarchical tool. Because Hercules starts at the lowest level and compares the smallest blocks first, you should also start debugging your problems at that level. In general, the fewer devices there are in a block, the easier it is to debug.

In some cases all of the lower-level blocks have the same problem and there is no need to debug all of these blocks. Generally, a situation like this is reported as a global problem in the LVS Errors file, so you do not need to look through all of the mismatched blocks. This was the case with the filter options in this example. If you look back, you see that fixing that problem significantly reduced the number of non-equivalent blocks.

At this phase of the example, there are only a few mismatched blocks in the lower levels that are not dependent on each other, so we work our way through each of these. Hercules VUE shows these blocks as First Priority blocks for debugging. If you debug a lower-level block

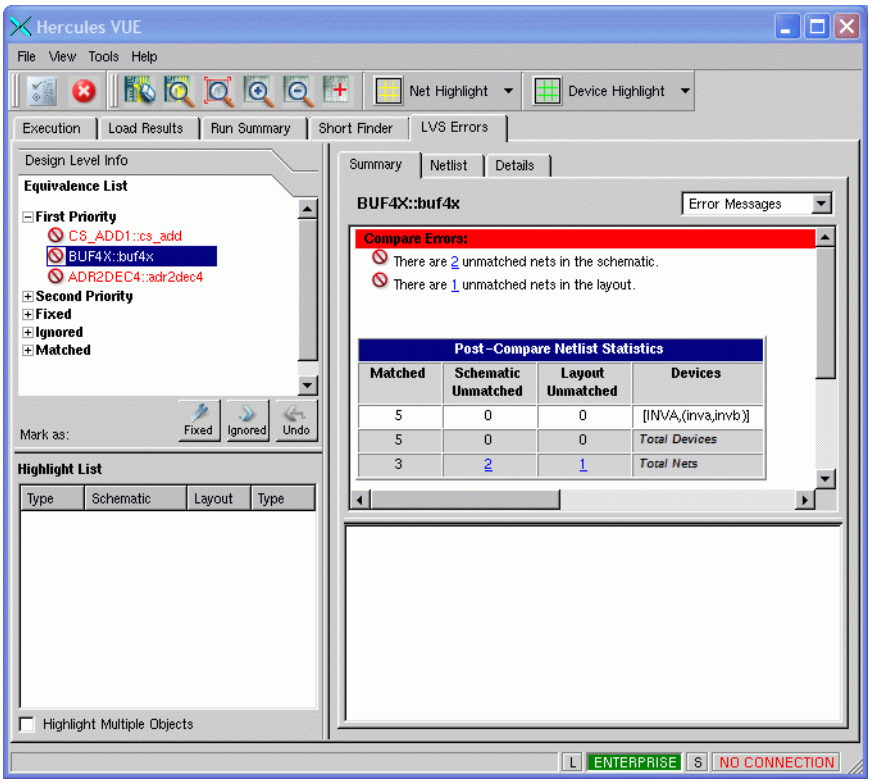
and you have a mismatched block at the next level that is dependent on that lower-level block, there is no need to debug the higher-level block. For example, buf4x does not compare and is listed at level 9. Level 8 has buf4x4 listed, and higher up is buf4x8. When we fix the problem in buf4x and it compares, then, most likely, buf4x4 and buf4x8 will also compare. We start debugging with buf4x.

Loading buf4x for Debug

Select First Priority block BUF4X::buf4x in the Hercules VUE window.

A summary of the errors is loaded in the main VUE window. If you click any of the summary information in the lists, more detailed information is shown at the bottom. Figure 9-18 shows the Hercules VUE window after you selected BUF4X::buf4x.

Figure 9-18 Selecting buf4x in Hercules VUE



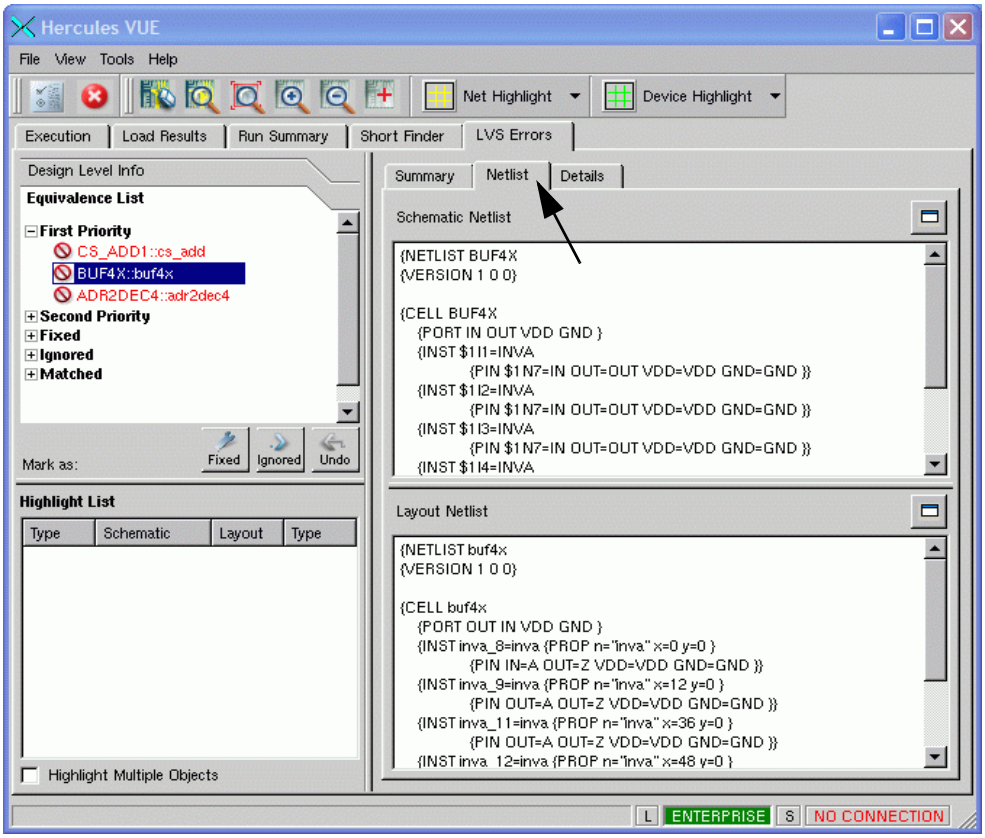
Files Available in Hercules VUE

In Chapter 7, “Introduction to Hercules HLVS,” we reviewed the summary files, explaining that these were the files for each block that contain all of the detailed error information. We also reviewed the cell-level schematic netlist (sch.block) and explained that it is a flat netlist, where all blocks that were successfully compared at a lower level are now only black-box instances.

When debugging a comparison error, it is often helpful to have these two files open. You can use these files through Hercules VUE for cross-probing and probing in the layout. The next few operations involve using these files.

Select the Netlist tab to view the Equivalence Schematic and Layout Netlist. Figure 9-19 shows the Equivalence Schematic and Layout Netlist that is opened.

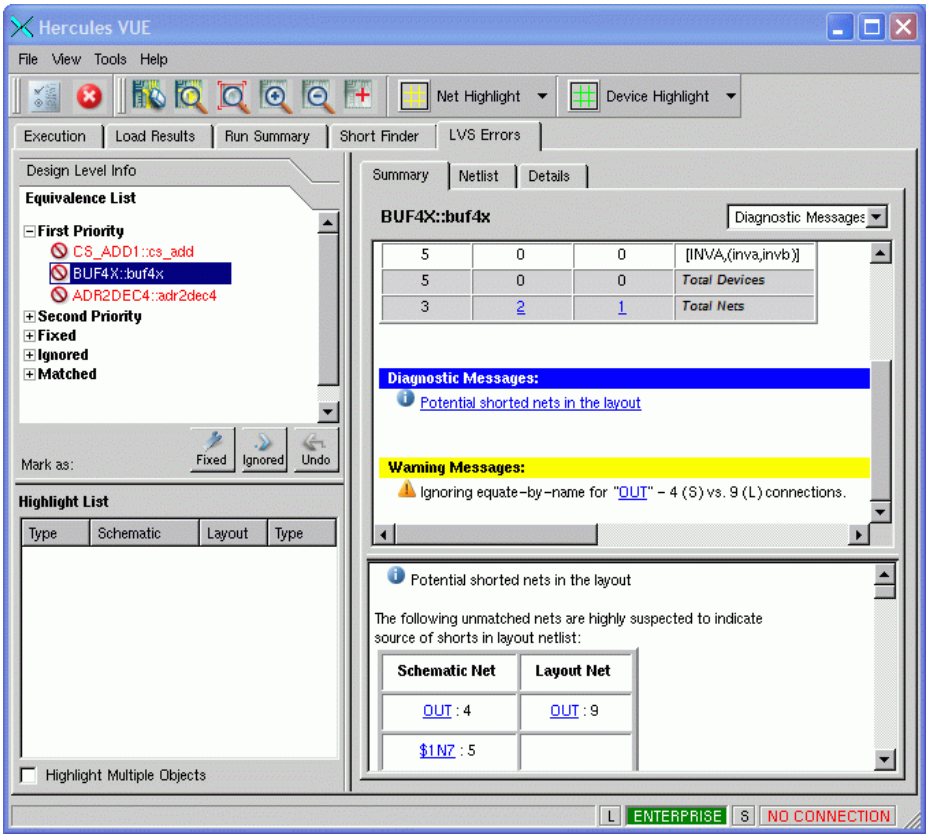
Figure 9-19 Equivalence Netlist in Hercules VUE



Select any device or net in the Netlist tab to highlight in the Schematic Netlist and Layout Netlist panes. As long as you are connected to the layout and schematic editors, selecting matched devices/nets will show them in both editors. The Highlight List provides more details on a selected device or net in the VUE window.

In the Hercules VUE window, select the Summary tab to view errors for buf4x. Block buf4x should be automatically opened in your Enterprise window. Scroll down to the Diagnostic Messages. Select the message regarding potential shorted nets in the layout. Details of this short appear in the bottom window as shown in Figure 9-20

Figure 9-20 Diagnostic Messages of the Summary Tab

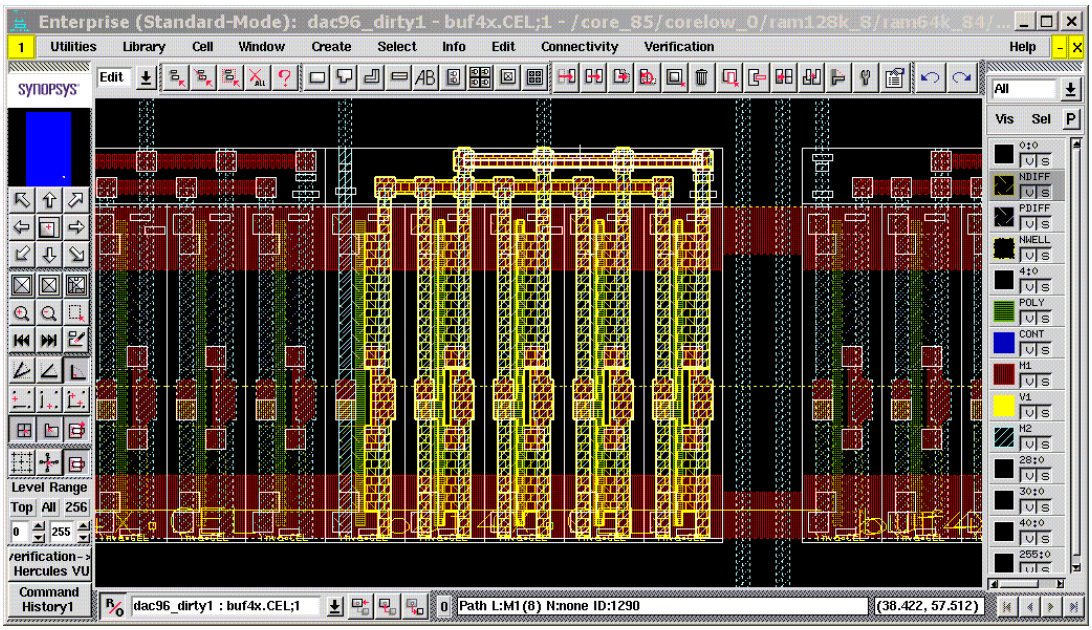


Highlighting Nets and Devices in the Layout

Based on the details in the Summary tab, it appears that there is a short between the two nets in the layout that match the schematic nets $IN7 and OUT. The easiest way to find a short between two fairly short nets is to highlight the problem in the layout.

In our example, we want first to highlight the layout net, OUT. Use your mouse to highlight the net in the table. You should see the highlighted net in your Enterprise window, as shown in Figure 9-21.

Figure 9-21 Highlight of OUT Net in Enterprise Window



Using Information About Matched Devices Connected to Unmatched Nets

Once you have identified the unmatched net in the layout, the next step is to locate matched devices that are connected to this net, in order to help determine where the short is located. If you scroll down to the bottom pane of the Summary tab, you see a list of Matched Devices Connected to Unmatched Nets. Select a device that is supposed to be connected to both. We use $1I2.

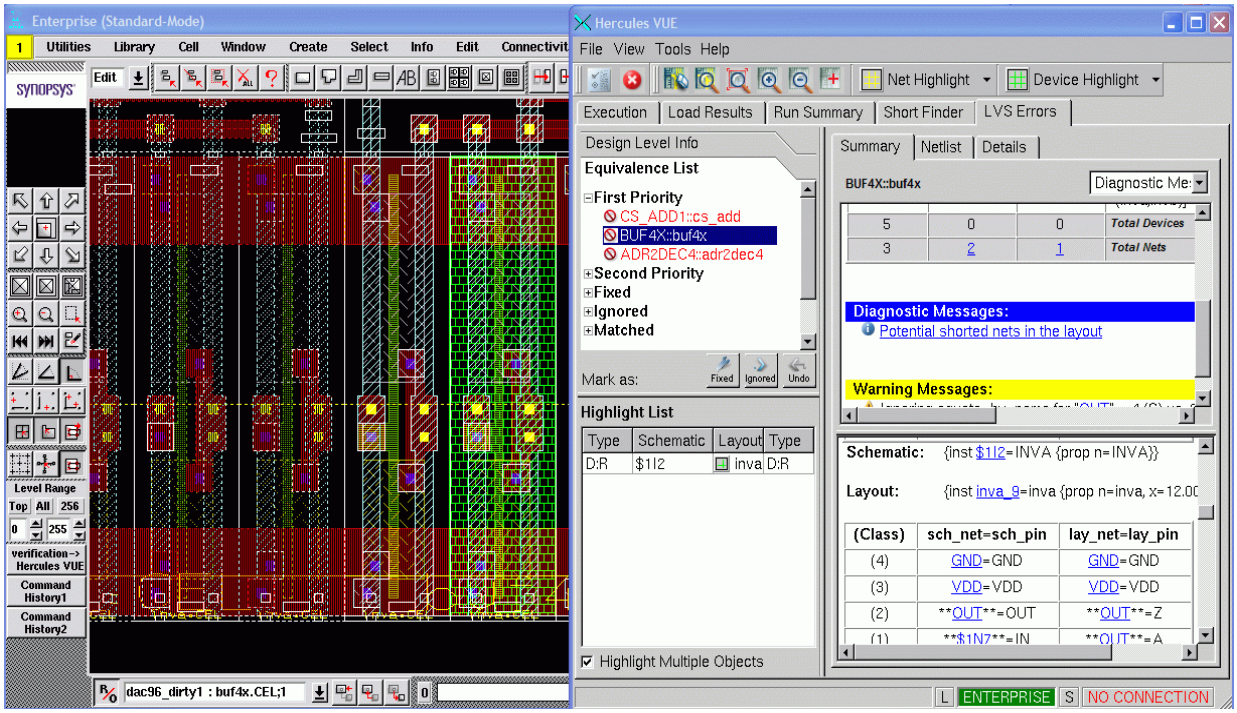
In the Hercules VUE window, click the Clear Highlights button located in the VUE Main tool bar.

Select $1I2 from Matched Devices Connected list. Figure 9-22 shows the $1I2 instance of INVA highlighted in the Enterprise and VUE windows.

Note:

You might want to re-highlight the OUT net to see the relationship between the net and the instance.

Figure 9-22 $1I2 Instance of INVA Highlighted in Layout



Analyzing the Highlight Information

Now you know the location of the OUT net in the layout and you can see how it connects to the $1I2 instance of INVA. The summary file shows that the output, Z, should be connected to OUT, and that the input, A, should be connected to $1N7. By looking at the design of the INVA cell, you can see that the input, which is tied to the gate, is supposed to be connected to the bottom net, and that the output, which is tied to the source/drain, is supposed to be connected to the top net.

There is an extra via on the output of $1I2 that shorts $1I2 to the metal 1 line, $1N7. If you have a full Enterprise license, remove this via.

Loading cs_add for Debug

The next block we load is cs_add.

Follow the same steps you used to load the buf4x block. The summary tab and Enterprise windows should automatically update with the new block information.

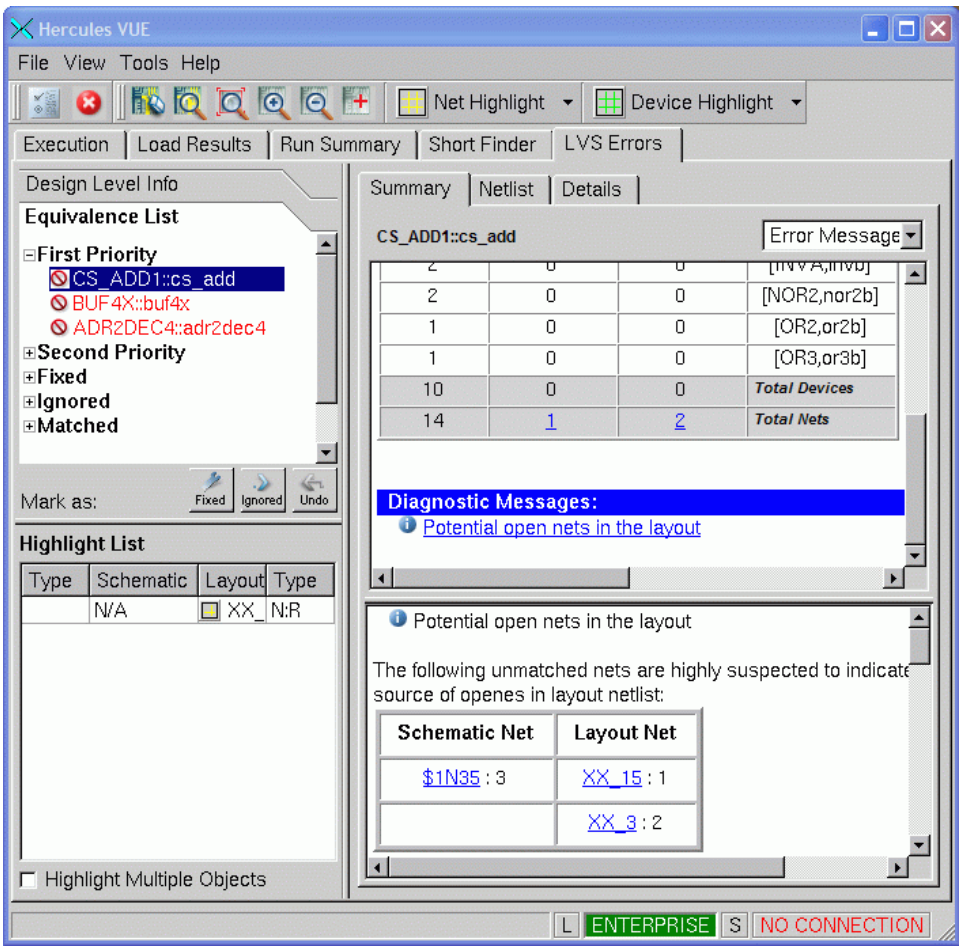
In the Summary tab, scroll to the Diagnostic Messages and look for a table similar to the one you saw for buf4x.

The block cs_add appears to have two nets in the layout and one in the schematic that are unmatched. All devices match. After reviewing the diagnostic message, you should be able to determine that you most likely have an open between XX_15 and XX_3.

Highlighting to Find an Open Between Two Nets

Select the XX_15 layout net in VUE to highlight the first net in the Enterprise as shown in Figure 9-21.

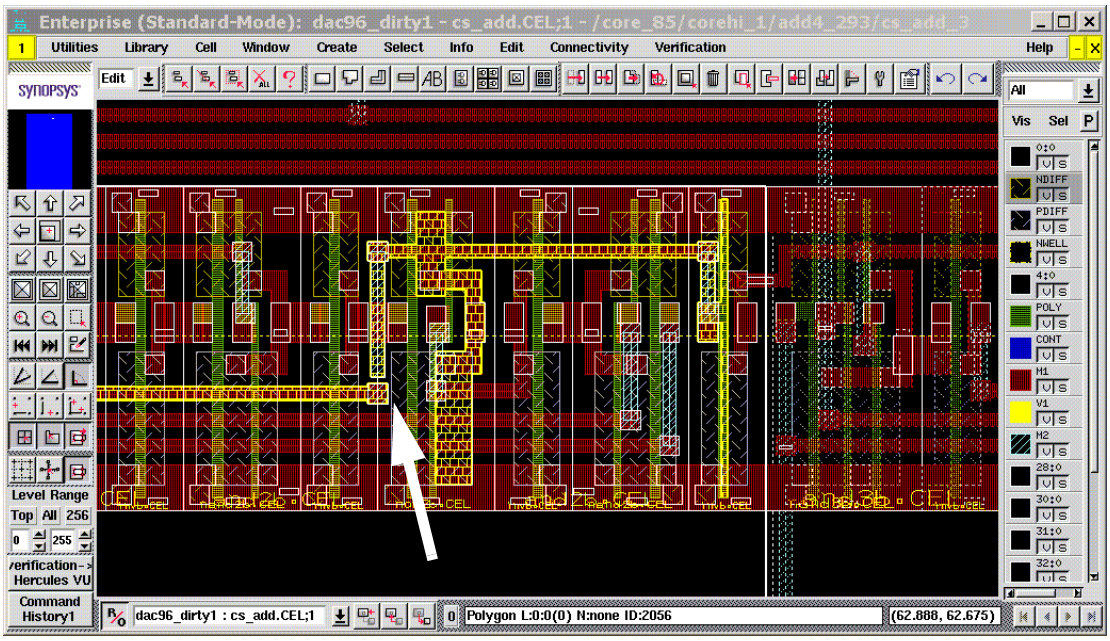
Figure 9-23 Highlighting XX_15 Net



To view multiple nets/devices highlighted, select the Highlight Multiple Objects button located at the bottom right of the Hercules VUE window. Now, select the XX_3 net to highlight the second net in the layout.

Figure 9-24 shows the XX_15 and XX_3 nets highlighted in the Enterprise window.

Figure 9-24 Highlighted Open between XX_15 and XX_3



Notice the obvious open at coordinates x=60, y=30. By using the Enterprise command s, you can stretch the lower metal 2 line over the via. Use save cell to save your changes.

An Exercise for the Reader

We have left the last first priority block as an exercise for you. Using the information you have learned in this chapter you should be able to load the adr2dec4 block and determine the error. In the dac96lvs4 directory, the file dac96lvs4.solution contains an explanation of the error if you are not able to complete the exercise successfully. When finished, exit Hercules VUE and Enterprise.

HINT: The summary file has a section called Suspicious Connection. If you have mismatched nets that are not shorts or opens, this is the section that, in most cases, should be able to help you.

When Do You Rerun Hercules?

As we mentioned earlier, you should debug most of the lower-level blocks that do not depend on other blocks that failed to compare. In our example, once you have completed debugging the first priority equivalence errors, you would be ready to rerun Hercules.

What's Next?

Chapter 10 is for Dracula physical verification users who need to convert Dracula physical verification LVS runsets to Hercules LVS runsets and debug Hercules LVS in the Virtuoso Layout Editor environment. You learn how translate a Dracula physical verification LVS runset to Hercules, run Hercules in the Virtuoso Layout Editor environment, and debug your run using Hercules VUE interfaced to Virtuoso Layout Editor and Virtuoso Schematic Editor.

10

HLVS Migration with Hercules-VUE

In this chapter you convert a simple HLVS Dracula physical verification rule set to a Hercules runset using the translation options you learned in the chapter about Hercules DRC Migration. Following that, you run Hercules on this runset and view the results using Hercules-VUE interfaced to Virtuoso Layout Editor and Virtuoso Schematic Editor, which are the Cadence layout and schematic tools.

Summary of Progress to This Point

If you are completing the exercises in the recommended order, you should now be familiar with Hercules and its associated input and output files, know how to execute Hercules from a UNIX command line, and be familiar with the Hercules to Dracula physical verification translator, Drac2He. Now that you have the basic information, you can move on to a more realistic example in your design environment.

Learning Objectives for This Chapter

In this part of the tutorial, we simulate a real design environment so that you have a better understanding of how to apply Hercules to *your* application. You will:

- Review the overall Dracula physical verification to Hercules migration flow

- Convert a Dracula physical verification LVS rule set to a Hercules runset using Drac2He
- Set up the Hercules and Hercules-VUE menus in Virtuoso Layout Editor
- Run Hercules on the output from Drac2He, automatically translating the CDL schematic, generating an equivalence file, and completing a hierarchical LVS
- Introduce and run Hercules-VUE connected to Virtuoso Layout Editor and Virtuoso Schematic Editor to show how the error detection process can be made easier through cross-probing
- Explain in detail the LVS run from the Dracula physical verification input
- Rerun Hercules on a clean database using all of the automatically generated data from the first Hercules run

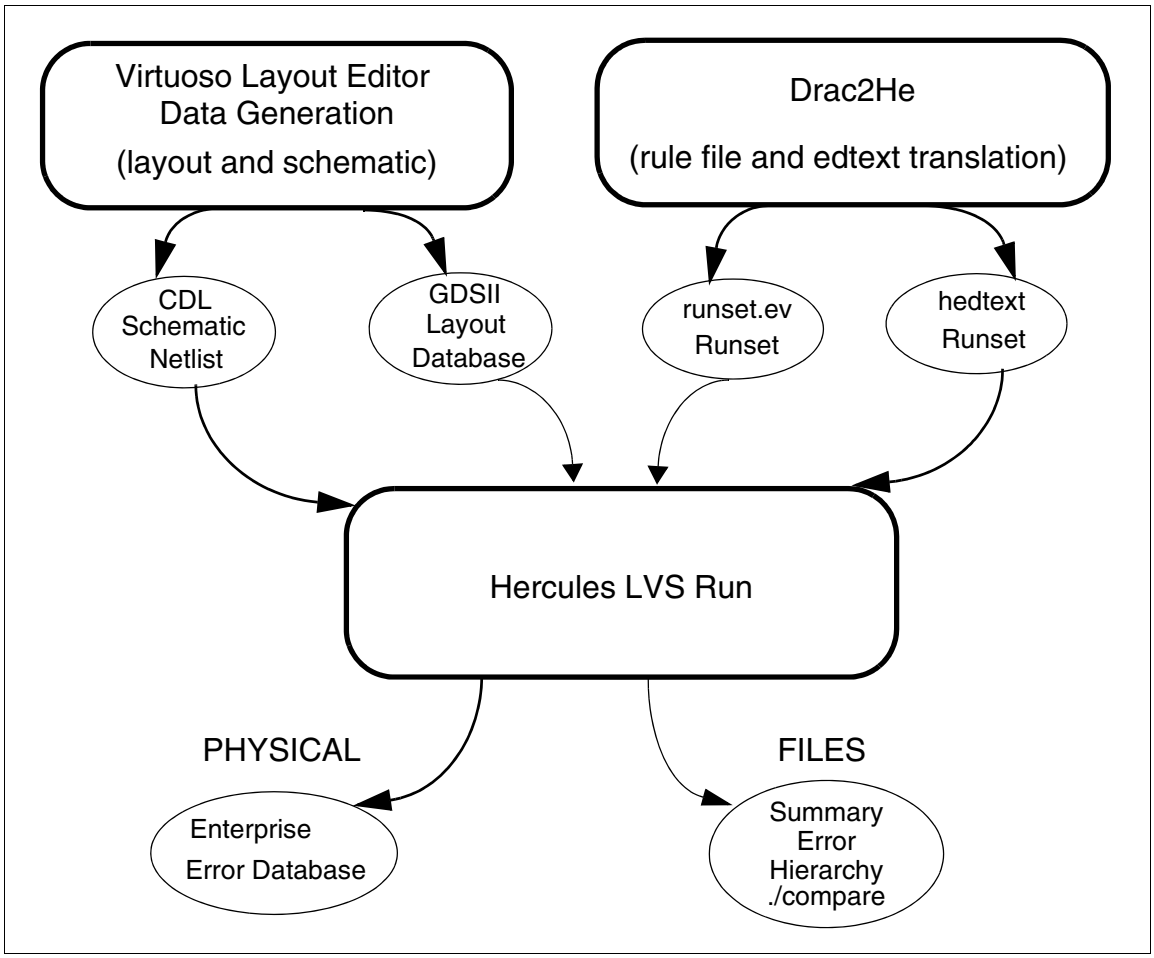
Before You Start

Before you start the tutorial, make sure that you have completely gone through the installation and setup procedure described in Chapter 1, “Installation and Setup.”

Overview of HLVS Migration Flow

In this chapter, you translate a Dracula physical verification rule file by setting the option on the command line to generate the necessary options for Hercules-VUE with Virtuoso Layout Editor and Virtuoso Schematic Editor. You also run Hercules and then view the ERROR output with Hercules-VUE in Virtuoso Layout Editor and Virtuoso Schematic Editor. Figure 10-1 shows the major steps that are reviewed in this chapter.

Figure 10-1 Dracula Physical Verification to Hercules HLVS Migration Flow



Details of Flow Diagram

The following is a brief explanation of the flow shown in Figure 10-1.

- Generate CDL schematics and a GDSII layout database from your Virtuoso Layout Editor environment. The files are provided, but you can also generate them from the AD4FUL library provided in the tutorial.
- Translate the Dracula physical verification rule file and hedtext or edtext file using the Synopsys Drac2He translator. This is the first step in this part of the tutorial.
- Run Hercules LVS using the CDL schematics, GDSII layout, translated rule file, and translated edtext file as input. First you bring up the Hercules-VUE/Virtuoso Layout Editor/Virtuoso Schematic Editor debug environment and then run Hercules LVS from this environment.

- Finally, review and, if necessary, debug the Hercules LVS results using Hercules-VUE connected to Virtuoso Layout Editor and Virtuoso Schematic Editor. There is one error in the example, which is identified and left for you to correct.

Generating a Runset With Drac2He

The first step in the flow is to translate the Dracula physical verification rule file to a Hercules runset using Drac2He.

Go to the directory that contains the LVS migration test, *your_path*/hercules-Examples/Getting_Started_Drac2he_LVS/migration_lvs

This directory contains the necessary technology files for the Virtuoso Layout Editor environment, as well as the following versions of the AD4FUL library:

- gdsii: ad4ful_error.gds
- Cadence 4.4.x library: ./lib4
- gdsii: ad4ful_fix.gds

In all three cases the top block is AD4FUL. The Cadence AD4FUL library is the same as the ad4ful_error.gds library. Use the following command:

```
drac2he -E error.out migration_lvs.drac > ad4ful_lvs.ev
```

Note:

At some point before or after the translation, make sure the library and schematic netlist files are correctly named in your Dracula physical verification rule file or Hercules runset. If you choose to edit the Dracula physical verification rule file, make sure the INDISK and SCHEMATIC variables in the DESCRIPTION section are correct. If you choose to edit the Hercules runset, you want to check the INLIB and SCHEMATIC variables in the HEADER section.

Your screen does not display data while the translator runs. When the translation is complete, ad4ful_lvs.ev contains the Hercules runset ready for execution. Before you run Hercules on this runset, you review its contents.

Translation Results for Migration.lvs Example

This runset was output from the Drac2He translation you just completed. Run Hercules from the Hercules-VUE/Virtuoso Layout Editor/Virtuoso Schematic Editor debug environment using this runset.

Example 10-1 ad4ful_lvs.ev HEADER and OPTIONS Sections

```
HEADER {
   INLIB = ad4ful_error.gds
   OUTLIB = EV_OUT
   BLOCK = AD4FUL
   GROUP_DIR = group
   FORMAT = GDSII
   OUTPUT_FORMAT = LTL
   OUTPUT_LAYOUT_PATH = .
   SCHEMATIC = adder4.cdl
   SCHEMATIC_FORMAT = CDL
}
OPTIONS {
   IGNORE_CASE=TRUE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.050
   WIDTH=0.100
   NET_PREFIX = N_
   LAYOUT_POWER = { VDD VCC }
   LAYOUT_GROUND = { VSS ground }
}
EVACCESS_OPTIONS {
   PATH = evaccess
   CREATE_VIEWS = TRUE
}
```

Translated Header and Option Sections

Example 10-1 shows the HEADER, OPTIONS, and EVACCESS_OPTIONS sections you should have in the ad4ful_lvs.ev file that Drac2He output. Notice that:

- INLIB is ad4ful_error.gds and FORMAT is GDSII.
- OUTPUT_FORMAT is LTL. Edit the runset ad4ful_lvs.ev to change OUTPUT_FORMAT to MILKYWAY for Hercules-VUE.

  The following example shows you how the new HEADER section should appear:

Example 10-2 New HEADER Section

```
HEADER {
   INLIB = ad4ful_error.gds
   OUTLIB = EV_OUT
   BLOCK = AD4FUL
   GROUP_DIR = group
   FORMAT = GDSII
   OUTPUT_FORMAT = MILKYWAY
   OUTPUT_LAYOUT_PATH = .
   SCHEMATIC = adder4.cdl
   SCHEMATIC_FORMAT = CDL
}
```

- SCHEMATIC_FORMAT is CDL. Hercules automatically translates this to the Hercules format.
- No EQUIVALENCE file is specified. It is generated automatically.
- No SCHEMATIC_GLOBALS are specified. They are generated automatically.
- In the OPTIONS section, CREATE_VUE_OUTPUT = TRUE must be set to generate the extra data necessary for Hercules-VUE to highlight errors.

The following example shows you how the new OPTIONS section should appear:

Example 10-3 New OPTIONS Section

```
OPTIONS {
   CREATE_VUE_OUTPUT = TRUE
   IGNORE_CASE=TRUE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.050
   WIDTH=0.100
   NET_PREFIX = N_
   LAYOUT_POWER = { VDD VCC }
   LAYOUT_GROUND = { VSS ground }
}
```

Translated EDTEXT and HEDTEXT Files

The Drac2He translator converts all EDTEXT and HEDTEXT (hierarchical EDTEXT) files to the Hercules EDTEXT file format. Example 10-4 shows an example of the ad4ful_text.herc file generated by Drac2He. The translator takes the input file listed by the EDTEXT or HEDTEXT command in the Dracula physical verification rule file, converts it to the Hercules format, and then appends a .herc extension to the file name. Example 10-5 shows the ASSIGN and OPTIONS sections with the EDTEXT command, and the TEXT section in the Hercules runset with these new text layers listed.

Example 10-4 Hercules EDTEXT File

```
structure INV
IN      50      255     12.500000       22.000000
structure ADFULAH
VSS     51      255     1.400000        2.900000
VDD     51      255     1.700000        45.050000
structure INV
VSS     51      255     17.300000       2.100000
VDD     51      255     15.800000       43.750000
structure DGATE
VSS     51      255     48.300000       3.750000
VDD     51      255     47.700000       45.100000
structure INVH
VSS     51      255     18.200000       2.350000
VDD     51      255     17.750000       42.850000
structure TGATE
```

```
VSS     51      255     16.750000       -5.250000
VDD     51      255     16.450000       36.150000
```

Example 10-5 Hercules Runset ASSIGN, EDTEXT, and TEXT Sections

```
/*  BEGIN INPUT LAYER BLOCK  */
/*  *INPUT-LAYER  */
ASSIGN {
   PWELL (31)  /* PWELL = 31  */
   TOX   (1)   /* TOX = 1  */
   POLY  (5)   /* POLY = 5  */
   CONT  (6)   /* CONT = 6 ;  */
   MET1  (8)   text(100)  /* MET1 = 8 TEXT = 100 ATTACH MET1*/
   MET2  (10)  text(110)  /* MET2 = 10 TEXT = 110 ATTACH MET2*/
   PSEL  (14)  /* PSEL = 14 ;  */
   VIA   (19)  /* VIA = 19 ;  */
   MET1_GEN_TEXT_LAYER1 (51;255) text(51;255) /* VIA = 19 ;*/
   POLY_GEN_TEXT_LAYER2 (50;255) text(50;255) /* VIA = 19 ;*/}

/*  HEDTEXT = ad4ful_text;  ATTACH TEXT file */
OPTIONS {
   EXPLIST = edtextnoexplode
   EDTEXT = ad4ful_text.herc
}
TEXT {
   MET1 BY MET1.TEXT
   MET2 BY MET2.TEXT
   MET1 BY MET1_GEN_TEXT_LAYER1.TEXT
   POLY BY POLY_GEN_TEXT_LAYER2.TEXT
}
```

Notice that under the OPTIONS section Drac2He also creates an EXPLIST file. The edtextnoexplode file contains a list of the cells in the EDTEXT file, telling Hercules not to explode these cells for any reason. Exploding the cells prevents the text in the ad4ful_text.herc EDTEXT file from being attached. For more details on EXPLIST files, see the *Hercules Reference Manual*.

We go over the automatic steps in the Hercules LVS run later when we execute Hercules. First, we set up the Virtuoso Layout Editor environment.

Netlisting Consistency Between Virtuoso Schematic Editor and Hercules-VUE

When you generate your CDL netlist from Virtuoso Schematic Editor, single-letter prefixes are automatically generated and attached to each instance name and device name (if they do not already exist in Virtuoso Schematic Editor) to make the netlist SPICE-compatible. For example, all MOSFETs are prefixed with an M and all instances are prefixed with an X.

When Hercules-VUE tries to cross-reference the device names in the schematic netlist generated by NetTran (from the CDL), it is not able to match XI10 in the netlist file to I10 in the Virtuoso Schematic Editor schematics due to this prefix.

To correct this cross-referencing problem when CDL is used and the Virtuoso Schematic Editor naming convention does not match SPICE, use the NetTran -cdl-chop option that must be set in the runset. You need to set this in the tutorial example.

Edit the ad4ful_lvs.ev file using vi or the editor of your choice. Search for the OPTIONS section, and add the following syntax:

```
NETTRAN_OPTIONS = "-cdl-chop"
```

Example 10-6 shows how your new OPTIONS section should appear.

Example 10-6 New OPTIONS Section

```
OPTIONS {
   NETTRAN_OPTIONS = "-cdl-chop"
   CREATE_VUE_OUTPUT=TRUE
   IGNORE_CASE=TRUE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.050
   WIDTH=0.100
   NET_PREFIX = N_
   LAYOUT_POWER = { VDD VCC }
   LAYOUT_GROUND = { VSS ground }
}
```

Case Sensitivity

Hercules and Dracula physical verification have different rules for working with case sensitivity. When you are streaming out from Virtuoso Layout Editor, select the PRESERVE_CASE option to guarantee that Hercules-VUE cross-probes correctly. The Drac2He translator sets IGNORE_CASE to TRUE, which does not allow you to cross-probe to the Virtuoso Schematic Editor schematic if you use mixed case. In our example, we execute Hercules directly from VUE & thus edit the runset to change this option.

```
OPTIONS {
   NETTRAN_OPTIONS = "-cdl-chop"
   CREATE_VUE_OUTPUT=TRUE
   IGNORE_CASE=FALSE
   DRACULA_DEFAULTS=TRUE
   RESOLUTION=0.050
   WIDTH=0.100
   NET_PREFIX = N_
   LAYOUT_POWER = { VDD VCC }
   LAYOUT_GROUND = { VSS ground }
}
```

Setting Up Hercules in the Virtuoso Layout Editor Environment

This section of the tutorial shows you how to set up Hercules, Hercules-VUE, the Virtuoso Layout Editor interface, and the Virtuoso Schematic Editor interface in the Virtuoso Layout Editor environment.

First, verify that your XPROBE environment variable is set. If not, set it using the command:

setenv XPROBE /tmp/xprobe-$user

When Hercules-VUE is started, it creates a file at the XPROBE location. Be sure to set XPROBE to a location where you have read/write privileges. Also, you should choose a location on a local disk drive.

Starting Virtuoso Layout Editor

Next, enter the command:

icfb &

The icfb command starts the Virtuoso Layout Editor, version 4.2.2 or higher. See the *Hercules Reference Manual* for earlier versions.

Loading Synopsys SKILL Code

In your CIW window, enter the command:

load "{$HERCULES_HOME_DIR}/etc/VUE/SkillVueMenu.il"

A user must set the HERCULES_HOME_DIR environment to Hercules version 2004.12-SP2 or higher. A successful setup of the Virtuoso Layout Editor environment with the Synopsys tools results in the following message in your *icfb* window:

```
************************************
* Welcome to Synopsys Hercules VUE Skill Interface.
* Running under Skill version 'SKILL09.10'.
*
************************************
Loading SkillVueMenu.il..
Done.
.
Loading SkillVueFilter.il..
Done.

Synopsys Hercules SKILL environment setup is done
/************************************/
```

Note:
If your SKILL load fails, include an explicit path. For example:

load "/l0/synopsys/code/etc/VUE/SkillVueMenu.il"

At this time you start a layout window in Virtuoso Layout Editor, begin your Virtuoso Layout Editor and Virtuoso Schematic Editor sessions, and then open the AD4FUL library.

Opening Your Layout

From the Tools menu in the CIW window, open the Library Manager.

If the library is not in the search path, execute the next three steps:

1. Under Library Manager, select Edit > Library Path.
2. Under Library Path Editor, select Edit > Add Library to add the library path.
3. Use the browser to locate the AD4FUL library and add it to the path.

Now you should have the AD4FUL library in your path. Before continuing, save these changes so they appear in the Library Manager.

Under Library, select AD4FUL.

Under Cell, select AD4FUL.

Under View, select layout by double-clicking.

Under View, select schematic by double-clicking.

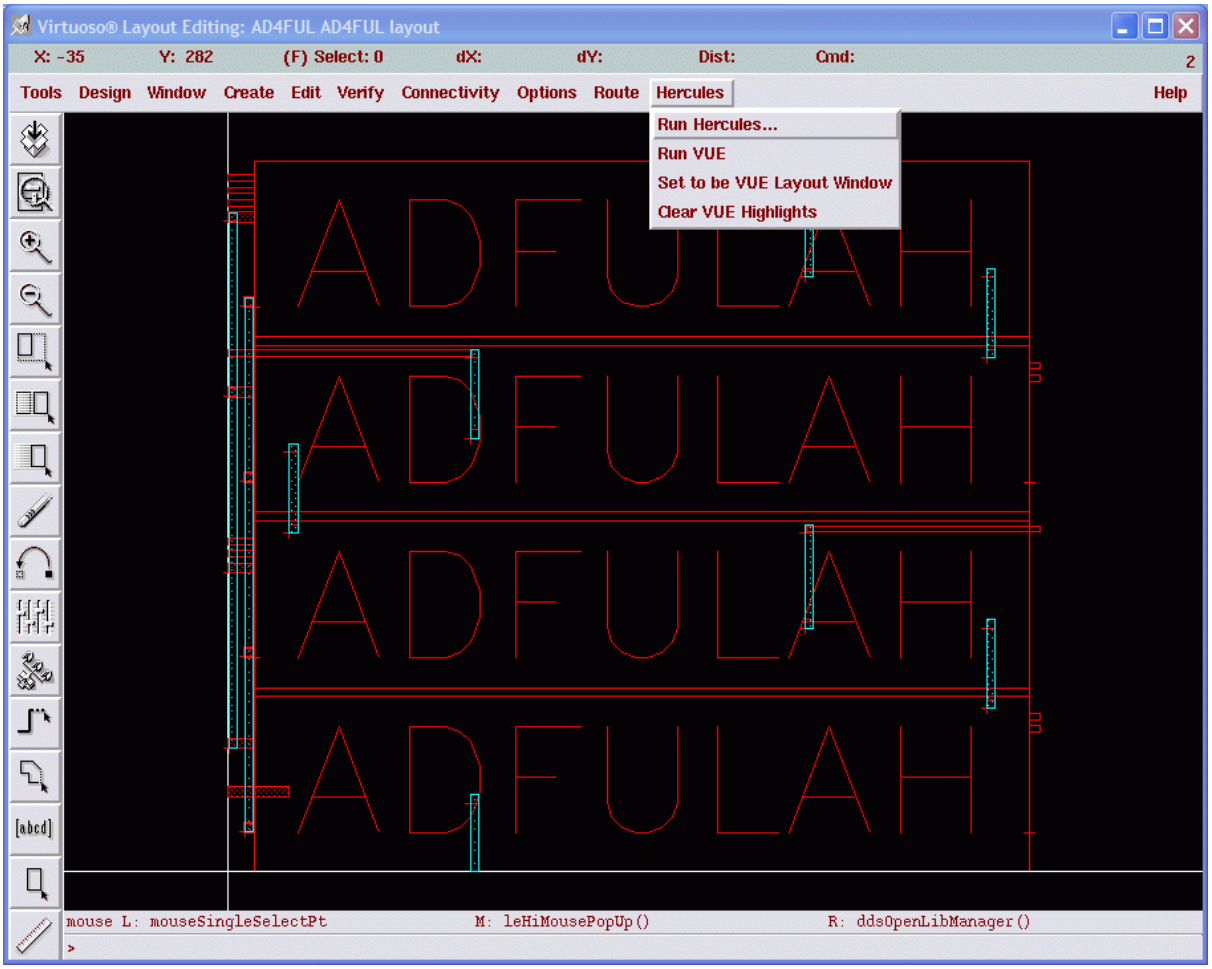
Now you should have the AD4FUL layout open in Virtuoso Layout Editor and the schematic open in Virtuoso Schematic Editor. Finally, before you can run Hercules you need to bring up the Synopsys GUI-based tools.

Connecting Hercules-VUE to Virtuoso Layout Editor and Virtuoso Schematic Editor

The next set of steps start Hercules-VUE and connect it to both Virtuoso Layout Editor and Virtuoso Schematic Editor. For successful cross-probing of nets, it is very important that Hercules-VUE be linked to both tools.

From the Hercules menu (shown in Figure 10-2), select Run VUE from Virtuoso Layout Editor layout window.

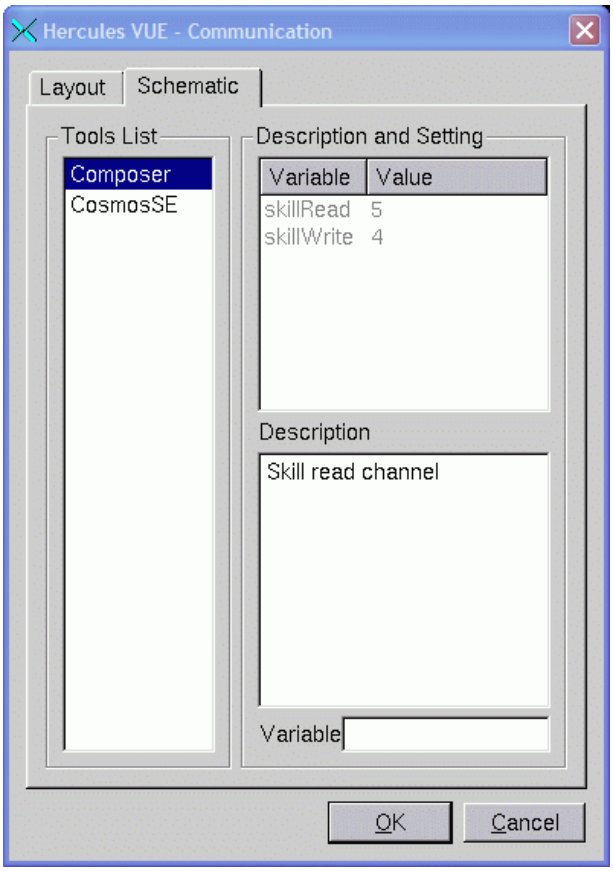
Figure 10-2 Hercules Menu



When the Hercules-VUE window starts, make sure that Virtuoso Layout Editor is connected:

To establish a connection with Virtuoso Schematic Editor, select Tools -> Communication in Hercules-VUE. Select the Schematic tab, select Virtuoso Schematic Editor, and click OK.

Figure 10-3 Setting Up a Connection with Virtuoso Schematic Editor



Hercules-VUE now shows that you are connected to Virtuoso Layout Editor and Virtuoso Schematic Editor.

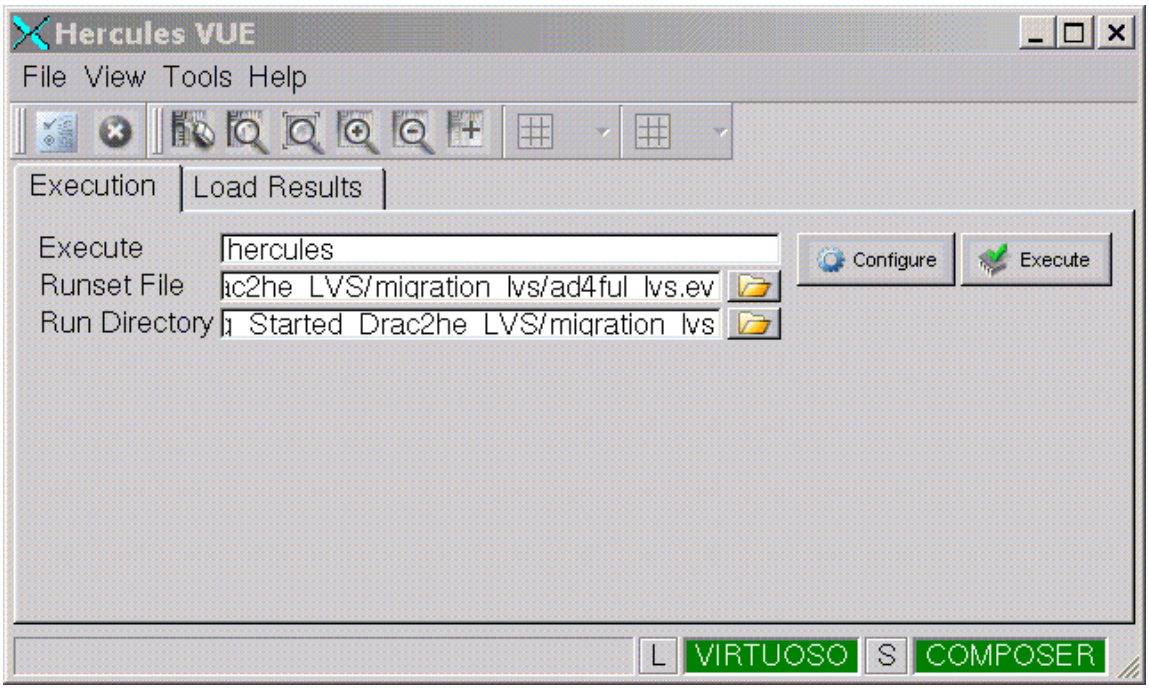
Executing Hercules LVS in the Virtuoso Layout Editor Environment

You have now set up your Hercules/Hercules-VUE/Virtuoso Layout Editor environment. The next step in the flow is to execute Hercules from this environment. Before you execute Hercules LVS, verify that all the necessary files are in your run directory.

- adder4.cdl: CDL schematic netlist, output from Virtuoso Schematic Editor (in our example we use a SPICE netlist similar to CDL)
- ad4ful_error.gds: GDSII layout, output from Virtuoso Layout Editor
- ad4ful_text.herc: EDTEXT file, output from Drac2He
- ad4ful_lvs.ev: Hercules runset, output from Drac2He

Fill in the Runset File field, ad4ful_lvs.ev, as shown in Figure 10-4. Also, verify that the Run Directory is *your_path*/hercules-Examples/Getting_Started_Drac2he_LVS/migration_lvs.

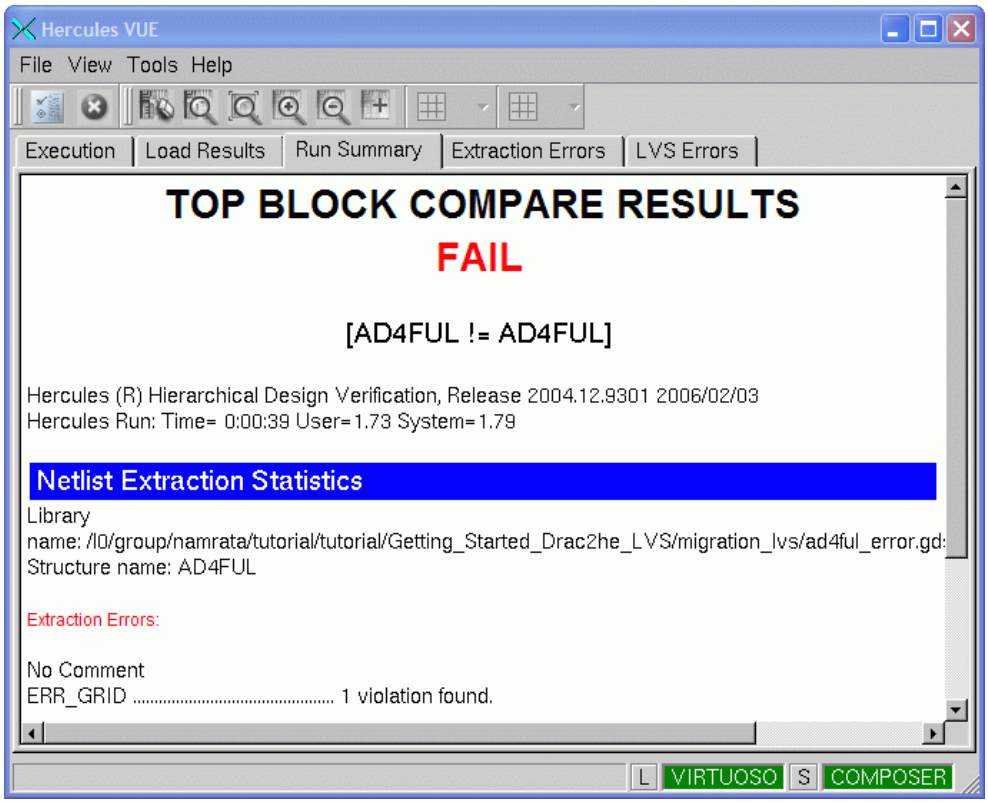
Figure 10-4 Execute Hercules Menu



Select Execute.

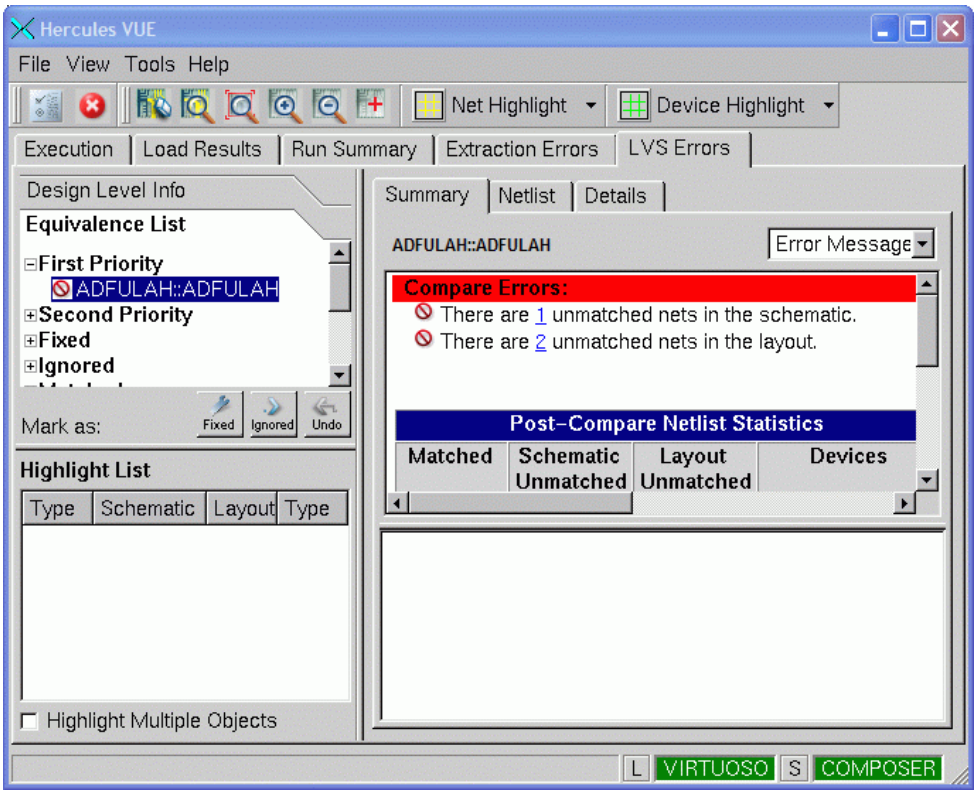
When the job completes, the block automatically loads and the window should update, as shown in Figure 10-5.

Figure 10-5 Hercules-VUE after LVS Error Data Loaded



Select the Extraction Errors tab to see one grid check violation. Now select the LVS Errors tab and select the first priority block ADFULAH. You will see LVS errors of the block ADFULAH loaded in VUE.

Figure 10-6 LVS Errors for block ADFULAH



Both the Virtuoso Layout Editor and Virtuoso Schematic Editor windows load and display the ADFULAH block as shown in Figure 10-7 and Figure 10-8.

Figure 10-7 Example of Layout Block ADFULAH

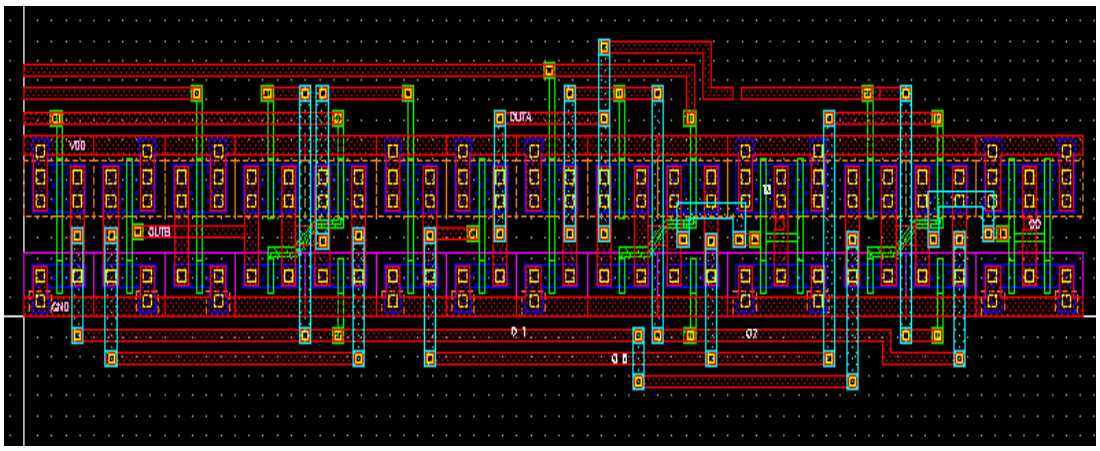
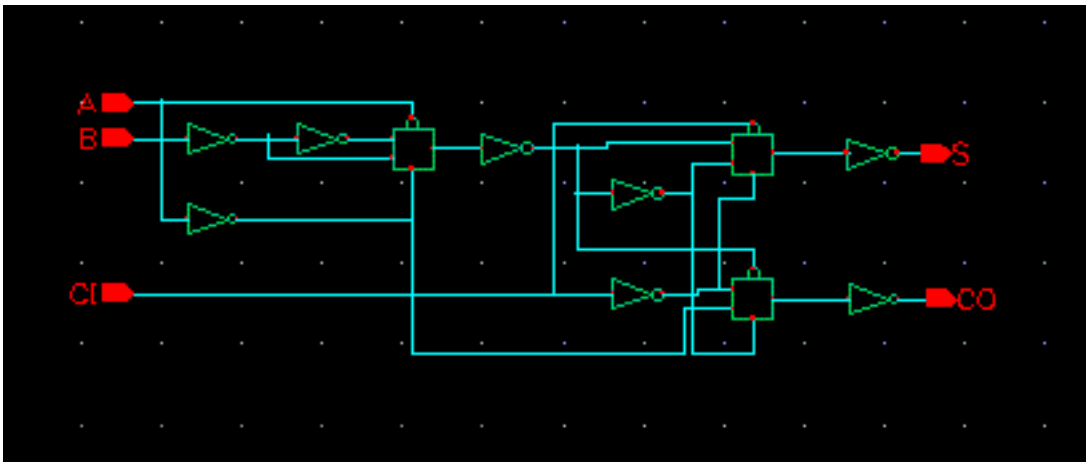


Figure 10-8 Example of Schematic Block ADFULAH



You have now set up your Hercules/Hercules-VUE/Virtuoso Layout Editor environment and executed Hercules on the ad4ful_lvs.ev runset. If you had any problems, or if want a more detailed description of the commands available in the SKILL files supplied by Synopsys, refer to the *Hercules VUE User Guide* for a complete description of the Hercules-VUE/ Virtuoso Layout Editor interface.

Highlighting in Virtuoso Layout Editor and Virtuoso Schematic Editor

Our example has one error in ADFULAH, which we use to show you how to highlight and cross-probe between your layout and schematic.

You can select a net to highlight in the Virtuoso Schematic Editor window, and if that net has a matching net in the layout, Virtuoso Layout Editor zooms to the net in the layout and highlights the matching net. The same is true for matching devices.

You can also select a net to highlight in the Virtuoso Layout Editor window and, if that net has a matching net in the schematic, Virtuoso Schematic Editor automatically highlights that matching net. The same is true for matching devices. To see the highlighted net or device in Virtuoso Schematic Editor, however, you might need to use the arrow keys, or simply refresh the window to see the highlight.

In this tutorial we explain in detail how to cross-probe using the files in Hercules-VUE. You can also Probe Point to select nets and devices to cross-probe.

Note:

In some cases the redraw command in Virtuoso Layout Editor or Virtuoso Schematic Editor might erase the highlights. If this happens, simply use the arrow keys to pan and redraw your screen.

Debugging With Hercules-VUE Connected to Virtuoso Layout Editor and Virtuoso Schematic Editor

The first step in debugging our design is to check for errors in our layout extraction. You can look at these errors using Hercules-VUE or view the AD4FUL.LAYOUT_ERRORS file. In either case, the only error you should find is a “non-90 path centerline” on POLY in the DGATE.

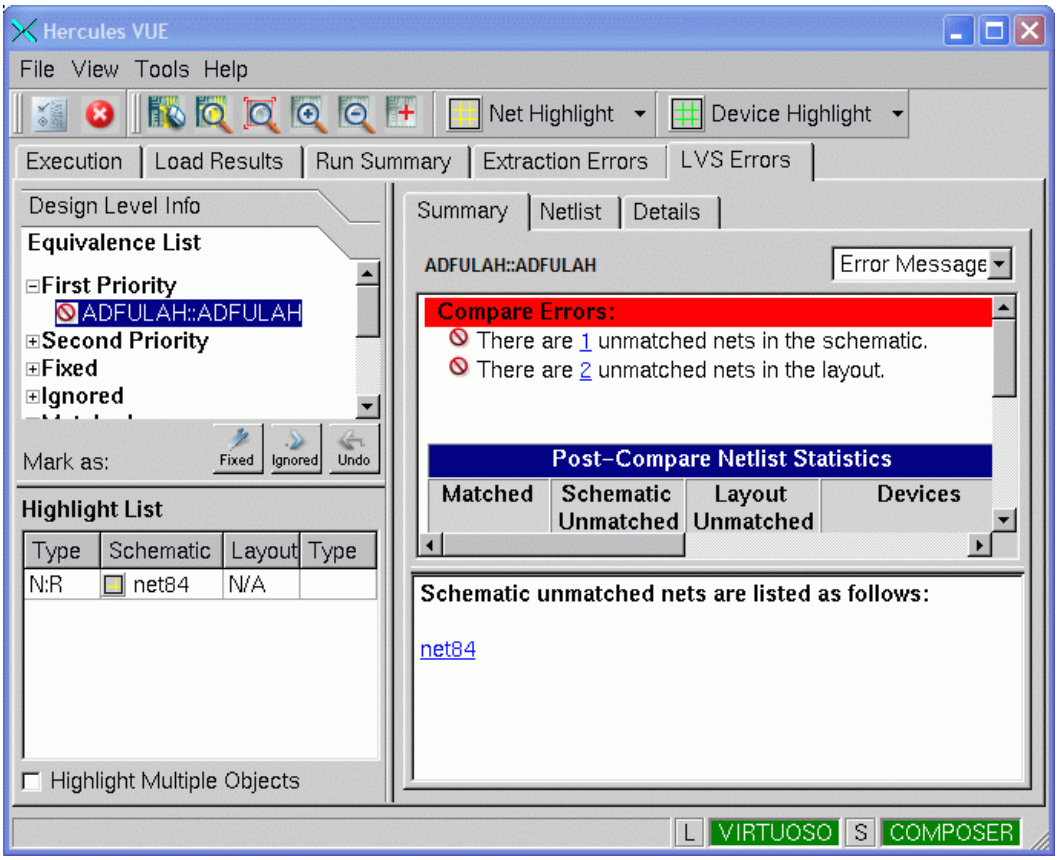
Next we debug the errors in our lowest level of hierarchy. In this case, there are errors only in AD4FUL, the top block, and in ADFULAH, the level 1 block. By fixing the errors in the ADFULAH block, we actually fix the errors in the AD4FUL block as well, because they were propagated up the hierarchy.

Examine Compare Errors

Select the first priority block ADFULAH on the left side of the Hercules-VUE GUI. This loads the summary tab with Compare Errors, Post-Compare Netlist Statistics, and Diagnostic Messages. Compare Errors tells you that there is one unmatched net in the schematic and two unmatched nets in layout.

Now select one unmatched net in the schematic. This will load details on this net at the bottom of the VUE window. See Figure 10-9.

Figure 10-9 Highlighting Unmatched Nets



Select the unmatched schematic net net84.Using the arrow keys, pan in Virtuoso Schematic Editor until you see your selected net.

Select nets N_18 and then N_7 from the Unmatched Layout Nets. At this time you should notice there is an OPEN between N_18 and N_7. A VIA sref is missing from the design. Figure 10-10 shows this missing VIA. The two highlighted nets intersect where the arrow points in the figure. This point is where the VIA sref should be located. Figure 10-11 shows the schematic net that these two layout nets should match.

Figure 10-10 Missing Via

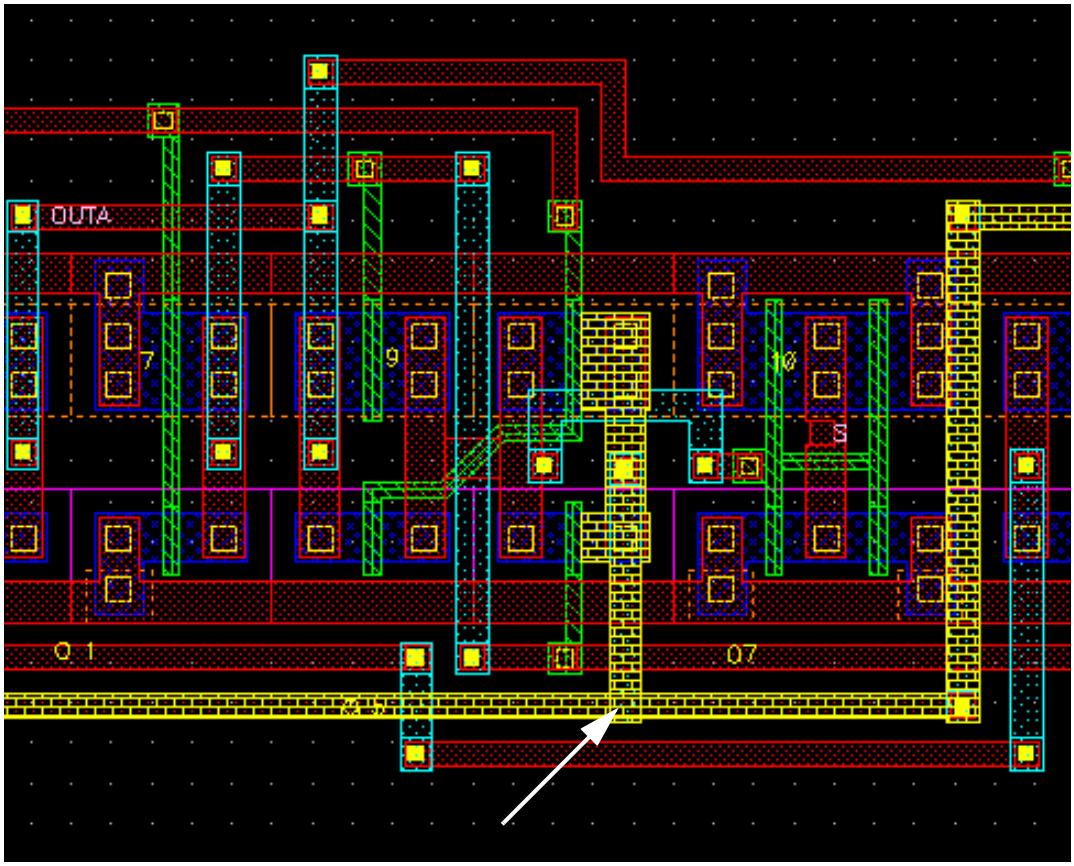
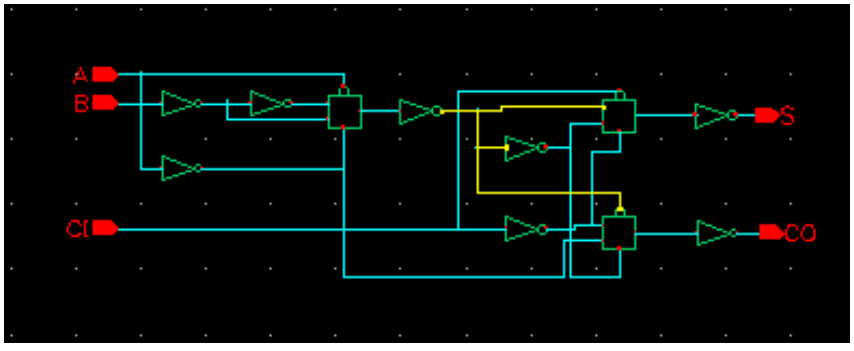


Figure 10-11 Schematic Net that N_21 and N_8 Should Match



If you would like to fix this error, you can add an SREF of the VIA cell to the ADFULAH cell and save the cell. Remember that you have to stream out a new GDS file for Hercules to see the edit. You can also simply rerun Hercules with the ad4ful_fix.gds file provided, by changing the INLIB variable in the ad4ful_lvs.ev runset to ad4ful_fix.gds. When finished, exit Hercules-VUE and Virtuoso Layout Editor.

Detailed Flow: Dracula Physical Verification Rule Files to Hercules LVS Output

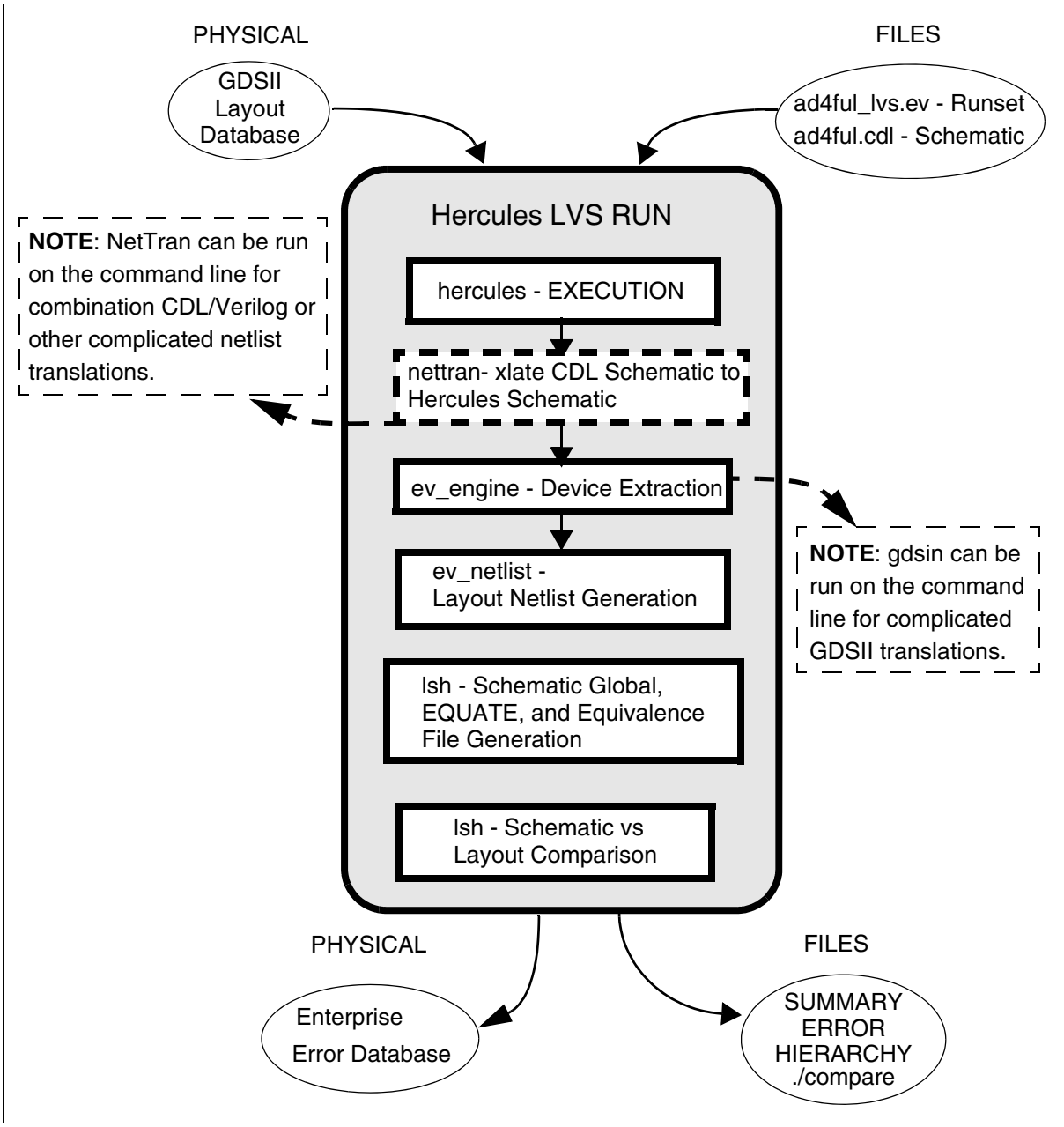
In the preceding example, all of the translation and setup from the Dracula physical verification rule file and CDL netlist to the Hercules LVS output required only the following operations:

- Translate the Dracula physical verification rule file with Drac2He.
- Stream out the LAYOUT to GDSII file.
- Generate the CDL netlist from Virtuoso Schematic Editor.
- Execute Hercules with output from Drac2He.

If you went through the LVS tutorials in Chapters 7 and 8, you know that many other steps are automatically taking place for this flow to work. For example, Hercules requires a Hercules-formatted netlist, so the CDL netlist is translated automatically. Also, the Hercules LVS uses a list of equivalence points for the hierarchical comparison. This file is generated automatically as well.

Figure 10-12 shows a flow diagram of the processes that take place automatically during the Hercules LVS execution. We go through each of these processes in detail, explaining how and when to run them individually for more complicated translation examples. We explain how to debug the results of each step if they do not complete successfully.

Figure 10-12 Detailed Hercules Execution during LVS Migration



The *block*.RESULTS file contains a summary of the steps found in this diagram. It shows the results of the run you just completed. In the following sections we describe each of these steps, the data created, possible errors you might encounter, and the proper way to debug these errors.

Example 10-7 AD4FUL.RESULTS File from a Hercules HLVS Migration Run

```
HERCULES (R) Hierarchical Design Verification, DATA OMITTED

  (C) Copyright Synopsys.  All rights reserved.

Called as: hercules ad4ful_lvs.ev

- Parsing runset "/l0/group/namrata/tutorial/03.12/tutorial/
Getting_Started_Drac2he_LVS/migration_lvs/ad4ful_lvs.ev" ...
DONE

- Checking input conditions ... DONE

- Translating input schematic netlist ... DONE

- Performing DRC checks and/or device extraction ... DONE
    ### Layout errors, refer to AD4FUL.LAYOUT_ERRORS

- Outputting Hercules netlist ... DONE

- Performing layout vs schematic comparison ... DONE
    ### LVS errors, refer to AD4FUL.LVS_ERRORS

- Creating EVaccess data ... DONE

This Hercules run has performed operations that do not need to be
duplicated in subsequent runs. Use the following Hercules
command-line option(s) to avoid that unnecessary processing:
   -s AD4FUL.sch_out

Hercules Run: Time=  0:00:21 User=5.110 System=1.46
Hercules is done.
```

Hercules Parses the Output of Drac2He as Input

The first step in the Hercules run is for Hercules to read the runset output from Drac2He and parse it. Below is a list of possible problems you might face at this stage.

- If you did not check the *error.out* file you created during the Drac2He translation, you might encounter a parser error at this point in the run, due to a translation error that you missed.
- If the paths and/or names of your GDSII or CDL files were not correct in your original Dracula physical verification rule file, Hercules is not able to find these files and generates an error.

You want to edit the Hercules runset and see if you can correct any translation errors and path or file names. If you are not sure of the reason for the translation error, contact Synopsys Technical Support with a description of the translation problem. If you feel uncomfortable editing the Hercules runset, you can fix the path or file name errors in the Dracula physical verification rule file and retranslate before running Hercules.

Reading and Converting the CDL Netlist

Hercules requires a Hercules-formatted netlist for LVS. You have specified the SCHEMATIC_FORMAT as CDL, so Hercules must call the NetTran utility to execute the CDL-to-Hercules format translation. Hercules generates a *block*.sch_out file, where BLOCK in our example is AD4FUL. This automatic process works only if you are using a single CDL file. Below are a list of other flows that you might encounter during an LVS migration.

- Multiple CDL Files
- Combination of CDL and Verilog files
- Special CDL translation options necessary for NetTran to translate your CDL netlist correctly

For any of these flows, there are two ways to run the NetTran utility with options, rather than the defaults that are set during the Drac2He translation:

1. Run the NetTran utility on the UNIX command line prior to executing Hercules. Or,
2. Set the NETTRAN_OPTIONS in the OPTIONS section of the Hercules runset.

The first way to use NetTran options is to run the NetTran utility on the UNIX command line prior to executing Hercules. In this case, you need to update your SCHEMATIC and SCHEMATIC_FORMAT variables in the HEADER section of your runset to reflect the new schematic netlist and format. You choose the name of the schematic netlist, but the SCHEMATIC_FORMAT should be changed to Hercules. For a complete list of the available NetTran options, refer to the *Hercules Reference Manual*.

The second way to use NetTran options is to set the NETTRAN_OPTIONS in the OPTIONS section of the Hercules runset. NETTRAN_OPTIONS allows you to specify a variety of options, such as multiple schematic netlists and formats, special CDL translation options, and other format translation options. For a list of the options available through the NETTRAN_OPTIONS variable, refer to the *Hercules Reference Manual*. NETTRAN_OPTIONS is listed under the section on OPTIONS and in the Alphabetical List of All Options.

Streaming in Your GDSII File With gdsin

The next step in the Hercules run is to translate the GDSII layout data to the LTL format. Hercules calls the gdsin utility for this translation. Hercules uses the default gdsin utility options. There are two ways to run the gdsin utility with options other than the default:

1. Run the gdsin utility on the UNIX command line prior to executing Hercules. Or,

   You need to update your INLIB variable to reflect the name of the LTL database you create, and set your LAYOUT_PATH to the location of the ./layout directory and your FORMAT to LTL (instead of GDSII). For a complete list of the available gdsin options, refer to the *Hercules Reference Manual*.

2. Set the GDSIN_OPTIONS in the OPTIONS section of the Hercules runset.

   GDSIN_OPTIONS allows you to specify any special gdsin option inside the Hercules runset, and therefore you do not need to run the gdsin utility on the UNIX command line. Hercules still automatically runs the gdsin utility. For a complete list of the options available through the GDSIN_OPTIONS variable, refer to the *Hercules Reference Manual*. GDSIN_OPTIONS is listed under the section on OPTIONS and in the Alphabetical List of All Options.

Device Extraction and Connectivity

After Hercules has successfully parsed the runset and translated the CDL and GDSII files, the main Hercules DRC and extraction program, ev_engine, is run. You should *not* run this program from the UNIX command line, but instead always have Hercules call this program.

Ev_engine executes all of the BOOLEAN and other data creation operations. It also extracts the devices and connects the devices and nets. Remember, your device extraction errors appear in the *block*.LAYOUT_ERRORS file or under Layout Extraction Errors in the Hercules-VUE LVS window. The errors or warnings from this step in our example are found in the AD4FUL.LAYOUT_ERRORS file.

Generic Differences in Hercules and Dracula Physical Verification Device Extraction

There are fundamental differences in how Dracula physical verification and Hercules process data; thus, in rare cases, there is no clearly defined translation from Dracula physical verification to Hercules. Specifically, the two tools do not work the same way when they are extracting devices. The next few paragraphs explain the fundamental differences between Dracula physical verification and Hercules tool definitions of device extraction layers, and some general rules to keep in mind when debugging any device extraction errors that occur using a translated runset.

Hercules and Dracula physical verification both require a recognition layer for every device extraction command. The main difference between the two tools is that the device terminal layers in Dracula physical verification do not necessarily contain the device polygons in their layers. The Dracula physical verification terminal layers might be conductor layers that only *connect* to the device terminals instead of actually *being* the device terminals. In the case of MOSFETS, the Dracula physical verification conductor layer polygons are the actual polygons that make up the device terminals, but this is not guaranteed and, in the case of DIODES and RESISTORS, is frequently not true.

Note:

You might experience some translation errors in versions of Drac2He older than 99.4 because of the use of conductor layers. If this occurs, try translating with a newer version of Drac2He.

MOSFET Device Extraction

In Dracula physical verification, a typical MOSFET extraction command looks like the one in our sample rule file, migration_lvs.drac:

```
ELEMENT MOS[N] NGATE POLY NSD NWELL
```

The POLY, NSD, and NWELL layers are called conductor layers. These are usually the actual device polygons, but they might be layers that are merely connected to the device polygons.

Drac2He translates this as follows:

```
NMOS MN_dev ngate nsd nsd PWELL
```

Because Hercules requires the actual gate polygon, the device recognition layer (ngate) is used instead of the gate conductor layer (POLY). The NSD layer is the Dracula physical verification source/drain conductor layer and is almost always the correct diffusion layer for Hercules.

Although Hercules makes a distinction between PMOS and NMOS devices, both extractions work the same way. The translator converts Dracula physical verification MOS commands to NMOS Hercules commands unless the device name has a P in it that is not preceded by N, E, or D.

BIPOLAR Device Extraction (BJTs)

In Dracula physical verification, a lateral PNP device extraction might look like this:

```
ELEMENT BJT[PL] COLL CCNT BASE EMIT
```

Here the COLL layer is the device recognition layer and the CCNT, BASE, and EMIT layers represent conductor layers for the collector, base, and emitter device layers, respectively. As with the MOS device, these layers are frequently, but not always, the same as the actual device polygons.

Drac2He would translate this device as:

```
PNP QPL_dev COLL BASE EMIT {
bjt_topology = lateral;
bjt_multi_term_layer = collector;
}temp=_generated_output_layer
```

A Dracula physical verification command for a vertical NPN device extraction might look like this:

```
ELEMENT BJT[NV] EMIT COLL BASE ECNT BULK
```

The emitter is the device recognition layer, and the conductor layers for the collector, base, emitter, and bulk are, respectively, COLL, BASE, ECNT, and BULK.

Drac2He would translate this device as:

```
NPN QNV_dev EMIT BASE COLL SUBSTRATE {
bjt_topology = normal;
}temp=_generated_output_layer
```

The BJT translation is the device extraction translation that is most likely to need manual intervention. The translator must determine the correct topology from the name of the device. If there is an L in the device name that is not preceded by a V, Drac2He converts the topology to lateral. Otherwise, the BJT device is assumed to have a normal vertical topology. Dracula physical verification is not as strict about the topology and you can specify the Dracula physical verification command incorrectly and still get a clean Dracula physical verification device extraction.

The key to identifying this problem is to look at the graphical data. If the emitter and collector polygons interact at all, the device is definitely not a lateral device. If the collector polygon is completely enclosed in the emitter polygon, the device topology is inverted.

DIODE Device Extraction

In Dracula physical verification, a diode extraction command might look like this:

```
ELEMENT DIO[P] PDIO PSD NSUB
```

The PDIO layer is the recognition layer for this device. The PSD and NSUB layers are conductor layers for the diode terminals.

Drac2He would translate this device as:

```
DIODE DP_dev pdio psd nsub { diode_type = PN }
temp=_generated_output_layer
```

The translator decides on the DIODE_TYPE by looking for the first occurrence of an N or P in the device name. Because Hercules and the Dracula physical verification have similar device definitions for diodes, there are seldom translation problems related to diodes. As with all devices, a problem might occur if the Dracula physical verification command uses layers that are merely connected to device polygon layers rather than themselves actual device polygon layers. If you are using a version of Drac2He older than 99.4 and this occurs, try a newer version.

CAPACITOR Device Extraction

In the Dracula physical verification, a capacitor extraction command might look like this:

```
ELEMENT CAP[CP] PDIO PSD NSUB
```

The PDIO layer is the recognition layer for this device. The PSD and NSUB layers are conductor layers for the capacitor terminals.

Drac2He would translate this device as:

```
CAP CP_dev pdio psd nsub {
}temp=_generated_output_layer
```

Because Hercules and the Dracula physical verification have similar device definitions for capacitors, there are seldom translation problems related to capacitors. As with all devices, a problem might occur if the Dracula physical verification command uses layers that are merely connected to the device polygon layers rather than themselves actual device polygon layers. If you are using a version of Drac2He older than 99.4 and this occurs, try a newer version.

RESISTOR Device Extraction

In the Dracula physical verification, a resistor extraction command might look like this:

```
ELEMENT RES[PD] PDIF RES_CONT
```

The PDIF layer is the recognition layer for this device and the RES_CONT layer is the conductor layer for the terminals.

Drac2He would translate this device as:

```
RES RPD_dev pdif res_cont res_cont {
}temp=_generated_output_layer
```

One situation where this translation might cause problems is where there is more than one terminal layer polygon at each end of the resistor body. If you are using a version of Drac2He older than 99.4 and this occurs, try a newer version. In rare cases, Hercules is not able to resolve the connectivity of the multiple terminals and you might need to use a SIZE OVER_UNDER command on the RES_CONT layer to merge the multiple polygons.

Layout Netlist Generation

Following the completion of the device extraction and connectivity generation, Hercules calls the ev_netlist program. This program generates the *block*.net file, which contains the layout netlist in Hercules format. The next program, lsh, reads this file along with the Hercules-formatted schematic netlist, *block*.sch_out.

In rare cases, this operation does not complete correctly. For example, you might run out of disk space during the layout netlist generation of a very large design. If this occurs, you can clean up your disk space and re-execute Hercules with the -N option, which tells Hercules to skip the device extraction and connectivity phase that already completed, and simply call ev_netlist to generate your layout netlist.

Generating Schematic Globals, Equates, and the Equivalence File

Once the Hercules ev_netlist program has completely generated the layout netlist, Hercules calls the lsh program to complete the generation of the Hercules runset needed for the schematic versus layout netlist comparison. If you look at the ad4ful_lvs.ev runset output from Drac2He, you notice that there are no SCHEMATIC_GLOBALS, SCHEMATIC_POWER, or SCHEMATIC_GROUND variables defined in the HEADER section. The EQUATE commands are generic and you did not provide an equivalence file. In Example 10-8 the generic EQUATE commands and COMPARE section in the ad4ful_lvs.ev runset are shown.

Example 10-8 Generic EQUATE Commands Generated by Drac2He

```
EQUATE PMOS auto_sch_pmos=MP_dev _auto_gate _auto_src _auto_drn
       _auto_bulk {
   FILTER=TRUE
   MERGE_PARALLEL=TRUE
   MERGE_PARALLEL_CHAINS=TRUE
   MERGE_SERIES=TRUE
   MERGE_PATHS=TRUE
   CHECK_PROPERTIES = {Length Width }
   FILTER_OPTIONS =  { PMOS-1 }
   TOLERANCE[Length]= { +5.000, -5.000 }
   TOLERANCE[Width]= { +5.000, -5.000 } }
EQUATE NMOS auto_sch_nmos=MN_dev _auto_gate _auto_src _auto_drn
       _auto_bulk {
   FILTER=TRUE
   MERGE_PARALLEL=TRUE
```

```
    MERGE_PARALLEL_CHAINS=TRUE
    MERGE_SERIES=TRUE
    MERGE_PATHS=TRUE
    CHECK_PROPERTIES = {Length Width }
    FILTER_OPTIONS =  { NMOS-1 }
    TOLERANCE[Length]= { +5.000, -5.000 }
    TOLERANCE[Width]= { +5.000, -5.000 } }
COMPARE {
    COMPARE_PROPERTIES=TRUE
    DETECT_PERMUTABLE_PORTS=TRUE
    EQUATE_BY_NET_NAME=TRUE
    EXPLODE_ON_ERROR=TRUE
    FILTER=TRUE
    MERGE_PARALLEL=TRUE
    MERGE_PATHS=TRUE
    MERGE_SERIES=TRUE
    PARALLEL_MERGE_RATIO=TRUE
    PROPERTY_WARNING=TRUE
    PUSH_DOWN_PINS=TRUE
    PUSH_DOWN_DEVICES=TRUE
    REMOVE_DANGLING_NETS=TRUE
    SHORT_EQUIVALENT_NODES=FALSE
    RETAIN_NEW_DATA=TRUE
    RETAIN_PREVIOUS_DATA=FALSE
    TEXT_RESOLVES_PORT_SWAP=FALSE
    MERGE_PATHS_DEVICE_LIMIT=30
    NET_PRINT_LIMIT=100
}
```

The lsh program creates a new directory within the directory in which you executed the hercules command. The new directory, ./lvsflow (in the run_details directory) contains a file, lvs_include.ev, that is added to the ad4ful_lvs.ev runset. Example 10-9 shows the file that was created in our tutorial example.

Example 10-9 ./run_details/lvsflow/lvs_include.ev File from the AD4FUL Migration

```
OPTIONS {
    SCHEMATIC_GLOBAL = { GND! VDD! }
    SCHEMATIC_GROUND = { GND! }
    SCHEMATIC_POWER = { VDD! }
}
HEADER {
    EQUIVALENCE = run_details/lvsflow/equiv
}
EQUATE NMOS MN=MN_DEV GATE SRC DRN BULK {
    FILTER=TRUE
    MERGE_PARALLEL=TRUE
    MERGE_PARALLEL_CHAINS=TRUE
    MERGE_SERIES=TRUE
    MERGE_PATHS=TRUE
    RESTRICT_MERGE_SERIES=TRUE
    RESTRICT_MERGE_BY_LENGTH=TRUE
    RESTRICT_MERGE_BY_WIDTH=TRUE
```

```
   USE_TOTAL_WIDTH=TRUE
   CHECK_PROPERTIES = {Length Width }
   FILTER_OPTIONS = { NMOS-1 }
   TOLERANCE[Width]= { +5.000, -5.000 }
   TOLERANCE[Length]= { +5.000, -5.000 } }
EQUATE PMOS MP=MP_DEV GATE SRC DRN BULK {
   FILTER=TRUE
   MERGE_PARALLEL=TRUE
   MERGE_PARALLEL_CHAINS=TRUE
   MERGE_SERIES=TRUE
   MERGE_PATHS=TRUE
   RESTRICT_MERGE_SERIES=TRUE
   RESTRICT_MERGE_BY_LENGTH=TRUE
   RESTRICT_MERGE_BY_WIDTH=TRUE
   USE_TOTAL_WIDTH=TRUE
   CHECK_PROPERTIES = {Length Width }
   FILTER_OPTIONS = { PMOS-1 }
   TOLERANCE[Width]= { +5.000, -5.000 }
   TOLERANCE[Length]= { +5.000, -5.000 }
}
```

You should notice the OPTIONS section that is appended to the one output from Drac2He. It now includes SCHEMATIC_GLOBALS, SCHEMATIC_POWER, and SCHEMATIC_GROUND. For a complete explanation of how these are generated, review the explanation of these variables in Chapter 4 of the *Hercules Reference Manual*.

Also notice the HEADER section that is appended to the one output from Drac2He. It now includes a path to the EQUIVALENCE file, which is generated automatically after the EQUATES are generated.

Finally, notice the new EQUATEs for NMOS and PMOS. These now reflect the correct device names and terminal names based on a search of the layout and schematic netlists.

Details on Filter Option Translation

Part of the function of the EQUATE command for each device is to specify the FILTER_OPTIONS. Filtering options in a Dracula physical verification rule file are specified by the commands FILTER-OPT, FILTER-LAY-OPT, and FILTER-SCH-OPT. If the F or G option is used in the LVSCHK command, device filtering is enabled. The F option specifies different filter options for the layout and schematic, while the G option results in the two netlists having the same filter options. There is a predetermined set of filter options for both the F and G options. The FILTER-OPT commands allow you to override the preset options. The preset options are:

LVSCHK[F]:
FILTER-LAY-OPT = BCDEFGHIJKO
FILTER-SCH-OPT = FGHIJKO
LVSCHK[G]
FILTER-OPT = BCDEFGHIJKO

Filtering options in a Hercules runset are specified by the FILTER_OPTIONS, FILTER_LAYOUT_OPTIONS, and FILTER_SCHEMATIC variables that are part of the EQUATE command. There are tables in the *Hercules Reference Manual* (in Chapter 5 under the EQUATE command) that explain each of the FILTER_OPTIONS in detail. Some of the Dracula physical verification filter options involving filtering based on path tracing to PADs are not currently supported by Hercules.

Details of Merging, Gate Formation, and Filtering Options

Hercules and the Dracula physical verification have different terminology relating to the merging of devices, gate formation, and filtering devices. Below is a list of some of these terms.

Table 10-1 Dracula Physical Verification Versus Hercules Terminology

Dracula Physical Verification Term	LVSCHK Option	Hercules Term	COMPARE Option	Note
Smash parallel devices	b	Merge parallel devices	MERGE_ PARALLEL = TRUE	Dracula physical verification merges (smashes) all MOS, LDD, resistor, capacitor, and diode devices by default, but does not merge BJT devices. MERGE_PARALLEL in Hercules merges all devices including BJTs. You must also set MERGE_PARALLEL = FALSE in the EQUATE commands for PNP or NPN to disable the merging of these devices.
Smash Pseudo Parallel Devices or MOS split-gates	S	Short equivalent nodes	SHORT_ EQUIVALENT_N ODES= TRUE	
Form CMOS gates	C	Gate Recogniti on	GATE_ RECOGNITION= TRUE	

Table 10-1 Dracula Physical Verification Versus Hercules Terminology(Continued)

Dracula Physical Verification Term	LVSCHK Option	Hercules Term	COMPARE Option	Note
Filtering Layout and Schematic separately	F	Filtering in the layout and/or schematic with separate options	FILTER=TRUE	The F option in the Dracula physical verification turns on a default set of filtering options under the FILTER-LAY-OPT command and the FILTER-SCH-OPT command. You must specify the type of layout filtering and type of schematic filtering under the EQUATE command to cause Hercules to do any filtering. The options to do this are FILTER_SCHEMATIC_OPTIONS and FILTER_LAYOUT_OPTIONS. See the *Hercules Reference Manual* for more details on filtering options.
Filtering Layout and Schematic with the same options	G	Filtering in the layout and schematic	FILTER=TRUE	The G option in the Dracula physical verification turns on a default set of filtering options under the FILTER-OPT command. The FILTER=TRUE option in Hercules only enables filtering. You must still specify the type of filtering options under the EQUATE command to cause Hercules to do any filtering. The option to do this is FILTER_OPTIONS. See the *Hercules Reference Manual* or more details on filtering options.

How the EQUIVALENCE File Is Generated

Once the ./lvsflow directory is complete and the lvs_include.ev file is created, lsh automatically generates an equivalence file. The lsh program does a complete name matching and substring name matching analysis, as well as a device and netlist count analysis to determine equivalence points. For this reason, you do not need to have schematic and layout blocks with the same names to get a complete set of equivalence points.

Because lsh might generate a number of equivalence points that do not match, your initial LVS run might take slightly longer than is optimal to work through the bad equivalence points. After the first LVS run, a ./run_details/lvsflow/*block*.ignore_equiv file is generated that includes all equivalence points that do *not* match. In the case of our example, ADDER1=ADFULAH is included in this file because that block had an error. From this file you can remove any equivalence points that you want as part of your EQUIVALENCE file and append the file to the end of the ./run_details/lvsflow/equiv file to generate a more accurate EQUIVALENCE file.

Comparing the Layout and Schematic Netlists

Hercules is now ready to perform the layout versus schematic comparison. Hercules calls lsh to run this process. The COMPARE section created during the Drac2He translation contains all the necessary changes to default settings to allow Hercules to match your Dracula physical verification results. If you have gone through the tutorial in Chapter 7, as suggested, you should recognize and understand the purpose of all the variable settings in the COMPARE section. Review Chapter 7 for a complete explanation of all the output files generated by Hercules during the comparison operation.

If you need to correct something in the COMPARE section of the runset, you can rerun only the COMPARE section using the -C option with the hercules command.

What's Next?

You have now completed a Dracula physical verification to Hercules translation, job execution, and debug example for an LVS rule file. If you plan to write Hercules runsets in the future, or if you want more details on Hercules commands, go back and complete Chapter 8.

A

Using Enterprise With Hercules-VUE

In this section you learn how to use the Hercules-VUE viewing tool in the Enterprise layout editing system. You should already have Enterprise installed on your system and ready for use. As you go through the steps, you might see alternate ways to navigate and activate various components of the GUI, and you are encouraged to try them out.

Introduction to Running Enterprise

Go to the addertest2/ directory you used for the tutorial in Chapter 3, “Single Design Error for DRC,” *your_path*/hercules-Examples/Getting_Started_Hercules_DRC/addertest2. To bring up the Enterprise window, execute:

```
Enterprise &
```

This starts Enterprise in background mode with the default start-up settings. Two windows appear, as shown in the following figures.

Figure A-1 Top Window of Enterprise

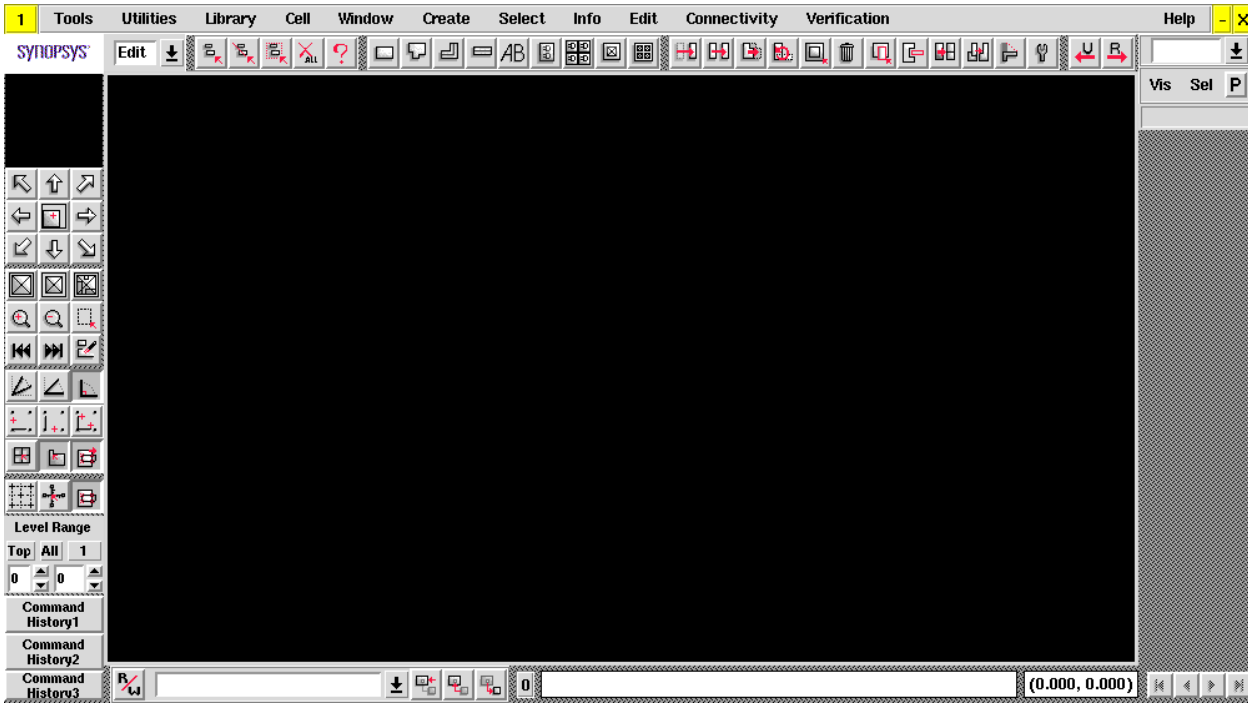
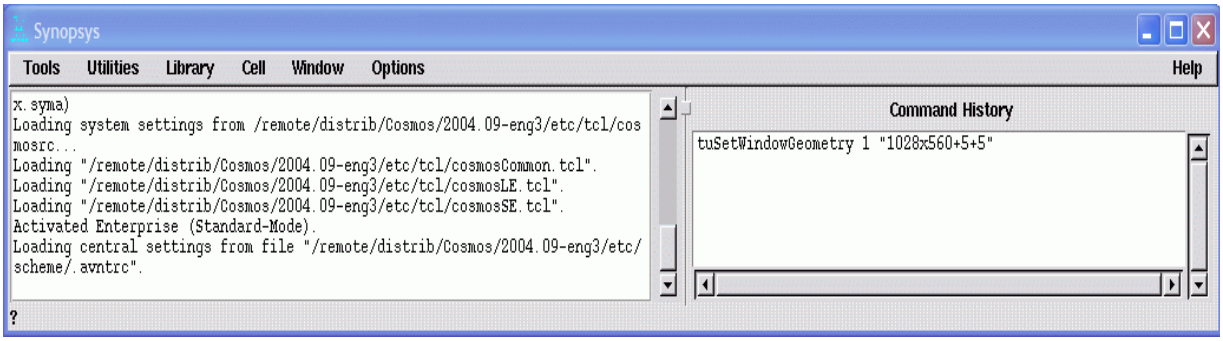


Figure A-2 Bottom Window of Enterprise



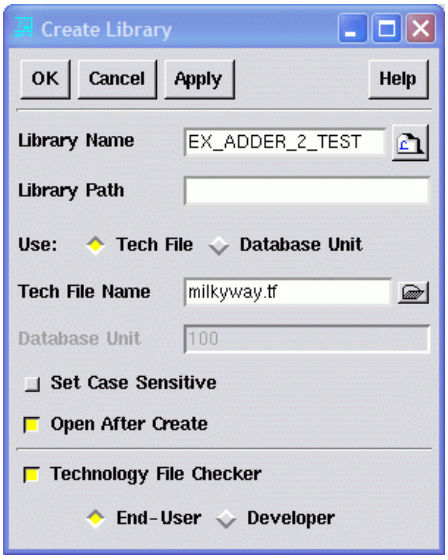
In order to stream in a library in Enterprise, create a new library and open it first.

Create Library

To create a new library in Enterprise, choose Library > Create from the pull-down menu, in either the top or bottom window.

The Create Library dialog box is displayed, as shown in Figure A-3. Fill in the Library Name and Tech File Name, as shown. The milkyway.tf tech file is in your addertest2 directory and the Library Path should also be set to the addertest2 directory.

Figure A-3 Create Library Dialog Box



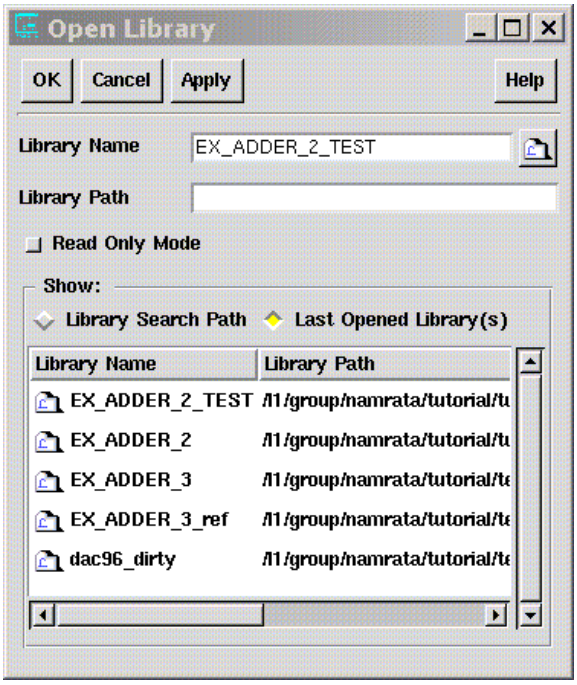
Click OK in the Create Library dialog box to dismiss it.

Open the Library

To open a library in Enterprise, choose Library > Open from the pull-down menu, in either the top or bottom window.

The Open Library dialog box is displayed, as shown in Figure A-4.

Figure A-4 Open Library Dialog Box



Note:

Enterprise keeps track of the last five libraries you opened and displays the library information in the Show: Last Opened Library(s) field. Using this list, you can open a library by double clicking its name.

Make sure EX_ADDER_2_TEST is in the Library Name field. If it is not, enter the command:

EX_ADDER_2_TEST

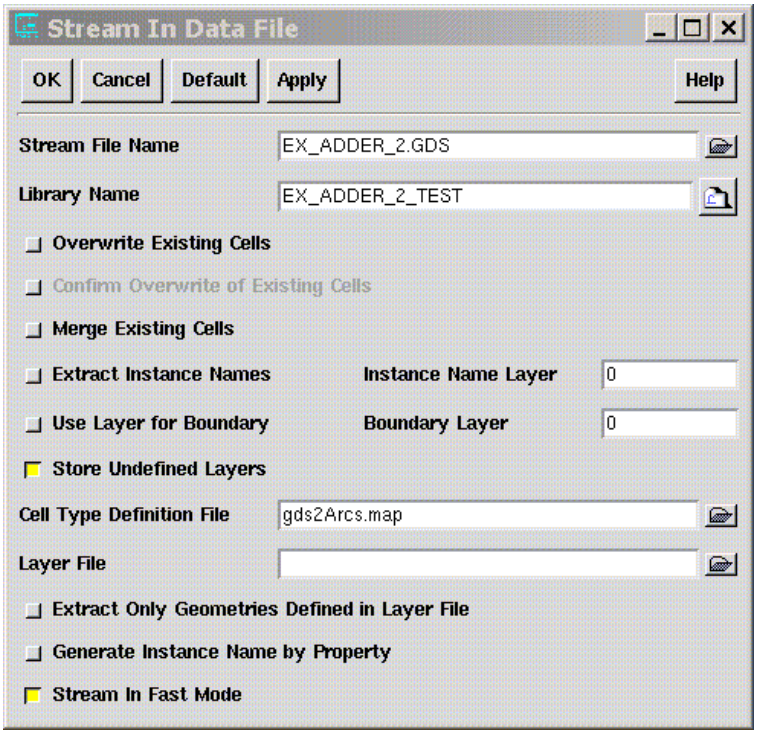
Click OK in the Open Library dialog box to dismiss it.

Stream In the Library

Now that we have created a new library and opened it, we are ready to stream the EX_ADDER_2.GDS file into Enterprise.

Using the pull-down menus in the top Enterprise window, choose Utilities > Stream In. You see an empty version of the dialog box shown in Figure A-5.

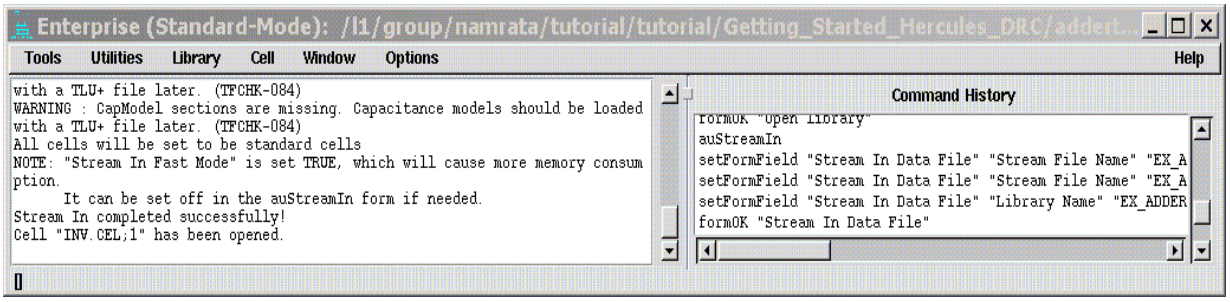
Figure A-5 Stream In Dialog Box



Fill in the text fields with the information above, including the Stream File Name and Library Name. Click OK when you are finished.

In the console window you should see the messages shown in Figure A-6.

Figure A-6 Enterprise Window After Filling in Stream In Dialog Box



Because the library was opened before streaming in, we can now open a cell from the EX_ADDER_2_TEST library.

Open a Cell

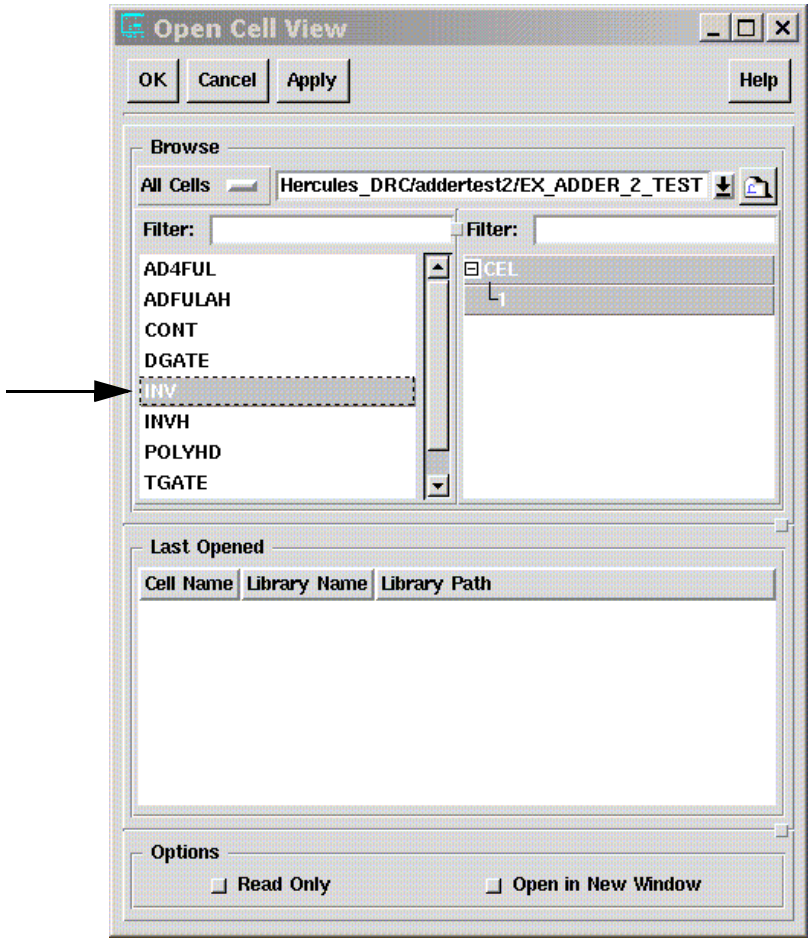
From the console window, enter:

gxwOpenCell

Or, from the pull-down menu, choose Cell > Open.

In either case, you see the Open Cell dialog box.

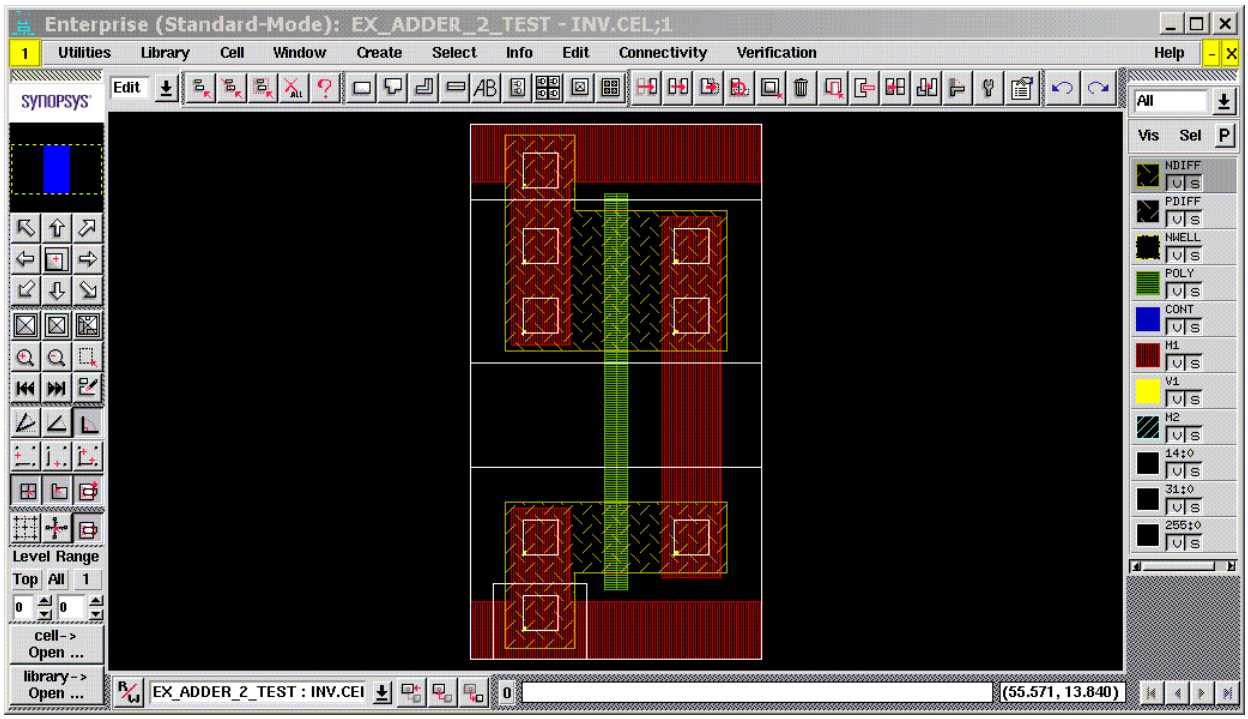
Figure A-7 Open Cell Dialog Box



Select a Structure

Select INV and click OK. The image shown in Figure A-8 appears in the main display window.

Figure A-8 INV in Main Display Window

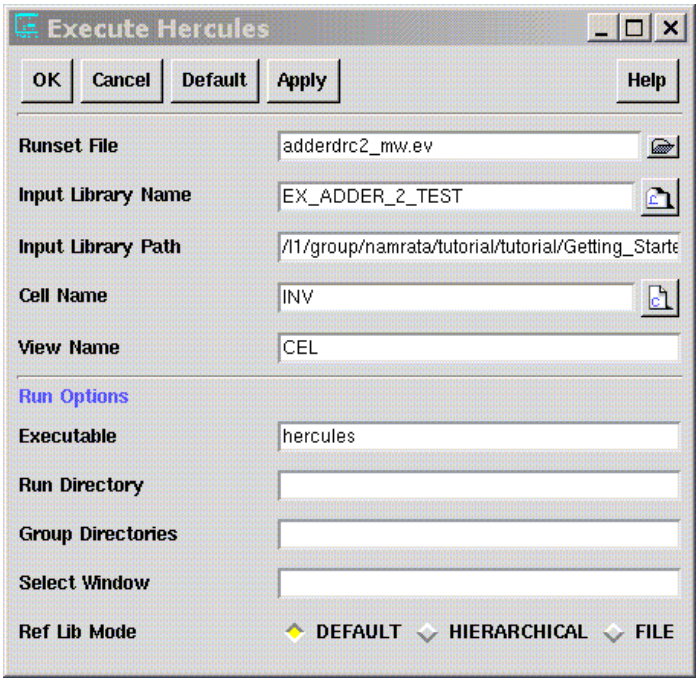


Running Hercules From Enterprise

Enterprise has a Verification pull-down menu that allows you to execute Hercules through a GUI interface.

On the right side of the menu bar, choose Verification > Hercules-User DRC Runset. You should get the Execute Hercules dialog box shown in Figure A-9.

Figure A-9 Execute Hercules Dialog Box



Fill in the runset file name with adderdrc2_mw.ev, as we did in Figure A-9. Click OK once you have filled in the runset name.

After you click OK, Hercules executes. Quickly check your UNIX shell to make sure the Hercules job is running correctly. Hercules must be able to write to the shell, so be sure you do not have any files open while Hercules is executing. Select File->Close Window to close UNIX shell.

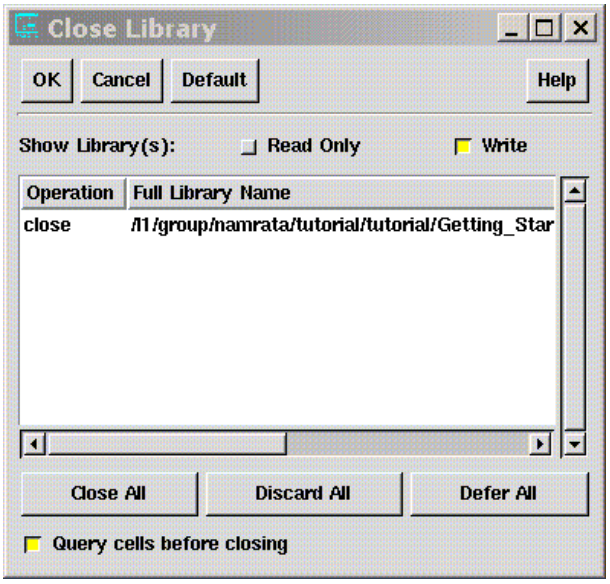
Using Hercules-VUE With Enterprise

Now that you know how to stream in and create a Milkyway database, we manually open a library and cell and execute Hercules from the Enterprise pull-down menus. We now look at how to use Hercules-VUE to open our library and cells with DRC errors automatically.

Close Any Open Libraries

Make sure that all libraries are closed. From the pull-down menu in either the top or bottom window, choose Library > Close. The Close Library dialog box appears, as shown in Figure A-10.

Figure A-10 Close Library Dialog Box



Click OK in the Close Library dialog box. You should get the following message in the console window indicating your library closed successfully:

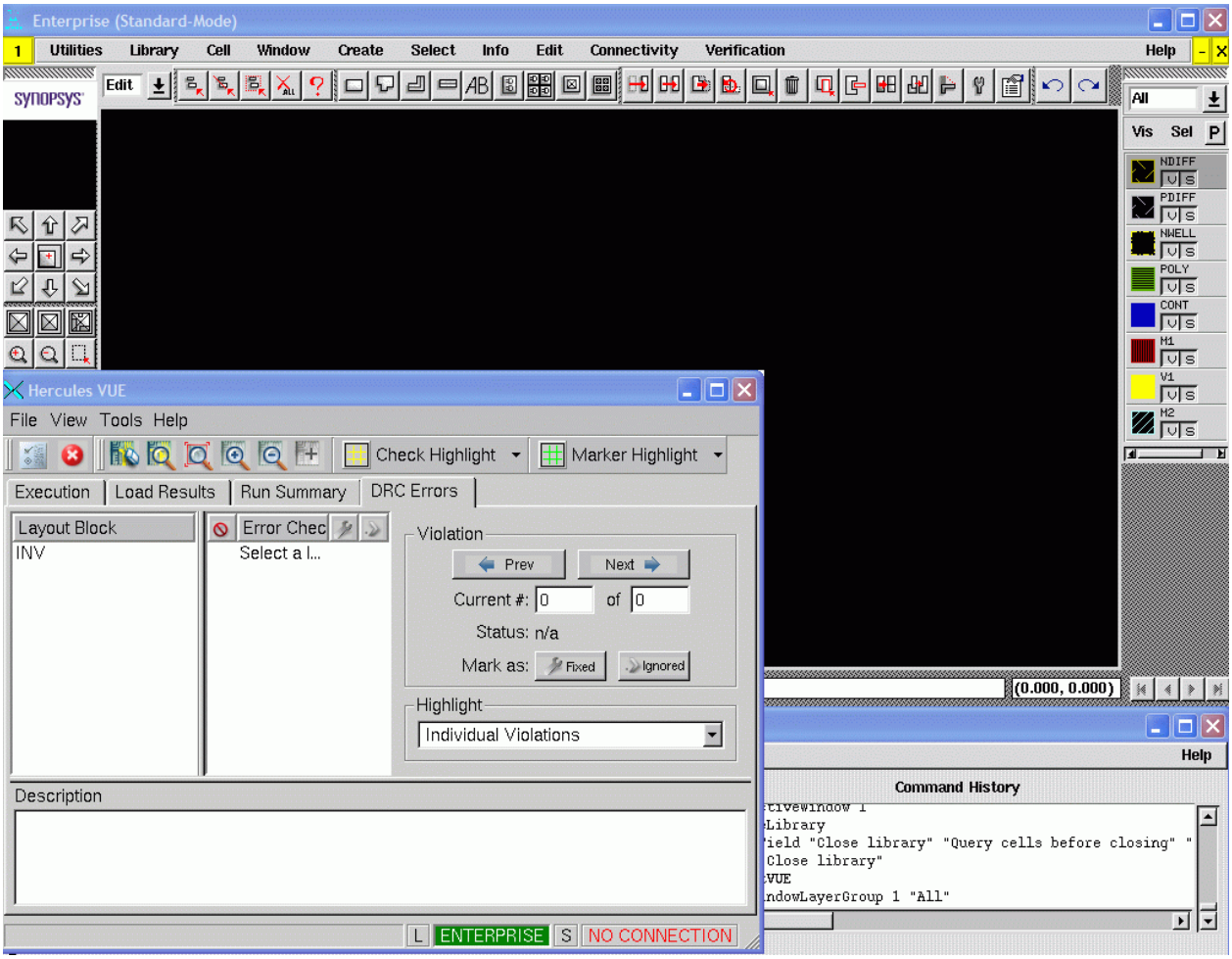
```
Saved and closed library "your_path/hercules-Examples/
addertest2/EX_ADDER_2" successfully
```

Load Hercules-VUE

In the pull-down menu of the top Enterprise window, choose Verification > Hercules VUE.

Because you already executed Hercules in your current directory, Hercules-VUE can load the Hercules output when you click on Load Results > Load to load INV.vue file. Select the DRC Errors tab once you review the run summary. You should see the following Hercules-VUE and Enterprise windows on your screen. Slide your Hercules-VUE window over so it is not overlaying the Enterprise window, as we did in Figure A-11.

Figure A-11 Hercules-VUE and Enterprise Windows

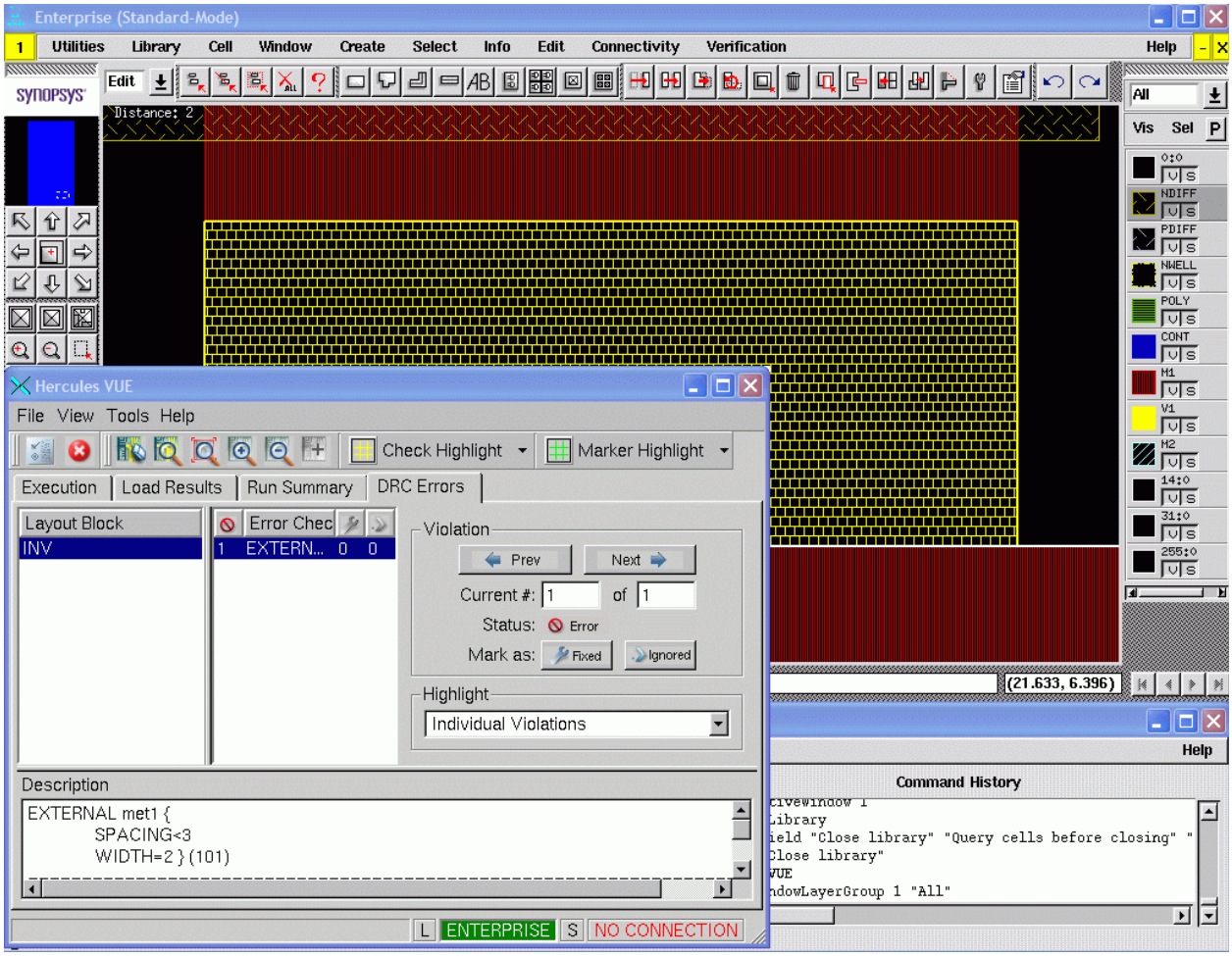


Now we check for errors.

In the Hercules-VUE window, select INV in the Layout Block field, automatically opening the EX_ADDER_2 library.

Next select 1 : EXTERNAL met1 in the Error Check field. This displays the DRC error in the layout window, as shown in Figure A-12.

Figure A-12 DRC Error Display of INV



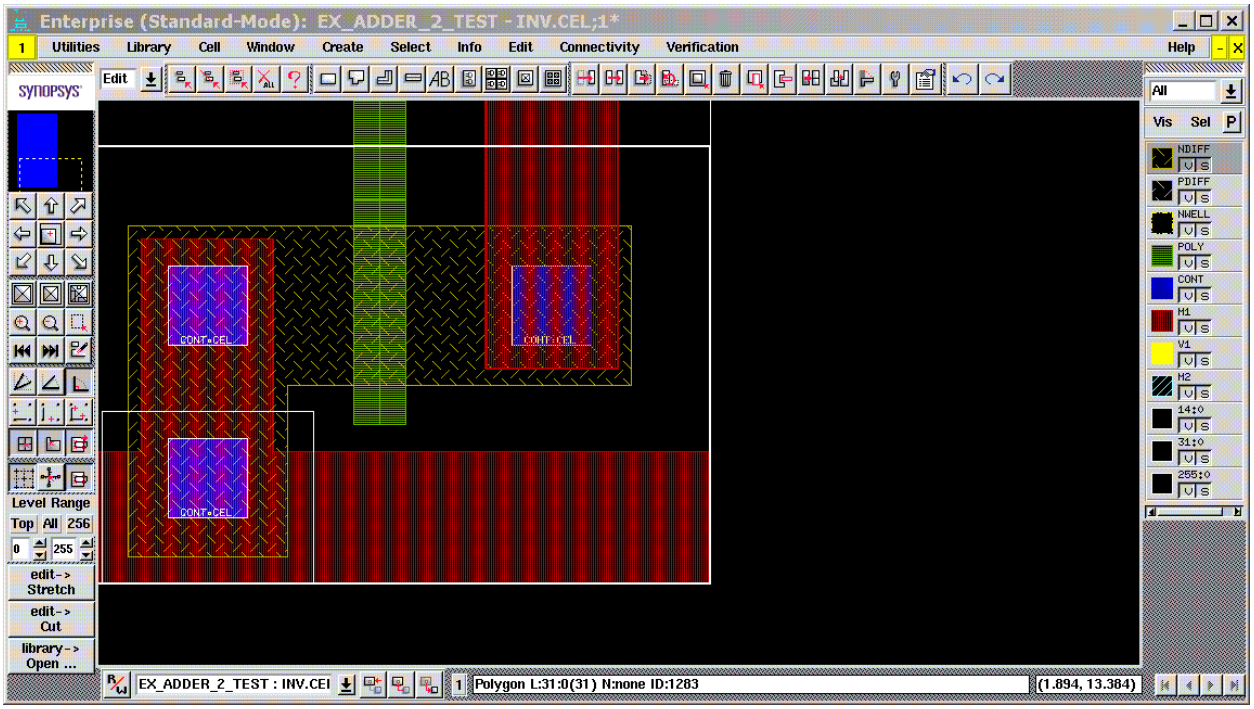
Fixing the Error

Now we fix the error and rerun Hercules from the Hercules-VUE GUI. You should still have the EX_ADDER_2_TEST library and the INV cell open. You must have the library open in write mode to edit the cell INV.

Click on the Zoom to All button located in VUE Main tool bar. This shows you the entire cell INV.

Edit the vertical M1 polygon to fix the error. You need to shorten this polygon by 1 µm. There are various ways to do this. For this example, we choose Edit > Cut in Enterprise and cut a 1-µm piece of the polygon. Figure A-13 shows how the new INV should appear on your screen.

Figure A-13 New INV



Save the INV cell by choosing Cell > Save from the pull-down menu.

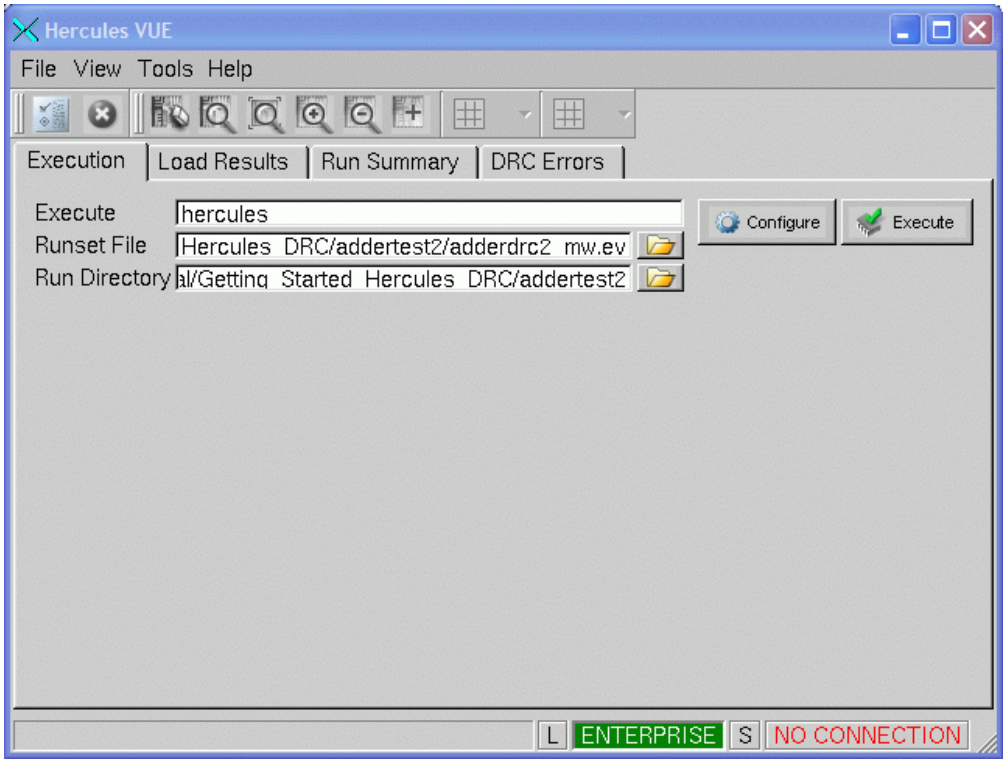
Rerunning Hercules From Hercules-VUE

Finally, we rerun Hercules from Hercules-VUE to determine whether our design is now clean.

From the top of Hercules-VUE choose Execution, select the runset and click on Execute.

Figure A-14 shows of the dialog box that should appear.

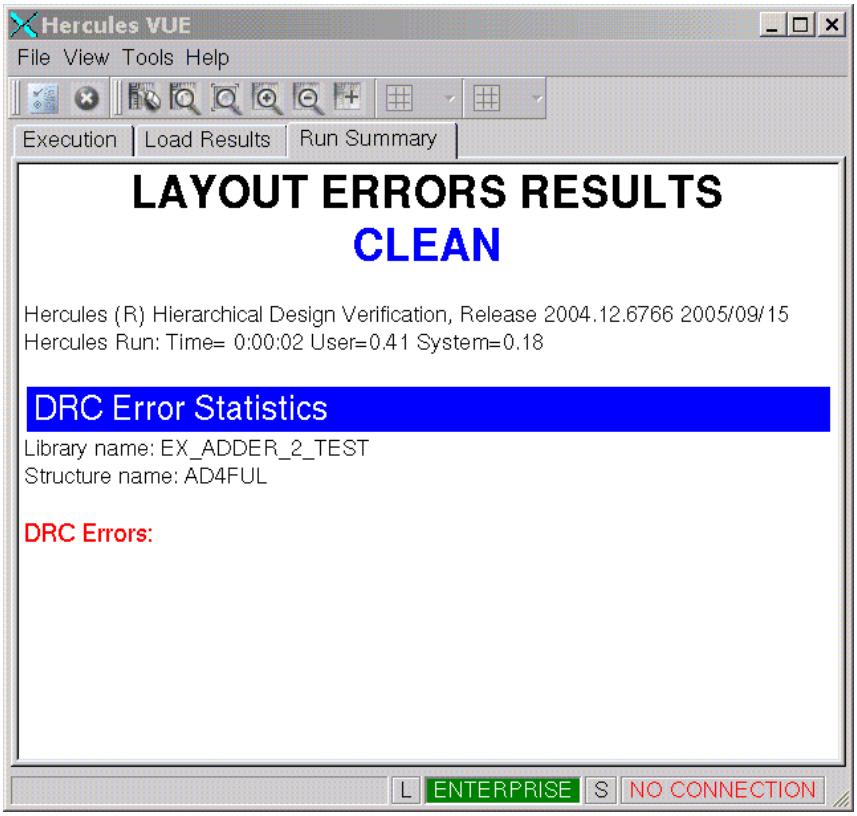
Figure A-14 Execute Hercules Dialog Box



Verify that the dialog box is correct and then click Execute at the top of the box. The Runset File should be adderdrc2_mw.ev.

Once Hercules has completed its run, the results are loaded automatically in Hercules-VUE. If you correctly fixed the design error, you should see *no errors* in the Hercules-VUE GUI, as shown in Figure A-15.

Figure A-15 Clean Hercules-VUE Results



Note:

If you did not have the Enterprise Grid or Snap feature on, you might get SNAP errors on the polygon you edited.

Summary

You have now completed an example using Hercules, Hercules-VUE, and Enterprise together to run DRC commands, debug the commands, fix errors, and then rerun in order to verify that you have a clean design. We suggest that you go through a similar exercise on your own with the runset and GDS stream file used in Chapter 4, “A Complete Design for DRC,” which is an example of multiple errors in a design.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

W

X

Z