# Hercules™
# General Usage Information

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

What's New in This Release . . . . . xiv

About This Guide . . . . . xiv

Customer Support. . . . . xvii

**1. Introduction to Hercules**

What Is Hercules? . . . . . 1-1

Hercules Features and Benefits . . . . . 1-2

How Does Hercules Fit Into My Design Methodology?. . . . . 1-2

Using Hercules in Your Design Flow. . . . . 1-3

Running Hercules within IC Compiler. . . . . 1-4

**2. Licensing and System Requirements**

Licensing Hercules and Related Products . . . . . 2-1

Licensing Software . . . . . 2-1

Which Hercules Licenses Do I Need? . . . . . 2-2

Product Licensing. . . . . 2-2

Determining the System Requirements . . . . . 2-5

Requirements for Database Space . . . . . 2-6

Output Files . . . . . 2-6

Memory . . . . . 2-6

Operating Systems . . . . . 2-8

SUN Operating System Requirements for Hercules. . . . . 2-8

HP Operating System Requirements for Hercules . . . . . 2-8

Hercules Performance In a Linux Environment . . . . . 2-9
NFS Settings . . . . . 2-10
Linux Kernel Version . . . . . 2-10

**3. Overview of Input Files**

Hercules Input Files . . . . . 3-1
Runset File . . . . . 3-1
Edtext File . . . . . 3-2
Equivalence File . . . . . 3-3
Explode List File . . . . . 3-4
Schematic Netlist File . . . . . 3-5
CELL_LIST File . . . . . 3-5
Parsing the Input Files With the EVP Utility . . . . . 3-5
Runset Flow Control and Macros . . . . . 3-8
Runset Flow Variables . . . . . 3-8
Declaring Runset Flow Variables . . . . . 3-8
Declaring Runset Internal Variables . . . . . 3-9
Declaring Environment Variables . . . . . 3-9
Using Runset Flow Variables . . . . . 3-10
Runset Flow Control Constructs . . . . . 3-11
Rules for Boolean Expressions in IF-ELSE . . . . . 3-11
Version Variable in Runset . . . . . 3-13
Runtime Flow Control . . . . . 3-14
Runtime Flow Control by Checking LAYER_EMPTY . . . . . 3-15
Runtime Flow Control by Testing TEXT_SHORT . . . . . 3-19
Macro Preprocessing for Hercules Runsets . . . . . 3-20
The gcpp Preprocessor . . . . . 3-20
Directives . . . . . 3-21
Predefined Symbols . . . . . 3-22
The MACROS Control Line . . . . . 3-27
Passing Arguments by Include Files . . . . . 3-28
Aliasing Keywords . . . . . 3-28

**4. Generating Input for Hercules**

Design Databases . . . . . 4-1
GDSII . . . . . 4-2
Using Hercules to Stream in the GDSII Database . . . . . 4-2

Using Command GDSIN to Stream In the GDSII Database . . . . . 4-6
Using Multiple GDSII Files . . . . . 4-11
Using GZIPPED GDSII Files . . . . . 4-13
Direct GDSII Output from Hercules . . . . . 4-14
GDSOUT . . . . . 4-15
OASIS . . . . . 4-18
Milkyway . . . . . 4-18
Milkyway Views . . . . . 4-18
Milkyway Objects . . . . . 4-19
Hercules Milkyway Flows . . . . . 4-19
Full . . . . . 4-19
Runsets . . . . . 4-24
DRAC2HE . . . . . 4-24
DRAC2HE Error and Warning Messages . . . . . 4-26
DRAC2HE Text Map File . . . . . 4-26
A2drc . . . . . 4-30
A2lvs . . . . . 4-32
Netlists . . . . . 4-34
Hercules Netlist Format . . . . . 4-35
NetTran . . . . . 4-37
Using Hercules to Translate into the Hercules Netlist Format . . . . . 4-37
Using NetTran to Translate into the Hercules Netlist Format . . . . . 4-38
NetTran Command-Line Syntax . . . . . 4-39
Translating Standard Format Netlists to Hercules Format . . . . . 4-44
Creating a Skeletal Equivalence File . . . . . 4-54
Creating a Wire Resolution Log File . . . . . 4-55
Using Two Steps to Complete a Netlist Translation . . . . . 4-55
NetTran Error/Warning Messages . . . . . 4-56
SPICE Netlist Format . . . . . 4-62
DEVICES . . . . . 4-62
CELL DEFINITION . . . . . 4-63
INSTANCE DEFINITION . . . . . 4-63
.GLOBAL . . . . . 4-63
.INCLUDE . . . . . 4-64
COMMENTS . . . . . 4-64
Converting SPICE Netlist Property Units . . . . . 4-64
CDL Netlist Format . . . . . 4-65
*.BIPOLAR . . . . . 4-65
*.BUSDELIMITER . . . . . 4-65

*.CAPA . . . . . 4-65
*.DIODE . . . . . 4-65
*.EQUIV . . . . . 4-65
*.RESI . . . . . 4-66
*.SCALE . . . . . 4-66
.OPTION SCALE . . . . . 4-66
Converting CDL Netlist Property Units . . . . . 4-66
Converting CDL and SPICE Netlist Formats . . . . . 4-68
Shorting Nets in CDL Netlist . . . . . 4-69
Verilog Netlist Format . . . . . 4-69
CELL DEFINITION . . . . . 4-69
PORT . . . . . 4-70
NET . . . . . 4-70
INSTANCE . . . . . 4-70
CONNECTION . . . . . 4-70
ASSIGN STATEMENTS . . . . . 4-71
BUS (VECTOR) . . . . . 4-71
Defining Global Nets During a Translation . . . . . 4-72
Verilog Global Net Mapping Table (-verilog-voltmap *filename*) . . . . . 4-72
The -verilog-busLSB Option . . . . . 4-73
Implicit Port Mapping . . . . . 4-75
General Netlist Guidelines . . . . . 4-75
Translating Multiple Netlists . . . . . 4-75
Merging Multiple Hercules-Format Netlists . . . . . 4-75
Duplicate Cell Definitions . . . . . 4-76
Case Sensitivity . . . . . 4-77
Handling Parametrized Cells (pcells) . . . . . 4-79
Three-Terminal Resistors . . . . . 4-79
GNF Error/Warning . . . . . 4-80

**5. General Hercules Execution**

Hercules Command-Line Syntax . . . . . 5-1

Hercules High Performance . . . . . 5-9
Multithreading . . . . . 5-10
Distributed Processing . . . . . 5-11
Distributed Processing Components . . . . . 5-12
Distributed Processing Autostart . . . . . 5-13
Distributed Processing Quick Start . . . . . 5-16
Dual Mode Processing . . . . . 5-16
Monitoring a Distributed Run . . . . . 5-17

Adding CPU to a Distributed Run . . . . . 5-18
Distributed Output . . . . . 5-19
Subcommands . . . . . 5-20
Combining Multithreading and Distributed Processing . . . . . 5-21
Troubleshooting Distributed Processing. . . . . 5-21
Error Messages . . . . . 5-24
Exit Codes . . . . . 5-25
Error Message Suppression . . . . . 5-26

**6. Hierarchical Verification Concepts and Processing**

Principles of Hierarchical Design . . . . . 6-1
Hierarchical Design . . . . . 6-2
Hierarchical Verification . . . . . 6-3
Efficient vs. Inefficient Hierarchy . . . . . 6-6
How Hercules Processes Hierarchy . . . . . 6-9
General Explodes . . . . . 6-11
VCELL Explodes . . . . . 6-12
VCELL Sets Pass . . . . . 6-14
VCELL Pairs Pass . . . . . 6-16
Post VCELL Explodes . . . . . 6-20
Summary Listing of Cells Created by Hercules Preprocessing . . . . . 6-21
Debugging Problems Related to Hierarchy . . . . . 6-23
Excessive Memory Use During Group File Creation and Sorting . . . . . 6-23
Group File Creation Uses Too Much Memory . . . . . 6-24
Excessive Runtimes . . . . . 6-25
Memory Pool Dump During Preprocessing . . . . . 6-26
VCELL Passes Run Very Slowly . . . . . 6-26

**7. General Results and Analysis**

Description of Output Files . . . . . 7-1
Account File . . . . . 7-2
Summary File . . . . . 7-2
Results File . . . . . 7-2
Error File . . . . . 7-3
Flat Error File . . . . . 7-8

LVS Debugging File . . . . . 7-9
Extracted Layout Netlist File . . . . . 7-9
LPE Statistics File . . . . . 7-9
Extracted Layout SPICE Netlist File . . . . . 7-9
Compare Log File . . . . . 7-9
Compare Directory Output Files . . . . . 7-9
Individual Equivalence Summary File . . . . . 7-10
Partitioned Schematic Netlist . . . . . 7-10
Partitioned Layout Netlist . . . . . 7-11
Compare Tree Files . . . . . 7-11
Equivalence Files . . . . . 7-11
TECHNOLOGY_OPTIONS File . . . . . 7-11
Tree Structure Output File . . . . . 7-11
WAIVE File . . . . . 7-12
Milkyway View and Object Type Statistics . . . . . 7-12
Graphical Error Structures . . . . . 7-14
Graphical Error Hierarchy . . . . . 7-14
Error and Permanent Output . . . . . 7-15
Placing the Error Structures and Permanent Output . . . . . 7-15
Editing Errors . . . . . 7-15
HTML Interface . . . . . 7-16
Invoking HTML Output . . . . . 7-16
Example of HTML Output . . . . . 7-16
Saving Output from Runset Commands . . . . . 7-18
Output Formats . . . . . 7-19
Error Hierarchy Syntax . . . . . 7-19
Temporary Result Syntax . . . . . 7-19
Permanent Result Syntax . . . . . 7-19

**8. Scheme Interface**

Introduction to Scheme Usage . . . . . 8-2
Flexible Device Netlisting . . . . . 8-3
Device Routines . . . . . 8-4
ev-devcon-dev-type-get . . . . . 8-4
ev-devcon-dev-name-get . . . . . 8-4
ev-devcon-dev-class-get . . . . . 8-4

ev-devcon-dev-is-shorted . . . . . 8-5
ev-devcon-dev-is-netlisted . . . . . 8-5
ev-devcon-dev-is-parasitic . . . . . 8-5
ev-devcon-dev-prop-list . . . . . 8-5
ev-devcon-dev-prop-value-get . . . . . 8-6
ev-devcon-dev-conn-list . . . . . 8-6
ev-devcon-dev-conn-net-get . . . . . 8-6
ev-devcon-net-name-get . . . . . 8-6

GENDEV Command . . . . . 8-7
Example for AUTO mode . . . . . 8-8
Example for MANUAL Mode . . . . . 8-9
Device Initialization Routine . . . . . 8-13
ev-dev-property-type-set . . . . . 8-13
Device Definition Routines . . . . . 8-13
ev-dev-property-value-set . . . . . 8-13
ev-dev-device-create . . . . . 8-14
ev-dev-device-write . . . . . 8-14
Output Data Routines . . . . . 8-14
ev-dev-output-set . . . . . 8-14
ev-dev-error-output-set . . . . . 8-15
ev-dev-error-message . . . . . 8-15
Data Manipulation Routines . . . . . 8-15
ev-dev-edges-get . . . . . 8-16
ev-dev-area-get . . . . . 8-16
ev-dev-coordinates-get . . . . . 8-16
ev-dev-polygon-select . . . . . 8-17
ev-dev-polygon-boolean . . . . . 8-17
ev-dev-spacing-get . . . . . 8-18
ev-dev-cell-name-get . . . . . 8-19
ev-dev-processing-layers-get . . . . . 8-19
ev-dev-polygons-get . . . . . 8-20
Data Measurement Routines . . . . . 8-20
ev-measure . . . . . 8-20

EXPLODE Language . . . . . 8-22
Data Information Routines . . . . . 8-23
ev-exp-is-name-equal . . . . . 8-23
ev-exp-is-non-orthogonal . . . . . 8-23
ev-exp-name-get . . . . . 8-24
ev-exp-area-get . . . . . 8-24

ev-exp-top-cell-area-get . . . . . 8-24
ev-exp-data-count-get . . . . . 8-25
ev-exp-text-count-get . . . . . 8-25
ev-exp-layer-list-get . . . . . 8-25
ev-exp-sref-count-get . . . . . 8-26
ev-exp-aref-count-get . . . . . 8-26
ev-exp-aref-as-sref-count-get . . . . . 8-26
ev-exp-hier-placement-count-get . . . . . 8-26
ev-exp-flat-placement-count-get . . . . . 8-27
ev-exp-explode . . . . . 8-27
ev-exp-flatten . . . . . 8-27
ev-exp-delete . . . . . 8-28
ev-exp-explode-all . . . . . 8-28
ev-exp-no-explode . . . . . 8-28

LVS User-Defined Property Comparison . . . . . 8-29
LVS Routines . . . . . 8-31
lvs-connection-inst-id-get . . . . . 8-31
lvs-inst-name-get . . . . . 8-31
lvs-inst-primitive-name-get . . . . . 8-31
lvs-inst-prop-get . . . . . 8-31
lvs-inst-member-get . . . . . 8-32
lvs-inst-tolerance-get . . . . . 8-32
lvs-inst-tolerance-is-absolute . . . . . 8-32
lvs-inst-tolerance-is-relative . . . . . 8-32
lvs-inst-is-parallel . . . . . 8-33
lvs-inst-is-series . . . . . 8-33
lvs-inst-is-parallel-chain . . . . . 8-33
lvs-inst-is-extracted . . . . . 8-33
lvs-inst-is-member . . . . . 8-34
lvs-inst-is-nmos . . . . . 8-34
lvs-inst-is-pmos . . . . . 8-34
lvs-inst-is-cap . . . . . 8-34
lvs-inst-is-res . . . . . 8-35
lvs-inst-is-npn . . . . . 8-35
lvs-inst-is-pnp . . . . . 8-35
lvs-inst-is-np . . . . . 8-35
lvs-inst-is-pn . . . . . 8-36
lvs-inst-is-generic . . . . . 8-36

Advanced Filtering . . . . . 8-36
Filtering Routines . . . . . 8-38

lvs-pins-get . . . . . 8-38
lvs-connection-count-get. . . . . 8-39
lvs-is-pin-net-ground. . . . . 8-39
lvs-is-pin-net-power . . . . . 8-39
lvs-is-pin-net-port . . . . . 8-39
lvs-are-pins-shorted . . . . . 8-40
lvs-connections-get. . . . . 8-40
lvs-connection-type-get. . . . . 8-40
Available Scheme Routines . . . . . 8-41

**Index**

# Preface

This preface includes the following sections:


- What's New in This Release
- About This Guide
- Customer Support

## 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 Guide

The Hercules hierarchical physical verification tool allows you to perform design rule checking (DRC), electronic rule checking (ERC), and layout-versus-schematic (LVS) on your integrated circuit design. This user guide assists the advanced designer in running applications with the Hercules software. Use the guide in conjunction with the *Hercules Reference Manual* and *Hercules Getting Started Tutorial*.

This preface provides an overview of the chapters included in *Hercules General Usage Information*, as well as information on how you can contact Customer Support and access SolvNet.

### Audience

This guide is a compilation of applications designed to enhance both the beginning and advanced users knowledge of the Hercules tool. This user guide includes

- Details of usage flows and applications for the Hercules tool, including debug information.
- Index of Hercules applications and particular Hercules commands.
- Detailed explanation of hierarchy and of how Hercules takes advantage of hierarchy.
- Advanced Hercules application information, runset debugging options, and hints on how to get the most out of Hercules.

## 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

# Introduction to Hercules

*This chapter gives a brief overview of the Hercules physical verification tool, telling how it fits into your design methodology.*

## What Is Hercules?

Hercules is a hierarchical physical verification tool that allows you to perform design rule checking (DRC), electronic rule checking (ERC), and layout-versus-schematic (LVS) on your integrated circuit (IC) design. Hercules is a powerful suite of tools that verifies integrated circuit layouts. Hercules

- verifies layout design rules
- performs electrical rule checks
- extracts layout structures
- compares extracted layout structures to an original design netlist

Hierarchical checking algorithms make Hercules particularly well-suited for large and complex IC verification.

When using Hercules, your tasks generally fall into one of two categories: design rule checking (DRC) or layout-versus-schematic (LVS). While completing DRC runs, Hercules is checking the layout files for errors against predefined rules. While performing LVS runs, Hercules creates a netlist file that describes the electrical connectivity of the design layout. This netlist is then compared against a netlist generated from a design's schematic.

## Hercules Features and Benefits

Hercules allows you to verify ultra-deep submicron (UDSM) IC designs. Using Hercules shortens runtimes because of its

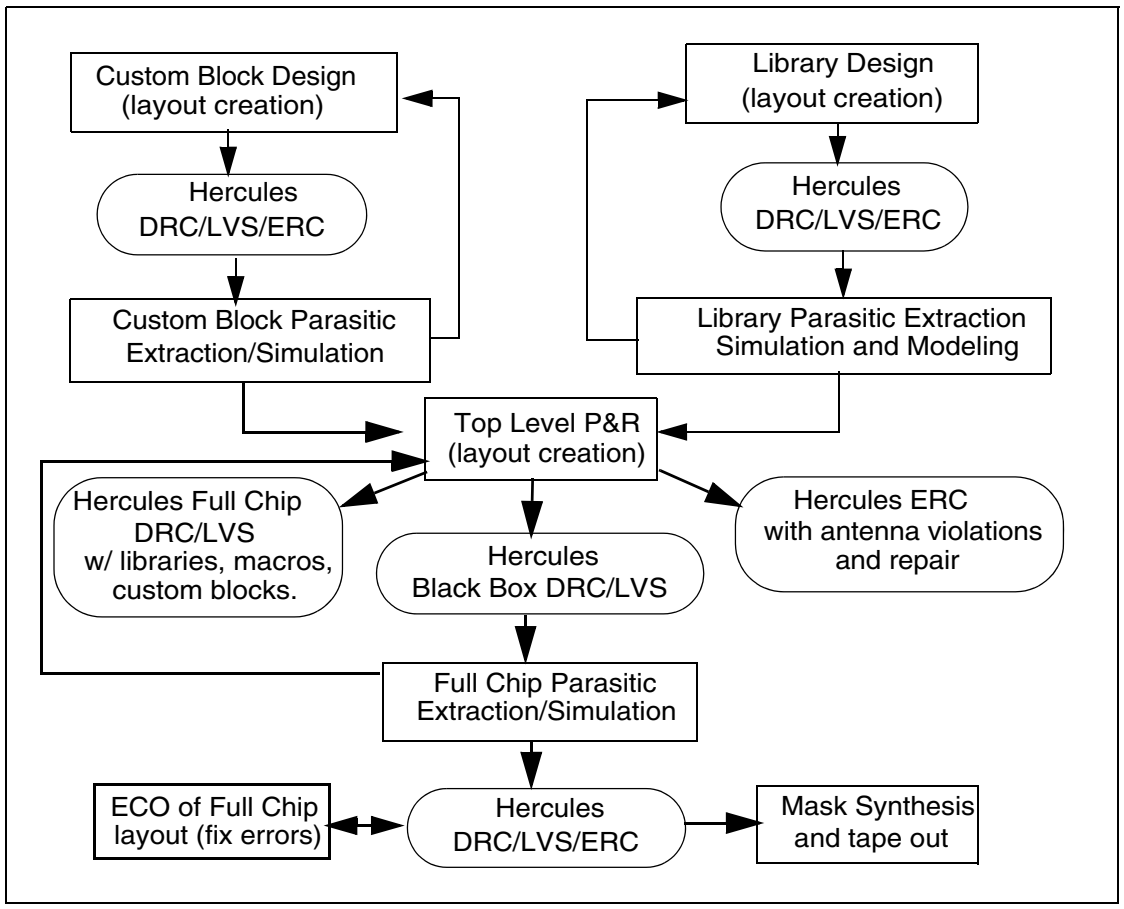
- Sophisticated algorithms for hierarchical design manipulation
- Optimized engine for flat processing
- Ability to automatically decide how to process each area of data, increasing the verification accuracy
- Distributed processing and multithreading

Also, the powerful graphical user interface (GUI) in Hercules creates fast design debugging by helping you find and fix design errors quickly.

## How Does Hercules Fit Into My Design Methodology?

Physical verification is typically the final step of the design process, prior to the tape out of your design. Due to the Hercules tool tight integration into the MIIkyway™ design environment, you can easily perform physical verification in parallel with other design processes, such as top-level place-and-route and parasitic extraction and simulation. Figure 1-1 shows how Hercules can fit into your design methodology.

*Figure 1-1 Design Flow Methodology*




## Using Hercules in Your Design Flow

All Hercules runs are executed based on commands listed in the Hercules runset. You can execute Hercules using command-line execution, interactive execution using the Hercules-VUE GUI debugging tool, or from various other Synopsys tools (such as Cosmos™ or IC Compiler).

In order to run Hercules successfully in your design flow, follow these general steps:

Create and debug a runset file.

Or

Convert existing runsets that are not in the Hercules format (such as Cadence® Dracula® physical verification or place-and-route) to Hercules runsets. You can also download runsets from the foundry.

The runset file includes the following information:

- Runset Header with layout database specifics
- General options
- Layer assignments
- Database checks (such as SNAP and GRID checks)
- DRC, ERC, fill patterns, or LVS

Run Hercules DRC and ERC on your Milkyway database, or GDSII, or OASIS file with your runset, verifying your design against the design rule document.

Note:
Many design flows separate the ERC antenna checking into a different runset.

Review and debug errors generated in the Hercules DRC and ERC run using Hercules error files or the graphical debugger GUI.

Run Hercules to perform fill-pattern generation, including basic DRC checks on the layers that are filled.

Run Hercules layout-versus-schematic (LVS) checking.

Review and debug errors generated in the Hercules LVS run using Hercules HTML error files or the graphical debugger GUI.

## Running Hercules within IC Compiler

You can run Hercules inside IC Compiler using these IC Compiler commands:

- signoff_drc. Performs DRC checking using foundry-qualified Hercules DRC runset, and reports all errors in the top-level design by exploding the lower-level errors without merging them.
- signoff_metal_fill. Performs metal filling using foundry-qualified Hercules metal-fill runset. Fills are written as array references (AREFs) into FILL views of Milkyway input design.
- verify_drc. Generates a Hercules DRC runset based on the design rules defined in the Milkyway technology file, and then runs Hercules to check for DRC violations.

Refer to the *IC Compiler Implementation User Guide* manual, which is available on SolvNet, or the man pages of IC Compiler help for detailed command syntax and usage.

# 2

# Licensing and System Requirements

*This chapter contains the File Transfer Protocol (FTP) installation and setup instructions for Hercules. Make sure that you have all the correct directories and files, that your environment accommodates Hercules, and that you have incorporated the proper licensing information.*

## Licensing Hercules and Related Products

Hercules is licensed using Combined Vendor Daemon licensing technology. The Hercules commands in your runset will depend on the license required. A listing of the required licenses for each command is shown below. Hercules and its utilities (NetTran, VUE, DRC, and LVS) are all licensed separately. It is important to have your license server set up before running any Hercules job.

### Licensing Software

The Synopsys Licensing QuickStart Guide is available at http://www.synopsys.com/Support/Licensing/Licensing/Pages. Hercules supports license waiting in accordance with the standard Synopsys Common Licensing (SCL) guidelines. To enable this function, you must set the HERCULES_LICENSE_WAIT environment variable.

Information about installing Hercules is available at http://www.synopsys.com//Support/Licensing/Installation/Pages.

## Which Hercules Licenses Do I Need?

This section lists the Hercules licenses that are sold with each package and the license each command requires.

## Product Licensing

Every time you run Hercules, Hercules uses a HERCULES_MANAGER license and a HERCULES_MASK license. In addition to those two licenses, Hercules uses licenses based on the commands you use in your runset. See the descriptions of each license below and/or the application you run.

| License | Executable | Runset Command | Command Line |
|---|---|---|---|
| HERCULES_DEVICE | ev_engine | See "HERCULES_DEVICE" on page 2-2. | N/A |
| HERCULES_DRC | ev_engine | See "HERCULES_DRC" on page 2-3. | N/A |
| HERCULES_ERC | ev_engine | See "HERCULES_ERC" on page 2-3. | N/A |
| HERCULES_LVS | lsh | COMPARE | hercules - C runset.ev |
| HERCULES_MANAGER | hercules | N/A | hercules runset.ev |
| HERCULES_MASK | ev_engine | See "HERCULES_MASK" on page 2-4. | N/A |
| HERCULES-NETLIST | ev_netlist | NETLIST SPICE | hercules -N runset.ev |
| HERCULES-RUN_TRAN | drac2he | N/A | drac2he |
| NET-TRAN | nettran | ! SCHEMATIC_FORMAT = HERCULES | nettran |

### HERCULES_DEVICE

Hercules requires a HERCULES_DEVICE license when you use any of the following commands:

CAPACITOR DEVICE DIODE

| | | |
|---|---|---|
| GENDEV | INDUCTOR | NMOS |
| NPN | RESISTOR | PMOS |
| PNP | | |

**HERCULES_DRC**

Hercules uses a HERCULES_DRC license when you use any of the following commands:

| | |
|---|---|
| AREA | ANALYZE_WINDOW |
| CENTER_TO_CENTER | DENSITY |
| ENCLOSE | EXTERNAL |
| INSIDE_EDGE | INTERNAL |
| LENGTH | MASK_ALIGN |
| MOSCHECK | MULTI-RULE ENCLOSE |
| MULTI-RULE INSIDE_EDGE | NOTCH |
| RESCHECK | |

**HERCULES_ERC**

Hercules uses a HERCULES_ERC license when you use any of the following:

| | |
|---|---|
| ANTENNA_FIX | C_THRU |
| DEV_CONNECT_CHECK | DEV_NET_COUNT |
| DEVICE_COUNT | NET_FILTER |
| NET_PATH_CHECK | RATIO |

TEXT_OPTIONS with FIND_SHORTEST_PATH_BETWEEN_TEXT_SHORTS set

**HERCULES_LVS**

Hercules uses a HERCULES_LVS license when you use the COMPARE command. The command-line option `-lvs-license` enables conditional licensing to omit a DRC license during an LVS run. See Hercules Command-Line Syntax.

Note:
Some DRC commands can be used in the LVS runset if they are exclusively used to form devices as input to a device extraction command.

**HERCULES_MASK**

Hercules uses a HERCULES_MASK license when you use any of the following commands:

| | | |
|---|---|---|
| ASSIGN | ATTACH_PROPERTY | BOOLEAN |
| CELL_EXTENT | COMPARE_GROUP | CONNECT |
| CONNECTION_POINTS | COPY | CUT |
| DATA_LIMIT | DELETE | EXPLODE |
| EXPLODE_ALL | FILL_PATTERN | FIND_WIDE |
| FLATTEN | GRAPHICS | GRID_CHECK |
| LABEL | LAYER_STATS | LEVEL |
| LOAD_GROUP | METAL_SLOT | NEGATE |
| POLYGON_FEATURES | PROCESS_TEXT_OPENS | PUSH |
| RECTANGLES | RELOCATE | REMOVE_OVERLAP |
| REVERSE | SELECT | SELECT_CELL |
| SELECT_CONTAINS | SELECT_EDGE | SELF_INTERSECT |
| SIZE | SIZE_RECT | SNAP |
| TEXT | TEXT_BOOLEAN | TEXT_OPTIONS |
| TEXT_POLYGON | TEXT_PROPERTY | VECTORIZE |
| VERTEX | | |

**NET-TRAN**

NETTRAN application - translation of netlists

**HERCULES-NETLIST**

EV_NETLIST application - generates a netlist for LVS run.

**HERCULES-RUN_TRAN**

DRAC2HE application - converts a Dracula physical verification runset

**HERCULES-DP_MT**

Distributed Hercules.

**HERCULES-CELL_***

Hercules uses a HERCULES_CELL license when you use the -j command-line option. Two examples of HERCULES-CELL_* licenses are HERCULES-CELL_DEVICE and HERCULES-CELL_LVS. See Hercules Command-Line Syntax for information about the -j option.

## Determining the System Requirements

The following instructions assume that you know UNIX operating system commands. You should also have the GNU software to unzip files. If you do not, the file gnu.tar.Z is provided on the Synopsys FTP site.

Installation has four groups of steps:

- Registering with Synopsys
- Creating directories and getting the zipped applications files, documentation, and tutorials
- Extracting the zipped files using the Synopsys Common Installer.
- Ensuring that your environment accommodates Hercules

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. Please note that these scripts are not supported.

Hercules installation requires about 380 megabytes of disk space. With some knowledge of the design to be checked, it is possible to estimate physical memory, disk space, and possible swap space requirements. Designs which are optimized during preprocessing have the advantage of using Hercules algorithms and layout checks that operate faster and require less memory than a flat design style. Flat designs or designs with unwanted data interactions require more disk and swap space to be executed successfully. The initial database file and the files that Hercules creates and uses determine the system requirements.

## Requirements for Database Space

The Hercules program performs its layout checking operations on group files created from the design database and stored in the group directory. Hercules first creates the primary group files—one file for each layer listed in the ASSIGN section of the runset file. Each file is a database describing that layer's existence throughout the design, maintaining all hierarchy. As the Hercules run progresses it encounters PERM (for permanent) and TEMP (for temporary) outputs from certain checks. These outputs are also written as group files. An example of fils in a group/directory are:

```
HRCHY.ERROR    inst_indx.dat  pgate.group  toxcont.group
cont.group     met1.group     poly.group   via.group
gate.group     met2.group     psel.group   well.group
inst_data.dat  ngate.group    tox.group
```

A general rule of thumb is to have a minimum of two to three times the space of the initial database (in GDSII file format) reserved for group file creation, assuming a design with a good hierarchical style. A mostly flat design, or one with unwanted data interactions, might require at least four to six times the initial database space for its group file creation. Thus, for a GDSII file database of 10 megabytes, a minimum of 20 to 30 megabytes is required for group file creation space, if the design is hierarchical in nature. By contrast, at least 40 to 60 megabytes are required for a flat design.

## Output Files

The other significant output files from Hercules are the Hercules run summary files and the OUTLIB output database file. The summary files are the *block*.LAYOUT_ERRORS and *block*.sum files, which are ASCII log files documenting the results of the Hercules run on the block named in the runset. The output database file contains the graphical error data in GDSII, OASIS, or Milkyway format.

The size of the *block*.LAYOUT_ERRORS file and the output database depends on the number of errors encountered in the Hercules run. As with the group files, if the design is hierarchical, the graphical output database will be smaller than the graphical output for a more flat design. This is because Hercules only needs to report errors internal to repeated modules once, while for a flat design the error is reported for every instance of the module.

## Memory

Managing the memory space needed for error output reporting can be accomplished with a logical approach to error location. When possible, you should apply a bottom-up approach to an application while using Hercules. Start by running Hercules on small modules of the design and address errors as they are encountered. After errors are fixed, run Hercules on

hierarchically higher modules. This keeps your *block*.LAYOUT_ERRORS file and output database reasonably small. Working up through the hierarchy in this way manages the use of memory as error output storage space.

Hercules also requires physical memory and swap space to complete its checks. The program attempts to load into memory all data required to complete any current check. Any overflow of physical memory utilizes swap space. Generally speaking, any time the Hercules program needs to access swap space, you can expect performance degradation, especially if swap space is distributed over a large network with considerable network traffic. If enough physical memory is available to load all data and complete the check without accessing swap space, you can expect maximum throughput. Set swap space so that any overflow of physical memory will not exhaust the swap space and abort the Hercules run.

The output *block*.sum file includes a list of the memory used and the number of page faults required to complete each check. If certain checks require more memory than the machine has allocated, with large amounts of data swapped out to disk, you might need a machine with more memory for future runs.

*Example 2-1 Partial Example of Memory Usage Information in block.sum File*

```
EXTERNAL met1 met2 {
  SPACING<1.000
  TOUCH=TRUE } (103)
WARNING - 2 spacing violations found.
  Check time = 0:00:00  User=0.01 Sys=0.08 Mem=4.084

INTERNAL poly {
  SPACING<2.000
  EDGE_45<2.500 } (104)
WARNING - 1 width violation found.
    Check time = 0:00:01  User=0.05 Sys=0.05 Faults=11 IO=0 Mem=7.452
AREA via { RANGE = [3.000,5.000]} (107)
WARNING - 1  violation found.
   Check time = 0:00:00  User=0.07 Sys=0.02 Faults=2 IO=0 Mem=7.437

BOOLEAN poly AND tox { } PERM=gate (111)
8 unique polygons written.
   Check time = 0:00:01  User=0.06 Sys=0.14 Faults=4 IO=1 Mem=7.452

ENCLOSE toxcont BY met1 {
  SPACING<1.000
  OVERLAP=TRUE } (123)
WARNING - 4 enclose violations found.
   Check time = 0:00:00  User=0.17 Sys=0.02 Faults=1 IO=0 Mem=8.530

Checks complete.
  Total check time = 0:00:06  User=2.21 Sys=1.11 Faults=84 IO=4 Mem=8.546

Saving error structures in block "TOP_ERR", in library
"TUT_ADDER_3_bogus.DB".
  Saving time = 0:00:05  User=0.16 Sys=0.89 Faults=1 IO=34 Mem=7.421
```

```
Overall ev_engine time = 0:00:26  User=5.27 Sys=5.27 Faults=235 IO=93
Mem=8.546
```

For maximum performance on a networked system, keeping as many files as required on the local machine results in faster completion. Enough disk space local to the machine on which the job is run should be available to accommodate the Milkyway format database (or, GDSII or OASIS files), the group files being created, and all ASCII or physical error files being created.

## Operating Systems

Hercules supports many hardware architectures and operating systems. Large Hercules runs require certain system limits. See the installation guide for a list of the supported operating systems.

## SUN Operating System Requirements for Hercules

The following CSH commands set the proper system limits for a Hercules run on SUN OS. The numerical values below are set in kilobytes.

```
unlimit filesize
unlimit datasize
limit descriptors 128
limit stacksize 8192
```

Stacksize should remain limited. If your system's stacksize is unlimited, it defaults to two gigabytes. Hercules does not use much stack; therefore, allocating more stack prevents Hercules from using that RAM for general use. The descriptor setting controls how many files you can have open at one time during a run. If UNIX system limits are set incorrectly, you may encounter an error during the Hercules run. For more information on system-related errors, see the *Hercules Reference Manual*, Running Hercules chapter.

## HP Operating System Requirements for Hercules

For HP operating systems, make sure that kernel parameters are set properly. These parameters can be adjusted using the SAM utility.

1. Execute the SAM utility using the command: `/usr/sbin/sam &`
2. Double-click the Kernel Configuration icon.
3. Double-click the Configurable Parameters icon.
4. Adjust maxdsiz, maxdsiz_64bit, maxfiles, maxssiz, maxssiz_64bit, maxswapchunks, maxtsiz, and maxtsiz_64bit parameters as suggested in Table 2-1, below.
5. Click the Actions button.

6. Select Process New Kernel.
7. Confirm that you want to process a new kernel.
8. Confirm that you want to reboot using this new kernel.

*Table 2-1 HP Operating System Kernel Settings*

| Name | Current Value | Pending Value | Type | Associated Module | Description |
|---|---|---|---|---|---|
| maxdsiz | 1879048192 | 1879048192 | Static | N/A | Max Data Segment Size (Bytes) |
| maxdsiz_ 64bits | 1073741824 | 1073741824 | Static | N/A | Max Data Segment Size (Bytes) |
| maxfiles | 500 | 500 | Static | N/A | Soft File LImit Per Process |
| maxssiz | 8388608 | 8388608 | Static | N/A | Max Stack Segment Size (Bytes) |
| maxssiz_ 64bit | 8388608 | 8388608 | Static | N/A | Max Stack Segment Size (Bytes) |
| maxswap chunks | 4096 | 4096 | Static | N/A | Max Number of Swap Chunks |
| maxtsiz | 67108864 | 67108864 | Static | N/A | Max Text Segment Size (Bytes) |
| maxtsiz_ 64bit | 1073741824 | 1073741824 | Static | N/A | Max Text Segment Size (Bytes) |

## Hercules Performance In a Linux Environment

Many things can affect the performance of Hercules in a Linux-based environment. Most notably, accessing files over a networked environment (such as when a remote GROUP directory is used). Synopsys has identified two items which have been shown to degrade Hercules performance:

- NFS mount settings (for both auto-mounted and static mounts).
- The version of the Linux kernel which you are running.

In both cases, if left uncorrected, you can observe large differences in the wall time (the overall time) and total CPU time (user + system times) reported by Hercules. This difference is attributable to the overhead the system encounters while attempting to satisfy the requests of your Hercules run, and can be caused by:

- Other processes (users) running on the same system.
- Network transfer overhead.
- System (kernel) overhead.

## NFS Settings

The read- and write-transfer sizes (rsize and wsize) the mounts should be set to are at least 32768.For static mounts this would look something like:

```
mount -t nfs -o rsize=32768,wsize=32768 server:/path /mnt/nfs
```

Note:

We have observed that RedHat AW2.1 installations on Intel Itanium2 platforms appear to be set to 4096 out-of-the-box. This dramatically degrades Hercules performance when using networked volumes.

## Linux Kernel Version

Synopsys recommends that you run RedHat Linux kernel version RHEL3, or later. Refer to the Red Hat web site for information.

# 3

# Overview of Input Files

*This chapter describes the different input files required to run Hercules. It includes information on syntax rules, runset variables, and macro usage that is useful when working with Hercules input files.*

## Hercules Input Files

This section describes the input files used by Hercules to perform DRC, LVS, LPE, and ERC design verification. The primary input file is the runset file. In addition to this file, Hercules can use an Edtext file, an Equivalence file, an Explode List file, a Schematic Netlist file, and a Cell List file. The following sections describe these files.

### Runset File

The runset file is a control file that instructs Hercules where to find your input data, which checks to perform, and where to write the output files. Runsets are typically tailored for the type of Hercules run you are completing, such as DRC or LVS. For information on runset rules and conventions, see the *Hercules Reference Manual*, Key Concepts chapter.

A DRC runset is one that instructs Hercules to check layout files for physical design errors. (See the *Hercules DRC and ERC User Guide*, starting with Introduction to DRC and ERC for a detailed explanation of DRC.)

An LVS runset is one that instructs Hercules to compare the layout netlist versus the schematic netlist of a design. (See the *Hercules LVS User Guide*, starting with Introduction to Hercules LVS for a detailed explanation of LVS. See "Runsets" on page 4-24 for information on runset sections.)

## Edtext File

The Edtext file contains a list of text placements to include in the extracted layout netlist, along with the text found in the layout. The text is specified for a specific structure and x- and y-coordinate. An EDTEXT text placement option at the exact coordinates of an existing layout text placement overrides the layout text placement.

The path to the Edtext input file must be specified in the runset file OPTIONS section with the EDTEXT option.

Comments enclosed by /* */ are allowed in this file, but nested comments are not allowed. Blank lines are allowed.

The syntax is:

```
STRUCTURE structure_name1 text1 layer_num1 datatype1 x_coordinate1

y_coordinate1

. . .

STRUCTURE  structure_nameN textN layer_numN datatypeN x_coordinateN
y_coordinateN
```

| Argument | Description |
|---|---|
| *structure_name* | Name of the layout structure in which the text is placed. |
| *text* | The text string that is placed. |
| *layer_num* | Layer number specified in the ASSIGN section TEXT statement. (It must be the same as in a corresponding ASSIGN section TEXT entry.) |
| *datatype* | Datatype specified in the ASSIGN section TEXT statement. (It must be the same as those in a corresponding ASSIGN section TEXT entry.) Default is 0 (zero). |

| Argument | Description |
|---|---|
| *x_coordinate* | The absolute local x-coordinate within the named structure where the text is placed. |
| *y_coordinate* | The absolute local y-coordinate within the named structure where the text is placed. |

## Equivalence File

The Equivalence file contains a list of EQUIV and/or BLACK_BOX command entries that associate schematic module names with layout structure names. These entries determine which schematic module to compare to which layout structure. Comments enclosed by /* */ are allowed in this file, but nested comments are not allowed. Blank lines are allowed.

Syntax for the EQUIV command is:

```
EQUIV schematic_name = layout_name { Option_Definitions EQUIV

Process_Definitions }
```

Syntax for the BLACK_BOX command is:

```
BLACK_BOX schematic_name = layout_name { BLACK_BOX Process_Definitions }
```

| Argument | Description |
|---|---|
| *schematic_name* | The schematic module name. |
| *layout_name* | The layout structure name. |
| *Option_Definitions* | Optional comparison features within a module. |
| *Process_Definitions* | EQUIV: Processing control for a specific module. (For more information, see the Equivalence File in *Hercules LVS User Guide*, Layout Versus Schematic Comparison chapter.)<br>BLACK_BOX: Processing control for a specific BLACK_BOX module. |

## Explode List File

The Explode List file contains a list of options with specified structure names that limits the amount of hierarchy that is checked. The Explode List file uses the following options: DELETE, EXPLODE, EXPLODE_ALL, FLATTEN, and NO_EXPLODE. Comments enclosed by /* */ are allowed in this file, but nested comments are not allowed. Blank lines are allowed.

These options have the following syntax:

```
delete = {structure_name1, ..., structure_nameN}

explode = {structure_name1, ..., structure_nameN}

explode_all = {structure_name1, ..., structure_nameN}

flatten = {structure_name1, ..., structure_nameN}

no_explode = {structure_name1, ..., structure_nameN}
```

| Argument | Description |
|---|---|
| *structure_name* | A layout structure. |

The path to the Explode List input file must be specified in the runset file EXPLODE_OPTIONS section with the EXPLIST option. See the *Hercules Reference Manual*, Detailed Options chapter.

The Explode List file option is an alternative to specifying multiple EXPLODE structure assignments in the runset. Each structure in the Explode List file is processed as specified by its keyword: exploded, flattened, or deleted.

The format of the Explode List file includes all of the EXPLODE_OPTIONS assignments, as indicated above. For all of these options, a keyword must be specified.

An example explode list file:

```
/*This is an example*/
explode = {via, contact}                          /*explode*/
explode_all = {register1 controlAB, ALV1}         /*explode_all*/
flatten = {array6, fifo3}                         /*flatten*/
delete = {testreg7 controlJK buffer6 testpoint3} /*delete*/
```

Another explode list file:

```
/*explode = not required if all structures are to be exploded*/
```

```
{via
   contact
   polyhd
   power1
   power2}
```

## Schematic Netlist File

The Schematic Netlist file is used during an LVS comparison. Hercules requires that the input netlist be in Hercules netlist format. The translators provided accept standard netlist formats and generate the Hercules format. The path to the Hercules format netlist must be specified in the runset file with the HEADER section SCHEMATIC option. For more information on the Hercules netlist format and on creating a Schematic Netlist file, see *Hercules LVS User Guide*, Netlisting chapter.

## CELL_LIST File

A CELL_LIST file can be used with the SELECT_CELL command and the CELL_EXTENT command. The name of the CELL_LIST file is placed in the CELL_FILE option of these commands. (For more information on these commands, refer to the *Hercules Reference Manual*, Detailed Commands chapter.) For example, the CELL_LIST file:

```
cell_list = {
     via
     contact
     register1 }
```

# Parsing the Input Files With the EVP Utility

EVP is the runset parser. It checks a runset for correctness and handles macro expansion, syntax analysis, and runset optimization. When processing a runset, EVP can optimize the runset by removing unnecessary commands and replacing duplicate commands with COPY commands.

To see what the optimizer removed from a runset, compare the outputs of the EVP utility with and without the disable option. For example:

```
evp -O opt.ev input.ev
evp -disable all -O no-opt.ev input.ev
diff opt.ev no-opt.ev
```

The syntax of the EVP utility is:

```
evp [-h | -usage] [-q] [-V] [-Version]
   [-disable [prune | rmdup | all | none | merge]] [-R]
```

```
[-prune-assign] [-emit-wxvdb] [-srf filename]
[-select_rule rule | -select_rule rule_1 ... rule_n \;]
[-unselect_rule  rule | -unselect_rule rule_1 ... rule_n \;]
[-O outfile] runsetfile
```

| Argument | Description |
|---|---|
| `-h` and `-usage` | Displays command-line information. |
| `-q` | Quiet—do not print copyright information. |
| `-V` | Print product version of EVP. |
| `-Version` | Print the version of EVP and the libraries used by Hercules. |
| `-disable prune` | Do not optimize runset by pruning commands with no effect. That is, disable removal of all commands in a runset that do not contribute to either ERROR or PERM layers. Using this option can result in longer execution times. |
| `-disable rmdup` | Do not optimize runset by replacing duplicate commands with COPYs. That is, disable duplicate command recognition and removal. Using this option can result in longer execution times. |
| `-disable all` | Do not optimize runset in any way. That is, disable all parse-time runset optimizations, and can result in substantially longer execution times. |
| `-disable none` | All optimizations are enabled (default behavior). |
| `-disable merge` | Replace related commands with a single command. |
| `-R` | Remove commands not required by commands in RUN_ONLY section of *runsetfile*, and save output to another runset named run_only.ev unless overridden by the -O option. The new runset contains only the functions and utility commands singled-out by the RUN_ONLY command. For more details, refer to the *Hercules Reference Manual*, Detailed Commands chapter. |

| Argument | Description |
|---|---|
| `-prune-assign` | Prune unnecessary ASSIGN layers. This command line option provides users with the option to prune out unused ASSIGN layers from a run so that reading in the design data is faster. This can have a significant impact toward runtime particularly when running a subset of a runset, either through selectable rule use, or run_only use.<br>ASSIGN layer pruning must be activated through this command line option. The default behavior is to keep all ASSIGN layers even if they are unused. |
| `-srf` *`filename`* | Allows for a selectable rules file. The file format follows the same format as the command line.<br>• It supports the same wildcard globbing as does –sr and SELECT_RULE.<br>• Each line in the file must start with either:<br>-sr or –select_rule for selecting a rule<br>-ur or –unselect_rule for unselecting a rule<br>Lines that start with the # character are ignored (comment).<br>Example:<br>`-sr "M1*" "V1*"`<br>`-ur "M1B*"`<br>`#-sr "M2*"`<br>This example is equivalent to:<br>`-sr "M1*" "V1*" –ur "M1B*"`<br>The first line could also be two separate lines and have the same result:<br>`-sr “M1*”`<br>`-sr “V1*”`<br>`-ur “MIB*”` |
| `-select_rule` *`rule`* \| `-select_rule` *`rule_1 ... rule_n \;`* | Run only commands with the same rule(s) in COMMENT option. Abbreviate as -sr. Use quotes and an asterisk. For example, “M1*”. -select_rule *regex-pattern*, is added to select certain matching commands and must be abbreviated, -sr *regex-pattern*. More than one regex-pattern can be specified if followed by a semicolon. Regex-patterns must not contain a leading minus “-”. The command string, *-sr “A*” “B*” “C*” \; t.ev* is equivalent to *-sr “A*” -sr “B*” -sr “C*” t.ev*. |

| Argument | Description |
|---|---|
| `-unselect_rule` *`rule`* \| `-unselect_rule` *`rule_1 ... rule_n \;`* | Discard commands with the same rule(s) in COMMENT option. Abbreviate as -ur. Use quotes and an asterisk. For example, “M1*”. -unselect_rule *regex-pattern*, is added to disable certain matching commands. This must be abbreviated -ur *regex-pattern*. More than one regex-pattern can be specified followed by a semicolon. Regex-patterns must not contain a leading minus “-”. |
| `-O` *`outfile`* | Save runset to *outfile*, instead of printing to stdout. |
| *`runsetfile`* | Hercules runset. |

# Runset Flow Control and Macros

This section describes different customization techniques for using runset flow control and macros. Runset flow control allows a runset to be written so that certain commands are conditionally executed, depending on the value of internal runset variables and environment variables. Using a construct similar to the IF-ELSE construct in the C programming language, single runsets can be written to perform various conditional operations.

Macros allow you to create blocks of parametrized Hercules code that can be instantiated as many times as needed in a runset. This can save time and avoid copy/paste errors while writing runset code. For more information on different customization techniques for runsets, see the *Hercules Reference Manual*, Overview of Runset Organization chapter.

## Runset Flow Variables

Runset flow variables are used to control the execution of a runset. Variables can be defined anywhere in the runset, but they must be defined before they are used. After you define a variable, it can be used inside subsequent commands and reset to different values throughout the runset.

Note:
All variables are case-sensitive.

## Declaring Runset Flow Variables

There are two types of runset flow variables, each of which is declared differently. An internal runset variable is declared and assigned an initial value, whereas environment variables receive their initial value from the shell environment.

Note:
After runset flow variables are declared, they are treated exactly the same whether they are internal runset variables or environment variables. The only difference is in how they receive their initial values.

## Declaring Runset Internal Variables

Internal variables are declared using the VARIABLE command at the beginning of the runset. Every internal variable must be assigned a type of either string or double. The type double is used for all numeric internal variables; the type string is used for all variables containing text. All internal variables must also be given an initial value, and all internal variable declarations of the type double must end with a semicolon (;). Initial values can be assigned using standard math operations and any previously declared variables. Parentheses can be used, as well as any of the standard C math functions. A complete list of the available math functionality, which can include any of the math functions available for the device equations, is found in the *Hercules LVS User Guide*, Using HLVS Device Extraction chapter. Any use of a literal string with respect to internal runset variables must be enclosed in quotation marks (" "). For a detailed explanation of the syntax of the VARIABLE command, see the *Hercules Reference Manual*, Detailed Commands chapter.

Note:
The following examples use commands that are explained later in the manual. The examples are intended to be generic and demonstrate principles that are clearer after the various commands are explained and understood.

```
VARIABLE string PROCESS = "CMOS";
VARIABLE double metal_width = 1.5;
VARIABLE double poly_width = metal_width / 2;
VARIABLE double group = 2;
```

## Declaring Environment Variables

Environment variables are declared using the environment construct at the beginning of the runset. Like internal variables, environment variables must be assigned a type and those of type double must end with a semicolon (;).

Environment variables differ from internal runset variables in that environment variables can receive their initial value from the shell environment variable of the same name or the DEFAULTS TO declaration in the ENVIRONMENT command. Hercules reports an error for all declared environment variables that do not exist in the environment from which Hercules was invoked. For example:

```
ENVIRONMENT string USER;
ENVIRONMENT double LAMBDA;
ENVIRONMENT double CAPLIMIT DEFAULTS TO 2.5e-8;
```

For a detailed explanation of the syntax of the ENVIRONMENT command, see the *Hercules Reference Manual*, Detailed Commands chapter.

## Using Runset Flow Variables

After runset flow variables are declared, they can be used several different ways:

- To define a check value for design rule checking commands.
- In user-defined equations for device extraction commands.
- In the Boolean expression of the IF-ELSE flow control construct (see "Rules for Boolean Expressions in IF-ELSE" on page 3-11).

### Defining Check Values

Variables can be used to define check values in many Hercules commands. With this you can write design rule checks that are dependent upon some environment variable or runset variable. For example:

```
ENVIRONMENT double LAMBDA;
VARIABLE double metal_width = 1.5;
INTERNAL metal {
SPACING < metal_width * LAMBDA; } (99)
```

### User-defined Device Syntax Equations

Runset variables can be used inside of device equations as global variables. A local device variable cannot be assigned the same name as a runset variable, and runset values cannot be changed inside of device equations. For example:

```
VARIABLE double metal_width = 1.5;
CAPACITOR m1p metal OVERLAP poly {
   length = EV_AREA/metal_width;
   EV_CAP = length * EV_PERIM_CAPVAL;
}TEMP = m1p_out
```

### Resetting Runset Flow Variables

Runset flow variables can be reset to different values throughout the runset. The SET command is used to change the value of either internal or environment variables. All SET commands must follow all variable declaration commands in the runset and must end with a semicolon (;). If an environment variable is reset in the runset using a SET command, it has no effect on the value of the environment variable in the shell environment from which Hercules was called. For a detailed explanation of the syntax of the SET command, see the *Hercules Reference Manual*, Detailed Commands chapter.

*Example 3-1*

```
VARIABLE double metal_width = 1.5;
VARIABLE double poly_width = 0;
```

```
ENVIRONMENT double LAMBDA;

INTERNAL metal {
   SPACING < metal_width;
} (99)
SET poly_width = (metal_width + LAMBDA) * 2;
INTERNAL poly {
   SPACING < poly_width;
} (99)
```

## Runset Flow Control Constructs

Hercules supports a construct similar to the C language IF-ELSE construct to allow conditional execution of portions of a runset. Nested IF statements are allowed. Curly braces { } are required for all IF statements, regardless of how many commands are inside the IF. This example shows the IF construct.

```
IF (boolean_expression) {
   HERCULES commands
}
ELSE {
   HERCULES commands
}
```

Limitation:

Any Hercules runset command can be included inside the IF-ELSE construct, but the IF-ELSE construct cannot be used inside a command. (An exception to this rule is the ability to write equations within LPE Device Extraction commands. This functionality is explained in detail in the *Hercules LVS User Guide*, Using HLVS Device Extraction chapter.)

An example of the legal runset flow control is:

```
VARIABLE double PROCESS = 3;
IF (PROCESS == 3) {
   INTERNAL metal { SPACING < 3 } (99)
}
ELSE {
   INTERNAL metal { SPACING < 2 } (99)
}
```

## Rules for Boolean Expressions in IF-ELSE

The expression must be enclosed in parentheses ( ) and must evaluate to either true or false.

Note:

Because the IF-ELSE constructions and the @if directives have different precedence of operators, use parentheses to insure that the order of precedence is correct.

In Boolean expressions involving variables of type string, the following operators are allowed:

| | |
|---|---|
| == | equal to |
| != | not equal to |

Expressions involving variables of type double can use the following operators:

| | |
|---|---|
| == | equal to |
| != | not equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |

Also, expressions can be separated by the following operators:

| | |
|---|---|
| && | logical and |
| \| \| | logical or |

Additional parentheses can be added to improve clarity and accuracy. For example:

```
VARIABLE double PROCESS = 3;
ENVIRONMENT string USER;
IF (PROCESS >= 3 && USER == "SMITH") {
   INTERNAL metal { SPACING < PROCESS } (90)
}
```

*Example 3-2 Runset Flow Control*

```
ENVIRONMENT string PROCESS;
ENVIRONMENT double DSGN_GROUP;
VARIABLE double LAMBDA = 0.2;
IF (PROCESS == "CMOS" && DSGN_GROUP == 2) {
   HEADER {
      layout_path = /u1/cmos/dsgn2
      format = GDSII
      inlib = cmos_chip
      outlib = cmos_chip_out
      block = CORE_LOGIC
      group = group
```

```
   }
}
ELSE {
   HEADER {
      layout_path = /u1/bicmos/dsgn
      format = GDSII
      inlib = bicmos_chip
      outlib = bicmos_chip_out
      block = DRAM
      group = group
   }
}
OPTIONS {
   resolution = 0.03
   ignore_case = TRUE
   explist = dsgn.exp
}
ASSIGN {
   tox       (1)
   poly      (5)
   cont      (6)
   metal     (8)          text (100)
   psel      (14)
   pwell     (31)
}
IF (PROCESS == "CMOS" && DSGN_GROUP == 2) {
   ASSIGN {
      metal2  (9)
   }
}
BOOLEAN psel AND tox { cell_level = true } temp = ptox
BOOLEAN poly AND ptox { cell_level = true } temp = pgate
SIZE pwell { oversize = LAMBDA * 2; } perm = sizewell (51)
IF (LAMBDA < 0.2) {
   SET METAL_SPACING = LAMBDA * 3.2;
}
ELSE {
   SET METAL_SPACING = .5 + POW(LAMBDA, 2);
}
INTERNAL metal { spacing < METAL_SPACING } (200)
IF (PROCESS == "CMOS" && DSGN_GROUP == 2) {
   EXTERNAL metal metal2 { spacing < METAL_SPACING } (201)
}
```

## Version Variable in Runset

The version variable can be used to write a single runset that can support multiple code versions. The version is extracted from the ev_engine version, which can be obtained by entering hercules -V from the command line.

```
Printing individual version numbers ...
```

```
Hercules : U-2003.03.0060    2003/12/01
drac2he  : U-2003.03.0006    2003/05/08
evp      : U-2003.03.0283    2004/02/03
ev_engine: U-2003.03.0416    2004/03/12
```

Example 3-3 shows the use of the version variable with the SIZE_RECT command.

*Example 3-3 Version Variable*

```
@if __VERSION__ >= "2003.03.0416"
 SIZE_RECT M4 {
        RANGE1 = [0, 1]
        RANGE2 = [0, 2]
        RANGE1_EDGESIZE = 10
        RANGE2_EDGESIZE =10
        CORNER_METHOD = clipped
 } (222)
 @else
 SIZE_RECT M4 {
        RANGE1 =[0, 1]
        RANGE2 =[0, 2]
        RANGE1_EDGESIZE = 10
        RANGE2_EDGESIZE =10
 } (222)
@endif
```

If a 2003.03.0416 or later version of code is used, the SIZE_RECT command with the CORNER_METHOD option will be executed. If an earlier version of code is used, the CORNER_METHOD option will not be used.

**Caution:**

Do not use a version number prior to 2000.2 in the conditional. Earlier versions of Hercules do not define the _ _VERSION_ _ variable, but the default is less than 2000.2. You may run runsets with the _ _VERSION_ _ variable with pre-2000.2 Hercules versions as long as the versions in the conditional are at least 2000.2.

Note:

The version variables use the MACROS control statement. The MACROS statement must be the first line in the runset. For more information, refer to “Macro Preprocessing for Hercules Runsets” on page 3-20.

## Runtime Flow Control

Unlike runset flow control IF statements, which evaluate before a run, runtime flow control statements are evaluated while Hercules is running.

There are two categories of runtime flow control. One is for checking the layer's emptiness in a design and the other is for text_short runtime detection. Both are explained in the two sections that follow.

| Arguments | Definitions |
|---|---|
| *var* | A variable name. |
| *num1* | Initial value of variable. |
| *num2* | Final value of variable. |
| *valid commands* | The commands used within runtime flow control. (See Table 3-1 on page 3-17.) |
| *condition* | There are two layer_empty conditions that can be checked.<br>• for empty layers: `layer_empty( layer_name )`<br>• for layers that are not empty: `!layer_empty( layer_name )` |

## Runtime Flow Control by Checking LAYER_EMPTY

Runtime flow control allows you to use conditional statements to check for empty layers in a design. An empty layer is defined as containing no data. The evaluation of these conditional statements is based on the existence of data in a group file.

The available conditional statements are:

IF
WHILE
FOR
EXIT WHEN

The arguments for the conditional statements are:

The runtime IF syntax is:

```
if ( conditional ) { valid commands} [else{ valid commands }]
```

For example:

```
if(layer_empty(tox)) {
   BOOLEAN via not pwell {
   } temp = t
}
else {
   BOOLEAN via not poly {
```

```
   } temp = t
}
```

The runtime WHILE loop syntax is:

```
WHILE ( condition ) { valid commands }
```

For example:

```
while(!layer_empty(t1)) {
   size t1 {
      undersize = 0.5
      verbose
   } temp = t1 (101)
}
```

The runtime FOR loop syntax is:

```
FOR var in num1 to num2 { valid commands }
```

For example:

```
for K in 10 downto 1 {
   size t1 {
      undersize = 0.5
      verbose
   } temp = t1 (101)
}
```

Or:

```
/* A safer while loop */
for K in 1 to 1000 {
   exit when (layer_empty(t1))
   size t1 {
      undersize = 0.5
      verbose
   } temp = t1 (101)
```

Or, use this syntax:

```
FOR var in num1 downto num2 { valid commands }
```

The runtime EXIT WHEN command syntax is:

```
EXIT WHEN ( condition )
```

Note:
EXIT WHEN is valid only in the body of a WHILE or FOR loop.

### Warning and Error Messages

The runtime flow control program issues warning or error messages for the following syntax constructs:

- An error generated when an invalid command is used. Only the commands listed in Table 3-1 on page 3-17 and their options can be used within a runtime flow control IF or ELSE statement.
- A warning message is issued when the same permanent layers are defined in the IF and ELSE statements. Runtime flow control allows you to redefine the permanent layer throughout the IF-ELSE statements as long as the layer is defined in different branches of the if statement. As a reminder, the warning message alerts you when a permanent layer is stated more than once.

Runtime flow control issues both a warning message and an error message for these Example 3-4 and Example 3-5.

*Example 3-4*

```
if(layer_empty(tox)) {              /*The permanent layer is
   BOOLEAN via not pwell {          /*redefined. A warning
                                      message is issued.*/
   } perm = t(9)
}
else {
   BOOLEAN via not poly {
      } perm = t(9)
}
```

*Example 3-5*

```
while(!layer_empty(t1)) {
    size t1 {
        undersize = 0.5
        verbose
    } temp = t1 (101)
}
```

*Table 3-1 Valid Commands for Runtime Flow Control*

| Data Creation Commands | Dimensional Checks | Utility Commands |
|---|---|---|
| ATTACH_PROPERTY | AREA | COMPARE_GROUP |
| BASIC_SIZE | CENTER_TO_CENTER | GRID_CHECK |
| BOOLEAN | DENSITY | LOAD_GROUP |
| BOUNDED_SIZE | ENCLOSE | SELF_INTERSECT |

*Table 3-1 Valid Commands for Runtime Flow Control(Continued)*

| Data Creation Commands | Dimensional Checks | Utility Commands |
|---|---|---|
| CELL_EXTENT | EXTERNAL | |
| CONNECTION_POINTS | INSIDE_EDGE | |
| COPY | INTERNAL | |
| CUT | LENGTH | |
| C_THRU | MASK_ALIGN | |
| DELETE | MOSCHECK | |
| EXPLODE_ALL | Multiple Rules<br>INSIDE_EDGE | |
| EXPLODE | NOTCH | |
| FLATTEN | RESCHECK | |
| LEVEL | | |
| NEGATE | | |
| POLYGON_FEATURES | | |
| PUSH | | |
| RELOCATE | | |
| REMOVE_OVERLAP | | |
| SELECT | | |
| SELECT_CELL | | |
| SIZE | | |
| TEXT_POLYGON | | |

### Missing Files

Hercules may print out a warning and continue when certain files specified in the runset do not exist. The warning can be changed to an error by using the MISSING_FILES runset directive.

It applies to:

- HEADER section files: SCHEMATIC, EQUIVALENCE, and BLACK_BOX_FILE
- CELL_FILE from these commands and Option sections: EXPLODE_ALL, FLATTEN, EXPLODE_OPTIONS, EXPLODE, SELECT_CELL, and CELL_EXTENT
- DATABASE_OPTIONS section: DB_CELL_LIST_FILE
- TEXT_POLYGON command: CELL_FILE and TEXT_FILE

The syntax of the MISING_FILES runset directive is:

```
#runset MISSING_FILES = ERROR|WARNING;
```

The default setting is WARNING.

Note:
You may also use the -mfe command-line option.

## Runtime Flow Control by Testing TEXT_SHORT

Runtime flow control allows you to use an IF conditional statement to check whether a text_short occurs after the execution of the TEXT command. The syntax is:

```
if ( conditional ) { valid commands } [else { valid commands }]
```

| Argument | Description |
|---|---|
| *condition* | text_short (default is FALSE) or !text_short. |
| *valid commands* | Commands used within runtime flow control (See Table 3-1.) |

*Example 3-6*

```
TEXT{
   layer_name by layer_name.text
   }
if (!text_short)
{
   copy layer_name { } temp = not_short
}
else
{
   copy layer_name { } temp = short
}
```

Note:
The check of the text_short would be valid only after executing the TEXT command, since it triggers the text_short flag.

## Macro Preprocessing for Hercules Runsets

The purpose of the macro preprocessing system is to process @include and conditional compilation instructions and macros. A runset is processed and an intermediate runset is created. This intermediate runset has the .evx extension. It is then used by Hercules as the original runset would have been for the commands to execute. When reporting any syntax errors, Hercules reports the file and line number in the original runset, not in the .evx file.

To use the macro preprocessing system, add a MACROS control statement to the beginning of the runset. Only comments and white space can precede the MACROS control statement.

When it is detected, Hercules invokes the gcpp preprocessor to generate the .evx file. This file has the same name as the runset but with the .evx extension and is placed in the current working directory.

This file is then used as the runset, and Hercules runs normally. After Hercules is finished running, the .evx file is not deleted. Examine this file to see what includes and substitutions were performed. Use the evp program to perform macro expansion, syntax analysis, and runset optimization without running the commands.

## The gcpp Preprocessor

Hercules is different from the gcpp preprocessor in that it requires the first statement in the runset to be the line, MACROS –I <includepath>, in order to invoke the gcpp preprocessor. Hercules uses the @ symbol in place of the # symbol for all preprocessor directives like @define and @include. Hercules uses @@ in place of ## for symbol concatenation. To ensure backward compatibility for Hercules runsets, the # symbol is used only for preprocessing directives such as #run_only.

Aside from this difference, Hercules functions the same as gcpp. For example, macros must be defined on a single line or incorporate the \ line-continuation character at the end of a line to make multiple lines look like a single line.

Example:

```
@define long_routine(X) \
     Subroutine1(X) \
     Subroutine2(X) \
     Subroutine3(X)
```

This is treated like a single line:

```
@define long_routine(X) Subroutine1(X) Subroutine2(X) Subroutine3(X)
```

For information on what happens during gcpp expansion in macro encryption, see the Macro Encryption section in the *Hercules Reference Manual*, Overview of Runset Organization chapter.

## Directives

The macro preprocessor's directives start with the @ symbol. This @ symbol should be in column 1 of the line.

### @define

The @define directive can be used to create a symbolic name and some associated fragment of text. If the symbolic name occurs at any point later in the runset, it will be replaced with the fragment of text. (However, this substitution does not occur within strings).

For example, if a runset contains:

```
@define Lamda 0.35
size WIN_XCPT {
     oversize = Lamda
} temp = ovrsz_win_up
```

the .evx file will contain:

```
size WIN_XCPT {
      oversize = 0.35
} temp = ovrsz_win_up
```

After an @define is expanded, the result is re-scanned for more symbolic names.

The @define directive can also have arguments. These arguments denote names within the text fragment that are replaced at the point the symbolic name is used.

For example, if a runset contains:

```
@define Squared(x) (x * x)
variable double big = Squared(2.3)
```

the .evx file will contain:

```
variable double big = (2.3 * 2.3);
```

It is important not to have any space between the symbolic name and the open parentheses symbol, as shown above.

## Predefined Symbols

The following symbolic names are already @defined:

| | |
|---|---|
| __FILE__ | a string containing the name of the file |
| __LINE__ | a number containing the line number in file |
| __DATE__ | a string containing the month, day, and year |
| __TIME__ | a string containing the hour, minute, and seconds. |

### @include

The @include directive is used to include the contents of another file. With the preprocessor enabled, it is an error to use the existing #include and include runset statements.

Since the contents of the files will not be scanned by the macro preprocessor, they have been disabled. For example:

```
@include "cmos_013.tech"
```

The preprocessor looks in the active directory for files. The preprocessor can be instructed to look elsewhere. The -I option in the MACROS control line adds another entry to the search list. (The active directory contains the file that has the @include directive.).

### @if, @else, and @endif

You can conditionally control which part of a runset is executed by using the conditional directives.

If the @if *expression evaluates to nonzero, the* directive selects the text in the following @ include statement. The @endif directive marks the end of the section, and the @else directive separates the section that is selected when the expression is true from the section that is selected when the expression is false. For example:

```
@define PROCESS "cmos05"

@if (PROCESS == "cmos05")
@  include "cmos5.tech"
@else
@  include "cmos7.tech"
@endif
```

The symbolic name PROCESS would first be substituted, resulting in:

```
@if ("cmos05"== "cmos05")
```

which would evaluate to:

```
@if 1
```

Therefore, the cmos5.tech file would be included, and the cmos7.tech file would not be included.

*Example 3-7 Using a Defined Name as a Flag That Contains 1 or 0*

```
@define IS_CMOS_5 1
@define IS_CMOS_7 0

@if IS_CMOS_5
@  include "mos5.tech"
@endif

@if IS_CMOS_7
@  include "cmos7.tech"
@endif
```

*Example 3-8 Using the Existence of a Name as a Flag*

```
/*HAS_METAL_5 is defined, but its text fragment is empty.*/
@define HAS_METAL_5

@if !defined(HAS_METAL_6)
@  if defined(HAS_METAL_5)
@    include "metal5.rules"
@  endif
@else
@  include "metal5_6.rules"
@endif
```

The @if defined *name* directive has a shortcut of @ifdef *name*. Also, @if !defined *name* can be abbreviated as @ifndef *name*. See Example 3-9.

*Example 3-9 @if Defined Name*

```
@define HAS_METAL_5
@ifndef HAS_METAL_6
@  ifdef HAS_METAL_5
@    include "metal5.rules"
@  endif
@else
@  include "metal5_6.rules"
@endif
```

Another shortcut is the @elif directive. It can be used to select from a set of values. See Example 3-10.

*Example 3-10 @elif Directive*

```
@define PROCESS "cmos5"

@if (PROCESS == "cmos5")
        ...
```

```
@elif (PROCESS == "cmos7")
        ...
@elif (PROCESS == "cmos8")
        ...
@endif
```

Example 3-11 can be used, but when working with big sets you might find it easier to use the shortcut directive shown in Example 3-10.

*Example 3-11 Working with Big Sets*

```
@define PROCESS "cmos5"

@if (PROCESS == "cmos5")
...
@else
@  if (PROCESS == "cmos7")
   ...
@  else
@    if (PROCESS == "cmos8")
     ...
@    endif
@  endif
@endif
```

Example 3-12 shows macro usage for instantiating blocks of parametrized Hercules code. A macro was used for this wide metal check because the design rule requirements were equivalent for METAL1 and METAL2. This not only saves time but avoids copy/paste errors while coding a runset.

Note:
Because the macro does not fit on one line, end the new line with a backslash (\), making sure it is the last character on the line.

*Example 3-12 Macro Usage for Instantiating Blocks of Parametrized Hercules Code*

```
@define widemetal(METAL)\
INTERNAL METAL {\
   SPACING < 5.00\
}TEMP = junk1\
SIZE junk1 {\
OVERSIZE = 0.010 } TEMP = junk1\
BOOLEAN METAL NOT junk1 {} TEMP = junk2\
EXTERNAL junk2 METAL{\
   SPACING < 1.5\
}(222)\

widemetal(METAL1)
widemetal(METAL2)
```

### @if Expressions

The expressions that are used for conditions can contain integers, floating point numbers, and strings. Each expression has several operators (defined below).

Note:
Because the IF-ELSE constructions and the @if directives have different precedence of operators, use parentheses to insure that the order of precedence is correct.

Octal (base 8) integers begin with a 0 and hexadecimal (base 16) integers begin with 0x. For example, 015 (base 8) is 13 (base 10) and 0x23 (base 16) is 35 (base 10).

The OP expr operators, ordered by precedence, are:

| | |
|---|---|
| - | the negative of the value |
| ! | the logically inverse value (not) |
| + | the same integer or floating value |
| ~ | the ones-complement of the value |

The expr OP expr operators, ordered by precedence, are:

| | |
|---|---|
| * | multiplication |
| / | division |
| % | (mod) remainder |
| + | addition (concatenate for strings) |
| - | subtraction |
| << | left shift |
| >> | right shift |
| == | equal |
| != | not equal |
| <= | less than or equal |
| >= | greater than or equal |
| < | less than |

| | |
|---|---|
| > | greater than |
| & | bitwise and |
| ^ | bitwise exclusive or |
| \| | bitwise or |
| && | short circuit and; for (a && b) don't evaluate b if a was false |
| \|\| | short circuit or; for (a \|\| b) don't evaluate b if a was true |

Note:
Hercules runset variables cannot be used in expressions. Because this evaluates before the runset is parsed, runset variables have not yet been discovered, and have therefore not yet been assigned a value.

*Example 3-13*

```
/*There is an @define for the d2t_width.*/
@if (RULES == "ALL" || RULES == "TOX" || RULES == "LOWLEVS")
@  include "tox.rule"
@endif

@if (RULES == "ALL" || RULES == "D2T" || \
     (RULES == "ALLMET" && d2t_width != 0))
@  include "d2t.rule"
@endif
```

**@undef**

The @undef directive is used to remove an @define. Do this to change its text fragment or to remove it if it is being used as an existing flag.

*Example 3-14*

```
@define VOLTAGE 80.0
@define SPACING 0.48

@if (VOLTAGE > 60.0)
@  undef SPACING
@  define SPACING 0.70

/* since the operating voltage is high, make sure
          that low voltage options are off */
@  ifdef DO_LOW_VOLTAGE
@    undef DO_LOW_VOLTAGE
@  endif
@endif
```

### @error and @warning

The directive @error causes the preprocessor to report a fatal error causing Hercules to abort. The rest of the line that follows @error is used as the error message. Use @error inside of a conditional that detects a combination of arguments the runset does not properly support. For example:

```
@if METAL_LAYERS > 4
@  error Hey, this runset can only do up to 4 layers of metal
@endif
```

@warning is similar, but Hercules continues.

## The MACROS Control Line

The MACROS control line enables the preprocessor and begins at the first line of your runset. It can also contain options that control the preprocessor.

The -I option appends another path to the set of directories that are searched when looking for an include file.

The -D option defines a macro in the same manner as the @define directive.

The MACROS control line can be continued to additional lines by ending new lines with a backslash (\). This can help when there are many include directories and defined options.

The MACROS control line is special in that the value of environment variables can be substituted. Environment variables are marked by the dollar sign ($). There are three forms: $name, $(*name*), and ${*name*}. For the first form name can contain A to Z, a to z, 0 to 9, and _. The environment variable name for the latter two can contain even more characters. They are not replaced within strings. For example:

```
MACROS -I/u/tech/cmos5 -DRULES=$(RULES) -DTECH=$(TECH) \
     -DTOP_METAL=$(TOP_METAL) -DVOLTAGE=$(VOLTAGE
```

In this example RULES, TECH, TOP_METAL, and VOLTAGE are strings. The environment was set up with:

```
setenv RULES     '\"ALL\"'
setenv TECH      '\"CMOS025\"'
setenv TOP_METAL '\"4LM\"'
setenv VOLTAGE   '\"ALL\"'
```

Note:

Double quotation marks (") have to be escaped so they make it to the preprocessor. To protect them from the csh, place csh within single quotation marks (' ').

## Passing Arguments by Include Files

While running Hercules from shell scripts, you might want to control the values used in a runset. For instance, using the -D option to MACROS (as mentioned above) is one method; however, it is limited to simple values. The -D option can have problems with strings containing blanks. A more flexible method is to use the shell script to write an include file. For example:

```
set tech = CMOS025
   echo '@define RULES "ALL"' > control.tmp
   echo -n '@define "TECH"' >> control.tmp
   echo -n $tech >> control.tmp
   echo    '"' >> control.tmp
   echo '@define TOP_METAL "4LM"' >> control.tmp
   echo '@define VOLTAGE "ALL"' >> control.tmp
```

Run:

```
hercules my_runset.ev
```

where my_runset.ev contains:

```
MACROS -I/u/tech/cmos5
@include "control.tmp"
```

In this way, the arguments in the runset are passed from the shell into the runset.

## Aliasing Keywords

This section lists all the keywords that may receive an alias. The alias may be whatever you choose, but it must be defined in your runset before you use the alias. To define aliases in your runset, use the following syntax:

```
ALIAS {
   ci = cut_inside
   . . .
}
```

The following list shows all the keywords that may receive an alias.

| | |
|---|---|
| ADJ_SPACING | LEVEL_NON_ORTHOGONAL |
| AREA<br>/* Alias for AREA option of the DENSITY Command, not AREA Command*/ | LEVEL_ONLY_RAM_ROM |
| BENDING_LONGEDGE | LONGEDGE |

| | |
|---|---|
| BORDER | LONGEDGE_RANGE |
| CELL_LEVEL | LONGEDGE_TO_EDGE |
| CHECK_ENCLOSED | MAXIMUM_SPACING |
| CLEANUP | NO_CUT |
| COLUMNS | NO_CUT_FILTER |
| CONCAVE_TO_CONCAVE | NODAL |
| CONCAVE_TO_EDGE | NON_PARALLEL |
| CONCAVE_TO_CONVEX | NOTCH |
| CONTINUE | OTHER_SPACING |
| CONVEX_TO_EDGE | OUTSIDE |
| CONVEX_TO_CONVEX | OUTSIDE_CONCAVE_TO_INSIDE_ CONVEX |
| CONVEX_TO_CONVEX_FILTER | OUTSIDE_CORNER_TO_INSIDE_EDGE |
| CONVEX_TO_EDGE_FILTER | OVER_UNDER |
| CONVEX_TO_CONVEX_RANGE | OVERLAP |
| CONVEX_TO_EDGE_RANGE | OVERSIZE |
| CORNER_TO_PARALLEL_EDGE | PARALLEL |
| DELETE_NOT | PLACE |
| DELTA_WINDOW | POINT_TOUCH |
| DELTA_X | PUSH_AND |
| DELTA_Y | PUSH_BOUNDARY |
| DEVICE_WIDTH | PUSH_CHIP_HIERARCHY |
| DEVICE_LENGTH | PUSH_GROUP_HIERARCHY |
| DIMENSION | RANGE |

| | |
|---|---|
| EDGE_45 | RATIO<br>/* Alias for RATIO option of the DENSITY Command, not RATIO Command*/ |
| EDGE_45_RANGE | RECTANGULAR |
| EDGESIZE | ROWS |
| ENCLOSED | SEGMENT |
| ERROR_COORD | SET_CORNERS_TO_SPACING |
| ERROR_GRAPHICS | SIZE |
| FILL_ORTH_CORNER | SPACING |
| FILTER_VECTOR | STYLE |
| FLAG_ACUTE_ANGLE | TOP_NET |
| FLAG_CONVEX_90 | TOGGLE_S_CORNER |
| FLAG_OBTUSE_ANGLE | TOGGLE_BOX_CORNER |
| FLAT_OUTPUT | TOUCH |
| FLAG_WIDTH | UNDER_OVER |
| GLOBAL_TEXT | UNDERSIZE |
| INCREMENT | VERBOSE |
| INSIDE | VIOLATIONS |
| INSIDE_CORNER_TO_OUTSIDE_EDGE | WIDTH_LONGEDGE |
| INSIDE_CONCAVE_TO_OUTSIDE_ CONVEX | WIDTH_RANGE |
| LAYER2_WIDTH | WINDOW |
| LEVEL_LAYER | XY_OFFSET |

# 4

# Generating Input for Hercules

*This chapter describes the input files required to run Hercules. You can create some of the input files automatically using the utilities described in this chapter. The chapter is divided into three sections: The design databases section of this chapter discusses methods of reading in different database formats; the runset section explains converter utilities; and the netlists section explains the Hercules netlist format and the use of the netlist translator.*

## Design Databases

In the beginning of a Hercules run, primary group files are created that consist of one file per layer listed in the ASSIGN section of the runset file. Each file is a database describing that layer's existence throughout the design—a design database. This section explains how to read in different design databases using

GDSII
GDSOUT
OASIS
Milkyway

## GDSII

This section describes methods for reading in GDSII data, including:

- Using Hercules to Stream in the GDSII Database
- Using Command GDSIN to Stream In the GDSII Database
- Using Multiple GDSII Files
- Using GZIPPED GDSII Files

### Using Hercules to Stream in the GDSII Database

Direct GDSII Input reads a GDSII-formatted stream file directly in to Hercules. When compared to the GDSIN utility, GDSII Direct Input enhances performance and reduces disk-space consumption by bypassing the GDSII-to-LTL database translation step. See Table 4-1 for a list of GDSIN options supported by GDSII Direct Input.

Note:
In cases where the -z option is used, an uncompressed file is created and disk consumption is comparable to the GDSIN utility.

Both methods of GDSII data input are supported by Hercules. The default method is to use the GDSII Direct Input to Hercules option. In cases where GDSII Direct does not support all of the options (gdsin_options =) specified in the OPTIONS section of the runset, the program switches to the GDSIN utility.

*Table 4-1 GDSIN Options Supported by GDSII Direct Input*

| Option | Description |
|---|---|
| -af | Aborts if write fails (that is, causes GDSIN to abort if data cannot be written to a disk). |
| -b | Converts BOX elements to RECTANGLE elements. |
| -box w \| e | Prints a warning (w) or aborts with an error (e) when a BOX element is encountered in the GDSII file. Normally, a BOX is translated without notification. |
| -cc u \| l | Forces the character case of structures to uppercase (u) or lowercase (l). |
| -lf *file_name* | Specifies the name of the MAPPING_FILE, such as layer-mapping.file. See “Creating a MAPPING_FILE” on page 4-3. |

*Table 4-1 GDSIN Options Supported by GDSII Direct Input(Continued)*

| Option | Description |
|---|---|
| -ma m | Merges POLYGON/PATH/TEXT/BOX/NODE data into existing cells. This option is always implied even if it is not specified in the runset. |
| -mr | Merges references (AREF/SREF) into existing cells. This option is always implied even if it is not specified in the runset. |
| -nl | Creates a database capable of supporting structure names up to 127 characters in length. The default maximum structure name length is 32 characters. |
| -pt1 w \| e | Prints a warning (w) or aborts with an error (e) when a PathType=1 PATH element is encountered in the GDSII file. Normally, a PATH of PathType=1 is translated without notification. |
| -pt4 w \| e | Prints a warning (w) or aborts with an error (e) when a PathType=4 PATH element is encountered in the GDSII file. Normally, a PATH of PathType=4 is translated without notification. During translation, a PathType=4 PATH is converted into a PathType=0 PATH after applying the BGNEXTN and ENDEXTN attributes associated with PathType=4 PATH elements. |
| -u *ValueUOM* | Sets default smallest library user unit and unit of measurement for the output library (for example: `-u 0.001`, which is microns). |
| -wc | Causes GDSIN to continue to completion instead of aborting if the expected ENDSTRUCT and ENDLIB records are missing. A warning is issued. |
| -z | Unzips input files into a group directory using gzip. The files are removed after the run. |

### Creating a MAPPING_FILE

The purpose of the MAPPING_FILE is to filter layers and datatypes from the input library file to be written to the output GDSII file. This file also can be used to map layers or datatypes to new layers or datatypes.

Filtering is accomplished by listing layers and datatypes in the following format: (layer_list)/(datatype_list). When GDSOUT translates the library, only the layers and datatypes listed in this file are translated to the output GDSII file.

Mapping is accomplished by listing layers and datatypes in the following format: (layer_list)/(datatype_list) > (new_layer_list)/(new_datatype_list). When GDSOUT translates the library, the layers and datatypes listed to the left of the greater-than sign (>) are mapped to the layers and datatypes listed to the right of the greater-than sign in the output GDSII file.

Several example entries from a MAPPING_FILE are shown in Table 4-2.

*Table 4-2 MAPPING_FILE Example Entries*

| Example | What is translated |
|---|---|
| (5) | layer 5; all dtypes |
| 5 | layer 5; all dtypes |
| 8/7 | layer 8; dtype 7 |
| (0-63) | layers 0 through 63; all dtypes |
| 0-63 | layers 0 through 63; all dtypes |
| (0-255)/0 | all layers; dtype 0 only for each layer |
| */0 | all layers; dtype 0 only for each layer |
| (1 3 5 7 9)/22 | layers 1, 3, 5, 7, and 9; dtype 22 only for each layer |
| 22/(1 3 5 7 9) | layer 22; dtypes 1, 3, 5, 7, and 9 only |
| (0-63)/(0-63) | layers 0 through 63; dtypes 0 through 63 for each layer |
| 1 > 2 | layer 1; all dtypes *maps to* layer2; all dtypes |
| (64-255) > 63/63 | layers 64 through 255; all dtypes *maps to* layer 63; dtype 63 |
| (41-50) > (8-17) | layers 41 through 50; all dtypes *maps to* layers 8 through 17; all dtypes |
| 0-63)/255 > 255/(0-63) | layers 0 through 63; dtype 255 *maps to* layer 255; dtypes 0 through 63 |
| 255/(0-255) > (0-255)/255 | layer 255; dtypes 0 through 255 *maps to* layers 0 through 255; dtype 255 |

*Syntax Rules for the MAPPING_FILE.* You must follow these syntax rules for creating the mapping file.

- Each layer in a layer_list must be separated by a space. Or, if specified in a range, the range must consist of a starting layer and an ending layer joined by a dash (-).
- Generally, parentheses ( ) should surround the layer list. However, you can omit parentheses if you specify only one layer, or if you specify one layer range and no datatype list.
- When an asterisk (*) is used in the place of a layer list, it represents layers 0 through255.
- A slash (/) is used to separate layers and datatypes.
- Each datatype in a datatype_list must be separated by a space. Or, if specified in a range, the range must consist of a starting datatype and an ending datatype joined by a dash (-).
- Generally, parentheses ( ) should surround the datatype list. However, you can omit parentheses if you specify only one datatype.
- When mapping layers and/or datatypes to new layers and/or datatypes, use the greater-than sign (>) to separate the layers and datatypes. The layers/datatypes on the left of the greater-than sign are mapped to the layers/datatypes on the right of the greater-than sign. The wildcard-asterisk (*) cannot be specified on the right of the greater-than sign. If a sequence or range of layers/datatypes is specified on the left-hand side of the greater-than sign, an equivalent number of layers/datatypes must be on the right-hand side of the greater-than sign, or just one layer/datatype.
- When mapping layers and datatypes, all layers are translated if the MAPPING_FILE does not contain any filtering instructions. For example, if your MAPPING_FILE contains only 6 > 8, all data is translated and the data on layer 6 is mapped to layer 8. However, if your MAPPING_FILE contains a 6 and then 6 > 8, only layer 6 is translated and it is mapped to layer 8.
- A pound sign (#) can be placed anywhere on a line to indicate that the rest of the line is a comment.

### Setting the Database Resolution

Because coordinates are stored as integers, increasing the database resolution with the -u option will require more data bits of storage and, therefore, reduce the maximum allowable coordinate. The limits on the coordinate according to the smallest library unit are as follows:

*Table 4-3 Database Resolution*

| Library Unit | Maximum Coordinate |
|---|---|
| -u 0.001 | 1073741.000 |
| -u 0.000125 | 134217.625 |
| -u 0.0001 | 107374.1000 |
| -u 0.00001 | 10737.41000 |
| -u 0.000001 | 1073.741000 |
| -u 0.0000001 | 107.3741000 |
| -u 0.00000001 | 10.73741000 |

## Using Command GDSIN to Stream In the GDSII Database

GDSIN translates a GDSII-formatted stream file to an LTL-formatted database. The GDSIN translation utility accepts many command-line options that control its execution. Direct GDSII Input will be the default method for reading in GDSII data (see Table 4-1). To force Hercules to use the GDSIN utility, specify the -force-gdsin option on the command line.

Note:

When merging an input GDSII file into an existing library (using `-ea m`), the library might not be open for edit.

The syntax is:

```
gdsin [-a] [-aa] [-af] [-b] [-box w|e] [-cc u|l] [-ea m|r] [-f] [-fs val1
val2 val3 val4] [-g directory] [-if filename propN] [-ip propN
text_layerN] [-lf filename] [-ln value] [-lp password] [-m value] [-ma
a|n|o|q|r] [-ma m] [-mr] [-n] [-nl] [-pl value] [-pr "oldstring"
"newstring"] [-ps oldchars newchars] [-pt1 w|e] [-pt4 w|e] [-tm value] [-
tp] [-ts oldchars newchars] [-U] [-u ValueUOM] [-v] [-version] [-v3]
[-wc] [-xf file_name] [-xp] [-z] gdsii_file [library]
```

| Argument | Description |
|---|---|
| `-a` | Creates the output LTL library in the machine architecture's non-native format. (This option is available only when GDSIN outputs an LTL-formatted library.) By default, GDSIN creates LTL data in the machine architecture's native byte-storage format: HighByte-LowByte for HP, IBM/RS000, SGI, and SUN; LowByte-HighByte for DEC/ALPHA and INTEL/PC. |
| `-aa` | Forces the absolute-angle attribute of all TEXT elements to be enabled |
| `-af` | Aborts if write fails. |
| `-b` | Converts BOX elements to RECTANGLE elements. |
| `-box w \| e` | Prints a warning (w) or terminates with an error (e) when a BOX element is encountered in the GDSII file. Normally, a BOX is translated without notification. |
| `-cc u \| l` | Forces the character case of structures to uppercase (u) or lowercase (l). |
| `-ea m \| r` | When the output library already exists, this option causes GDSIN to merge (m) GDSII stream data into the existing library or to replace (r) the existing library. When choosing merge options, all of the input libraries must have the same hierarchy; otherwise there can be phantom data generated after the libraries are merged. |
| `-f` | Directs all translation messages to a log file named *gdsii_file*.log. *gdsii_file* represents the input GDSII file name. |
| `-fs val1 val2 val3 val4` | Defines the four font-scale factors for the output library. When this option is not used, the default scale factors are: -1.5 2.0 5.0 10.0 |
| `-g directory` | Defines the directory where the output library is to be written. The directory path can be relative or absolute and the directory does not need to exist already |
| `-if filename propN` | Specifies the instance data file and property number. See "Property Value" on page 4-10 for information about the property number. |

| Argument | Description |
|---|---|
| `-ip` *`propN text_layerN`* | Converts instance text to property. If a TEXT element on *text_layerN* immediately follows an SREF element in a GDSII file, the value of the text string is applied to the SREF as a property defining the property number (*propN*). |
| `-lf` *`filename`* | Name of MAPPING_FILE. See "Creating a MAPPING_FILE" on page 4-3. |
| `-ln` *`value`* | Specifies the library index number to translate from a multi–library GDSII stream file. By default, only the first library is translated from a multi–library file. |
| `-lp` *`password`* | Specifies the password for the current library. |
| `-m` *`value`* | Specifies a magnification factor to scale all data translated into the output library. |
| `-ma a \| n \| o \| q \| r` | When libraries are merged (using `-ea m`), this option specifies when structures in the input GDSII file should replace structures with the same name in the output library: always (a), never (n), only if older (o), query user for each structure (q), or rename (r).<br>Note: q is an interactive option that is not recommended for Hercules use. |
| `-ma m` | Merges (m) POLYGON/ PATH/TEXT/BOX/NODE data into existing cells |
| `-mr` | Merges reference AREFs or SREFs data into existing cells. |
| `-n` | Specifies that GDSIN should query you when it translates each structure name. |
| `-nl` | Creates a database capable of supporting structure names up to 127 characters in length. The default maximum structure name length is 32 characters. |

| Argument | Description |
|---|---|
| `-pl` *value* | The maximum point checking value for the BOUNDARY element is 600 The maximum point checking value for the PATH element is 200. While the point number of an element exceeds the polygon maximum point limit value specified by the -pl option, Hercules prints out a warning message, but keeps this element and continues running, except when the element exceeds the maximum capacity of Hercules. The maximum capacity point number for the PATH element is 32767. The maximum capacity point number for the BOUNDARY element is 2147483647. |
| `-pr "`*oldstring*`" "`*newstring*`"` | Defines a property string replacement. GDSIN replaces all property occurrences of oldstring with newstring. |
| `-ps` *oldchars newchars* | Substitutes the property character of *oldchars* with *newchars*. |
| `-pt1 w \| e` | Prints a warning (w) or aborts with an error (e) when a PathType=1 PATH element is encountered in the GDSII file. Normally, a PATH of PathType=1 is translated without notification. |
| `-pt4 w \| e` | Prints a warning (w) or aborts with an error (e) when a PathType=4 PATH element is encountered in the GDSII file. Normally, a PATH of PathType=4 is translated without notification. During translation, a PathType=4 PATH is converted into a PathType=0 PATH after applying the BGNEXTN and ENDEXTN attributes associated with PathType=4 PATH elements. |
| `-tm` *value* | Forces the specified magnification value to be applied to all the TEXT elements. |
| `-tp` | Creates text from properties. |
| `-ts` *oldchars newchars* | Specifies the text character substitution of *oldchars* with *newchars*. |
| `-U` | Queries you for the output library's user units. The default units are those defined in the input GDSII file or those overridden by the -u option value *ValueUOM*. |
| `-u` *ValueUOM* | Defines the default smallest library user unit value and unit of measurement for the output library. See "Setting the Database Resolution" on page 4-6. |

| Argument | Description |
|---|---|
| `-v` | Prints the GDSIN version number. |
| `-version` | Prints the GDSIN version number and the version numbers of the libraries GDSIN uses. |
| `-v3` | Causes GDSIN to support only GDSII Stream Version-3. When this option is used, GDSIN translates only Version-3 data elements and attributes contained in the input GDSII file. Data attributes such as external, template, and plea are ignored and not translated. |
| `-wc` | Causes GDSIN to continue to completion instead of aborting if the expected ENDSTRUCT and ENDLIB records are missing. |
| `-xf` *`filename`* | Specifies the name of the structure cross-reference file. The filename file should contain a list of GDSII-structure-name LTL-structure-name definitions. |
| `-xp` | Explodes PATH elements in the input GDSII file into BOUNDARY elements. |
| `-z` | Ungzips a file before translating it into LTL-format data. |
| *`gdsii_file`* | Specifies the name of the GDSII-formatted stream input file. |
| *`library`* | Specifies the name of the output library GDSIN creates. You can use the default library name contained in the input GDSII file by specifying an underscore (_) in this variable.<br>Note: You can use the library variable only when you have also specified the gdsii_file variable. |

### Property Value

Each line of the instance data file (a text file) specified in the if command contains five or seven fields:

Property_value
Cell_name
Parent_name
x
y
angle
reflection

The last two fields, angle and reflection, are optional. The x, y, and angle fields are floating point numbers. Additional fields are strings. The reflection fields N, No, 0, and Off signify NO REFLECTION. The reflection fields Y, Yes, R, Refl, 1, and On signify REFLECTION. Notice that all of these fields are case-insensitive.

While GDSIN runs, it attempts to match *cell_name*, *parent_name*, *x, y*, *angle*, and *reflection*. If a match is found, the *property_value* string is assigned to the property number (*propN*) for the instance.

## Using Multiple GDSII Files

Hercules has the ability to read in and merge together multiple GDSII files. The GDSII files assigned to the INLIB option in the HEADER section of the runset are used as input library files.

If multiple paths and filenames are assigned to the INLIB variable, the output of the GDSIN utility merges all of these data files into one library. Each file can contain an optional path to indicate the file location. A set of specific layers and datatypes are extracted from each GDSII file. This set of layers and datatypes is determined by the *Layer_String* field and can be mapped to new layers and datatypes by adding a positive integer, which is provided at the *Adding_Number* field.The syntax is:

```
INLIB = Path1/FileName1.gds[LayerString1]+AddingNumber1; ... ; PathN/
FileNameN.gds[LayerStringN]+AddingNumberN
```

| Argument | Description |
|---|---|
| *PathN* | Directory path of the nth GDSII input file. |
| *FileNameN* | Name of the nth GDSII input file. |
| *LayerStringN* | Layers are extracted from the nth GDSII output file by the GDSIN utility. Comprised of layer entries. Each layer entry is separated from other layer entries by a comma (,). The syntax of the layer entry follows. If *LayerStringN* is not provided, all layers are extracted from the nth GDSII input file. |
| *AddingNumberN* | The adding number of the nth GDSII input file. All layer entries of *LayerStringN* are mapped to new layers whose layer numbers are the original ones added by *AddingNumberN*. To those layer entries where the greater-than sign (>) is used, the new layers are the ones on the right of the greater-than sign. The adding number does not take any effect. |

The syntax rules for layer entry are almost the same as those for the layer mapping file in the GDSIN utility, except that the wildcard-asterisk (*) is not allowed for layer entry.

- Each layer in a *layer_list* must be separated by a space. Or, if specified in a range, must consist of a starting layer and an ending layer joined by a dash (-).
- Generally, parentheses ( ) should surround the layer list. However, they can be omitted if only one layer is specified, or one layer range and no datatype list.
- A slash (/) is used to separate layers and datatypes.
- Each datatype in a *datatype_list* must be separated by a space. Or, if specified in a range, it must consist of a starting datatype and an ending datatype joined by a dash (-).
- Generally, parentheses ( ) should surround the datatype list. However, they can be omitted if only one datatype is specified.
- When mapping layers or datatypes to new layers or datatypes use the greater-than sign (>) to separate the layers and datatypes. If the layers or datatypes to the left of the greater-than sign are mapped to the layers or datatypes to the right of the greater-than sign, an equivalent number of layers or datatypes must be on the right-hand side of the greater-than sign (>), or just one layer or datatype.
- When mapping layers and datatypes, all layers are translated if the MAPPING_FILE does not contain any filtering instructions. For example, if the MAPPING_FILE contains only 6 > 8, all data is translated and the data on layer 6 is mapped to layer 8. However, if the MAPPING_FILE contains a 6 then 6 > 8, only layer 6 is translated and mapped to layer 8.Examples are shown in Table 4-4.

*Table 4-4 Layer Entry Examples*

| Syntax | What is translated |
|---|---|
| (5) | layer 5; all dtypes |
| 5 | layer 5; all dtypes |
| 8/7 | layer 8; dtype 7 |
| (0-63) | layers 0 through 63; all dtypes |
| 0-63 | layers 0 through 63; all dtypes |
| (0-255)/0 | all layers; dtype 0 only for each layer |
| (1 3 5 7 9)/22 | layers 1, 3, 5, 7, and 9; dtype 22 only for each layer |
| 22/(1 3 5 7 9) | layer 22; dtypes 1, 3, 5, 7, and 9 only |

*Table 4-4 Layer Entry Examples(Continued)*

| Syntax | What is translated |
|---|---|
| (0-63)/(0-63) | layers 0 through 63; dtypes 0 through 63 for each layer |
| 1 > 2 | layer 1; all dtypes *maps-to* layer2; all dtypes |
| (64-255) > 63/63 | layers 64 through 255; all dtypes *maps-to* layer 63; dtype 63 |
| (41-50) > (8-17) | layers 41 through 50; all dtypes *maps-to* layers 8 through 17; all dtypes |
| 0-63)/255 > 255/(0-63) | layers 0 through 63; dtype 255 *maps-to* layer 255; dtypes 0 through 63 |
| 255/(0-255) > (0-255)/255 | layer 255; dtypes 0 through 255 *maps-to* layers 0 through 255; dtype 255 |

For example:

```
INLIB = /home/usr1/a.gds[3/2 , 6, (10-15)],
   usr2/b.gds[1 4 , 8>100]+50
```

The information extracted is:

- Extract from the file /home/usr1/a.gds "layer 3; dtype 2, layer 6; all dtypes, and layer 10 through 15; all dtypes".
- Extract from the file ./usr2/b.gds "layer 1; all dtypes > layer 51; all dtypes, layer 4; all dtypes > layer 54; all dtypes, layer 8; all dtypes > layer 100; all types".

## Using GZIPPED GDSII Files

Hercules has the ability to read or write gzipped GDSII library files directly. Specify the -z option in the GDSIN_OPTIONS variable of the OPTIONS section in the runset to read a gzipped input library. Specify the -z option in the GDSOUT_OPTIONS variable of the OPTIONS section in the runset to write a gzipped output library. An example of the syntax is:

```
HEADER  {
         INLIB = /path/gdsii_library.gds.gz
         FORMAT = GDSII
         OUTLIB = gdsii_output.gsd
         ...
        }
OPTIONS {
         GDSIN_OPTIONS = "-z"
```

```
 GDSOUT_OPTIONS = "-z"
}
```

## Direct GDSII Output from Hercules

Direct GDSII Output outputs GDSII-formatted data directly from Hercules. When compared to the GDSOUT utility, GDSII Direct Output enhances performance and reduces disk usage. See Table 4-5 for a list of GDSOUT options supported by GDSII Direct Output.

Both methods of GDSII data output are supported by Hercules. The default method is to use Direct GDSII Output to generate the output file. If all the options specified in the OPTIONS section of the runset are not supported by Direct GDSII Output, you may turn on the command-line option, -force-gdsout.

*Table 4-5 GDSOUT Options Supported by GDSII Direct Output*

| Option | Description |
|---|---|
| -lf *file_name* | Specifies the name of the MAPPING_FILE. See "GDSOUT" on page 4-15 for details on how to create a MAPPING_FILE. |
| -m *value* | Specifies the magnification factor to scale all data translated to the output GDSII file. |
| -nl *value* | Specifies the maximum length for structure names. Because the internal Hercules database (created with gdsin -nl, cifin -nl, or INIT_LIB_LN) supports structure names up to 127 characters long, this option should be used with a value of 32 when creating a GDSII Stream-compliant file; the GDSII Stream standard specifies a maximum structure name length of 32 characters. Structure names exceeding *value* characters in length are truncated and renamed if necessary to avoid naming conflicts. |
| -pl *value* | Defines the maximum point limit for BOUNDARY and PATH elements. The user-specified value may range between 5 and 500. This option should be used with a value of 200 when creating a GDSII Stream-compliant file; the GDSII Stream standard specifies a maximum polygon point limit of 200. However, most vendors' stream translators accept a greater number of points, beyond the 200 polygon point specification. |
| -w y \| n | Specifies whether to overwrite the filename *gdsii_file* if it already exists. If no (n) is specified, GDSOUT aborts if the file already exists. |
| -z | Pipes the output of the stream file to gzip. |

## GDSOUT

The GDSOUT utility translates an LTL-formatted database to a GDSII-formatted stream file. The GDSOUT translation utility accepts many command-line options that control its execution.

Note:
GDSOUT can be executed on the input library at any time.

The syntax is:

```
gdsout [-af] [-f] [-g directory] [-lf file_name] [-m value] [-n] [-nl
value] [-pl value] [-q y|n] [-r y|n] [-t font1 font2 font3 font4] [-v] [-
version] [-v3] [-w y|n] [-x password] [-x1 Ref1_password] [-x2
Ref2_password] [-xf file_name] [-z][Library] gdsii_file [Pattern]
```

| Argument | Description |
|---|---|
| `-af` | Aborts if data cannot be written to a disk. |
| `-f` | Directs all GDSOUT translation messages to a log file named *gdsii_file*.log, where *gdsii_file* represents the GDSII file name. |
| `-g directory` | Specifies the directory where the library to be translated is stored. |
| `-lf file_name` | Specifies the name of the MAPPING_FILE. |
| `-m value` | Specifies a magnification factor to scale all data translated to the output GDSII file. |
| `-n` | Specifies that GDSOUT should not output empty cells. |
| `-nl value` | Specifies the maximum length for structure names. Because the internal Hercules database (created with gdsin -nl, cifin -nl, or INIT_LIB_LN) supports structure names up to 127 characters long, this option should be used with a value of 32 when creating a GDSII Stream-compliant file; the GDSII Stream standard specifies a maximum structure name length of 32 characters. Structure names exceeding *value* characters in length are truncated and renamed if necessary to avoid naming conflicts. |

| Argument | Description |
|---|---|
| `-pl` *value* | Defines the maximum point limit for BOUNDARY and PATH elements. The user-specified value may range between 5 and 500. This option should be used with a value of 200 when creating a GDSII Stream-compliant file; the GDSII Stream standard specifies a maximum polygon point limit of 200. However, most vendors' stream translators accept a greater number of points, beyond the 200 polygon point specification. |
| `-q y \| n` | Specifies wether or not GDSOUT should query the user when it translates each structure name. |
| `-r y \| n` | Outputs referenced structures. When this option is set to yes (y) GDSOUT translates reference structures (that are in the input library) but are not specified by the -structure_pattern option, and also translates structures contained in reference libraries. |
| `-t` *font1 font2 font3 font4* | Specifies the text fonts (font1 through font4) to be referenced in the output GDSII file. An underscore (_) may be used for one or all font numbers to attach the font reference name GDSII:CALMAFONT.TX. |
| `-v` | Prints the GDSOUT version number. |
| `-version` | Prints the GDSOUT version number and the version numbers of the libraries GDSOUT uses. |
| `-v3` | Causes GDSOUT to support only GDSII Stream Version-3. When this option is set, GDSOUT translates only Version-3 data elements and attributes to the output GDSII file; data attributions such as external, template, and plex are ignored and not translated. By default, GDSOUT translates data elements and attributes to a GDSII Stream version of 5. |
| `-w y \| n` | Specifies whether to overwrite the filename *gdsii_file* if it already exists. If no (n) is specified and the file already exists, then GDSOUT aborts. |
| `-x` *password* | Specifies the password for the library that is to be translated. If a password does exist for the library and this option is not specified, a standard (non-echoing) interactive password prompt is issued. |

| Argument | Description |
|---|---|
| `-x1 Ref1_password` | Specifies the reference library 1 password. If a *Ref1_password* exists for reference library 1 and this option does not specify the correct password, a standard (non-echoing) interactive password prompt is issued. The *Ref1_password* is required if reference cells from the Permanent Reference library 1 are to be translated (using `-r y`). |
| `-x2 Ref2_password` | Specifies the reference library 2 password. If a *Ref2_password* exists for reference library 2 and this option does not specify the correct password, a standard (non-echoing) interactive password prompt is issued. The *Ref2_password* is required if reference cells from the Permanent Reference library 2 are to be translated (using `-r y`). |
| `-xf file_name` | Names a structure cross-reference file that is used with the -nl command line option. Any structure names are truncated and then renamed. GDSOUT creates the *file_name* if it does not exist and records all truncated/ renamed structures. If *file_name* does exist, GDSOUT uses the cross-referenced structure name definitions in the *file_name* when structure names need to be truncated. The format for the cross-references is a list of LTL-structure_name GDSII-structure_name definitions. |
| `-z` | Pipes the output of the stream file to gzip. |
| *library* | This variable specifies the name of the library to be translated to GDSII-stream format. |
| *gdsii_file* | This variable specifies the name of the GDSII-formatted stream output file. The default filename Library.gds may be used by specifying an underscore (_) for this variable. |
| *Pattern* | The *Pattern* variable specifies the structure names to be translated to the output GDSII file. To avoid wildcard interpretation by the operating system's shell, a *Pattern* containing the asterisk (*) and question mark (?) wildcards must be encased in single quotation marks (that is, '*Pattern*'); single quotation marks are additionally required when several structure names are specified for the *Pattern*. An underscore (_) may be specified to represent the translation of all structures contained in the library. |

## OASIS

Hercules can directly read OASIS input files. Intermediate layout translators are not needed. Information about the options that control Hercules processing of OASIS input data is in the *Hercules Reference Manual*, Detailed Commands chapter, OASIS_OPTIONS section. Refer to this section for the settings to use when reading OASIS input.

Note:
Hercules does not contain a translator for translating OASIS files to other layout formats.

## Milkyway

Milkyway is the Synopsys layout database. Milkyway extends the capabilities of GDSII by storing additional data, which provides easier processing within Hercules as well as better integration across EDA tools.

To make proper use of the Milkyway database using Hercules, it is important to understand the following key concepts of the database:

- Views
- Object Types

## Milkyway Views

Traditionally, a layout is made of a top cell that contains polygons and placements of other cells. In turn, each cell can also contain polygons and placement. All the information stored inside such a hierarchical database represents the chip design. In some cases, it can be useful to view only a subset of all the polygons. For example, during routing, IC Compiler is only concerned with data on blockage polygons, vias, and pins. Within Milkyway it is possible to create subsets of data. These subsets are called FRAM views. Having only blockage, vias, and pins inside the FRAM view allows IC Compiler to route quickly and efficiently. Each FRAM view is related to a CEL view. The FRAM view will have the same name as its associated CEL, only the extension changes. For example, TOP_DIGI.FRAM would be associated with TOP_DIGI.CEL.

Another example of a Milkyway View is the FILL view. Again, separating all FILL data into its own view allows tools running on Milkyway to conveniently access and process the data. Aside from common views such as FRAM and FILL, you might create your own view and name it as you like.

## Milkyway Objects

Milkyway can store additional information for each polygon. This additional information is saved as the Object Type. It can be used to identify objects such as wires, pins, paths, and so on.

Object Types supported by Hercules:

RECTANGLE
TEXT
POLYGON (maps to POLYGON and EXPANDEDPOLYGON)
BOUNDARY
PATH (maps to PATH and PACKEDPATH)
HORIZONTALWIRE
VERTICALWIRE
PIN (maps to DBT_PIN and DBT_PININST)

Note:
BOUNDARY is a reserved term in Milkyway. Do not use it as a layer name.

## Hercules Milkyway Flows

Milkyway flows within Hercules are set by the DB_MODE option with the DATABASE_OPTIONS section of the runset.

## Full

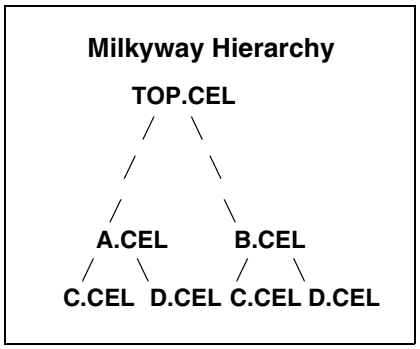
By default, Hercules reads all the data contained in the CEL view and the pins from the FRAM view.

### DRC Black Box

In some cases, you will want to black-box cells before running DRC checks. Hercules will read pin and blockage polygons from FRAM view only for cells that are specified as black-boxed. For cells that are not black-boxed, Hercules will read all polygons from the FRAM view. In a black-box flow, time will be saved because the FRAM views representing black-boxed cells contain considerably less data than CEL views.

To set up a black-box flow, set DB_MODE to DRC_BLACK_BOX and specify the black-box cells using either the DB_CELL_LIST or DB_CELL_LIST_FILE option. The following DRC flow example shows how to black box cell A.

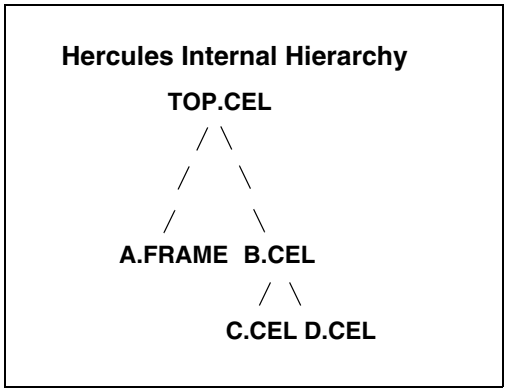
*Figure 4-1 DRC Black-Box Application*




In this example, because DB_MODE = DRC_BLACK_BOX, Place and Route Hercules reads the BLACK_BOX_FILE to determine what data to read in.

```
**************
DRC_BBOX = { A }
**************
ASSIGN {
       metal1     (23)
}
DATABASE_OPTIONS { DB_MODE = DRC_BLACK_BOX
   DB_CELL_LIST_FILE = my_db_cell_list.txt }
```

The following hierarchy tree and table show the outcome:




| CELL | VIEW | OBJECT_TYPE |
|---|---|---|
| TOP | CEL | polygons |

| CELL | VIEW | OBJECT_TYPE |
|---|---|---|
| A | FRAM | blockage, pins |
| C | ***skipped because A.FRAM has no instances in it*** | |
| D | ***skipped because A.FRAM has no instances in it*** | |
| B | CEL | polygons |
| C | CEL | polygons |
| D | CEL | polygons |

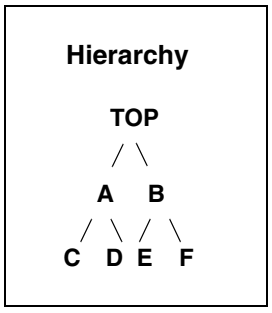
The result is that metal1 will contain a combination of polygons, blockage, and pins according to the above table.

### LVS Black Box

It is possible to black box cells for LVS. Hercules will read pin polygons from FRAM view only for cells that are specified as black-boxed. For cells that are not black-boxed, Hercules will read all polygons found in the CEL views. Time will be saved during the Extract phase because the FRAM views representing black-boxed cells contain considerably less data than CEL views. Time will also be saved during the COMPARE phase because Hercules will compare only the ports of the cells.

The following example shows how to black box cell A.

*Figure 4-2 LVS Black-Box Application*




An example of a Black-Box File:

```
**************
DATABASE_OPTIONS {
DB_MODE = LVS_BLACK_BOX
```

```
DB_CELL_LIST = { A = A }
}
ASSIGN {
metal1 (23)
}
```

In this example, because DB_MODE = LVS_BLACK_BOX, Hercules reads the DB_CELL_LIST to determine what data to read in. The following table shows the outcome.

| CELL | VIEW | OBJECT_TYPE |
|---|---|---|
| TOP | CEL | polygons |
| A | FRAM | pins |
| C | *** skipped because A.FRAM has no instances in it *** | |
| D | *** skipped because A.FRAM has no instances in it *** | |
| B | CEL | polygons |
| E | CEL | polygons |
| F | CEL | polygons |

The result is that metal1 will contain a combination of polygons and pins according to the previous table.

### Application Specific

Leveraging the additional information saved in a Milkyway database, it is possible to write specialized checks. Taking advantage of OBJECT TYPES, it is possible to quickly identify data that traditionally would need to be derived with data creation layers. For example, if you want to report all pins that do not fall on a cell's boundary, you could assign all pin data to a layer by setting VIEW = FRAM and OBJECT_TYPE = PIN. The runset would be as shown in Example 4-1.

*Example 4-1*

```
DATABASE_OPTIONS {
   db_mode = full
}
ASSIGN {
   metal1_pin     (16)   { VIEW=FRAM, OBJECT_TYPE=PIN }
   boundary (255)
}
select metal1_pin edge_touch boundary {
   INSIDE RANGE = [1,3]   }temp = goodMet1Pins
```

```
boolean metal1_pin not goodMet1Pins temp=badMet1Pins
```

### Summary of How Hercules Reads in Milkyway Data

When in DB_mode FULL and EXCLUDE_FRAM_VIEW is set to TRUE, Hercules only reads data from the CEL view for all cells. FULL mode ignores DB_CELL_LIST and DB_CELL_LIST_FILE options.

Normal cells are not specified in DB_CELL_LIST or DB_CELL_LIST_FILE. Black-boxed cells (DB_CELL) are specified in DB_CELL_LIST or DB_CELL_LIST_FILE.

*Table 4-6 FULL Mode*

| FULL | Normal Cell | DB_CELL |
|---|---|---|
| TOP | CEL & FRAM (PIN) | not affected |
| Exclude_Fram_View | CEL | not affected |
| Others | CEL & FRAM (PIN) | not affected |
| Exclude_Fram_View | CEL | not affected |

When in DB_mode DRC_BLACK_BOX or LVS_BLACK_BOX, Hercules only reads data from the CEL view for the top block when it is not listed in DB_CELL_LIST. Exclude_Fram_View option only affects FULL mode. DRC_BLACK_BOX and LVS_BLACK_BOX ignore the Exclude_Fram_View option.

The only different between LVS_BLACK_BOX and DRC_BLACK_BOX is that when reading FRAM view data DRC_BLACK_BOX mode reads in pins and blockage polygons, but LVS_BLACK_BOX mode reads in only pins. TOP cell always reads in CEL view data, no matter in which mode.

*Table 4-7 DRC_BLACK_BOX*

| DRC_BLACK_BOX | Normal Cell | DB_CELL |
|---|---|---|
| TOP | CEL | CEL & FRAM (pins and blockage polygons) |
| Others | CEL | FRAM (pins and blockage polygons) |

*Table 4-8 LVS_BLACK_BOX*

| LVS_BLACK_BOX | Normal Cell | DB_CELL |
|---|---|---|
| TOP | CEL | CEL & FRAM (PIN) |
| Others | CEL | FRAM (PIN) |

# Runsets

Three utilities will translate the files necessary to create a Hercules runset:

- The DRAC2HE utility translates a Dracula physical verification command file.
- The A2drc utility translates the design rules from a Place & Route (P&R) Milkyway database.
- The A2lvs utility translates netlisting information from a P&R Milkyway database.

The following sections describe how to use each of these utilities to translate files to the desired Hercules runset.

## DRAC2HE

The DRAC2HE utility translates a Dracula physical verification command file to a Hercules runset. The utility translates DRC, extraction, LVS, ERC commands, the INPUT_LAYER section, and some description section options. The output goes to stdout but you can redirect it with the UNIX command >.

The syntax is:

Note:
Some translated runsets might need modifications. See “DRAC2HE Error and Warning Messages” on page 4-26 for details.

```
drac2he [-BaseCell cellname] [-DupDevice] [-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`* | Allows you to specify one or more base cell names from a gate-array design. If you want to specify more than one base cell, use quotation marks (for example, "foo foo2"). |
| *`commandfile`* | Name of input Dracula physical verification command file. |
| `-DupDevice` | Turns on the MOS_ALLOW_DUPLICATE_DEVICE option in the LPE_OPTIONS section of the runset. |
| `-E` *`errorfile`* | Writes all error and warning messages generated by the translator to the user-specified file. |
| `-FE` *`errorfile`* | Writes only fatal errors to the user-specified file. |
| `-GateArray` | Sets up TECHNOLOGY_OPTIONS for gate array design. |
| `-IncTechDefs` | Includes TECHNOLOGY_OPTIONS defaults in a translated runset. |
| `-LvlSize` | Sets LEVEL_NON_ORTHOGONAL = TRUE for all size commands. |
| `-mosFilter` | Allows you to specify filtering options for MOS devices. These filter options will apply to all MOS devices in the runset and must be specified by number, and in quotation marks, for example "3 4". |
| `-MinCellOverlap` *`number`* | Sets the MIN_CELL_OVERLAP option to the user-specified *number* for sets vcell passes. |
| `-N` | Generates a runset with no comments. |
| `-nofixtemp` | Leaves Dracula physical verification TEMPORARY-LAYERS as they are in the runset. If you do not use this option, all instances of the TEMPORARY-LAYERS will be assigned unique names. |
| `-outgds` | Sets OUTPUT_FORMAT = GDSII. |
| `-Parse` | Parses the command file only. |
| `-rc` | Retains the case of all layer names. |

| Argument | Description |
|---|---|
| `-ShareBaseCell` | Turns on SHARED_BASE_CELL for gate-array designs. |
| `-Size` *number* | Sets the increment value for a SIZE INSIDE command. |
| `-text_map` *textfile* | Uses a text_map file for lower-level text. For more information, see "DRAC2HE Text Map File" on page 4-26. |
| `-V` | Displays DRAC2HE version. |
| `-Version` | Displays product and library version. |

## DRAC2HE Error and Warning Messages

Three types of cautionary messages are produced by DRAC2HE:

- *Error messages* appear when commands or options are not translated. They appear with the translated commands/options as comments within the runset.
- *Warning messages* display information that should be reviewed. For example, warning messages appear if two or more commands are combined, or if a command is moved to a new location. (Be aware that warning messages appear for many possible reasons. The previous two examples are representative of only two possible scenarios.)
- *Fatal Error messages* appear when a major runset error occurs or the runset will not parse. You might need to rewrite certain sections of the runset.

Note:

All error and warning messages have the string DRAC2HE in the message. The runset can be searched for this text string to locate the error and warning messages. Also, using the E command-line option prints all error and warning messages to a user-specified file.

## DRAC2HE Text Map File

The text map file is used to identify lower-level text in a design and associate it with a layer. This file consists of three columns, which correspond to the layer number of the text, the datatype number of the text, and the name of the layer to which the text is attached. The column containing the text datatype is optional. For example.

```
1   2   metal1
3       poly
24  32  metal2
```

The text map file must contain at least a layer number and a layer name on each line. There cannot be blank lines or comments in the file. Columns must be separated by spaces or by tabs.

Note:
DRAC2HE does not produce a default graphics command for LVS. The settings are:

```
create_views  = FALSE
no graphics command
```

### Description Block Translation Rules

The following are the options available for translation:

CNAMES-CSEN
CONNECT_LAY
DELCEL
ERROR-WIDTH-PATH
FILTER-LAY-OPT
FILTER-OPT
FILTER-SCH-OPT
FLAG-ACUTEANGLE
FLAG-NON45
FLAG-OFFGRID
GROUND-NODE
HCELL-FILE
INDISK
KEEP_SHORT_CAP
KEEP_SHORT_MOS
KEEP_SHORT_RES
POWER-NODE
PRIMARY
RESOLUTION
SCALE
SCHEMATIC
SMASH-CAP-TYP
SMASH-RES-TYP
SYSTEM
TEMPORARY-LAYERS
WINDEL
WINDOW

### INPUT_LAYER Block Translation Rules

- Layer-name definitions are translated.
- EDTEXT is translated.

### Translated Operation Block Commands

The following is a list of Dracula physical verification commands that are translated by DRAC2HE and their translation restrictions.

- The following Logical Operations are translated: OR, NOT, AND, XOR
  - Booleans passing connectivity are now translated.
  - Use logical equivalence to remove processing on the SUBSTRATE layer.
- The SELECT command is translated and all operators or options have a corresponding Hercules translation.
- The FLATTEN command translates to the COPY command.
- The CONNECT command is translated.
- The SCONNECT command is translated to a C_THRU command with the SCON_MODE option. If there is a corresponding SOFTCHK command, DRAC2HE adds another C_THRU command, which will output all the C_THRU errors. The additional C_THRU command outputs layer2 errors.
- The STAMP command translates to the CONNECT command.
- The LINK command translates to the TEXT command.
- For the SIZE command, the O option is the only option directly translated.
- The ENC command is translated.
  - Operations translated: LT, LE, RANGE, SELGT, SELGE, SELLT, SELLE, SELRA.
  - Options translated: C, C', E, N, N', O, P, P', R, R', S, T.
- The EXT command is translated.
  - Operations translated: LT, LE, RANGE, SELGT, SELGE, SELLT, SELLE, SELRA.
  - Options translated: C, C', E, G, H, N, N', O, P, P', R, R', S, T, V.
- The INT command is translated.
  - Operations translated: LT, LE, RANGE, SELGT, SELGE, SELLT, SELLE, SELRA.
  - Options translated: P, P', N, N', R, R', S, T.
- The WIDTH command is translated.
  - Operations translated: LT, LE, RANGE, SELGT, SELGE, SELLE, SELLT, SELRA.
  - Options translated: C, C', P, P', R, R', S.
- The LENGTH command is only translated if it is being fed by a conjunctive.

- The AREA command is translated.
- The PLENGTH command translates when being fed by a dimensional with the -tr options set.
- The CORNER command translates to the INTERNAL command.
- The following Device Extraction commands are translated: ELEMENT MOS, LDD, CAP, DIO, BJT, RES, PARAMETER CAP, PARAMETER RES.
- The ECONNECT command is translated to the DEV_CONNECT_CHECK command.
- The ELCOUNT command is translated to the DEVICE_COUNT command.
- The NDCOUNT command is translated to the DEV_NET_COUNT command.
- The PATHCHK command is translated to the NET_PATH_CHECK command.
- The LCONNECT command is translated to the SELECT command with the @co operator.
- The PROBE command is translated to the SELECT command with the TEXTED_WITH operator.
- The CHKPAR command is translated to the RATIO command if it is used with a COMPUTE SUM and an LEXTRACT command.
- The EQUATION command is translated to equations in the appropriate device extraction command.
- The GROW, SHRINK, and RELOCATE commands are translated.

### DRACULA_DEFAULTS for Dimensional Checks

The following defaults are set by DRAC2HE using the option in the runset, DRACULA_DEFAULTS = TRUE.

*Table 4-9 DRACULA_DEFAULTS*

| Dracula Physical Verification Command | Hercules Command | Hercules Options |
|---|---|---|
| EXT | EXTERNAL | FLAG_ACUTE_ANGLE = FALSE<br>NON_PARALLEL = TRUE<br>SHADOW = TRUE<br>POINT_TOUCH = FALSE<br>SHADOW_OTHER_LAYER = TRUE |

*Table 4-9 DRACULA_DEFAULTS(Continued)*

| Dracula Physical Verification Command | Hercules Command | Hercules Options |
|---|---|---|
| ENC | ENCLOSE | FLAG_ACUTE_ANGLE = FALSE<br>POINT_TOUCH = TRUE<br>NON_PARALLEL = TRUE<br>SHADOW = TRUE<br>SHADOW_OTHER_LAYER = TRUE |
| WIDTH | INTERNAL | NON_PARALLEL = TRUE<br>POINT_TOUCH = TRUE |
| INT | INSIDE_EDGE | NON_PARALLEL = TRUE<br>SHADOW = TRUE<br>SHADOW_OTHER_LAYER = TRUE<br>FLAG_INSIDE_PNT_TOUCH = FALSE<br>FLAG_INSIDE_EDGE_TOUCH = FALSE<br>FLAG_OUTSIDE_PNT_TOUCH = FALSE<br>FLAG_OUTSIDE_EDGE_TOUCH FALSE |

### Special Translation Operation

- When translating a SIZE UNDERSIZE operation on the bulk layer, Hercules uses the CELL_EXTENT command to create the bulk layer. The layer is then passed to the SIZE UNDERSIZE operation.

## A2drc

The A2drc utility translates the design rules from a Place & Route (P&R) Milkyway database to a Hercules DRC runset file. A2drc creates the DRC runset based on the design rules taken from the Layer section, the DesignRule section, and the ContactCode section of your P&R technology file. (For information on Hercules DRC use models, see the *Hercules DRC and ERC User Guide*, Executing Hercules DRC and ERC chapter.)

When executed, A2drc creates a directory named DRC in the current directory, as well as a runset named *cellname*.ev, where *cellname* represents the name of the top cell in the design. A2drc then runs Hercules on the new runset, unless you use the -pause command-line option. The results from the Hercules run are stored in the DRC directory. The syntax is:

```
A2drc -b cellname -l library_name [-batw] [-bbc cell_list] [-bbf

file_name] [-f] [-fill][-g group_dir] [-graf] [-help] [-ib] [-m mode] [-
nc] [-o output_layout_path] [-p layout_path] [-pause] [-pdl] [-r
filename] [-t] [-V]
```

*Table 4-10 Required Command-Line Arguments*

| Argument | Description |
|---|---|
| `-b cellname` | Name of top cell to check (required argument). |
| `-l library_name` | Name of Milkyway library on which to perform DRC (required argument). |

*Table 4-11 Optional Command-Line Arguments*

| Argument | Description |
|---|---|
| `-batw` | Treats all blockage data as thin wire. Default is false. |
| `-bbc cell_list` | Black-box cell list to be added to the black-box file in the HEADER section. |
| `-bb file_name` | Black-box file name in HEADER section |
| `-fill` | Uses Hercules to perform notch/gap filling of routing layers on a Place and Route design. Uses the spacing value from the technology file as the rule, and the notches found are written to the FILL view of the input Milkyway database. If a FILL view already existed, it will be overwritten. Default is false. |
| `-g group_dir` | Defines group_dir in DRC/runsetfile. Default is the group directory under DRC. Default is group. |
| `-graf` | Writes Graphics section in DRC/*runsetfile*. Default is false. |
| `-help` | Displays command-line information. Default is false. |
| `e` | Intra-blockage data checking. Default is false. |
| `-e mode` | Either FULL or DRC_BLACK_BOX. Default is full. |

*Table 4-11 Optional Command-Line Arguments(Continued)*

| Argument | Description |
|---|---|
| `-f` | Extension of the -fill option. This option performs notch filling, and then the spacing checks of other design rules in the technology file, such as minimum width and external spacing rules. Default is false. |
| `-nc` | Disables technology file checks. Default is false. |
| `-o` *`output_layout_path`* | Defines output_layout_path in DRC/*runsetfile*. Default is the layout directory under DRC. |
| `-p` *`layout_path`* | Path to Milkyway input library. Default is the current directory where A2drc was invoked. |
| `-pause` | Creates the runset then stops. Does not start Hercules. Default is false. |
| `-pdl` | Uses Astro predefined mask layers only. Default is false. |
| `-r` *`filename`* | Includes a user runset file in the DRC/ *runsetfile* |
| `-t` | Hercules output to TEMPORARY layers. The default is to output to the error hierarchy. Default is false. |
| `-V` | Displays version number. Default is false. |

In this example, a runset named TOP.ev is created within the directory DRC. All Hercules output files are also stored within the DRC directory.

```
A2drc -l DESIGNS -b TOP
```

## A2lvs

The A2lvs utility translates netlisting information from a Place & Route Milkyway database to a Hercules LVS runset file, equivalence file, and a schematic netlist file.

Note:
You cannot translate a GDSIN-created Milkyway library with A2lvs.

A2lvs creates the LVS runset based on the Layer and ContactCode sections of your Place & Route technology file. If you do not specify an equivalence file or a schematic netlist file on the command line, A2lvs creates those files from the Logical Design Data stored in the Milkyway database. The schematic netlist file is written in Hercules format. Only the top-level comparison is done.

When executed, A2lvs creates a directory named LVS in the current directory, as well as a runset named *cellname*.ev, where *cellname* represents the name of the top cell in the design. A2lvs then runs Hercules on the new runset. The results from the Hercules run are stored in the LVS directory.

For information on LVS Use Models, see the *Hercules LVS User Guide*, Executing LVS Extract and Compare chapter.

The syntax is:

Note:
The command-line options -milkyway, -verilog, and -hercules are mutually exclusive. In addition, the -schblock option must be used in conjunction with either -verilog or -hercules.

```
A2lvs -b cellname -l library_name [-equiv_list filename]
   [-g group_dir] [-help] [-hercules hercules_netlist]
   [-graf] [-milkyway milkyway_netlist]
   [-o output_layout_path] [-p layout_path] [-pause]
   [-r filename] [-schblock schematic_block_name] [-V]
   [-verilog verilog_netlist]
```

*Table 4-12 Required Command-Line Arguments*

| Argument | Description |
|---|---|
| `-b cellname` | Name of top cell to check (required argument). |
| `-l library_name` | Name of Milkyway library on which to perform LVS (required argument). |

*Table 4-13 Optional Command-Line Arguments*

| Argument | Description |
|---|---|
| `-equiv_list filename` | File containing list of cells to compare in addition to top block. The default is no list. |
| `-g group_dir` | Defines group_dir in LVS/runsetfile. Default is the group directory under LVS. The default is group. |
| `-graf` | Writes Graphics section in LVS/*runsetfile*. The default is false. |
| `-help` | Displays A2lvs usage. The default is false. |

*Table 4-13 Optional Command-Line Arguments(Continued)*

| Argument | Description |
|---|---|
| `-hercules` *`hercules_netlist`* | Uses Hercules-format schematic netlist. |
| `-milkyway` *`milkyway_netlist`* | Uses Milkyway-format schematic netlist. |
| `-o` *`output_layout_path`* | Defines output_layout_path in LVS/*runsetfile*. Default is the layout directory under LVS. |
| `-p` *`layout_path`* | Path to Milkyway input library. Default is the current directory where A2lvs was invoked. |
| `-pause` | Creates the runset then stops. Does not start Hercules. The default is false. |
| `-r` *`filename`* | Includes a user runset file in the LVS/*runsetfile* |
| `-schblock` *`schblock_block_name`* | Schematic block name, if different from -b *block_name*. |
| `-V` | Displays version number. The default is false. |
| `-verilog` *`verilog_netlist`* | Uses verilog format schematic netlist. |

For example:

```
A2lvs -l DESIGNS -b TOP
```

From this syntax, a runset named TOP.ev is created within the directory LVS. Two other files are also created: TOP_equiv, which is the input of the equivalence field in the runset, and TOP_sch, which is the input of the schematic field in the runset. For more information on Hercules COMPARE, see the *Hercules Reference Manual*, Detailed Options chapter.

## Netlists

This section describes the Hercules netlist format used by Hercules during COMPARE, as well as the two methods of translating other formats into a Hercules netlist format. The first method is to call the NetTran utility within the Hercules runset and is appropriate if your flow always requires the same NetTran options from run to run. However, if the flow requires

changes in the NetTran options, it is recommended to use the second method—to have a script call the NetTran directory with the appropriate command-line options before Hercules is invoked. You will find information on these tasks in the following sections:

## Hercules Netlist Format

The Hercules netlist format is an explicit format; that is, it is not order-dependent and it provides complete net information with each cell. A cell entry contains instances, ports, and nets. (You can derive net information about a cell from the pins and instances information.) Ports are always made up of the nets in the cell, with the one exception of feedthroughs. For more information on the Hercules netlist format, see the *Hercules LVS User Guide*, Netlisting chapter.

The cell instance and pin information is designated by the keywords inst (instances) and pin (pins). When looking at the instance information, the information to the left of the equal sign (=) is the name of the instance. (The instance name must be unique.) The information to the right of the equal sign (=) is the name of the cell that is being instantiated.

When looking at the pin information, the information to the left of the equal sign is the name of the net in the current cell. The information to the right of the equal sign is the name of the port in the lower-level cell to which the current cell connects. Remember that this port does not exist in the current cell; it is the name of the port that exists in a lower-level cell.

For example:

```
cell A
   inst 1=B
      pin 1=X Y=2
```

In the preceding example of a Hercules-format netlist, Cell A contains one instance and two pins. The line `inst 1=B` identifies 1 as the name of the instance and B as the name of the cell that is being instantiated. The line `pin 1=X Y=2` identifies 1 and Y as the name of the net that is in Cell A. This line also identifies X and 2 as the name of the port from the lower-level cell to which Cell A connects.

Example 4-2 shows different ways to set property values.

*Example 4-2 Netlist Example for sample.ev*

```
{synopsys sample.ev
/* Shows different ways to set property values. */

{version 1 0 0}
                          /* Only supported version is 1.0.0 */

{param_global GlobalL=1.0 GlobalW=2.0}
                          /* Parameter globals */
```

```
{prop_default l=3.0 w=4.0}
                        /* Property defaults */

{cell p
   {port gate source drain bulk}
   {param_init pmosl=13.0 pmosw=14.0}
                        /* Parameter defaults */
   {inst p0=pmos
      {prop l=pmosl w=pmosw}
      {pin gate=gate source=source drain=drain bulk=bulk}
   }
}

{cell inv
   {port vdd in vss out}
   {param_init nl=5.0 nw=6.0 pl=7.0 pw=8.0}
   {inst n1=nmos
      {prop l=nl w=nw}
      {pin in=gate out=source vss=drain vss=bulk}
   }
   {inst p1=p
      {param pmosl=pl pmosw=pw}
      {pin in=gate out=source vdd=drain vdd=bulk}
   }
}

{cell tgate
   {port vdd sel selb vss in out}
   {param_init nl=9.0 pl=10.0}
   {inst n2=nmos
      {prop l=nl w=nw}
      {pin sel=gate in=source out=drain vss=bulk}
   }
   inst p2=pmos
      {prop l=pl w=pw}
      {pin selb=gate out=source in=drain vdd=bulk}
   }
}

{cell top
   {port in out sel selb vdd vss}
   {inst 1=inv
      {param pl=GlobalL pw=11.0}
                        /* Parameter passing */
      {pin in=in net1=out vdd=vdd vss=vss}
   }
   {inst 2=tgate
   {param nw=12.0}
   {pin net1=in out=out sel=sel selb=selb vdd=vdd vss=vss}
   }
   }
}
```

In Example 4-2 there are a total of four transistors in the sample.ev design: two NMOS and two PMOS devices. A flattened version of this netlist, with the resulting property values, is shown here, Example 4-3.

*Example 4-3 Flattened Netlist Example for sample.ev*

```
{sample.ev.flat        /* Flattened equivalent of sample.ev */

   {version 1 0 0}

{cell top
   {port in out sel selb vdd vss}
   {inst 1/n1=nmos
      {prop l=5.0 w=6.0}
      {pin in=gate net1=source vss=drain vss=bulk}
   }
   {inst 1/p1/p0=pmos
      {prop l=1.0 w=11.0}
      {pin in=gate net1=source vdd=drain vdd=bulk}
   }
   {inst 2/n2=nmos
      {prop l=9.0 w=12.0}
      pin sel=gate net1=source out=drain vss=bulk}
   }
   inst 2/p2=pmos
      {prop l=10.0 w=4.0}
      {pin selb=gate out=source net1=drain vdd=bulk}
      }
   }
}
```

## NetTran

NetTran is a netlist translation utility. The following can be accomplished with NetTran:

- Translate a standard format netlist to a Hercules-format netlist
- Translate a Hercules-format netlist to a standard format netlist
- Create a skeletal equivalence file (during any translation)
- Create a wire resolution log file (during any translation)

## Using Hercules to Translate into the Hercules Netlist Format

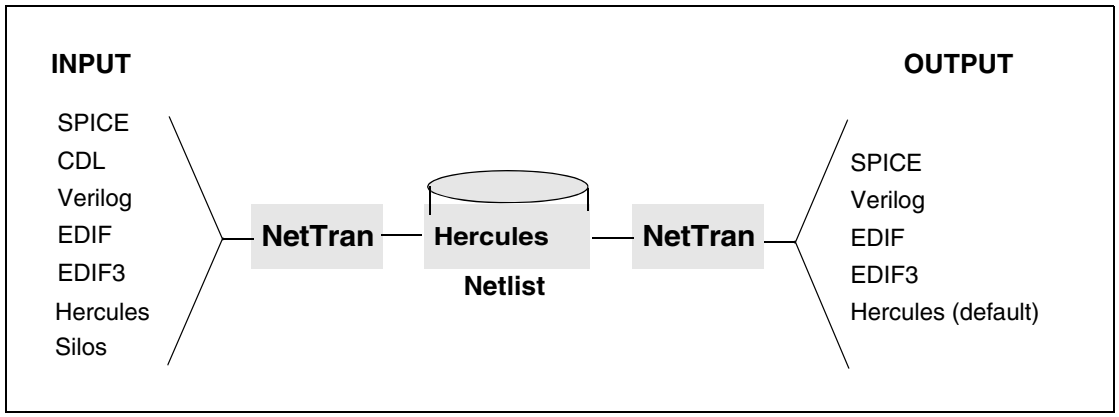
You can invoke the NetTran utility from within Hercules. To do this, use the NETTRAN_OPTIONS within the OPTIONS section as described in the following section. Whenever Hercules COMPARE is invoked on a schematic netlist that does not have a Hercules netlist format, Hercules will call the NetTran utility. It is possible to specify options in the NETTRAN_OPTIONS, as shown in the following example:

```
OPTIONS {
 NETTRAN_OPTIONS="-verilog AFF8.v -cdl EXL.cdl -outName AFF8.herc
}
```

## Using NetTran to Translate into the Hercules Netlist Format

Figure 4-3 lists types of netlists that can be converted using NetTran.

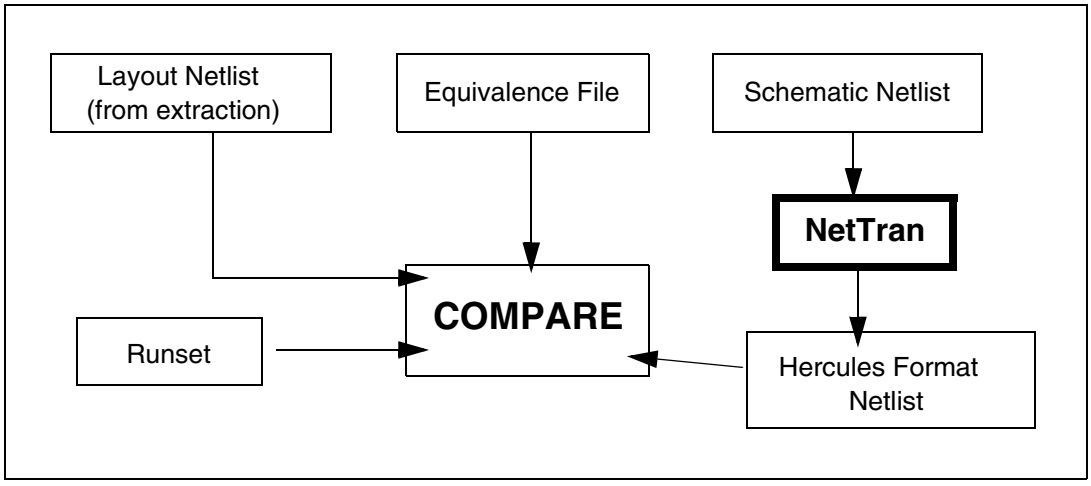
*Figure 4-3 Converting Types of Netlists*




### Netlist Translation

Schematic netlists are compared to the extracted layout netlist during LVS. The schematic netlist must be translated to the Hercules format before Hercules can use it for LVS comparison as shown in Figure 4-4. (NetTran selects net names from the netlist assign statements in ASCII order.) The Hercules-format netlist contains a hierarchical description of schematic devices and their interconnections.

*Figure 4-4 LVS Process Flow*



### Skeletal Equivalence File

A skeletal equivalence file contains an entry for each schematic module found in the netlist during translation. Each associated layout structure entry is named the same as the schematic module name. You must edit the layout name entries to set the proper equivalence point for any cases where the schematic name and layout name are not identical.

### Wire Resolution Log File

The wire resolution log file is a record of all the shorted nets for each cell. If two nets are shorted, the nets are replaced by one name. The file can be used to cross-reference new net names with the original net names. NetTran automatically resolves the wire constructs in Hercules netlists, but if you wish to have a log of all the wire constructs, you must use the wireLog command-line syntax.

## NetTran Command-Line Syntax

The NetTran command-line syntax is presented here.

```
nettran [-V] [-Version] [-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-x] [-sp-voltThresh number]
     [-verilog-b0 netName] [-verilog-b1 netName]
     [-verilog-addDummyDev devName][-verilog-R][-verilog-U]
     [-verilog-busLSB][-verilog-voltmap filename][-acct] [-acctFile file]
     [-dup] [-equiv file] [-forceGlobalsOn]
     [-herc-globalNets netName ...] [-logFile file]
     [-mprop] [-msgError NTR-N ...] [-msgId]
     [-msgSuppress NTR-N ...] [-msgTable]
     [-namePrefix prefix] [-noflatten] [-rootCell cell]
     [-slash] [-undef] [-wireLog file]
     -outName file
```

*Table 4-14 General NetTran Command-Line Syntax*

| Argument | Description |
|---|---|
| `-V` | Print product version. |
| `-Version` | Print product and library version. |

*Table 4-14 General NetTran Command-Line Syntax(Continued)*

| Argument | Description |
|---|---|
| `-usage` | Print NetTran usage. |
| `-cdl` *file* ... | Input CDL format netlists. |
| `-edif` *file* ... | Input EDIF 2 0 0 format netlists. |
| `-edif3` *file* ... | Input EDIF 3 0 0 format netlists. |
| `-hercules` *file* ... | Input Hercules-format netlists. |
| `-sp` *file* ... | Input Spice format netlists. |
| `-verilog` *file* ... | Input Verilog format netlists. |

*Table 4-15 SPICE/CDL Translation Command-Line Syntax*

| Argument | Description |
|---|---|
| `-cdl-a` | Preserve all passive devices (Res, Cap, Diode). |
| `-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 inductor, resistor, and capacitor devices with specific model name. Nets connecting the A and B pins of each device instance will be shorted in the translated netlist. When using this option, net naming rules for resolved nets are in order of highest to lowest priority:<br>1. Global nets are selected.<br>2. Port nets are selected.<br>3. Nets are selected based on alphabetical order. |
| `-cdl-M` *file* | CDL map file. |
| `-cdl-p` | Translate instance param to prop if the model has no instances. |

*Table 4-15 SPICE/CDL Translation Command-Line Syntax(Continued)*

| Argument | Description |
|---|---|
| `-cdl-P` | Multiply scale_factor by param_global.<br>For the following syntax in CDL:<br>• .PARAM PRD = 6.6u<br>• *.SCALE meter<br>If -cdl-P is used, NetTran will translate the PRD value to 6.6 x 10e-6 x 10e6 = 6.6<br>If -cdl-P is not used, NetTran will translate the PRD value to 6.6 x 10e-6 = 0.0000066 |
| `-cdl-R` | Include unused parameters in the Hercules netlist. |
| `-cdl-renameDev` | Prefix device names with type to resolve identical name conflict. |
| `-cdl-s` | Do not treat slash (/) as delimiter. |
| `-cdl-U` | Convert cdl input to uppercase. |
| `-cdl-x` | Trim prefix x of instance name. |
| `-sp-fopen` *`model_name...`* | Filter out devices with the specific model-name. The nets that used to be connected to the device terminal are left unconnected (open). This option applies to all devices. |
| `-sp-fshort` *`model_name...`* | Filter out inductor, resistor, and capacitor devices with the specific model-name. Nets connecting the A and B pins of each device instance will be shorted in the translated netlist. When using this option, net naming rules for resolved nets are in order of highest to lowest priority:<br>1. Global nets are selected.<br>2. Port nets are selected.<br>3. Nets are selected based on alphabetical order. |
| `-sp-finst` *`inst_name`* | Filter out devices with specific inst_name (accepts wildcard *). |
| `-sp-fp` *`delimiter`* | Filter out parasitic resistors. |
| `-sp-m` *`number`* | Device property multiplication factor. |
| `-sp-od` *`pattern`* | Defines the order of the output ports from verilog to spice. |
| `-sp-S` | Sets .options scale = 1 μ. |

*Table 4-15 SPICE/CDL Translation Command-Line Syntax(Continued)*

| Argument | Description |
|---|---|
| `-sp-topCell` *`cell`* | Set name of top cell. |
| `-sp-topCellPorts` *`portName ...`* | Add ports to SPICE/CDL topcell port list. Must use -sp-topCell option to specify the top cell. |
| `-sp-U` | Convert spice input to uppercase. |
| `-sp-voltThresh` *`number`* | Voltage source threshold value for shorts. |
| `-sp-x` | Trim prefix x of instance name. |

*Table 4-16 EDIF Translation Command-Line Syntax*

| Argument | Description |
|---|---|
| `-edif-f` | Force EDIF primitives in Hercules netlist. |
| `-edif-globalNets` *`netName ...`* | Specifies particular net names in the EDIF netlist as global nets. These net names take precedence when an EDIF net has multiple names. |
| `-edif-p` | Convert EDIF properties. |
| `-edif-pa` | Convert all EDIF properties associated with interface and instance types. Properties associated with other types, such as cell and port, are ignored. |
| `-edif-U` | Convert edif input to uppercase. |

*Table 4-17 SILOS Translation Command-Line Syntax*

| Argument | Description |
|---|---|
| `-silos` *`file ...`* | Input SILOS format netlists. |

*Table 4-18 Verilog Translation Command-Line Syntax*

| Argument | Description |
|---|---|
| `-verilog-addDummyDev` *devName* | Add dummy resistors to ports of root-cell in the Verilog netlist. |
| `-verilog-b0` *netName* | Verilog global ground net. |
| `-verilog-b1` *netName* | Verilog global power net. |
| `-verilog-busLSB` | Verilog bus starts with least significant bit (0) first. |
| `-verilog-voltmap` *filename* | Verilog global net mapping file. |
| `-verilog-U` | Convert Verilog input to uppercase. |
| `-verilog-R` | Retain backslash (\) on verilog net names. |

*Table 4-19 Hercules Translation Command-Line Syntax*

| Argument | Description |
|---|---|
| `-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. |
| List of Hercules global nets. | NetTran summary log filename (default is nettran.log). |
| `-mprop` | If device property m is greater than 1, this option netlists the device m number of times, each with m equal to 1. |
| `-msgError` *NTR-N* `...` | List of messages to upgrade to error. |
| -msgId | Include message ID in message. |
| `-msgSuppress` *NTR-N* `...` | List of messages to suppress reporting. |
| `-msgTable` | Print message table and exit. |

*Table 4-19 Hercules Translation Command-Line Syntax(Continued)*

| Argument | Description |
|---|---|
| `-namePrefix` *prefix* | Prefix for names used in output netlist (default is NT_). As NetTran translates a netlist, it detects characters that are illegal in the output netlist. NetTran substitutes valid names prefixed with *prefix* in the resulting output netlist. |
| `-noflatten` | Prevent flattening of the hierarchy by NetTran in order to resolve parameters. By default, NetTran flattens the cells whose parameters refer to other cells so that no parameters are included in the output netlist. The ability of NetTran to handle equations with parameters is lost if you use this option. |
| `-rootCell` *cell* | Set root cell for output netlist. Only cells contained in the hierarchy of the root cell are output. |
| `-slash` | Use slash (/) as the hierarchical delimiter instead of backslash (\) |
| `-hier_delimiter` *pattern* | User defined hierarchical delimiter to separate cell instances in an instance path when the -mprop option is used. |
| `-undef` | Write undefined cell names to the file undef.log |
| `-wireLog` *file* | Dissolved Nets log filename |
| `-outName` *file* | Output netlist name. The default name is *block*.sch_out. |

## Translating Standard Format Netlists to Hercules Format

The following types of netlists can be translated to the Hercules format:

EDIF 2 0 0
EDIF 3 0 0
Verilog
Silos
SPICE/CDL

### Netlist formats

CDL and SPICE netlists are different from Hercules netlists in that they have implicit referencing. The order of the ports is fixed. In contrast, in a Hercules-format netlist, referencing is explicit because the port names are given. Therefore, the ports can exist in any order.

Table 4-20 shows the default conversion units for various schematic netlist formats. The main difference between CDL and SPICE is that the default units for parameters is microns in CDL and meters in SPICE. These units can be manipulated using various command-line options with nettran. By default, NetTran does not translate passive devices (R, C, D) for CDL, but it does for SPICE. To translate the passive devices in a CDL netlist, use the NetTran option -cdl-a.

*Table 4-20 Default Conversion Units*

| Netlist Type | Default Unit |
|---|---|
| Hercules | micron |
| CDL | micron |
| SPICE | meter |
| EDIF | meter |

A Verilog netlist uses modules to define cells. It requires that every port and every net be previously defined (as input, output, inout, or wire) before being used. Hercules does not make use of this definition. The syntax for a Verilog netlist might be .BULK(VDD), whereas a Hercules netlist might be VDD=BULK. Notice the different order in the cases above. NetTran supports structural, not behavioral, Verilog.

An EDIF netlist is unlike all other netlists in that it is net-based and not instance-based. For an EDIF netlist, the instances are followed in the list by the nets and what they are connected to.

### Property Values

Although standard netlist formats allow properties in several formats, NetTran accepts only specific formats, as shown in Table 4-21.

*Table 4-21 Property Value*

| Character | Name | Value |
|---|---|---|
| f | femto | 1e-15 |
| p | pico | 1e-12 |
| n | nano | 1e-9 |
| u | micro | 1e-6 |

*Table 4-21 Property Value(Continued)*

| Character | Name | Value |
|---|---|---|
| m | milli | 1e-3 |
| K | kilo | 1e+3 |
| M | mega | 1e+6 |
| G | giga | 1e+9 |
| T | tera | 1e+12 |

### EDIF 2 0 0 Translation

This section describes translating an EDIF 2 0 0-format netlist to a Hercules-format netlist.The syntax is:

```
nettran -edif infile [infile2 ... infilen] [-edif-f] [-edif-globalNets
netName] [-edif-p] [-edif-U] -outName netlist
```

| Argument | Description |
|---|---|
| `-edif` *infile* [*infile2 ... infilen*] | Inputs EDIF 2 0 0-format netlists. |
| `-edif-f` | Forces EDIF primitives in Hercules netlist. |
| `-edif-globalNets` *netName1 ... netNameN* | Specifies particular net names in the EDIF netlist as global nets. These net names take precedence when an EDIF net has multiple names. |
| `-edif-p` | Converts EDIF properties. |
| `-edif-U` | Uppercase translation. All input text is converted to uppercase and the resulting output files are all in uppercase. This allows for an error-free translation of edif files that have mixed cases of the same text. Therefore, Inv is the same as inv. |
| `-outName` *netlist* | The output file. |

EDIF 2 0 0 arrays are supported by NetTran. Net names are not generated for unconnected pins.

Although EDIF allows properties in several formats, NetTran accepts only a specific format. An example of the EDIF property value syntax for NetTran is shown here.

Note:
The -edif-p option must be specified to report properties in the Hercules netlist.

```
(instance PFET4
(viewRef PFET_SCHEMATIC
(cellRef PFET))
(property W
(string "3.30u"))
(property L
(string "0.75u")))
```

The property names accepted are

A, AREA
C, CVAL
L
R, RVAL
W

with units as shown in Table 4-21 on page 4-45.

For example:

```
nettran -edif filename.edif -edif-p -edif-f -outName filename.nt
```

Note:
The -edif-p option includes properties in Hercules netlist. The -edif-f option includes primitive cells in Hercules netlist.

### EDIF 3 0 0 Translation

This section describes translating an EDIF 3 0 0-format netlist to a Hercules-format netlist. The syntax is:

```
nettran -edif3 infile [infile2 ... infilen] [-edif-p] [-edif-globalNets
netName] [-edif-f] [-edif-U] -outName netlist
```

| Argument | Description |
|---|---|
| `-edif infile [infile2 ... infilen]` | Inputs EDIF 2 0 0-format netlists. |
| `-edif-p` | Converts properties. |

| Argument | Description |
|---|---|
| `-edif-globalNets` *`netName1 ... netNameN`* | Specifies particular net names in the EDIF netlist as global nets. These net names take precedence when an EDIF net has multiple names. |
| `-edif-f` | Forces primitives in output netlist. |
| `-edif-U` | Uppercase translation. All input text is converted to uppercase and the resulting output files are all in uppercase. This allows for an error-free translation of edif files that have mixed cases of the same text. Therefore, Inv is the same as inv. |
| `-outName` *`netlist`* | The output file. |

EDIF 3 0 0 arrays are not supported by NetTran. Net names are not generated for unconnected pins.

Although EDIF allows properties in several formats, NetTran accepts only a specific format. The following example is acceptable EDIF property value syntax for NetTran.

Note:
You must specify the -edif-p option to report properties in the Hercules netlist. Properties are handled identically in EDIF 200 and EDIF 300.

Here is an example of EDIF 3 0 0 translation syntax:

```
nettran -edif3 filename.edif3 -edif-p -outName filename.nt
```

### Verilog Translation

This section describes translating a Verilog-format netlist to a Hercules-format netlist.

The NetTran -verilog option supports only structural Verilog and will not parse behavioral Verilog statements. It supports arrays and Logical Tied_High and Tied_Low. The syntax is:

```
nettran -verilog infile [infile2 ... infilen] [-verilog-b0 netName][-
verilog-b1 netName] [-verilog-U] [-verilog-R] -outName netlist
```

An example of the Verilog translation syntax is:

```
nettran -verilog infile infile2 -outName netlist
```

NetTran will short two nets if there is a Verilog assign statement. For example,

```
assign old_name = new_name ;
```

In the preceding example, NetTran shorts two nets, *old_name* and *new_name*. After shorting the two nets, *old_name* will be substituted with *new_name*. If the shorted net is connected to a port, it should retain the port name as the retained net, and in other cases the net name.

### Silos Translation

This section describes translating a Silos-format netlist to a Hercules-format netlist. The syntax is:

```
nettran -silos infile [infile2 ... infilen] -outName netlist
```

| Argument | Description |
|---|---|
| `-silos infile [infile2 ... infilen]` | Input Silos-format netlists. |
| `-outName netlist` | The output file. |

Note:

The Silos translator supports only standard macro expansion. It does not support, nor will it translate, predefined primitive elements (for example, .NAND and. NOR).

An example of the Silos translation syntax is:

```
nettran -silos filename.silos -outName filename.nt
```

### SPICE/CDL Translation

This section describes translating a SPICE- or CDL-format netlist to a Hercules-format netlist. The syntax is:

```
nettran -sp infile [infile2 ... infilen]
   -cdl infile [infile2 ... infilen] [-cdl-a] [-cdl-chop]
   [-cdl-chopDev] [-cdl-fopen model_name...]
   [-cdl-fshort model_name...] [-cdl-M map_file]
   [-cdl-p] [-cdl-P] [-cdl-R] [-cdl-renameDev]
   [-cdl-s] [-cdl-U] [-cdl-x]
   [-sp-fopen model_name...] [-sp-fshort model_name...]
   [-sp-finst inst_name ...] [-sp-fp delimiter]
   [-sp-m multiply_factor] [-sp-od] [-sp-S]
   [-sp-topCell cell] [-sp-topCellPorts portName ...]
   [-sp-U] [-sp-voltThresh number] [-sp-x]
   -outName netlist
```

| Argument | Description |
|---|---|
| `-sp` *`infile`* [*`infile2 ... infilen`*] | Inputs the SPICE/CDL-input-format netlists. |
| `-cdl-a` | The preservation of all passive devices (CDL mode). |
| `-cdl-chop` | The removal of the first character (X, R, C, D, M, Q, J) of all instance names of primitive subcircuits. |
| `-cdl-chopDev` | The removal of the first character (R, C, D, M, Q, J) of all device instance names. |
| `-cdl-fopen` *`model_name...`* | Filter out devices with specific model-name. |
| `-cdl-fshort` *`model_name...`* | Filter out inductor, resistor, and capacitor devices with specific model name. Nets connecting the A and B pins of each device instance will be shorted in the translated netlist. When using this option, net naming rules for resolved nets are in order of highest to lowest priority:<br>1. Global nets are selected.<br>2. Port nets are selected.<br>3. Nets are selected based on alphabetical order. |
| `-cdl-M` *`map_file`* | The input Cadence Virtuoso® Layout Editor map file. |
| `-cdl-p` | Translate instance param to prop if the model has no instances. |
| `-cdl-P` | Multiplies the PARAM_GLOBAL value with the scale factor.<br>For the following syntax in CDL:<br>• `PARAM PRD = 6.6u`<br>• `*.SCALE meter`<br>If -cdl-P is used, NetTran will translate the PRD value to 6.6 x 10e-6 x 10e6 = 6.6.<br>If -cdl-P is not used, NetTran will translate the PRD value to 6.6 x 10e-6 = 0.0000066. |
| `-cdl-R` | The inclusion of unused arguments in the netlist. |
| `-cdl-renameDev` | Prefix device names with type to resolve identical name conflict. |

| Argument | Description |
|---|---|
| `-cdl-s` | Do not treat slash (/) as delimiter. |
| `-cdl-U` | Convert to uppercase. |
| `-cdl-`*x* | Trim prefix *x* of instance name. |
| `-sp-fopen` *model_name...* | Filter out devices with the specific model-name. The nets that used to be connected to the device terminal are left unconnected (open). This option applies to all devices. |
| `-sp-fshort` *model_name...* | Filter out inductor, resistor, and capacitor devices with the specific model-name. Nets connecting the A and B pins of each device instance will be shorted in the translated netlist. When using this option, net naming rules for resolved nets are in order of highest to lowest priority:<br>1. Global nets are selected.<br>2. Port nets are selected.<br>3. Nets are selected based on alphabetical order. |
| `-sp-finst` *inst_name* | Filter out devices with specific inst_name (accepts wildcard *). |
| `-sp-fp` *delimiter* | Filter out parasitic resistors.*delimiter* should be a character. |
| `-sp-m` *multiply_factor* | Device property multiplication factor. |
| `-sp-od` *pattern* | Defines the order of the output ports. |
| `-sp-S` | Sets the .options scale = 1 μ. |
| `-sp-topCell` *cell* | Sets name of top cell. |
| `-sp-topCellPorts` *portName1... portNameN* | List of SPICE/CDL top-cell ports. |
| `-sp-U \| -cdl-U` | Uppercase translation. All input text is converted to uppercase and the resulting output files are all in uppercase. This allows for an error-free translation of spice files that have mixed cases of the same text. Therefore, Inv is the same as inv. |

| Argument | Description |
|---|---|
| `-sp-voltThresh` *number* | The voltage source threshold value for shorts. In the default mode, NetTran replaces all voltage sources in a CDL/SPICE design with an open. Use this option to override the open with a short by specifying a threshold value on the command line. All voltage sources with a DC voltage of less than or equal to the threshold value will be replaced by a short. |
| `-sp-`*x* | Trim prefix *x* of instance name. |
| `-outName` *netlist* | The output file. |

SPICE/CDL example of the syntax: Expressions or equations for argument values in the input SPICE file are not supported. NetTran generates a warning message if an equation is parsed.

```
MOS0 B BS B0 GND NFET L= '2*(sp+10u*shrink+2*actbias)' $unsupported
.PARAM sdp = '2*(sp+10u*shrink+2*actbias)'             $unsupported
```

SPICE/CDL example of the syntax: The command in converts a SPICE netlist to a Hercules netlist and resolves all wires or shorts. The wireLog file contains all the nets that were dissolved in the process.

```
nettran -sp filename.sp -outName filename.nt -wireLog filename.log
```

SPICE/CDL example of the syntax: The -sp-U option forces all identifier names to uppercase, making the netlist case-insensitive. The -equiv option generates an equivalence file for Hercules.

```
nettran -sp filename.sp -sp-U -equiv equivfile -outName filename.nt
```

SPICE/CDL example of the syntax to filter out all devices with the model name EFG:

```
nettran -sp-fopen "EFG"
```

SPICE/CDL example of the syntax `-sp-fp` *delimiter*: select a delimiter character, such as #. The file test.sp:

```
SUBCKT test
M1 Z#1 A B C pmos l=0.1 w=0.1
M2 D E Z#3 F nmos l=0.1 w=0.1
R1 Z#1 Z#2   res1 l=0.1 w=0.1
R2 Z#2 Z     res2 l=0.1 w=0.1
R3 Z Z#3     res3 l=0.1 w=0.1
R4 G H       res4 l=0.1 w=0.1
ENDS
```

If the NetTran command is:

```
nettran -sp-fp # -sp test.sp -outName out1
```

the test.sp file will be translated to a Hercules format netlist where the parasitic resistors R1, R2, and R3 is filtered out (shorted).

SPICE/CDL example of the syntax:

```
nettran -verilog file ... -sp-od pattern -outName outfile
```

The pattern should be iob, ibo, bio, and so on. Therefore, it consists of one of the following three characters: b (inout); i (input); o (output).The order of the three characters is the same as one of the output ports and pins.

The order of output ports is [input] [output] [inout] in the command. For example:

```
nettran -verilog test.v -sp-od iob -outName out.spi
```

SPICE/CDL example of showing the MOS difference: In CDL format, $BULK terminal is translated in the Hercules netlist. For example

```
SUBCKT test
   M0 d g s b b2 b3 nch $BULK4=b4
ENDS
```

If the NetTran command is:

```
nettran -sp input -outName output
```

the output is:

```
{inst M0=nch {pin d=DRN g=GATE s=SRC b=BULK b2=BULK2 b3=BULK3}}
```

If the NetTran command is:

```
nettran -cdl input -outName output
```

the output is:

```
{inst M0=nch {pin d=DRN g=GATE s=SRC b=BULK
              b2=BULK2 b3=BULK3 b4=BULK4}}
```

### CDL Map File

A map file is created by the Virtuoso Layout Editor CDL out process. The file contains name translations of Virtuoso Layout Editor database names to legal CDL names. You can map Net, instance, and subckt names using this method. The resulting netlist from NetTran contains the Virtuoso Layout Editor database names (the names on the left in Example 4-4).

*Example 4-4 Example Map File*

```
(( nil
```

```
version "2.1"
mapType "hierarchical"
repList "cdl schematic cmos.sch gate.sch symbol"
stopList "cdl"
globalList "vbb! vpp! vbleq! vblh! vint! gnd!"
   hierDelim "."
 )
( net
( "vbleq!" "vbleq" )
( "vpp!" "vpp" )
( "vbb!" "vbb" )
 )
( inst
( "IA<4>" "XIA4" )
( "IB<37>" "XIB37" )
( "IA<365>" "XIA365" )
 )
( model
( "nfetlo" "M1" )
)
)
```

SPICE/CDL example of the syntax: The -cdl-M option loads a Cadence-name mapping file. This file is sometimes available in a Cadence environment, and allows GUI debugging tool to use the original names prior to Cadence substitution.

```
nettran -cdl filename.cdl -cdl-M mapfile -outName filename.nt
```

## Creating a Skeletal Equivalence File

NetTran can create a skeletal equivalence file during any netlist translation. An equivalence file is used during LVS COMPARE to list each schematic structure and the corresponding layout structure. The equivalence file NetTran creates is skeletal because you must edit the layout name entries to set the proper equivalence point for any instances where the schematic name and layout name are not identical. The syntax is:

```
nettran -hercules infile -equiv file -outName netlist
```

| Argument | Description |
|---|---|
| `-hercules` *infile* | The input format netlist. |
| `-equiv` *file* | Instructs NetTran to create a skeletal equivalence file (where equivalence points are printed). |
| `-outName` *netlist* | The output format netlist. |

## Creating a Wire Resolution Log File

NetTran automatically resolves the { wire } constructs in Hercules netlists. The following syntax describes how to create a log of all wire constructs Hercules resolves during any netlist translation.

A { wire } construct can be created during translation of SPICE, CDL, and EDIF netlists. NetTran hierarchically resolves the netlist connectivity affected by the nets and ports involved in the wire constructs. For example, during SPICE translation, { wire } constructs are used to replace resistors that are specified for removal with the *.RESI command. You must remove the { wire } constructs prior to starting LVS because Hercules COMPARE does not accept these constructs.

The syntax is:

```
nettran -hercules infile -wireLog wire_log_file -outName netlist
```

| Argument | Description |
|---|---|
| `-hercules infile` | The input format netlist. |
| `-wireLog wire_log_file` | Command-line syntax that instructs NetTran to create a wire log file (where dissolved nets are printed). |
| `-outName netlist` | The output format netlist. |

## Using Two Steps to Complete a Netlist Translation

There is one situation where the netlist translation needs to be done in two steps. The scenario is as follows:

- Base-level cells are defined using EDIF.
- Cell ports are being renamed in the base-level cells.
- Higher-level cells are defined using a netlist that makes explicit reference to the original port names of base-level cells (Verilog/explicit, Hercules).

When such a scenario occurs, use code as shown in the following example to translate the netlist:

```
nettran -edif baseCells.edif -outName baseCells.sch
nettran -hercules baseCells.sch -verilog top.vlg -outName chip.sch
```

*not* this:

```
nettran -edif baseCells.edif -verilog top.vlg -outName chip.sch
```

NetTran generates errors for the single-invocation case because there is a discrepancy between the definition of the port name in the base-level netlist and the reference to that same port in the higher-level netlist. The problem in this case is that the original name for the ports of a base-level cell are retained only for the purpose of generating the Hercules netlist. Internally, all port references are expected (as mandated by the EDIF specification) to be in terms of the renamed label.

## NetTran Error/Warning Messages

You can control the printing of certain messages during netlist translation. In certain cases, you might want to reduce the size of the output log file by switching off particular messages. Normally, these messages are warnings you understand, or data that is not generally referenced. Use the -msgSuppress option to disable the printing of messages. You cannot suppress messages with a severity level of FATAL or ERROR. The -msgId option enables the listing of the message ID along with the message in the log file. You cannot use the -msgError option to upgrade messages to ERROR severity. A warning message allows NetTran to complete and exit with a successful status (0). An error message allows NetTran to complete and exit with an unsuccessful status (non 0).

To print the message table and exit, use this code:

```
nettran -msgTable
```

To print message identifiers in the log file:

```
nettran -msgId -sp foo.sp -outName foo.net
```

To avoid printing specific messages:

```
nettran -msgSuppress NTR-14 NTR-28 -sp foo.sp -outName foo.net
```

To upgrade message level to error:

```
nettran -msgError NTR-20 -sp foo.sp -outName foo.net
```

Table 4-22 lists all the controllable messages, in order of message identifier, along with the text of the message. All messages are ON by default.

*Table 4-22 NetTran Error Messages*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-1 | ON | MSG | NetTran banner. |
| NTR-2 | ON | FATAL | License Error: %s. |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-3 | ON | ERROR | ERRORS during translation (See log file for listing). |
| NTR-4 | ON | FATAL | Inserting NULL string in NetTran String. |
| NTR-5 | ON | FATAL | Cannot create %s hash table in %s(). |
| NTR-6 | ON | FATAL | Too Many Errors !!! |
| NTR-7 | ON | FATAL | At least one input netlist must be specified. |
| NTR-8 | ON | MSG | Parsing view-t translation table... |
| NTR-10 | ON | FATAL | Netlist pointer is NULL. |
| NTR-11 | ON | MSG | Reading "%s" netlist ... |
| NTR-12 | ON | MSG | Processing Hierarchy ... |
| NTR-13 | ON | WARNING | Duplicate Cell '%s' found. |
| NTR-14 | ON | FATAL | Cannot open output file "%s" for writing. |
| NTR-15 | ON | MSG | Writing %s netlist ... |
| NTR-16 | ON | FATAL | Cannot open file "%s" for reading. |
| NTR-17 | ON | FATAL | Unsupported netlist type "%s". |
| NTR-18 | ON | FATAL | -equiv %s and -outName %s Cannot have same name. |
| NTR-19 | ON | WARNING | "%s" output not supported, generating Hercules netlist. |
| NTR-20 | ON | FATAL | No read access to netlist "%s". |
| NTR-21 | ON | WARNING | Overwriting primitive cell '%s' previously declared. |
| NTR-22 | ON | FATAL | Cannot create summary Log file "%s" for writing. |
| NTR-23 | ON | WARNING | %s. |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-24 | ON | FATAL | %s at or near line %d in "%s" near token : "%s". |
| NTR-25 | ON | FATAL | PinCount for instance '%s' in cell '%s' doesn't match portCount defined in cell '%s'. |
| NTR-26 | ON | MSG | Adding primitives to Hercules Netlist. |
| NTR-27 | ON | INFO | Loading cell "%s" ... |
| NTR-28 | ON | FATAL | Cannot open file "%s" for writing. |
| NTR-29 | ON | ERROR | %s "%s" cannot open. |
| NTR-30 | ON | WARNING | Get net failed, netId = %x from cell instance "%s" (port instance "%s"). |
| NTR-31 | ON | WARNING | Duplicate Port '%s' at or near line %d, creating wire. |
| NTR-32 | ON | INFO | Creating Top Block "%s". |
| NTR-33 | ON | FATAL | Syntax error near line %d: Bad definition. |
| NTR-34 | ON | FATAL | Syntax error near line %d: Bad Instancename or reserve word. |
| NTR-35 | ON | WARNING | Parameter at or near line %d has been ignored. |
| NTR-36 | ON | ERROR | Verilog HDL Expression not handled near line %d. |
| NTR-37 | ON | ERROR | Invalid identifier '%s[%d]' at or near line %d. |
| NTR-38 | ON | ERROR | Invalid identifier '%s[%d:%d]' at or near line %d. |
| NTR-39 | ON | INFO | %s has been created for constant %d seen on line %d. |
| NTR-40 | ON | ERROR | Continuous assign (%s) statement is not supported near line %d. |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-41 | ON | ERROR | Duplicate '%s' port array definition in cell '%s,' on line %d. |
| NTR-42 | ON | WARNING | Duplicate Entry in Rename Table '%s' with '%s'. |
| NTR-43 | ON | FATAL | Undefined instance "%s" referenced by net "%s". |
| NTR-44 | ON | FATAL | Unexpected name "%s" for RIPPER cell. |
| NTR-45 | ON | FATAL | Cell '%s' not defined. |
| NTR-46 | ON | FATAL | Instance '%s' in cell '%s' line# %d has no CellRef. |
| NTR-47 | ON | INFO | Renaming net "%s" to match port "%s". |
| NTR-48 | ON | FATAL | Only (edifLevel 0) supported. |
| NTR-49 | ON | WARNING | Unsupported Number '%s' (Truncating..). |
| NTR-50 | ON | WARNING | Illegal value %s in the netlist (Ignoring..). |
| NTR-51 | ON | WARNING | Net "%s" in cell "%s" not connected to port with same name, add prefix %. |
| NTR-54 | ON | WARNING | %d error(s),%d warning(s) encountered during load. |
| NTR-55 | ON | WARNING | %s at or near line %d has been ignored. |
| NTR-56 | ON | ERROR | Bad token '%s' at or near line %d. |
| NTR-58 | ON | WARNING | Cannot change severity of FATAL message. 'NTR-%d' ignored. |
| NTR-59 | ON | WARNING | Cannot suppress ERROR and FATAL messages. 'NTR-%d' ignored. |
| NTR-60 | ON | WARNING | Invalid NetTran message id -%s '%s' ignored. |
| NTR-61 | ON | WARNING | Black-box cell '%s' not found in netlist. |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-64 | ON | ERROR | Library "%s" was not found. |
| NTR-65 | ON | ERROR | Open Library Failed, Return Code: %d. |
| NTR-66 | ON | ERROR | '%s' listed in portlist in cell '%s', but not in I/O variable list. |
| NTR-67 | ON | WARNING | Running NetTran with both -cdl-M and -cdl-chop might cause instance renaming conflicts, |
| NTR-68 | ON | FATAL | Must have permission to write in directory %s to open library. |
| NTR-69 | ON | ERROR | '%s=%s' in cell '%s' is a device. '%s' has been previously declared as subckt. Rerun NetTran with -cdl-renameDev option. |
| NTR-70 | ON | ERROR | Device '%s' in cell '%s' has type mismatch. Used as both '%c' and '%c', rerun NetTran with -cdl-renameDev option. |
| NTR-71 | ON | ERROR | Continuous assign with a list of identifiers is not supported near line %d. |
| NTR-72 | ON | ERROR | Assigning an array to a nonarray: Continuous assign (%s[%d:%d]) is not supported near line %d. |
| NTR-73 | ON | ERROR | Must specify global net name to substitute 1'b%s with, rerun NetTran with -verilog-b%s *globalNetName* option. |
| NTR-74 | ON | WARNING | Voltage must be a value not a parameter to compare with threshold value. Ignoring -sp-voltThresh for this source with dc=%s. |
| NTR-75 | ON | MSG | %s%c (This message is used to print the NetTran command-line syntax to the log file). |
| NTR-76 | ON | MSG | Translation completed. |
| NTR-77 | ON | INFO | Flattening cell "%s"%s... |
| NTR-78 | ON | WARNING | Global parameter "%s": unrecognized value. |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-79 | ON | WARNING | Parameter "%s", cell "%s": unrecognized initialization. |
| NTR-80 | ON | WARNING | Attribute "%s", instance "%s", cell "%s": bad parameter "%s". |
| NTR-81 | ON | WARNING | VHDL input format is no longer supported. Pursuing an alternate format netlist is recommended. |
| NTR-82 | ON | FATAL | Net '%s' previously equived to net '%s'. Line %d attempts to re-equiv with net '%s'. |
| NTR-83 | ON | WARNING | *.EQUIV found in the middle of CDL netlist, and will apply only to the netlist data following it. |
| NTR-84 | ON | FATAL | Output filename matches specified input file. |
| NTR-85 | ON | FATAL | Error opening black box file "%s". |
| NTR-86 | ON | FATAL | Root cell "%s" cannot be found in netlist. |
| NTR-87 | ON | WARNING | CDL control command "%s" in line %d is invalid or not supported. |
| NTR-89 | ON | MSG | Duplicate Cell '%s' found, change its name into '%s'. |
| NTR-90 | ON | WARNING | "%s = %s" and "%s = %s": *.EQUIV does not allow you to double-translate node names. |
| NTR-91 | ON | MSG | HN device: %s. |
| NTR-92 | ON | MSG | HN cell: %s. |
| NTR-93 | ON | MSG | Processing Verilog Vectors... |
| NTR-94 | ON | MSG | Writing Accounting File "%s"... |
| NTR-95 | ON | WARNING | Attribute "%s" of instance "%s" in cell "%s": %s ! |
| NTR-96 | ON | INFO | Duplicate cell "%s", changing cell name to "%s"... |

*Table 4-22 NetTran Error Messages(Continued)*

| ID | Status | Severity | Message |
|---|---|---|---|
| NTR-97 | ON | MSG | Processing duplicate cells. |
| NTR-98 | ON | ERROR | Line %d: value "%s". |
| NTR-99 | ON | WARNING | Line %d: value "%s" - %s. |
| NTR-100 | ON | MSG | Renamed file "%s" to "%s". |
| NTR-101 | ON | MSG | (NewLine "\n"). |
| NTR-102 | ON | MSG | %s. |
| NTR-103 | ON | MSG | Writing undefined cells to the file "undef.log"... |
| NTR-104 | ON | ERROR | Indices for nets of bus "%s" are not defined due to missing cell definition of "%s". |
| NTR-105 | ON | ERROR | Port "%s" of cell "%s" is defined as a bus, but a net which connects to the port is not. |
| NTR-106 | ON | ERROR | The input file '%s' is not in ASCII format. |
| NTR-107 | ON | WARN | The pattern of order is incompatible. Take original order instead. |
| NTR-108 | ON | FATAL | Syntax error near line %d in '%s': net '%s' was connected to two different nets: '%s' and '%s'. |
| NTR-109 | ON | WARN | Syntax '%s' is not supported. Ignore it... |

## SPICE Netlist Format

## DEVICES

- The MOSFET syntax is:

  `Mxxx DRN GATE SRC` *`bulk mname width length properties ...`*

- The RESISTOR syntax is:

  `Rxxx A B` *`mname`* `<R=>resistance` *`properties`* `...`

- The CAPACITOR syntax is:

```
Cxxx A B mname <C=>capacitance properties ...
```

- The DIODE syntax is:

```
Dxxx ANODE CATHODE mname properties ...
```

- The INDUCTOR syntax is:

```
Lxxx A B <L=>inductance properties ...
```

- The JFET syntax is:

```
Jxxx DRN GATE SRC bulk mname properties ...
```

- The BJT syntax is:

```
Qxxx COLL BASE EMIT bulk mname properties ...
```

## CELL DEFINITION

SPICE uses .SUBCKT syntax to define cells in a netlist. NetTran supports nested subcircuit definitions. The syntax is:

```
.SUBCKT cell_name port1 ... portN ... instances ...

.END<S> cell_name
```

## INSTANCE DEFINITION

An instance can be described as a copy of the cell. It is not possible to define two or more cells with the same cell name, but it is possible to define several instances with the same cell (model name). The instance names cannot be the same within a cell. If the instance names are in different cells, there is no such restriction.

An example of the Instance syntax is:

```
Xxxx pin ... pin model_name parameters
```

Note:

Device syntax can be considered as an instance of the specific model in the SPICE netlists.

## .GLOBAL

The .GLOBAL statement globally assigns a node name. All references to a global node name used at any level of the hierarchy in the circuit will be connected to the same node.

The .GLOBAL statement is most often used when subcircuits are included in a netlist file. This statement assigns a common node name to subcircuit nodes. Power supply connections of all subcircuits are often assigned using a .GLOBAL statement. For example,

.GLOBAL VCC connects all subcircuits with the internal node name, VCC. In a subcircuit, the node name is ordinarily given as the circuit number concatenated to the node name. When a .GLOBAL statement is used, the node name is not concatenated with the circuit number and is assigned only the global name. This allows exclusion of the power node name in the subcircuit or macro call.

The syntax is:

```
.GLOBAL node ... node
```

where *node* specifies global nodes such as supply and clock names, and overrides local subcircuit definitions.

## .INCLUDE

The syntax is:

```
.INCLUDE 'filepath filename'
```

| Argument | Description |
|---|---|
| *filepath* | The path name of a file for computer operating systems supporting tree-structured directories. |
| *filename* | The name of a file to include in the data file. |

The *filepath* plus *filename* can be up to 1024 characters in length and can be any valid file name for the computer's operating system. The file path and name must be enclosed in single (' ') or double quotation marks (" ").

## COMMENTS

There are two kinds of comments in a SPICE netlist format:

- asterisk (*): The asterisk is always placed at the beginning of a line, following the comment strings.
- dollar sign ($): The dollar sign is never placed at the beginning of a line. The strings after the dollar sign in a line are comments.

## Converting SPICE Netlist Property Units

By default, the unit for SPICE netlist properties is the meter. SPICE schematic files can be translated with different device property units using the following NetTran options. The syntax is:

`-sp-m` *`number`*

`-sp-S`

The -sp-m option allows you to multiply the device property numbers in a SPICE netlist by a factor.

The -sp-S option allows you to translate netlist properties in microns instead of the default unit of a meter.

---

## CDL Netlist Format

### *.BIPOLAR

The *.BIPOLAR command is used to preserve analog elements. In order to ignore specific analog elements, use the *.CAPA (capacitors), *.DIODE (diodes) and *.RESI (resistors) commands. Refer to the following definitions for proper syntax of each command.

### *.BUSDELIMITER

The *.BUSDELIMITER command specifies delimiter characters for bus pins inside a CDL netlist. This syntax is used by NetTran when it must apply a bus defined inside a separate netlist format, such as Verilog, to underlying bus pins used inside a CDL netlist. Valid delimiter characters understood by NetTran include a bracket ([), a curly brace ({), a less-than sign (<),or an underscore (_). Only the leftmost character is specified within the *.BUSDELIMITER command. If unspecified, the default bus delimiter character understood by NetTran is a bracket ([).

### *.CAPA

When using the *.BIPOLAR command, all analog elements are preserved. Using the *.CAPA command allows you to omit capacitors from the schematic netlist.

### *.DIODE

When using the *.BIPOLAR command, all analog elements are preserved. Using the *.DIODE command allows you to omit diodes from the schematic netlist.

### *.EQUIV

The *.EQUIV command equates node names in a layout database to the corresponding node number in the netlist.

## *.RESI

The *.RESI command is used to specify the threshold value of the shorted resistors. If the resistance between any two nodes in the resistor statement is less than or equal to the threshold value, the two nodes are shorted.

## *.SCALE

The default database unit in the netlist is microns. Setting the *.SCALE option to meter changes the default database unit from microns to meters. If you have multiple CDL netlists to be merged and translated, they should all have the same *.SCALE option at the top of the CDL file. If each CDL file has a scale setting, inconsistent scaling is avoided.

Because NetTran parses files sequentially, the scale setting only affects statements that come after it in the file.

## .OPTION SCALE

Another scaling factor adopted from SPICE-like syntax that is a global option. If it is in the CDL netlist, all elements following it are scaled. Multiple .OPTION SCALE are allowed, but a later .OPTIONS SCALE overrides the previous one.

If it is inside .SUBCKT, it must be the first line or NetTran reports an error. Even inside .SUBCKT, its scope is still global.

If .OPTION SCALE is set to a scale factor, it is resolved to a non-scale number, with a warning message from NetTran. For example, 1u is resolved to 1e-06.

## Converting CDL Netlist Property Units

Hercules converts the CDL netlist property units based on how the *.SCALE option is set in the CDL Netlist. *.SCALE can be set to micron (default) or meter. Setting this option to meter will change the default database unit from micron to meter. The parameters that represent distance are multiplied by $10^6$. When *.SCALE is not set, it does not exist in the CDL netlist. In this case, the u, p, and any other CDL value is ignored. Table 4-23 shows how the *.SCALE option modifies the length and width property values during netlist translation.

*Table 4-23 Property Conversion Table for Length and Width*

| CDL Value (L or W) | *.SCALE | Hercules Value |
|---|---|---|
| 1 | Not set | 1 |
| 1 | micron | 1 |
| 1 | meter | 10e+6 |

*Table 4-23 Property Conversion Table for Length and Width*

| CDL Value (L or W) | *.SCALE | Hercules Value |
|---|---|---|
| 1u | Not set | 1 |
| 1u | micron | 10e-6 |
| 1u | meter | 1 |
| 1p | Not set | 1 |
| 1p | micron | 10e-12 |
| 1p | meter | 10e-6 |

For example, the default property unit for a CDL netlist is the micron. The following two cases represent the same setting:

*.SCALE
*.SCALE micron

That means the u or p are considered, and become a multiplier of 10e-6 or 10e-12 on the value seen in the netlist. If the netlist has *.SCALE meter, there is an additional multiplier of 10e+6.

Thus, if the following netlist exists:

```
*.SCALE meter
M1 foo bar N w=1p
```

then, the value of the width is (1 * 10e-12 * 10e+6) = 10e-6. The value, when scaled out like this, has a final unit of microns. The result in Hercules will be 10e-6.

If the same table is created for area values, it gets more complicated.

*Table 4-24 Property Conversion Table for Area*

| CDL Value (area) | *.SCALE | Hercules Value |
|---|---|---|
| 1 | Not set | 1 |
| 1 | micron | 1 |
| 1 | meter | 10e+12 |
| 1u | Not set | 1 |

*Table 4-24 Property Conversion Table for Area(Continued)*

| CDL Value (area) | *.SCALE | Hercules Value |
|---|---|---|
| 1u | micron | 10e-6 |
| 1u | meter | 10e+6 |
| 1p | Not set | 1 |
| 1p | micron | 10e-12 |
| 1p | meter | 1 |

## Converting CDL and SPICE Netlist Formats

The *.LENGTH_UNIT, *.AREA_UNIT, and *.NON_UNIT commands are non-standard CDL and SPICE constructs that should be used only during NetTran conversion. Parameter names are case-insensitive.

The *.LENGTH_UNIT command can be used to treat parameter names not starting with L or W as L and W parameters during NetTran unit conversion. The syntax is:

```
*.LENGTH_UNIT parameter_1(device_type) ...
              parameter_n(device_type)
```

The *.AREA_UNIT statement can be used to treat parameter names not starting with A as an A parameter during NetTran unit conversion. The syntax is:

```
*.AREA_UNIT parameter_1(device_type) ...
            parameter_n(device_type)
```

The *.NON_UNIT statement can be used to exclude parameter names from NetTran unit conversion. The syntax is:

```
*.NON_UNIT parameter_1(device_type) ...
           parameter_n(device_type)
```

The device types supported are: MOS, RES, CAP, IND, BJT, DIODE, JFET. For example, to apply LENGTH_UNIT to S only in MOSFET devices:

```
*.LENGTH_UNIT S(MOS)
```

Use the property name without specifying a device type to apply the parameter to all device types. For example, to apply LENGTH_UNIT to S for all devices:

```
*.LENGTH_UNIT S
```

These commands only convert parameters to correct values. If you want to apply these commands to an empty subckt extracted as a non-standard device defined in CDL, you must also set -cdl-p on the NetTran command-line. In the following example, only non-standard devices defined by an empty subckt are handled with the unit S(model name):

```
*.AREA_UNIT S(syfns_res)
```

## Shorting Nets in CDL Netlist

For the LVS process, use the *.EQUIV option to short nets in the schematic netlist to match its connection to the layout netlist.

For example, a schematic netlist contains a standard cell with two ports, gnd! and gnda!, and a RAM cell with one port, vss. In a layout, gnd!, gnda!, and vss are connected to a global power net, vss. (When streaming out the netlist, these ports do not appear in the EDIF netlist because they are power.)

In the layout netlist during LVS, vss is properly connected to gnd!, gnda!, and the vss pin of the standard cell and RAM. In order to match layout nets to schematic nets, gnd!, gnda!, and vss have to be connected together. This can be done by adding the *.EQUIV option in the schematic netlist and retranslating it using NetTran.

The syntax is:

```
*.EQUIV new-name = old-name
```

Add the following to the schematic netlist:

```
*.EQUIV VSS=gnd! VSS=gnda! VSS=vss
```

---

# Verilog Netlist Format

## CELL DEFINITION

The Verilog format uses the keyword module to define cells in a netlist. A module can be an element or a collection of lower-level design blocks. Each module must have a *cell_name*, which is the identifier for the module, and a port list, which describes the input and output terminals of the module.

The syntax is:

```
module cell_name(port1, ..., portN);
    ...
endmodule
```

## PORT

The syntax for input is:

```
input port
```

The syntax for output is:

```
output port
```

The syntax for inout is:

```
bidirectional port
```

For example:

```
module AND( A, B, C);
    input A;
    input B;
    output C;
    ...
endmodule
```

## NET

An example of the syntax is:

```
wire a;         // Declare net a for the circuit
wire b,c;       // Declare net b and c for the circuit
wire d = 1'b0; // Net d is fixed to logic value 0 at declaration
```

where wire can be used to describe internal nets.

## INSTANCE

The syntax is:

```
model_name instance_name (connections);
```

For example:

```
AND I1 (net1, net2, net3);
```

## CONNECTION

The syntax is:

```
module AND (A, B, C )
    ...
endmodule
```

Example of connecting by order:

```
AND I1 (net1, net2, net3);
```

Example of connecting by name:

```
AND I1 (.B(net2), .C(net3), .A(net1));
```

## ASSIGN STATEMENTS

The syntax is:

```
assign old_name = new_name ;
```

For example:

```
assign net1=net2;
```

NetTran will short two nets, net1 and net2, and will use the name net2 to substitute for the name net1.

Note:

If the shorted net is connected to a port, it should retain the port name as the retained net and, in other cases, the net name.

## BUS (VECTOR)

Nets can be declared as vectors (multiple-bit widths). If the bit width is not specified, the default is scalar (1-bit).

For example, to declare the vectors:

```
output [3:0]  X // 4-bit X
input  [0:2]  Y
input  [2:12] Z
```

Then, the vectors would be used as follows:

```
X[3], X[2], X[1], X[0]

Y[0], Y[1], Y[2]

Z[2], Z[3], ..., Z[12]
```

## Defining Global Nets During a Translation

Nets are declared primarily with the keyword wire. In the following example, nets tie_high and tie_low are assigned a fixed value of 1 and 0 at declaration. In order to translate this correctly, use the NetTran options -verilog-b1 and -verilog-b0 to substitute 1'b1 and 1'b0 with the correct global net names. An error is generated if these options are not specified during the translation.

For example:

```
Input verilog netlist test.v2
wire tie_high, tie_low;
assign tie_high = 1'b1;
assign tie_low = 1'b0;
```

Use this command:

```
nettran -verilog test.v2 -outName herc.v2
```

The following messages are displayed:

```
Reading "test.v2" netlist ...
Loading cell "test_asic_pad" ...
ERROR: Must specify global net name to substitute 1'b1 with,
rerun nettran with -verilog-b1 globalNetName option.
ERROR: Must specify global net name to substitute 1'b0 with,
rerun nettran with -verilog-b0 globalNetName option.

ERROR: Errors during translation. See log file for listing.
```

As suggested by NetTran, use the -verilog-b0 and -verilog-b1 options during the translation to specify global nets for this netlist.

Use this command:

```
nettran -verilog test.v2 -verilog-b0 vss -verilog-b1 vdd
   -outName herc.v2
```

## Verilog Global Net Mapping Table (-verilog-voltmap *filename*)

The syntax of the mapping file is:

*cell-name inst-name* B0 *voltage-name* B1 *voltage-name*

Without specifying the instance name, NetTran translates the 1'b0 1'b1 within and under that cell to the assigned global netname. If the instance name is assigned, then the name mapping will be performed only on that specific instance.

Since this mapping processes down the hierarchy, unlike the current -verilog-b0 -verilog-b1 option, it can only be done at post-process. The NetTran -verilog-voltmap option cannot exist with -verilog-b0 or -verilog-b1. If these two options are set simultaneously, NetTran outputs a warning message and ignores the -verilog-voltmap option.

Example of the verilog-voltage mapping file:

```
sub1 B1 VDD1 B0 GND1
sub3 B1 VDD2 B0 GND2
top_io top_io1 B0 GND1 B1 VDD1
top_io top_io2 B0 GND2 B1 VDD2
```

## The -verilog-busLSB Option

The NetTran -verilog-busLSB option starts the Verilog bus with the least significant bit (LSB). The following example shows how it changes the translated netlist.

In a Verilog netlist:

```
test I1 (
        .byte_sel_n (byte_sel_n)
);
```

The net (wire) that is defined as bus:

```
wire    [4:0]   byte_sel_n;
```

The net contains 5 lines:

```
    byte_sel_n[4]
    byte_sel_n[3]
    byte_sel_n[2]
    byte_sel_n[1]
    byte_sel_n[0]
```

The port byte_sel_n from the cell test is not defined. The default is to treat the port as the net (from bus [4] to bus [0]). When the -verilog-busLSB option is set, NetTran treats the port from bus [0] to bus [4].

If the Verilog netlist is well defined (no undefined cells except primitives), the -verilog-busLSB option does not need to be set. NetTran will translate it to the correct order.

For example, with the Verilog netlist shown in Example 4-5, the Hercules netlist will be as shown in Example 4-6, and the Hercules netlist translated with the -verilog-busLSB option will be as shown in Example 4-7.

*Example 4-5 Verilog Netlist*

```
module top ();
```

```
    wire  [1:0] s_top;
    wire  [1:0] bus;

    mid X1 ( .i(bus), .s(s_top) );
endmodule

module mid ( .i({\i[0][0] , \i[1][1] }), s );

input  [1:0] s;
input  \i[0][0] , \i[1][1] ;

    nd02d4 X1 ( .a1(\i[0][0] ), .a2(\i[1][1] ), .zn(s[1]) );
    nd02d4 X2 ( .a1(\i[0][0] ), .a2(\i[1][1] ), .zn(s[0]) );

endmodule
```

*Example 4-6 Hercules Netlist*

```
{netlist herc.sch
{version 1 1 0 }
/* Options: -verilog bus.verilog -outName herc.sch */

{net_global }
{cell mid
{port s[0] s[1] i[1][1] i[0][0]}
{inst X2=nd02d4
{pin i[0][0]=a1 i[1][1]=a2 s[0]=zn}}
{inst X1=nd02d4
{pin i[0][0]=a1 i[1][1]=a2 s[1]=zn}}
}

{cell top
{inst X1=mid
{pin bus[0]=i[0][0] bus[1]=i[1][1] s_top[1]=s[1] s_top[0]=s[0]}}
}

}
```

*Example 4-7 Hercules Netlist: Translated with the -verilog-busLSB Option*

```
{netlist herc.sch.bus
{version 1 1 0 }
/* nettran:  DATA OMITTED */
/* Created:  DATE OMITTED  14:35 */
/* Options: -verilog-busLSB -verilog bus.verilog -outName herc.sch.bus */

{net_global }
{cell mid
{port s[0] s[1] i[1][1] i[0][0]}
{inst X2=nd02d4
{pin i[0][0]=a1 i[1][1]=a2 s[0]=zn}}
{inst X1=nd02d4
{pin i[0][0]=a1 i[1][1]=a2 s[1]=zn}}
```

```
}

{cell top
{inst X1=mid
{pin bus[0]=i[0][0] bus[1]=i[1][1] s_top[0]=s[1] s_top[1]=s[0]}}
}

}
```

## Implicit Port Mapping

NetTran can translate this type of Verilog netlist with no problems. However, you might get an error during translation that reads:

```
A port list is not defined for cell INV1
   Port IN1 not declared for cell INV1
...
```

This error generally means that you need to define a module for that particular cell with the correct port ordering. NetTran needs this module in order to correctly translate the Verilog netlist and get the correct port mapping.

# General Netlist Guidelines

## Translating Multiple Netlists

Translating multiple netlists to a Hercules-format netlist should be done in a single step. In order to convert a top-level Verilog netlist named topverilog.v with three SPICE and two CDL sublevel netlists, use the following syntax to avoid an error.

```
nettran -sp spice1.sp spice2.sp spice3.sp -cdl cdl1.sp cdl2.sp
-cdl-renameDev -verilog topverilog.v -verilog-b0 VSS
-verilog-b1 VDD -outName schematic.herc
```

The command-line options might not be the same, but it is important at this stage that each type of netlist is listed after the input format option (-sp, -cdl, and so forth). Multiple -sp or -cdl command-line options will result in a translation error. There should be no need to translate netlists in multiple steps.

## Merging Multiple Hercules-Format Netlists

Sometimes it is necessary to merge a top-level netlist, a standard cell library netlist, and a macro netlist. NetTran does this using the -noflatten option. For example:

```
nettran -noflatten -hercules top.net std.net macro.net -outName

output.net
```

The -noflatten option prevents NetTran from flattening the hierarchy to resolve parameters. By default, NetTran flattens the cells whose parameters refer to other cells so that no parameters are included in the output netlist. NetTran's ability to handle equations with parameters is lost if the -noflatten option is set. Even though the hierarchy is kept during translation, remember that during the COMPARE process these blocks do not serve as equivalence points.

## Duplicate Cell Definitions

When port numbers of cells are the same, NetTran accepts only the last cell. The following example demonstrates how NetTran processes duplicate cell definitions.

```
nettran -cdl s0.cdl -edif s3.edif s4.edif3 -sp s1.sp
   -verilog s2.v -hercules s5.net -outName output.net
```

The netlists are read in the following order:

```
Reading "s5.net" netlist ...
Reading "s4.edif3" netlist ...
Reading "s3.edif" netlist ...
Reading "s2.v" netlist ...
Reading "s1.sp" netlist ...
Reading "s0.cdl" netlist ...
```

If a specific block XYZ is defined in all the netlist files, NetTran will first read from the s5.net, and then write to the output file. NetTran then reads s4.edif3, and overwrites the whole XYZ block to the output file. Because s0.cdl is the last netlist to be read, XYZ in the s0.cdl will overwrite all the previous XYZ blocks in the output file.

If the -dup command-line option is used, NetTran looks to see if the multiple cell definitions are the same.

- If the definitions are the same, duplicate cells are ignored and the main cell is kept. A message is written to dupCell.log.
- If the two definitions are different, NetTran changes the name of the cell. A message is written to the dupCell.log file. For example:

```
"Duplicate cell "XYZ", changing cell name to "XYZ_NETTRAN_1".
```

Below is a explanation of the messages that are output in the dupCell.log:

```
CELL "XYZ" :
   ./home/user1/library/dir1/file1.sp : Line 919
   ./home/user1/library/dir2/file2.sp : Line 919 (master cell)
      portCount = 16
      instCount = 107
```

There are two places where the cell XYZ is defined. One is at line 919 of the file1.sp file, and the other is at line 919 of the file2.sp file. The cell that is designated as master cell becomes the reference against which all duplicate cells are checked. In this case, there are 16 pins and 107 instances in the master cell. Because there are no error messages in this section, the cell XYZ in the file1.sp file is the same as the cell in the file2.sp file.

The following section of logfile contains error messages for the FOOBAR cell.

```
CELL "FOOBAR" :
    ./home/user1/library/dir1/file1.sp : Line 311
        (ERROR - unmatched pin "#63" of instance "M1n1" !)
        (ERROR - unmatched pin "#63" of instance "M1p2" !)
        (ERROR - unmatched pin "#63" of instance "M1p3" !)
        (ERROR - unmatched pin "#61" of instance "M1p3" !)
     ...
    ./home/user1/library/dir2/file2.sp: Line 401 (master cell)
        portCount = 56
        instCount = 489
```

The error message

```
(ERROR - unmatched pin "#63" of instance "M1n1" !)
```

implies that there is a difference in the instance M1n1 within the two cell definitions.

Looking at the instance M1n1 at line 311 in ./home/user1/library/dir1/file1.sp:

```
M1n1  VSS  #63  NET#3  VSS  nmos l=0.25u w=2.00u
```

and at line 401 in the specified file ./home/user1/library/dir2/file2.sp:

```
M1n1  VSS  #50  NET#3  VSS  nmos l=0.25u w=2.00u
```

The error message indicates that the pin connect to #63 is different from the pin connect to #50. If NetTran finds an error, it will rename the instance to a different name in the form FOOBAR_NETTRAN_N. For this case, the following message is printed in the nettran.log file:

```
Duplicate cell "FOOBAR", changing cell name to "FOOBAR_NETTRAN_1"
    ...
```

It is recommended to use the -dup option with -rootCell. NetTran removes dangling cells (cells that cannot reach from the root) when -rootCell option is used.

## Case Sensitivity

When translating multiple netlists, a cell might be defined in lowercase in a sublevel netlist while an instance of this cell might be defined in uppercase in a top-level netlist. Because NetTran is case-sensitive, an error is generated during the translation.

For example, the top-level Verilog netlist is:

```
module top (
      ...
      ram_block ram_0 (
         .RD_ADDR[0]( rd_addr_0 ),
         .RD_ADDR[1]( rd_addr_1 ),
         ...
      ...
endmodule )
```

and the sublevel SPICE netlist is:

```
.subckt ram_block rd_addr[0] rd_addr[1]
```

This example has a sublevel SPICE netlist where the cell ram_block is defined. Ports of this cell are rd_addr[0] and rd_addr[1]. The top-level Verilog netlist has an instance of this cell where pins of the cell are now in uppercase. In order to translate the netlist, the following command is executed:

```
nettran -sp rams.sp -verilog top.verilog -outName sch.out
```

Executing this command results in a GNF warning due to case sensitivity. See “GNF Error/ Warning” on page 4-80 for more information.

```
NetTran - Netlist Translator (R), Release .... Synopsys. All rights
reserved.
Options: -sp rams.sp -verilog top.verilog -outName sch.out
Reading "top.verilog" netlist ...
Loading cell "top" ...
Reading "rams.sp" netlist ...
Loading cell "ram_block" ...
WARNING: GNFwarning: Port RD_ADDR[0] is not declared for cell ram_block.
WARNING: GNFwarning: Port RD_ADDR[1] is not declared for cell ram_block.
Writing Hercules netlist ...
Translation completed.
```

In order to solve this problem, use the NetTran -sp-U option to convert SPICE input to uppercase. NetTran also offers options to convert EDIF, CDL, and Verilog input to uppercase.

```
nettran -sp-U -sp rams.sp -verilog top.verilog -outName sch.out
```

In the preceding example, detecting the problem with case sensitivity was not difficult due to a GNF warning message. In a case of parameter passing, an error is not reported. The following example demonstrates this case:

```
* 4 terminal n-channel device
*
.SUBCKT xntran4 D G S BULK vdd vss wn=0 ln=0.4 MN1 D G S BULK EN LN WN
```

```
.ENDS
*
* 4 terminal p-channel device
*
.SUBCKT xptran4 D G S BULK vdd vss wp=0 lp=0.4 MP1 D G S BULK EP LP WP
.ENDS
```

In the sub-circuit definition for xntran4, the specified parameters are wn=0 ln=0.4, while the MN1 instance in this subckt specifies LN and WN. In all instantiations of xntran4, the parameters are specified in lowercase. NetTran is case-sensitive and functions as though these parameters are not used inside the xntran4 subckt. As a result, NetTran discards them. In order to solve this problem, change the WN, LN, LP, and WP to lowercase in the sub-circuit definitions.

## Handling Parametrized Cells (pcells)

The only way that NetTran can process a netlist with pcells is to flatten them. Even if the hierarchy could be maintained and Hercules COMPARE could check the devices and connectivity in the block, it would still need to compare all the properties of each instance in the parent cell, which is the same method as flattening the blocks. Instead of spending a lot of effort enhancing lsh to partially keep the hierarchy, it is a lot easier to let NetTran flatten the pcells.

The NetTran -noflatten option is used when comparing the logic of a pcell without comparing properties. There is no way to compare the logic of a pcell with the properties. For example:

```
{Cell FOO
   {INST I1 = Bar
      {Param wc = 4 lc = 5 }}
   {INST I2 = Bar
      {Param wc = 6 lc = 2 }}
}

{Cell BAR
   {INST M1 = NMOS
      {PROP w = wc l = lc}}
```

When comparing the Schematic Cell Bar to the Layout Cell Bar, properties 4 and 5, or 6 and 2 are used. There is no way to perform an instance-specific comparison of the Cell BAR. To compare properties in this case, BAR must be flattened.

## Three-Terminal Resistors

SPICE netlists do not support three-terminal resistors, but CDL netlists do. Because the two formats are almost identical, treat your SPICE netlist as a CDL netlist when translating it into Hercules format for an LVS compare. To do this, use the following syntax for the resistor definition statement:

```
r1 MINUS PLUS $SUB=BULK $.MODEL=SCH_RLDDB r=r w=w l=l
```

| Argument | Description |
|---|---|
| $SUB | The third terminal. |
| $.MODEL | The resistor's model name. |

Tell NetTran to treat the netlist as if it were CDL format by specifying the -cdl command-line option;

The -cdl-a option tells NetTran to retain passive devices, such as resistors, during netlist translation. The same task can be accomplished by adding the *.BIPOLAR command to the SPICE/CDL netlist.

## GNF Error/Warning

GNF stands for Generic Netlist Format. GNF is a data structure of NetTran designed to store various netlists. A GNF error indicates that there are errors in the GNF data structure that are a result of either a NetTran bug or incorrect input data.

# 5

# General Hercules Execution

*This chapter describes the syntax used to execute a Hercules runset, as well as information on command-line arguments, executing Hercules from the GUI debugging tool, and the Hercules high performance capabilities, distributed processing, and multithreading.*

## Hercules Command-Line Syntax

Hercules allows you to change many runset options on a command line. This is especially useful when you do not wish or are not allowed to edit a runset file, but you need to run with a different setting. All options set on a command line will override runset settings.

The Hercules command-line syntax is:

```
hercules [-A] [-C] [-F] [-L file] [-N]
       [-O format] [-P] [-R] [-V]
       [-Version] [-X file] [-a] [-af assign_file]
       [-dppath32 path_to_32-bit_executables]
       [-b block] [-c compare_dir]
       [-cg] [-d] [-da] [-df] [-dm] [-dna] [-dnrs]
       [-dpbatchslaves N] [-dpgui]
       [-dphosts host_name_1 group_cache_dir_1
       ... host_name_n group_cache_dir_n]
       [-dp[slaves]] [-dracula] [-disable [passnames]]
       [-e equivalence] [-early-compare]
       [-ece] [-ep err_prefix]
       [-ex] [-f format] [-force-gdsin]
```

```
[-force-gds-direct] [-force-gds-direct-in]
[-force-gdsout] [-force-gds-direct-out]
[-full_equiv] [-g group_dir] [-ga]
[-hst host_name] [-html] [-html-nobrowse] [-i inlib]
[-il incremental_layer, ..., incremental_layer]
[-iln layer_number, ..., layer_number]
[-j] [-k gdsin_dir] [-ke] [-lvs-license] [-m factor]
[-mfe] [-ncg] [-no-prune] [-no-rmdup] [-nro]
[-noSets] [-noVcell] [-np M | -np NxM] [-ntd]
[-o outlib]
[-ob output_block] [-p layout_path] [-psi]
[-rd run_details_dir] [-ro] [-prune-assign]
[-s schematic]
[-select_rule rule_1 |
-select_rule rule_1 ... rule_n \;]
[-sf CDL|SPICE|EDIF|EDIF3|VERILOG|SILOS|HERCULES]
[-srf filename] [-stb block] [-t output_layout_path]
[threads num_threads] [-tsub]
[-unselect_rule rule |
-unselect_rule rule_1 ... rule_n \;] [-use_hier_fill]
[-vue] [-vueshort] [-w working_dir] runsetfile
```

Table 5-1 describes the command-line syntax.

*Table 5-1 Command-Line Syntax Descriptions*

| Syntax | Description |
|---|---|
| -A | Prints the host ID number and stops. |
| -C | Runs only a netlist comparison between the existing schematic netlist and a previously generated layout netlist, using the information specified in the runset. DRC and extraction are not done. |
| -F | Stops Hercules after group file creation. |
| -L *file* | Redefines the Explode List file specified in the runset. |
| -N | Runs only netlist commands that use existing netlist databases in the specified runset. The specified runset must include a corresponding command for every device that exists in the netlist database. |
| -O *format* | Uses *format* as the format of the output data (GDSII, Milkyway, or OASIS), overriding the OUTPUT_FORMAT variable specified in the runset. |
| -P | Runs preprocessing only and generates *block*.tree files. Preprocessing includes generating tree files, grid check, check90, and check45. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -R | Generates an archive file containing the runset, Explode List files, Edtext files, and the equivalence file to recreate the current run. The file is called *block*.archive.tar and its contents can be used by first typing tar xvf *block*.archive.tar at the shell prompt. |
| -V | Prints version numbers for the individual components of Synopsys software. |
| -Version | Prints version numbers for the individual components and the library versions of Synopsys software. |
| -X *file* | Redefines the Edtext file in the runset. |
| -a | Produces only ASCII output. |
| -af *assign_file* | Autogenerates an ASSIGN section within this file. Only a HEADER section is necessary in the input runset. All layers within the hierarchy under the top cell will be listed. Does not handle text. By default, layer names are composed by layer number and data type. Default is NONE. |
| -dppath32 *path_to_32-bit_executables* | Path to 32-bit DP executables. |
| -b *block* | Uses *block* as the name of the block to be checked, overriding the block variable specified in the runset. Hercules looks for this block in the library specified by the INLIB variable. If the block is not found, Hercules exits with a message that the block could not be found. |
| -c *compare_dir* | Uses *compare_dir* as the path name of the directory in which files created for, and resulting from, the netlist comparison are written, overriding the COMPARE_DIR variable specified in the runset. If a given COMPARE directory does not exist, that directory is created. The default is ./run_details/compare. |
| -cg | Forces Hercules to store group files in memory. |
| -d | Restricts display of dots in screen output. |
| -da | Disable aliasing of group files. |
| -df | Forces a distributed run to use the distributed format for output to the screen. This is the same format used by the distributed log file. For more information, see “Distributed Log File” on page 5-19. |
| -dm | Automatically starts dp-monitor in distributed run. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -dna | Forces a distributed run not to abort on failure. Skips the command that failed and any other commands that are dependent on the failed command. |
| -dnrs | Forces a distributed run not to make an attempt to restart a failed task. |
| -dpbatchslaves *N* | *N* slaves are already running. |
| -dpgui | Start dpgui from Hercules directly. |
| -dphosts *host_name_1 group_cache_dir_1... host_name_n group_cache_dir_n* | Defines hosts to be used in a distributed run across a network. A host can be specified more than once to use multiple CPUs. *group_cache_dir_n* is an optional parameter used to specify the group_cache_dir for auto generation in the .dprc file. The default group_cache_dir is CACHE_DIR, in the group_dir (see -g command-line option). Use unique directory names for each user; the user must have read/write privileges in the directory. |
| -dp*slaves* | Starts a distributed run using the quick start method. You need to specify a number of processes to start after specifying this option; -dp2, for example. For more information, see "Distributed Processing Quick Start" on page 5-16. |
| -dracula | Allows Hercules to run off of a Dracula physical verification runset. |
| -disable [*passnames*] | Disables the named runset optimizations. By default, all passes are executed. The possible passnames are available in the help message for the command. The optimizations performed are:<br>• `prune` to discard commands with no effect.<br>• `rmdup` to replace duplicate commands with COPY commands.<br>• `merge` to replace related commands with a single command.<br>The *passnames* value can be a comma-separated list of the above passnames, or one of the two values `all` and `none`. |
| -e *equivalence* | Uses *equivalence* as the name of the file that lists the blocks to be equated by the netlist comparison, overriding the EQUIVALENCE variable specified in the runset. If the *equivalence* file is not found and LVS COMPARE is being run, Hercules exits with a message that it cannot open the file to read. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -early-compare | Instructs Hercules to distribute LVS Compare, and commands like GRAPHICS, resulting in faster overall lvs extraction and compare performance.<br>This option creates the layout netlist as soon as the netlist database is complete, and starts compare as soon as the layout netlist is available.<br>Since this option is a distributing processing feature, the -dp command line option must be used. It should be noted that by allowing these processes to run in parallel, additional memory will be used. |
| -ece | Reports an exit code if an error occurred in *block*.err. |
| -ep *err_prefix* | Redefines error prefix in runset. |
| -ex | Runs only Hercules extraction. |
| -f *format* | Uses *format* as the format of the input data (GDSII, Milkyway, or OASIS), overriding the FORMAT variable specified in the runset. |
| -force-gdsin | Forces use of GDSIN utility to process GDSII Stream data. |
| -force-gds-direct | Forces use of GDSII Direct Input to process GDSII-formatted stream file. |
| -force-gds-direct-in | Force direct input for GDSII data. |
| -force-gdsout | Forces GDSII data output using GDSOUT. |
| -force-gds-direct-out | Forces GDSII data output using direct gds. |
| -full_equiv | Use full unoptimized equivalence list. |
| -g *group_dir* | Uses *group_dir* as the path name of the directory in which group files are created, overriding the GROUP_DIR variable specified in the runset. If a given group directory does not exist, that directory is created. Default is ./group. Multiple directories can be specified with multiple -g entries. Use unique directory names for each user; the user must have read/write privileges in the directory. |
| -ga | Gate array design. |
| -hst *hostname* | For distributed runs, forces group file creation on the named host. |
| -html | Produces HTML data and invokes the browse script at the completion of the run. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -html-nobrowse | Produces HTML data and the browse script but does not invoke the script at the completion of the run. |
| -i *inlib* | Uses *inlib* as the name of the input library that contains the block to be checked, overriding the INLIB variable specified in the runset. For Milkyway data, Hercules looks for the library in the directory defined by the LAYOUT_PATH variable. If the input format is specified as GDSII, Hercules uses the file specified by INLIB for input. |
| -il *incremental_layer*, ..., *incremental_layer* | Defines the layers to be used in a DRC incremental-by-layer run. It overrides the INCREMENTAL_LAYERS option in the runset.<br>Either layer numbers or layer names can be used, but they cannot be mixed. Specify multiple layers with a comma or a space as separators. (Enclose arguments in quotation marks if spaces are used.) |
| -iln *layer_number*, ..., *layer_number* | Maps the layer numbers in the ASSIGN section of the runset. The run checks only those layers. It overrides the INCREMENTAL_LAYERS option in the runset.<br>This option accepts layer numbers only. Specify multiple layer numbers with a comma or a space as separators. (Enclose arguments in quotation marks if spaces are used.)<br>To specify layer names, use the -il option. The -il and -iln options can be used together on the command line. |
| -j | Runs hercules-cell. Hercules-Cell runs a verification on small designs or cells. It exits immediately with a warning if the size of the data exceeds the check limit. Hercules-Cell supports all Hercules commands and options, except the LOAD_GROUP command and the RESTART command. |
| -k *gdsin_dir* | Uses *gdsin_dir* as the path name of the directory in which the layout directory is temporarily created for the GDSIN command when FORMAT is set to GDSII. The layout directory is removed when Hercules completes the run. The default is group_dir. |
| -ke | Retains the *block*.err and *block*.errsum files after they have been used to create the *block*.LAYOUT_ERRORS file. |
| -lvs-license | Enables conditional licensing for omitting a DRC license during an LVS run. For more information see “HERCULES_LVS” on page 2-3. |
| -m *factor* | Scales the input database magnification. |
| -mfe | Provides the ability to parse missing files specified in the runset as fatal errors. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -ncg | Forces the OPTIONS section option CACHE_GROUP to FALSE. |
| -no-prune | Equivalent to -disable prune. |
| -no-rmdup | Equivalent to -disable rmdup. |
| -nro | Equivalent to -disable all. |
| -noSets | Turns off the inclusion of the SETS vcell pass when adding technology defaults. A SETS vcell pass that is already in the runset will still run. |
| -noVcell | Turns off the inclusion of all vcell passes when adding technology defaults. Any vcell passes that are already in the runset will still run. |
| -np M \| -np NxM | Redefines NUM_PARTITIONS or N_BY_M in the runset. For example, if you specify an integer (-np 2), then NUM_PARTITIONS is redefined as 2. If you specify -np 2x2, N_BY_M is redefined as 2x2. If -np is used, it takes precedence over NUM_PARTITIONS and N_BY_M settings in the runset. |
| -ntd | Turns off the inclusion of default TECHNOLOGY_OPTIONS in runset. |
| -o *outlib* | Uses *outlib* as the name of the output library where error structures are placed, overriding the OUTLIB variable specified in the runset. For GDSII data, Hercules creates the GDSII file specified by OUTLIB. The default is *block*.out. |
| -ob *output_block* | Uses *output_block* as the name of the top-level block in the output library, overriding the *OUTPUT_block* variable specified in the runset. The default is EV_OUTPUT. |
| -p *layout_path* | Uses *layout_path* as the path name of the directory that contains the input database, overriding the LAYOUT_PATH variable specified in the runset. |
| -psi | Performs the self-intersect testing before optimization of the hierarchy rather than after optimization. |
| -rd *run_details_dir* | Specifies the run details directory to use. This will override the RUN_DETAILS_DIR variable specified in the runset. |
| -ro | Executes a run_only run directly from the runset. For more information, see the *Hercules Reference Manual*, Detailed Commands chapter. |
| -prune-assign | Enable pruning of ASSIGN layers. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -s *schematic* | Uses *schematic* as the name of the schematic netlist file used by the netlist comparison, overriding the SCHEMATIC variable specified in the runset. If the schematic file is not found when LVS COMPARE is run, Hercules exits with a message stating it cannot open the file to read. |
| -select_rule *rule* \| -select_rule *rule_1* ... *rule_n* \; | Run only commands with the same rules in COMMENT option. Abbreviate as -sr. Use quotes and an asterisk. For example, “M1*”. -select_rule *regex-pattern*, is added to select certain matching commands and must be abbreviated, -sr *regex-pattern*. More than one regex-pattern can be specified if followed by a semicolon. Regex-patterns must not contain a leading minus “-”. The command string, *-sr “A*” “B*” “C*” \; t.ev* is equivalent to *-sr “A*” -sr “B*” -sr “C*” t.ev*. |
| -sf CDL \| SPICE \| EDIF \| EDIF3 \| VERILOG \| SILOS \| HERCULES | Specifies the input schematic netlist format. This option overrides the schematic_format option of the HEADER section. |
| -srf *filename* | Allows for a selectable rules file. The file format follows the same format as the command line.<br>• It supports the same wildcard globbing as does –sr and SELECT_RULE.<br>• Each line in the file must start with either:<br>-sr or –select_rule for selecting a rule<br>-ur or –unselect_rule for unselecting a rule<br>Lines that start with the # character are ignored (comment).<br>Example:<br>`-sr "M1*" "V1*"`<br>`-ur "M1B*"`<br>`#-sr "M2*"`<br>This example is equivalent to:<br>`-sr "M1*" "V1*" –ur "M1B*"`<br>The first line could also be two separate lines and have the same result:<br>`-sr “M1*”`<br>`-sr “V1*”`<br>`-ur “MIB*”` |
| -stb *block* | Defines root cell in schematic netlist for lvsgen. |

*Table 5-1 Command-Line Syntax Descriptions(Continued)*

| Syntax | Description |
|---|---|
| -t *output_layout_path* | Uses *output_layout_path* as the path name of the directory which contains the layout subdirectory of output Milkyway databases, overriding the OUTPUT_ LAYOUT_PATH variable specified in the runset. |
| -threads *num_threads* | Specifies the number of threads to use for a multithreaded run. |
| -tsub | Translates substrate from Dracula physical verification using the CELL_EXTENT command. |
| -unselect_rule *rule* \| -unselect_rule *rule_1* ... *rule_n* \; | Discard commands with the same rules in COMMENT option. Abbreviate as -ur. Use quotes and an asterisk. For example, “M1*”. -unselect_rule *regex-pattern*, is added to disable certain matching commands. This must be abbreviated -ur *regex-pattern*. More than one regex-pattern can be specified followed by a semicolon. Regex-patterns must not contain a leading minus “-”. |
| [-use_hier_fill] | When this command-line option is set, Hercules adds hierarchical fills. Default is top-level. |
| -w *working_dir* | Defines the Hercules working directory. Hercules will change directories to the specified directory and run from there. It will return to the original directory upon completion of the run. |
| -vue | Create VUE output. |
| -vueshort | Create VUE output and extract short data. |
| *runsetfile* | Name of the executed runset file. |

## Hercules High Performance

This section describes the Hercules high performance capabilities, which can take advantage of multiple CPUs on a single server and multiple servers across a network. The Hercules engine supports multithreading as well as distributed processing. To get the most out of Hercules, it is important to understand both capabilities and how they can interact. The following sections describe multithreading and distributed processing in detail, with subsections on combining the processes for the most efficient runtime, memory use, and disk use on high capacity servers with multiple CPUs.


- “Multithreading” on page 5-10
- “Distributed Processing” on page 5-11

- “Distributed Processing Components” on page 5-12
- “Distributed Processing Autostart” on page 5-13
- “Distributed Processing Quick Start” on page 5-16
- “Dual Mode Processing” on page 5-16
- “Monitoring a Distributed Run” on page 5-17
- “Adding CPU to a Distributed Run” on page 5-18
- “Distributed Output” on page 5-19
- “Subcommands” on page 5-20
- “Combining Multithreading and Distributed Processing” on page 5-21
- “Troubleshooting Distributed Processing” on page 5-21


## Multithreading

Hercules multithreading is handled by algorithms within each command. Each command is run on its own, but the work within that command is separated among multiple threads. Unlike distributed processing, multithreading allows parallelism within the command rather than at the runset level. Multithreading has the advantage of being more transparent. Multithreading is also more memory- and disk-efficient, because it runs only one command at a time. Multithreading is limited, however, in that it can be run only on a single server and multiple CPUs are required for maximum benefit.

Hercules multithreading is automatic. Hercules determines the number of CPUs on the server and automatically runs that number of threads, provided the appropriate high performance licenses are available. The one exception is when distributed processing is enabled. Distributed processing will not automatically run multithreaded; however, you can use command-line options to turn it on. Also, you can set the number of threads with the command-line argument -threads. The following command line will force Hercules to run with two threads:

```
hercules -threads 2 runset.ev
```

Verify the number of threads by checking the summary file (*block*.sum). A run with two threads will report the following:

```
EV_Engine (R) Hierarchical Design Rule Checker, Release U-2003.03.0238
2003/05/12

  (C) Copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003.
  Synopsys, Inc.  All rights reserved.
```

```
Running Multi-Threaded code with 2 thread(s)

Runset file .....................  lab.ev
Current Directory ..............  /var3/bane/training/feb03
Hostname ........................  u804
Platform type ...................  SUN64_58
MILKYWAY input library path ......  /var3/bane/training/feb03/
```

A run not utilizing threads will report the following:

```
EV_Engine (R) Hierarchical Design Rule Checker, Release U-2003.03.0238
2003/05/12

  (C) Copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003.
  Synopsys, Inc.  All rights reserved.

Running Single-Threaded code

Runset file .....................  lab.ev
Current Directory ..............  /var3/bane/training/feb03
Hostname ........................  u804
Platform type ...................  SUN64_58
MILKYWAY input library path ......  /var3/bane/training/feb03/
```

## Distributed Processing

Distributed processing takes advantage of multiple CPUs on a single host or multiple hosts across a network. By distributing Hercules processes to multiple CPUs, commands in a runset can be processed at the same time (in parallel), which reduces the time needed to complete a runset.

Two types of distributed runs are possible:

- One uses multiple hosts across a network.
- The second uses a single host with multiple CPUs.

You can distribute a Hercules run to any number of CPUs on any number of host machines of the same type. Hercules does not support running distributed processing on different host architectures. If a host has multiple CPUs, then multiple Hercules instances can run on it. Each instance is referred to as a slave.

You can automatically run Hercules distributed processing with command-line options, or manually through setup files. Hercules also has an X/Motif tool that you can use to monitor distributed runs; however, this tool is not required.

The following sections describe how to run Hercules Distributed Processing automatically and view the run with the X/Motif tool.

## Distributed Processing Components

Table 5-2 gives the terminology describing distributed processing files and components.

*Table 5-2 Components of Distributed Processing*

| File/Executable | Component | Definition |
|---|---|---|
| .dprc | | File name, not a suffix, consisting of a list of hosts to use for a distributed run |
| hercules -dp *runset_file* | | Executable and command line argument that starts distributed Hercules |
| | dp_master | Master program that schedules parallel commands, started by Hercules |
| | .evdp | Suffix appended to a parallel runset file generated by Hercules and used by dp_master |
| dp_slave | | Executable that starts dp_slave daemon. Must be started on each host in a distributed run |
| | dp_slave daemon | Resides on host and, when connected to dp_master, starts a slave process and connects the slave process to dp_master |
| | slave process | Communicates with dp_master and manages distributed processing on a host. Started by dp_slave daemon |
| | host | Individual machine on network |
| | dp_portmap | Executable that is used by other components to register or obtain portmapping information. |

### Removing Distributed Components

The slave daemons run as background processes. To stop these processes manually, use the UNIX `kill -2` command.

A distributed run automatically clears its process after a successful run. However, if a run abnormally terminates, some processes can remain. The following processes run on the slave host machines specified in the .dprc file:

- DP_SLAVE
- EV_ENGINE

The dp_slave forks when connected, so there might be multiple instances displayed. It is typical to have one dp_slave per user on a host. You may restart the dp_slave daemon after killing all of the processes in preparation for future runs.

The following processes run on the host in which a distributed run was executed:

- DP_MASTER

The DP_MASTER process forks during a run so there are two instances displayed per run. If any DP_MASTER processes remain after a run, they can be terminated using the `kill -2` command.

Note:
A host machine executing a run can also be included in the .dprc file and be used as a slave host. In this case, both master and slave processes can remain on the host.

## Distributed Processing Autostart

The autostart option for Hercules distributed processing allows you to start a distributed run on a single host or on multiple hosts across a network in one step. It can automatically start all distributed components and resolve network paths. Your system must meet certain requirements to utilize all of the features in this option.

The hosts you use to complete the distributed run must be accessible to one another using the UNIX commands, rsh or remsh (depending on your operating system).

**Caution:**
When running distributed Hercules, the path of the run_details directory, the group directory, and the directory from which Hercules is started must all be network paths. A network path is a path that has the same name on all the hosts being utilized for a distributed run.

An example of a typical network path is:

```
/net/orange/local/group
```

The automount daemon on many UNIX and LINUX systems is set up so that the network paths are actually symbolic links to temporary mount points, which are visible only on the local machine. This means that the distributed working directory might be different from the working directory selected by the user via the cd command. The following command, run on a host with such an automounter, demonstrates the problem:

```
(cd /tools; /bin/pwd)
/.autofs_direct/+/tools
```

In the preceding example, the user has changed to the automounted directory, /tools, but the automounter has actually mounted this directory at /.autofs_direct+/tools and created a symbolic link to make it appear that /tool exists.

When distributed Hercules is run in one of these automounted directories, it communicates the wrong working directory to the remote machines. This causes the starting of the remote slave processes to fail, and the run to be aborted.

This problem can be solved by using the -w option to tell Hercules what its working directory is. For example, to run in the /tools directory on the host used in the previous example, you could execute the following commands:

```
cd /tools
hercules -w /tools runset.ev
```

### System Setup

Check the file /etc/hosts for each host on which you are trying to complete the distributed run to make sure each host has been removed from the local host entry. Otherwise, you might experience connecting problems among dp_slaves, the executables that start the dp_slave daemons.

For example, if you are checking /etc/hosts on myhost1, you get:

```
127.0.0.1 myhost1 localhost.localdomain localhost
```

If you are unable to change the file, ask your system administrator to change it to the following:

```
127.0.0.1 localhost.localdomain localhost
```

### System Test

You can test your system before starting the distributed run by using the following method:

If you want to run your program on host_name_1 and host_name_2, type the following on host_name_1:

```
rsh host_name_2 ls
```

You will be able to determine if you can obtain the directory list from host_name_2. If you receive messages such as cannot access host_name_2 or access denied, contact your local network administrator to correct the problem before using the autostart option.

### Running Distributed Processing in Autostart Mode

The following is syntax for running distributed processing in auto mode from the command line. This is a very easy method and requires no setup, only some additional command-line arguments. You have the option to set up part of the distributed run manually and let the rest be set up automatically. See “Details on Autostart Distributed Runs” on page 5-15 for details on the operation of the autostart feature. Syntax is provided both for a run on a single host utilizing multiple CPUs and for a run on multiple hosts over the network.

The syntax for Single-Host Run is:

```
hercules -dpn runset.ev
```

- -dp*n*: Specifies distributed run with *n* number of distributed processes

The syntax for Network Run is:

```
hercules -dp -dm -dphosts host_name_1 [group_cache_dir1] ...
   host_name_2 [group_cache_dir2] runset.ev
```

- -dm: Option to autostart dp_monitor after starting dp_portmap and dp_slave
- -dphosts *host_name1* ... *host_namen*: Command-line option that specifies the hosts used to run distributed processing. You can specify a host more than once to utilize multiple CPUs on that host.
- *group_cache_dir*: Specifies the directory in which to store group files locally on the host. This command is optional.
- runset.ev: Hercules runset

This example starts a two-process distributed run on the local host.

```
hercules -dp2 runset.ev
```

This example starts a networked distributed run using two CPUs on host1 and one on host2. After dp_portmap runs, dp_monitor will start automatically and dp_slave runs on each remote host.

```
hercules -dp -dm -dphosts host1 host1 host2 runset.ev
```

This example starts a networked distributed run on host1, host2, and host3. Host1 can store temporary data locally on /l0/group/userx/cache1.

```
hercules -dp -dphosts host1 /l0/group/userx/chache1 host2 host3 runset.vc
```

**Details on Autostart Distributed Runs**

This section provides details on the operation of the autostart feature, which are important if part of the distributed setup process is done manually.

The autostart option has the following features:

- The environment variable, HERCULES_PMAP_HOST, should not be set.
- The environment variable, HERCULES_PMAP_PORT, should not be set.
- .dprc file does not have to be set.

If you do not have the .dprc file, you might need to use the command-line option -dphosts to tell Hercules which hosts you want to use.

- Automatically starts the DP_PORTMAP program.
- Determines if dp_slave is running on each host specified in the .dprc file and, if not, automatically starts the program.

## Distributed Processing Quick Start

The quick start option for Hercules distributed processing starts a distributed processing run in one step. However, the run is allowed to use only the current host. (Distributed processing quick start is most effective when the current host has multiple processors.)

Note:
The quick start option does not support distributed processing across multiple hosts.

The syntax is:

```
hercules -dpN runset.ev
```

- -dp*N*: The distributed processing quick start command-line option. *N* represents the user-specified number of processes to start.
- runset.ev: A Hercules runset

For example, to start a two-process distributed run on the current host:

```
hercules -dp2 runset.ev
```

For example, to start a three-process distributed run on the current host:

```
hercules -dp3 runset.ev
```

## Dual Mode Processing

Dual Mode processing extends the processing capability of distributed processing by cutting the design into smaller partitions of data. Most of the processing of a given partition can be done in parallel, entirely independent of any other partition, thereby increasing the scalability of parallel processing.

The method for specifying hosts for Dual Mode processing is the same as for distributed processing. Dual Mode processing can also use a .dprc file, specify the GROUP_CACHE_DIR, and invoke dp_monitor to keep track of jobs. Dual Mode processing is invoked by the USING PARTITIONS DO command. For more information, see the USING PARTITIONS DO processing command in the *Hercules Reference Manual*, Detailed Commands chapter.

## Monitoring a Distributed Run

An X/Motif™ tool is available for monitoring a distributed run. To execute, enter the following:

```
dp_monitor
```

The monitor connects automatically to the distributed run if the monitor tool is executed on the same host and executed before a distributed run.

Note:

dp_monitor must run after dp_portmap and dp_slave are started. If you use the autostart distributed run feature, dp_monitor cannot be started manually before the distributed run is started. In this case, you might need to use the command-line option -dm to start dp_monitor automatically. For more details, see "Distributed Processing Autostart" on page 5-13.

The following is an example of an execution command:

```
%>dp_monitor &
%>hercules -dp rset.ev
```

To manually connect the monitor after beginning a distributed run, type the host name in the text window and use the Connect option in the File menu. This also allows you to connect to a run on a different host. However, only one monitor can be connected to a run at a time.

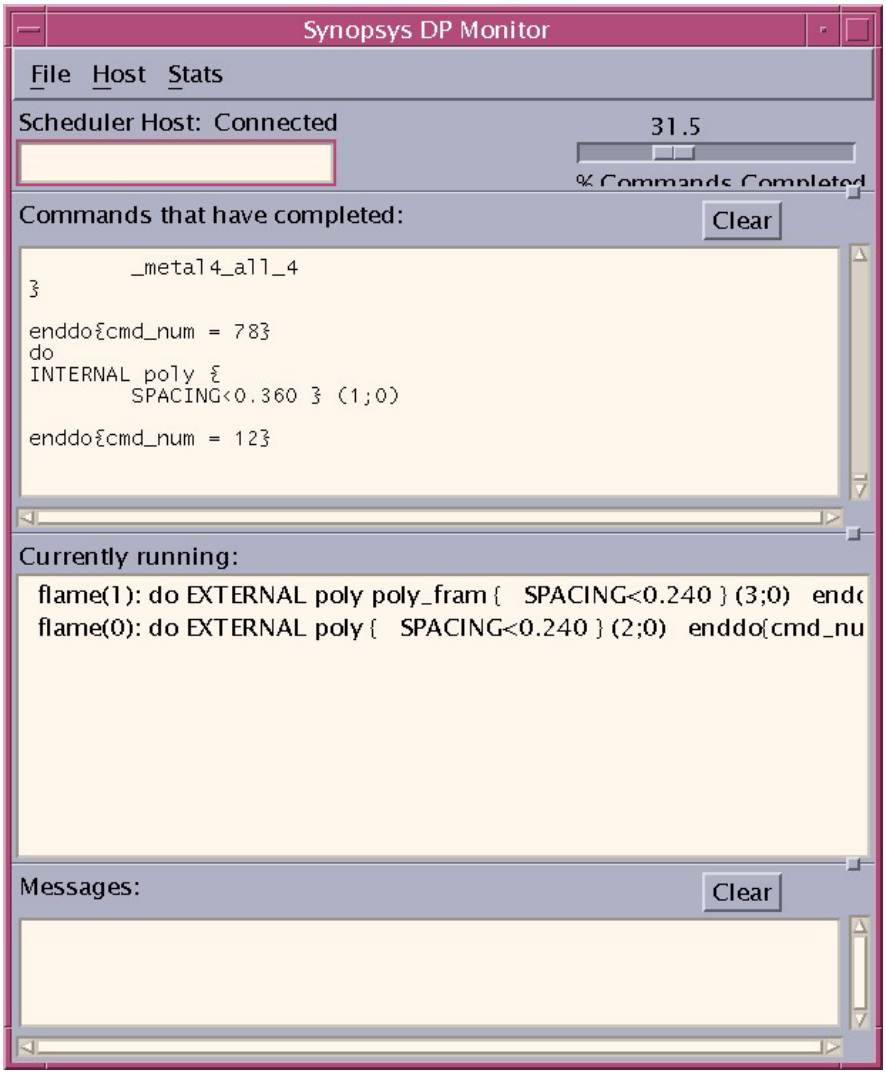
Table 5-3 describes the main windows of the dp_monitor.

*Table 5-3 DP_Monitor Windows*

| Window Title | Description |
|---|---|
| Commands That Have Completed | Completed commands in the order in which they were executed. Because commands are in parallel, they may not be executed in the same order as the runset. |
| Currently Running Host (#) | List of hosts executing the run and which command is currently running. The number in parentheses after the host name indicates the instance on the host. Double-clicking any of the host names opens a text window with summary output for that host. |
| Messages | Error and status messages from the monitor and the connected dp_master. |

There is a menu option for adding a host to the run through the monitor. The host name in the form is the only required field. The host must have a slave daemon running for the addition to be successful.

*Figure 5-1 DP_MONITOR*




## Adding CPU to a Distributed Run

When a Hercules run is executed using .dprc, you can add additional CPUs to the run by entering the following:

```
dp_addhost [-rd run_details_dir] [[host] [/cache/dir/path]]
```

You can access the command line description for this script by entering:

```
dp_addhost -h
```

Note:
dp_addhost cannot be run until after the dp_portmap and dp_slave processes have started. Also, dp_addhost may not be fully compatible with all queuing systems.

The following is an example of a command execution. This run is started with a single CPU and one additional host "host01" added to the run:

```
%>hercules rset.ev&
%>dp_addhost -rd . host01 /tmp/group &
```

If Hercules successfully adds the CPU, it returns the following message:

```
"ok"
```

Otherwise, an error message is returned.

Information on the added host is found in the run_details/<runsetname>.dp.log directory.

## Distributed Output

This section describes the files and directories created by a distributed run.

### Distributed Log File

The distributed log file name is the runset name with a dp.log extension. It is created in the run details directory. It contains more information than a summary file, including details about when a command is sent to a host, and the results produced when the host has completed the command. The distributed log file is in order of execution rather than in order of the runset. You can make this format the screen output by using the Hercules -df command line argument. See Table 5-1 on page 5-2 for a description of the argument.

A runset test.ev would generate a distributed log file test.dp.log

### Distributed Runset File

A distributed runset file is automatically generated in the current working directory from the standard runset file for a distributed run. This file has the runset name with a .evdp extension. It is deleted automatically after a successful run but remains if the run is terminated before completion. This file is overwritten and deleted by any following runs of the standard runset. It may also be deleted manually.

A runset test.ev would generate a distributed runset file test.evdp

### Distributed Statistics File

A distributed statistics file is created for a distributed run in the run details directory. The statistics file is the runset name with a dp.stats extension. For example, a runset named test.ev would generate a distributed statistics file named test.dp.stats. The following is an example distributed statistics file:

```
Task          Running      Idle         Total     Utilized
--------      --------    --------     --------     ----
machine(1)    00:01:01    00:01:44     00:02:46     37.2%
machine(0)    00:02:16    00:00:14     00:02:31     90.2%
totals        00:03:18    00:01:59     00:05:18     62.4%
```

- Task: The slave instance.
- Running: The time actively running.
- Idle: The time idle (waiting for a command, for example).
- Total: The combined running and idle time.
- Utilized: The percentage of the total time that the instance is running.
- totals: The combined values from each column. Each column that presents a time displays the combined times of all instances. The Utilized column displays the combined utilization percentage for all instances.

  Note:
  Time is displayed in hours:minutes:seconds.

### Distributed Work Directory

A distributed work directory is created for a distributed run in the run details directory. This directory is named dp_work. It contains subdirectories and files used during the distributed run. If this directory exists before a run, it and all of its contents are deleted and replacements are created. Because this directory is deleted before the run, it is important not to name any individual directories dp_work in the run details directory of a run.

## Subcommands

Distributed processing requires several internal subcommands that increase timing efficiency of a run between remote hosts. When a runset is executed on distributed hosts, these commands appear in the summary file.

These commands are inserted automatically by the distributed run as needed. They are internal to this process and are not intended for use in a regular runset. Table 5-4 lists the internal subcommands.

*Table 5-4 Internal Subcommands*

| Subcommand: | Helps to Facilitate |
|---|---|
| INIT_RUNSET | Group file creation |
| LOAD_LAYERS | Group file creation |

*Table 5-4 Internal Subcommands(Continued)*

| Subcommand: | Helps to Facilitate |
|---|---|
| SAVE_LAYERS | Group file creation |
| NO_OP | General timing problems |
| REMOVE | Smart delete |
| NET_CONNECT | Netlisting |
| TALLY | Summary file |
| STATS | Summary file |
| SHUTDOWN_RUNSETS | Turning off remote processes |

## Combining Multithreading and Distributed Processing

The Hercules engine supports multithreading as well as distributed processing. To get the most out of Hercules, it is important to understand both and how they can interact.

Distributed processing and multithreading are complementary. You can use distributed processing for servers across a network and also use multithreading to take advantage of multiple CPUs on each server. This allows you to get the most out of cost-efficient 2-CPU servers. You can utilize two 2-CPU servers over the network with two threads each, which can utilize all 4 CPUs on the two servers.

You can also run distributed processing and multithreading on a single multiple-CPU server. A 4-CPU server can be used effectively by combining a 2-CPU distributed run of two threads each. This will use all 4 CPUs (2 dp x 2 threads) with the advantages of parallelism at both the runset level and at the command level. This is an effective combination for high capacity servers with multiple CPUs.

This example combines a 2-CPU distributed run of two threads each.:

```
hercules -threads 2 -dp2 runset.ev
```

## Troubleshooting Distributed Processing

This section is provided to help troubleshoot problems encountered when setting up and running distributed Hercules.

*Problem:* Unable to register with the Synopsys Portmapper.

This error can occur when Hercules starts the dp_master, or when the dp_monitor is started.

Solution:

1. Make sure the environment variables, HERCULES_PMAP_HOST and HERCULES_PMAP_PORT are set.
2. Verify that the Portmapper program is running using the command pmap_info.

---

*Problem:* Cannot obtain licenses.

This error occurs when distributed Hercules attempts to obtain licenses.

*Solution:*

- Ensure that the LM_LICENSE_FILE environment variable is set. You must set it before starting dp_slave.
- Ensure that enough distributed Hercules licenses are available for a distributed run.

To run Distributed Processing, the formula to find out how many HERCULES-DP_MT licenses required is:

(DP-1) = # of HERCULES-DP_MT licenses required

For example, if you have 3 single-CPU servers and wish to run Distributed Processing across 3 CPUs:

(3-1) = 2 HERCULES-DP_MT licenses required

```
%>hercules -dp3 runset.ev
```

---

*Problem:* Directory access errors.

These errors are likely to occur when a host does not have access to a directory path in the runset or a directory path generated by Hercules. Directory access errors are most likely to occur when using local drives that have been temp mounted on other hosts.

*Solution:*

- Ensure that the drive has enough disk space.
- Ensure that the path of the run_details directory, the group directory, and your current working directory are all network paths.

---

*Problem:* Failure running Hercules.

This error occurs when a host cannot start Hercules for a distributed run.

*Solution:* Ensure that Hercules is in the PATH environment before starting the slave daemon (dp_slave).

---

*Problem:* Slave daemon stops working.

This error occurs for the slave daemon, which is run as a background task on the hosts for a distributed run.

*Solution:*

1. Ensure that the host running dp_slave has not crashed or been rebooted.
2. Ensure that the dp_slave executable has not been updated while dp_slave is running. This can occur when a new update is installed on your system.
3. If using an AFS network, ensure that the tokens have not expired. You will have to restart the slave daemons when tokens expire.

---

*Problem:* Hercules stops with multiple instances on a host.

This error occurs when multiple slave instances are on one host when running a memory-intensive runset.

*Solution:* Ensure that the host has enough memory to run multiple commands at the same time. The number of slave instances on one host indicates the number of commands the host tries to run in parallel. If too many commands are attempted at once, reduce the number of instances on the host.

---

*Problem:* Runset runs slower in distributed mode.

Some runsets might run slower in distributed mode. Distributed Hercules works best with runsets that contain large independent commands.

*Solution:*

- If the runset completes promptly in non-distributed mode (under five minutes), distributed mode offers little benefit. Execute the runset in non-distributed mode.

- If executing a nondistributed runset, in which each command uses a layer generated by the previous command, parallel execution will not occur. This type of runset forces the nondistributed operation. To use distributed processing on that type of runset, reduce the dependencies within the runset by separating the dimensional-checking commands from the connection-type commands.

---

*Problem:* Individual commands run slower in distributed mode.

There are multiple components to a distributed run. Running too many components on one host can impact the performance of individual Hercules commands during execution.

*Solution:*

1. Ensure that no other tasks are running on the hosts, specified in the .dprc file, before starting a distributed run.
2. If you are running multiple instances on an individual host, ensure that the host has multiple processors so that all of the instances can be run simultaneously.
3. Run Hercules with the -d option to turn off dots. The dots add more data to be passed across the network.

## Error Messages

The following tables list exit codes and error messages. The exit code tables contain exit code number, type, and explanation, and include general and license manager related exit codes. The error message tables contain the message ID, type, message text, and explanation. These tables include general, device extraction, parser/runset, file format, and operation system messages.

## Exit Codes

Table 5-5 lists the general exit codes. Note that Hercules reports an exit code if an error occurred in *block*.err only when the -ece option is specified on the command line.

*Table 5-5 General Exit Codes*

| Number | Type | Explanation |
|---|---|---|
| 0 | EXIT_COMPLETE | Run was completed |
| 30 | EXIT_COMPLETE_WERROR | Run was completed with errors in *block*.err file |
| 32 | EXIT_USER_INT | Interrupted by user with Ctrl-C |
| 33 | EXIT_USER_ERR | Error caused by user (parser error, file not found, ...) |
| 34 | EXIT_OS | Signal other than Ctrl-C, for example, division by zero, bus error, disk full, out of swap space...Could not create directory or file |
| 35 | EXIT_PANIC | An *impossible* situation occurred in an algorithm (badly designed algorithm) |

Table 5-6 lists the license manager related exit codes.

*Table 5-6 License Manager Related Exit Codes*

| Number | Type | Explanation |
|---|---|---|
| 64 | EXIT_LIC_TOO_MANY | Too many licenses checked out |
| 65 | EXIT_LIC_NOT_ALLOWED | Station not authorized to run this product |
| 66 | EXIT_LIC_NO_REPLY | Could not contact license server |
| 67 | EXIT_LIC_DENIED | License denied, no explanation given |
| (-1) | EXIT_NONE | An uninitialized exit status variable |

## Error Message Suppression

To suppress printing of any of the following messages, use the MESSAGE_SUPPRESS option in the runset OPTIONS section, and prefix the message ID number with ev- as follows:

```
message_suppress = { ev-ID, ev-ID, ... }
```

For further information on this option, see the *Hercules Reference Manual*, Detailed Options chapter.

The following tables describe the error messages displayed for various types of errors:

- Table 5-7 on page 5-26 lists general error messages
- Table 5-8 on page 5-32 lists device extraction error messages.
- Table 5-9 on page 5-34 lists parsing of runset/user error messages.
- Table 5-10 on page 5-38 lists operating system related warnings.

*Table 5-7 General Messages*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 101 | Warn | Area is not within set range. This AREA check is aborted. | AREA command RANGE Min value is greater than Max value. |
| 102 | Warn | Cannot have both boundary layer and window option defined. Negative image not created. | NEGATE has an INSIDE layer and a WINDOW defined in the same check. |
| 104 | Warn | The RANGE option can only be used with the CUTTING, ENCLOSING,TOUCHING, VERTEX and EDGE_TOUCH operators when using the SELECT command. | Same as Message Text. |
| 105 | Warn | A referenced cell was not found in any library. Check "%s.err" file. | There is a reference in a parent cell pointing to a child cell that is not found in any of the referenced libraries. |
| 106 | Warn | Cannot run DENSITY command with AREA or RATIO values less than or equal to zero. | Same as Message Text. |
| 107 | Warn | Cannot run DENSITY command with DELTA_WINDOW height value less than or equal to zero. | Same as Message Text. |

*Table 5-7 General Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 108 | Warn | Cannot run DENSITY command with DELTA_WINDOW width value less than or equal to zero. | Same as Message Text. |
| 110 | Warn | Command %s not completed due to invalid check value. | The referenced command's RANGE Min value may be greater than Max value. |
| 112 | Warn | Cannot read the hierarchy file HRCHY.ERROR from the remote directory %s. | The HRCHY.ERROR file under the group file directory that has been loaded with LOAD_GROUP is not accessible or does not exist. |
| 113 | Warn | VERBOSE option can only be used with the CLEANUP option. The CLEANUP option will be turned on. | Same as Message Text. |
| 114 | Fatal | The runset is empty. Nothing to execute. | No commands which write to TEMP, PERM, or the ERROR HIERARCHY were found in the runset. |
| 116 | Warn | LENGTH command only works on the output of a CUT command. | Same as Message Text. |
| 122 | Warn | Number of tiles exceeded MAX tiles of %d. | The design is too large to be divided into the maximum or fewer tile regions for processing by the DENSITY or NEGATE commands. Tiles are window areas across the chip in which DENSITY and NEGATE operate. |
| 123 | Warn | Attempted preprocess operation on %d cell(s), not found in Input Library. Check the summary file for a complete list. | Same as Message Text. |
| 124 | Warn | Expecting to EXPLODE cell "%s", not found in Input Library. | Same as Message Text. |
| 125 | Warn | Expecting to EXPLODE_ALL cell "%s", not found in Input Library. | Same as Message Text. |
| 126 | Warn | Expecting to FLATTEN cell "%s", not found in Input Library. | Same as Message Text. |

*Table 5-7 General Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 127 | Warn | Expecting to DELETE cell "%s", not found in Input Library. | Same as Message Text. |
| 128 | Warn | Attempted DELETE operation on %d cell(s), not found in Input Library. Check the summary file for a complete list. | Same as Message Text. |
| 129 | Warn | Attempted CHECK on %d cell(s), not found in Input Library. Check the summary file for a complete list. | This is related to message ID #105. A cell has been referenced which cannot be located in the reference libraries, therefore, the grid check cannot be performed. |
| 131 | Warn | Attempted to set noncat_text flag for Cell %s, which is not exploded. Flag will not be set for this cell. | The flag NONCAT_TEXT in the TEXT_OPTIONS section can only be set for cells which are exploded. |
| 132 | Warn | The resolution in the runset (%f) should not be less than the database units of the input library (%f). This runset resolution will have no effect. The database resolution will be used instead. | Same as Message Text. |
| 133 | Warn | The hierarchies to this cell level %s are mismatched. There is no correct hierarchy for the result. Performing hierarchical %s instead. | The result of a hierarchical NOT or SELECT has been used as an input to a command containing a cell_level operation. The hierarchy for the result of the cell_level operation could not be determined. The CELL_LEVEL option is automatically switched off so that the output can be generated. |
| 134 | Warn | Polygon_features equations created self_intersecting output polygon at (%.3f, %.3f). | Same as Message Text. |
| 136 | Warn | Border ignored for multi-polygon boundary layer. | The border option is not allowed when using NEGATE with the INSIDE option. |
| 137 | Warn | Cell "%s", for which text is defined in edtext file, not found in Input Library. | A structure name specified in the Edtext file does not exist. |

*Table 5-7 General Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 147 | Warn | Window coordinates incorrect. Negate exited. | The NEGATE command with WINDOW option must have lower-left coordinates specified first, then upper-right. |
| 148 | Warn | Self-intersection. Check "%s.err" file. | There exists a polygon in the design that self-intersects. |
| 149 | Fatal | Self-intersection. Aborting Hercules, please check "%s.err" file. | A self-intersecting polygon was detected in the database and PREPROCESS_OPTIONS{ abort_self_intersecting = true }. |
| 150 | Fatal | Hercules-Cell: Cannot process LOAD_GROUP command when running Hercules-Cell. Remove the LOAD_GROUP command from your runset and rerun. | Running Hercules-Cell with a LOAD_GROUP command in runset. |
| 151 | Fatal | Hercules-Cell: Block "%s" exceeds the Hercules-Cell size limit. Rerun Hercules without the -j Hercules-Cell option. | Running Hercules-Cell on too large a block. |
| 152 | Fatal | Hercules-Cell: Cell area of %f for this runset exceeds Hercules-Cell cell area limit of %f. | Same as Message Text. |
| 153 | Fatal | Grid violations found and SNAP=FALSE: cannot continue. | Same as Message Text. |
| 154 | Warn | Large negative border eliminates layer 1. | Negative BORDER option value larger than resultant data. |
| 155 | Warn | Number of output comments exceeds MAX comments of %d. No more output comments will be stored for error property attachment. | The limit for the number of comments to be attached as properties by the ERROR_PROPERTY{} command has been exceeded. |
| 156 | Warn | Overwriting layer '%s'. | The input layers and the prefix name for a LEVEL command is the same as for a previous LEVEL, causing the result of the previous LEVEL to be overwritten. |

*Table 5-7 General Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 157 | Warn | Oversize (%f) less than database resolution (%f). Oversize reset to %f. | OVERSIZE value for bounded size less than database resolution. |
| 158 | Warn | Oversize (%f) not a multiple of database resolution (%f). Oversize reset to %f. | OVERSIZE value for bounded size not multiple of database resolution. |
| 159 | Warn | Increment (%f) less than database resolution (%f). Increment reset to %f. | INCREMENT value for bounded size less than database resolution. |
| 160 | Warn | Increment (%f) not a multiple of database resolution (%f). Increment reset to %f. | INCREMENT value for bounded size not multiple of database resolution. |
| 161 | Warn | No layer1 polygons enclosed by layer2 polygons found. | There are no layer1 polygons inside layer2 polygons to be sized with the bounded size operation resolution. |
| 163 | Warn | Your copy of gdsin doesn't support the -af option. You should upgrade to the latest version. | Same as Message Text. |
| 164 | Warn | Your copy of gdsout doesn't support the -af option. You should upgrade to the latest version. | Same as Message Text. |
| 165 | Warn | Cell %s was exploded. No data were selected from this cell. | A SELECT_CELL has been attempted on a cell which has been exploded. If the data from this cell must be selected by cellname, this cell cannot be exploded. |
| 166 | Warn | Cell %s was deleted. No data were selected from this cell. | A SELECT_CELL has been attempted on a cell which has been deleted. If the data from this cell must be selected by cellname, this cell cannot be deleted. |
| 171 | Warn | No input layers specified. | GRID_CHECK command did not have any input layers specified. |
| 172 | Warn | WARNING. | An unexpected non-fatal condition has occurred. |
| 173 | Warn | Top block \"%s\" was deleted by DELETE command. | Same as Message Text. |

*Table 5-7 General Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 174 | Warn | Cell \"%s\", for which text is defined in DELTEXT file, not found in Input Library. | Same as Message Text. |
| 175 | Fatal | Invalid nested structure %s in input library. | The indicated cell is placed below itself in the hierarchy, which would result in an infinite loop upon traversing the hierarchy. |
| 176 | Warn | Cannot undersize undirected vectors - | This operation is not permitted. |
| 177 | Warn | Cannot edgesize vectors - | This operation is not permitted. |
| 178 | Fatal | Cannot mmap() %d bytes of file %s. | An attempt to use memory-mapped I/O on a file has failed. |
| 179 | Warn | Layer3 empty after %d iterations. | After %d iterations, the SIZE_OUTSIDE command has completely NOT'ed away all layer3 data. |
| 184 | Fatal | Hercules-Cell: Cannot process %s command when running Hercules-Cell. Remove the %s command from your runset and rerun. | Cannot run commands CHECK_POINT or RESTART with Hercules-Cell license. |
| 185 | Fatal | Tried to create a group with a preexisting layername <layername> in the destination filesystem. | May occur during a LOAD_GROUP command in which a layer being added already exists in the target filesystem. |
| 186 | Warn | Cannot both undersize and oversize vectors. | May occur during a SIZE command involving vectors. Vectors may not be both undersized and oversized at the same time. The input layer will be copied to the output layer when this warning occurs. |
| 869 | Warn | Did not find any %s text. | There is no text to find with a SELECT TEXTED WITH operation or the desired text was not found. |

*Table 5-8 Device Extraction Errors*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 300 | Warn | The %s devices have already been extracted. | Same as Message Text. |
| 301 | Warn | Command %s can't extract device "%s". | One or more of the layers needed for formation of the device may be missing. |
| 304 | Warn | Generic DEVICEs are not supported by SPICE. | Only capacitors, MOSFETS, diodes, bipolar transistors, and resistors can be written to a SPICE netlist. Generic devices can be defined, but will not be written to the netlist. |
| 307 | Warn | Three terminal MOSFETS may generate illegal SPICE netlist. | Some SPICE simulation programs require that all four terminals (including BULK connection) be specified for MOSFETs. Hercules allows three terminal devices (no BULK) to be written to the SPICE netlist. |
| 308 | Warn | Detected %d unrecognized devices. Check file "%s". | Same as Message Text. |
| 309 | Warn | Unknown substrate connection in CAP "%s". | Check to see that each res_layer specified in each runset command also has a corresponding CAPACITOR command from the res_layer to SUBSTRATE. If this capacitance is not desired then VC_CAP_VAL = 0 can be set. |
| 310 | Fatal | Each layer must also have a capacitance to substrate extracted. | Check to see if either layer1 or layer2 of each LATERAL CAPACITOR command is SUBSTRATE. This is not allowed. |
| 311 | Fatal | Cannot use SUBSTRATE as a lateral capacitance layer. | Check to see if resistors in layout contain any self-intersecting or doughnut shapes. Devices with these attributes will not be able to be properly extracted. |
| 312 | Warn | Resistor is cell \"%s\" at coordinates (%,3f, %,3f) (%,3f, %,3f) has bad geometry...discarding it. | Same as Message Text. |

*Table 5-8 Device Extraction Errors(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 313 | Warn | RC extraction found a floating polygon in layer "%s", cell \"%s\", coordinate (%f, %f). | Same as Message Text. |
| 315 | Warn | CAPACITOR command "%s", neither layer is a net layer. | Same as Message Text. |
| 317 | Warn | IO node name "%s" is too long, and will not be printed in the IO list. | Same as Message Text. |
| 318 | Warn | Layer "%s" completely eliminated by NOT operations. | Same as Message Text. |
| 319 | Warn | Group file associated with contacts for layer \"%s\" is empty | A virtual contact group file is empty. |
| 320 | Fatal | Unexpected use of layer SUBSTRATE. | The use of SUBSTRATE in this command is invalid. |
| 321 | Fatal | Too many %s INTERACT layers found. | Same as Message Text. |
| 322 | Warn | Missing terminal %d polygon at (%15.3f, %15.3f) in cell %s for device %s. | There are no terminals to connect to devices. |
| 323 | Warn | Undefined mathematical operation attempted in user equations. %s | Same as Message Text. |
| 324 | Warn | The '%s' parameter overflows a float; value set to zero | Same as Message Text. |
| 325 | Warn | Found LOAD/DRIVE layer polygon which is not part of a device. Location X = %.3f, Y = %.3f | SELECT_NET warning depicting a potential problem with device extraction commands. |
| 801 | Warn | Unable to connect %s. This device will be discarded. | Same as Message Text. |
| 802 | Warn | Unable to connect %s at coordinate (%f %f). This device will be discarded. | Low-level error codes. Same as Message Text. |
| 813 | Fatal | Cannot find %s type from runset in the global LPE list. | Low-level error codes. Same as Message Text. |

*Table 5-8 Device Extraction Errors(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 824 | Fatal | RC code unable to cope with data. | Low-level error codes. Same as Message Text. |
| 840 | Warn | Found a subtrap with NO virtual contacts. | Same as Message Text. |
| 855 | Warn | This may cause a connection problem. | Same as Message Text. |
| 861 | Warn | All angle data encountered in layer %s in a %s extraction may produce erroneous results. | Same as Message Text. |
| 862 | Fatal | Attempting to open too many layers in RC. | Same as Message Text. |
| 872 | Warn | Cannot perform resistance reduction on this polygon. | Same as Message Text. |
| 874 | Warn | Found a bad %s address in %s. | Same as Message Text. |
| 875 | Warn | %d %s from cell "%s" were not written to the SPICE netlist. | One or more of the layers needed for formation of the device may be missing. |

*Table 5-9 Parsing of Runset/User Errors*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 401 | Warn | Invalid option for %s command detected. Continuing. | Same as Message Text. |
| 402 | Warn | Non-printable characters detected in text string. Will replace text with string "bad_text". Check "%s.err" file. | Same as Message Text. |
| 403 | Warn | Cannot find an input layer in the layer list to perform CONNECT command. | Same as Message Text. |
| 404 | Fatal | No more than %d group files allowed. Aborting. | Exceeded maximum 256 group files for assigned layers. Aborting. |
| 405 | Warn | Need to specify %s for %s command. | Need to specify a range for LENGTH command. |

*Table 5-9 Parsing of Runset/User Errors(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 406 | Warn | Layer "%s" does not exist. | Layer "%s" does not exist or is empty. |
| 407 | Fatal | Unknown library path for library "%s". | Same as Message Text. |
| 408 | Fatal | Library "%s" does not exist. | Same as Message Text. |
| 409 | Fatal | License denied. | Same as Message Text. |
| 410 | Warn | Requesting (%s/%s/%s). | Same as Message Text. |
| 413 | Warn | No process layer assignments found. | Same as Message Text. |
| 416 | Warn | Only one window is allowed. Using only first window specified. | For NEGATE, only one window is allowed. Using only first window specified. |
| 417 | Warn | Structure "%s" does not exist for %s. | Same as Message Text. |
| 418 | Warn | Text string too long. Text string truncated. Check "%s.err" file. | Text string exceeded 512 characters. Text string truncated. Check "%s.err" file. |
| 419 | Warn | Unable to DELETE the top block "%s". | Same as Message Text. |
| 420 | Fatal | Top block "%s" does not exist. | Same as Message Text. |
| 421 | Warn | Unable to EXPLODE the top block "%s". Try using the FLATTEN option to flatten the top block. | Same as Message Text. |
| 422 | Warn | Two unique layers required for %s operation. Same layers "%s" specified. Continuing. | Same as Message Text. |
| 423 | Fatal | Unknown input format specified on the command line. Please specify either "gdsii" or "ltl" format. | Same as Message Text. |
| 424 | Fatal | Unknown output format specified on the command line. Please specify either "gdsii" or "ltl" format. | Same as Message Text. |
| 425 | Fatal | Parsing EXPLIST file "%s", %s near line %d with token : %s. Remove token from list and rerun. | Same as Message Text. |

*Table 5-9 Parsing of Runset/User Errors(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 426 | Warn | Text string all digits and net_prefix not defined in runset. Text string discarded. Check "%s.err" file. | If a text string in the layout (or Edtext file) is all digits and there is no NET_PREFIX specified in the OPTIONS section, the text string will be discarded to avoid a potential short between this text string and a net number. Because Hercules assigns numbers to untexted nets, it is possible that an all-numeric text string could be the same as the number assigned to another net, resulting in a short. Specifying a NET_PREFIX in the OPTIONS section, or changing the layout text, will resolve this problem. |
| 427 | Warn | Text string net_prefix plus all digits. Text string discarded. Check "%s.err" file. | If a NET_PREFIX is specified in the OPTIONS section and a text string in the layout (or Edtext file) is found that begins with the NET_PREFIX and is followed by only numeric characters, the text string will be discarded to avoid a text short. Because Hercules will assign untexted nets a name consisting of the NET_PREFIX and a number, any text found with this format could potentially cause a short. Specifying a different NET_PREFIX in the OPTIONS section, or changing the layout text, will resolve this problem. |
| 428 | Warn | Text string contains reserved character. String is discarded. Check "%s.err" file. | In addition to space and tab characters, the following characters are reserved and cannot be used in text strings: = { } , * " /.<br>If a text string in the layout is found containing one of these characters, the text string will be discarded. The TEXT_OPTIONS command, REPLACE_TEXT_CHARS, can be used to replace the illegal character. |

*Table 5-9 Parsing of Runset/User Errors(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 429 | Warn | Line %d: Array index exceeds array size. | Can occur if you specify an array index in a user-defined equation which exceeds the size that has been allocated for the array. User-defined equations are permitted with the POLYGON_FEATURES command as well as device extraction commands. |
| 430 | Warn | Uninitialized Geometric Parameter "%s". | Can occur because a parameter was incorrectly specified in an equations command. |
| 431 | Warn | Line: %d: TYPE MISMATCH! Incompatible types being evaluated in the expression. | Can occur if you have defined a run control variable to be of one type, such as double, but you assign the variable a value of a different type, such as a string. |
| 432 | Warn | Text string %s from OPTIONS section contains reserved character "-%s". Replace reserved character and rerun. | Occurs if a text string specified from a command in the OPTIONS section, such as NET_PREFIX, for example, contains one of the reserved characters, = { } , * " or /. You must replace the reserved character and rerun. |
| 433 | Warn | Control character detected in text string. Text string discarded. Check "%s.err" file. | Same as Message Text. |
| 434 | Warn | Structure "%s" does not exist. | Same as Message Text. |
| 436 | Warn | Size (%f) less than database resolution (%f). Reset to %f. | Same as Message Text. |
| 437 | Fatal | Vector with Polygon operation is not supported by %s . | This operation is not permitted. |
| 438 | Fatal | Multiple SELF_INTERSECT commands are not allowed. | Same as Message Text. |
| 439 | Fatal | Instance name cannot contain slashes, please change your data. Bad instance name \"%s\". | Same as Message Text. |

*Table 5-10 Operating System Related Warnings/Fatal Error Messages*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 701 | Fatal | Cannot change to directory "%s". Aborting. | Same as Message Text. |
| 702 | Fatal | Cannot create directory "%s". Aborting. | Same as Message Text. |
| 703 | Fatal | Cannot write to disk, errno %d. Aborting. | Same as Message Text. |
| 704 | Warn | Cannot open %s file "%s" for reading. Continuing. | Same as Message Text. |
| 706 | Warn | Cannot open %s file "%s" for writing. Continuing. | Same as Message Text. |
| 707 | Warn | Cannot create %s file for %s. Continuing. | Same as Message Text. |
| 708 | Fatal | Cannot open %s file "%s" for reading. Aborting. | Same as Message Text. |
| 709 | Fatal | Cannot open %s file "%s" for reading and writing. Aborting. | Same as Message Text. |
| 710 | Fatal | Cannot open %s file "%s" for writing. Aborting. | Same as Message Text. |
| 711 | Fatal | The program GDSIN which converts files from GDS-II to LTL format could not be found or was aborted. Make sure you have a copy of it in your path, and contact Synopsys if you cannot find one. Check the file "%s/%s.gdsin.log". Aborting. | Same as Message Text. |
| 712 | Fatal | The program GDSOUT which converts files from LTL to GDS-II format could not be found or was aborted. Make sure you have a copy of it in your path, and contact Synopsys if you cannot find one. Check the file "%s/%s.gdsout.log". Aborting. | Same as Message Text. |

*Table 5-10 Operating System Related Warnings/Fatal Error Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 713 | Fatal | Out of swap space, malloc failed to allocate %d bytes. Increase virtual memory and try again. (Temporary files have been removed.) Aborting. | Same as Message Text. |
| 714 | Warn | Unable to print netlist file. Continuing. | Can occur if the database has not yet been connected (using the CONNECT command), or if the netlist file could not be opened for writing (such as, no disk space or wrong file permissions). |
| 715 | Fatal | Could not rename "%s" to "%s". Aborting. | Same as Message Text. |
| 716 | Warn | Could not delete file "%s". Continuing. | Same as Message Text. |
| 717 | Fatal | Bus error. Aborting. | Same as Message Text. |
| 718 | Fatal | Insufficient swap space. Aborting. | Same as Message Text. |
| 719 | Fatal | Floating point exception. Aborting. | Same as Message Text. |
| 720 | Fatal | User interrupt. Aborting. | Same as Message Text. |
| 721 | Fatal | Broken pipe. Aborting. | Same as Message Text. |
| 722 | Fatal | Segmentation violation. Aborting. | Same as Message Text. |
| 723 | Fatal | SIGTRAP. Aborting. | Same as Message Text. |
| 724 | Fatal | Unknown signal happened. Aborting. | Same as Message Text. |
| 725 | Fatal | Cannot open %s file "%s" for writing. Either the limit for open file descriptors is reached or the group file directory is not writable. Aborting. | Same as Message Text. |
| 726 | Fatal | I/O error when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 727 | Fatal | Permission denied when opening file %s in mode %s. Aborting. | Same as Message Text. |

*Table 5-10 Operating System Related Warnings/Fatal Error Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 728 | Fatal | Attempt to open directory as a file when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 729 | Fatal | UNIX file table overflow occurred when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 730 | Fatal | Too many open files when opening file %s in mode %s. Aborting. | Usually a c-shell limitation that you can change by typing "unlimit descriptors." |
| 731 | Fatal | Device filled up when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 732 | Fatal | UNIX reported too many links when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 733 | Fatal | UNIX thought the filename %s in mode %s was too long. Aborting. | Same as Message Text. |
| 734 | Fatal | Disk quota exceeded (when opening file %s in mode %s.) Aborting. | Same as Message Text. |
| 735 | Fatal | Sun NFS reported stale NFS handle when opening file %s in mode %s. Aborting. | Same as Message Text. |
| 736 | Fatal | System call failed. Please rerun Hercules. | Same as Message Text. |
| 737 | Fatal | Max file name length exceeded. | Same as Message Text. |
| 738 | Fatal | Could not open a pipe to process %s. | Same as Message Text. |
| 739 | Fatal | System call failed. Rerun Hercules. | Same as Message Text. |
| 740 | Fatal | UNIX could not service the call: system(%s). | Same as Message Text. |
| 741 | Fatal | The runset requires %i open files. Only %i are available. Try typing "unlimit descriptors" and re-running. | Same as Message Text. |
| 742 | Warn | %s is not a directory. Will not use as a group file directory. | Same as Message Text. |

*Table 5-10 Operating System Related Warnings/Fatal Error Messages(Continued)*

| ID | Type | Message Text | Explanation |
|---|---|---|---|
| 743 | Warn | Don't have permissions to access %s. Will not use as a group file directory. | Same as Message Text. |
| 744 | Fatal | None of the group directories were usable for storing group files. | Same as Message Text. |
| 745 | Warn | Could not create %s. Will not use as a group file directory. | Same as Message Text. |
| 746 | Fatal | Could not find out the number of KBytes free for group dir %s. Aborting. | The group file directory specified with HEADER{GROUP_DIR =} either does not exist or Hercules does not have permission to create it. |
| 747 | Warn | Bad filename <%s> passed in to open, mode %s. | Created internally to the code when an empty string is passed to open( ), and a file is to be opened using the string as the name. |
| 748 | Fatal | The directory <%s> is not createable or accessible. The first group directory listed must be accessible. Subsequent directories don’t have to be. | The first group file directory path does not exist. |
| 749 | Warn | Cannot create the group directory "%s"... | The path to an additional group file directory does not exist. |
| 750 | Warn | Change output directory "%s" to a valid directory in the runset. | The path to the output library does not exist. |
| 753 | Fatal | Cannot open %s directory \"%s\" for writing. Aborting. | Same as Message Text. |

# 6

# Hierarchical Verification Concepts and Processing

*This chapter explains the principles of hierarchical IC design. It describes how hierarchy influences the verification process in order to show you how to design chips with an efficient hierarchy that allows Hercules to demonstrate its full potential. The sections describe how Hercules optimizes hierarchy during the verification process, how the optimization process might be modified (and why), and tips on troubleshooting hierarchy-related problems: A discussion of how Hercules reports on a design's hierarchy and uses it to facilitate error reporting and debugging is presented in Hercules General Usage Information, Chapter 7.*

## Principles of Hierarchical Design

A good hierarchical design can improve runtime and memory usage during your Hercules run. The three principles of good hierarchical design are:

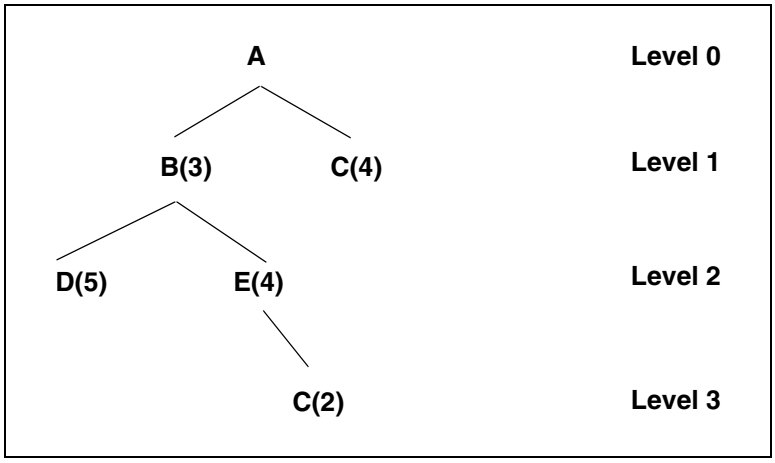
- Data patterns that occur throughout the design should be combined inside other cells, and these cells should be placed throughout the design.
- Keep as many checks at cell level as you can by making sure each cell contains a reasonable amount of data, avoiding excessive cell overlap.
- Minimize the number of cells that are placed only once in the design.

The following sections examine these three principles in detail in order to help you determine what makes a good hierarchical design and how to identify the hierarchy of current designs.

## Hierarchical Design

While Hercules is fully able to verify a flat design, it has been designed to take advantage of any hierarchy in the design to improve run-times and memory utilization. A hierarchical design is one in which cells containing different design elements are nested inside other cells. An inverter might be part of an adder, for instance, with the adder being part of a larger circuit element. Figure 6-1 shows a typical hierarchical structure. Cell A, at the top level (Level 0) of the design, contains three instances (or placements) of cell B and four instances of cell C. Cell B in turn contains instances of cells D and E, and so on.

*Figure 6-1 Example of a Vertical Design Hierarchy*




The different cells in a hierarchical design are said to have a parent-child relationship with each other. A *parent* cell is one that contains instances of one or more lower-level cells; each of the lower-level cells is considered a *child* of its parent. In Figure 6-1, cell A is the parent of cells B and C, while cell C is a child of both cells A and E. Since cells B and C have the same parent, they are considered *siblings*. Notice how one cell (B) can be both a child of one cell (A) and the parent of others (D and E), and how different placements of a cell (C) can have different parents (A and E).

The number of *hierarchical* cell placements in a design is the number of placements of a given cell at different levels of the design. In Figure 6-1, there are six hierarchical placements of cell C; four at Level 1 and two at Level 3. The number of *flat* placements is the number of placements of that cell you would actually see on the finished chip. For example, each instance of cell E contains two placements of cell C. There are four instances of E in each placement of B, and three instances of B in A. Four other instances of C are also placed directly in A. The number of flat placements of C is therefore (((2 x 4) x 3) + 4) = 28.

The ratio of flat placements to hierarchical placements in a design is a rough measure of the degree to which a design contains vertical hierarchy. A high ratio indicates that the design has vertical hierarchy, while a low ratio indicates a horizontal design. Different types of designs also have varying degrees of inherent hierarchy. Memory designs typically have a deep vertical hierarchy, for example, while ASIC designs are more likely to have a horizontal hierarchy.

## Hierarchical Verification

When a hierarchical design is verified, there are two aspects of the design that must be considered. The first is how the data within a cell interacts with other data in the cell, and the second is how it interacts with data from other cells. The former are considered cell-level interactions while the latter are considered hierarchical interactions. Generally, the more data that can be verified at cell level, the better Hercules will perform.

Consider, for example, the following check:

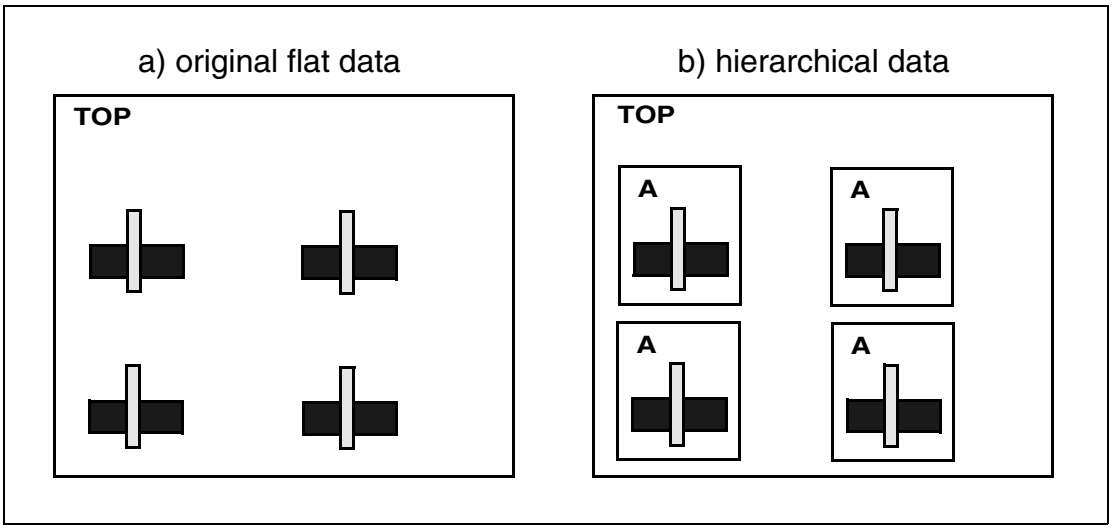
```
ENCLOSE CONT BY DIFF { SPACING < 2 } (99)
```

Hercules first tries to verify that this rule is satisfied using only data within the cell currently being processed. If it can do this, Hercules stops processing this data and goes on to the next set of data. If it cannot, Hercules flags the data to indicate that it has not yet been verified and then moves on to the next set of data. Later, when Hercules is considering hierarchical interactions in the design, this data will have to be re-examined. To put it another way, data that can be checked at the cell level only has to be processed once for each applicable check, while data that has to be examined hierarchically must be checked twice. The impact of this on Hercules tool runtimes is obvious.

Hercules looks at the hierarchical interactions in the data after it has completed its cell-level checks. At this time it completes any checks it was not able to resolve at the cell level. Hercules might not have been able to verify the ENCLOSE rule above at the cell level, for example, because the relevant data is in different, overlapping cells. After completing these hierarchical checks, any data that has not been shown to satisfy the relevant check is reported as a violation. Hercules reports the error in the lowest-level structure that contains the data involved in that particular check.

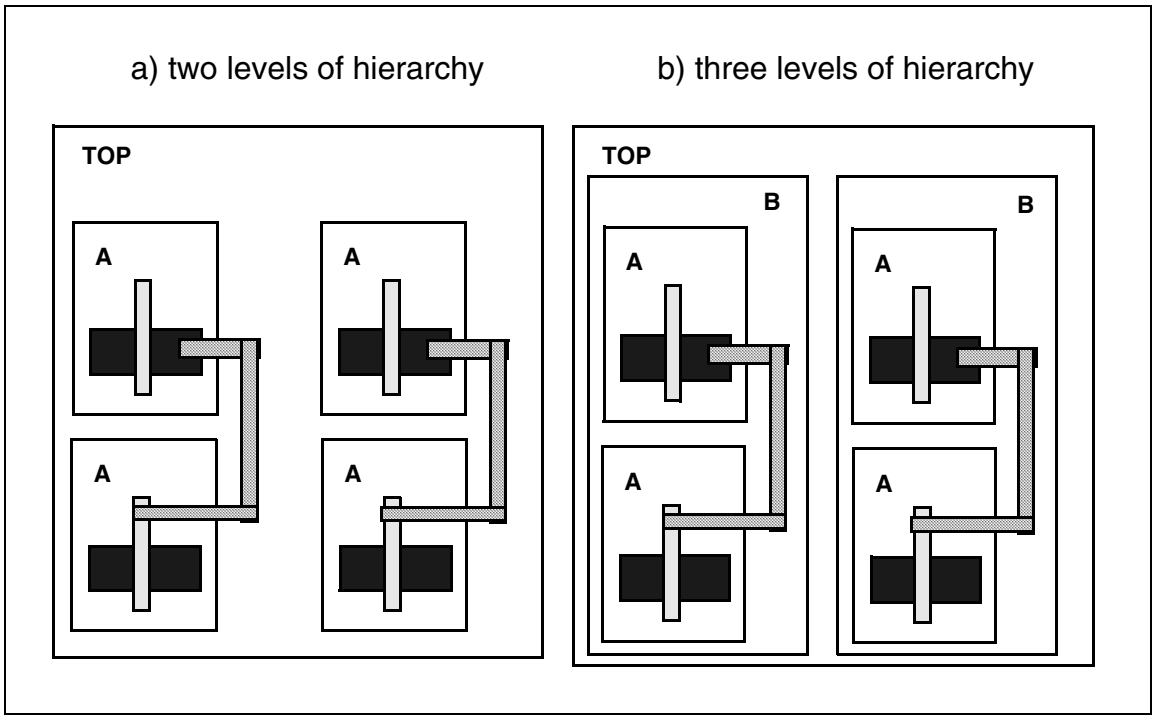
The primary way Hercules uses hierarchy is to take advantage of any repeating data so that a pattern that occurs many times in a design only has to be checked once. This principle is illustrated in Figure 6-2. In Figure 6-2(a), the same two-rectangle pattern occurs four times. In Figure 6-2(b), that pattern is drawn only once in cell A, but cell A is placed in the design four times. The two designs are equivalent, but Hercules has to make four checks in the first case (a) compared to only one in the second (b).

*Figure 6-2 Combining Repeating Data Into Multiple Placements of a Single Cell*



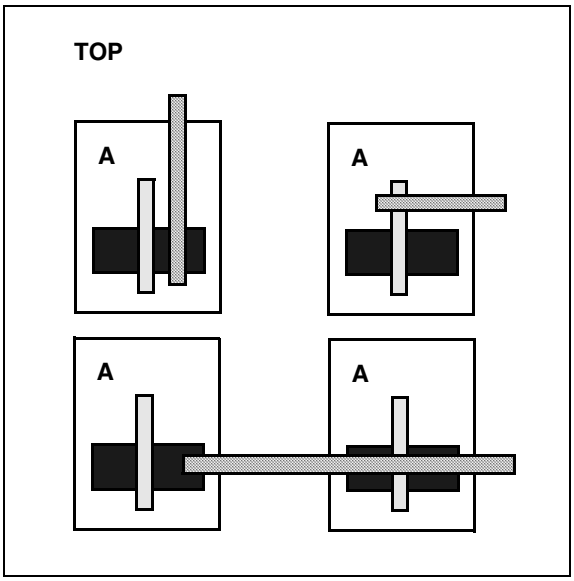

The benefits of encapsulating data in this manner continue as you move up through the hierarchy. Figure 6-3(a) shows the design in Figure 6-2(b) along with some routing material in the top cell. There are four unique interactions between the routing material and the contents of A that Hercules needs to consider, but two of these are duplicates of the other two. Figure 6-3(b) shows how these duplicate checks can be eliminated by combining two instances of A with their associated routing material to form a new cell, B. As before, the number of individual checks that must be made has been reduced, and Hercules will run much faster and require much less memory.

*Figure 6-3 Nesting Cells at Multiple Levels of Hierarchy*




It may not always be possible to take advantage of a design's hierarchy. Figure 6-4 shows a case where each instance of A interacts in a different way with the top-level routing material. There is no way to combine any of this data in another cell to reduce the number of checks Hercules must make, as was done in Figure 6-3(b). The result is a design that must be checked as if it was a flat design, and is therefore known as "hierarchy that results in flat processing."

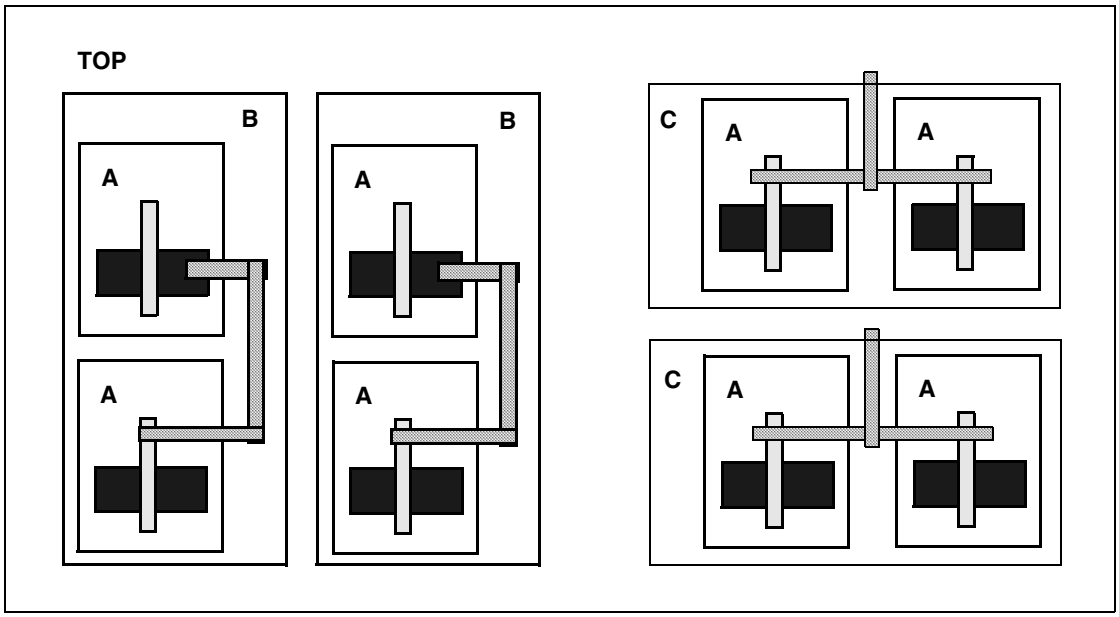
*Figure 6-4 Hierarchy that Results in Flat Processing*




## Efficient vs. Inefficient Hierarchy

The first principle of good hierarchical design has already been presented: data patterns that occur throughout the design should be combined inside other cells, and these cells should be placed throughout the design, as demonstrated in Figure 6-2. Cells at lower levels should also be considered as potential data for inclusion in cells higher in the hierarchy, as shown in Figure 6-3. Remember that different instances of a cell can have different parents, as shown in Figure 6-5.

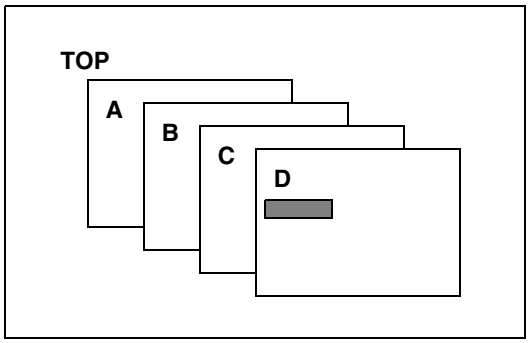
*Figure 6-5 Using a Single Child Cell in Different Branches of the Hierarchy*




The second principle of good hierarchical design is to keep as many checks at cell level as you can by making sure each cell contains a reasonable amount of data. This way Hercules is able to take advantage of the cell's multiple placements to minimize the amount of unique data it needs to check. If the cell contains very little data, Hercules spends most of its time checking interactions between cells. Since each cell interacts with (potentially) many other cells, the number of checks increases significantly and Hercules tool performance is reduced.

A good hierarchical design also avoids excessive cell overlap. Some overlap is inevitable, particularly with certain designs (such as gate arrays), but too much overlap causes an excessive number of hierarchical interactions. In Figure 6-6, for example, the polygon shown in cell D could interact with data from as many as four other cells.

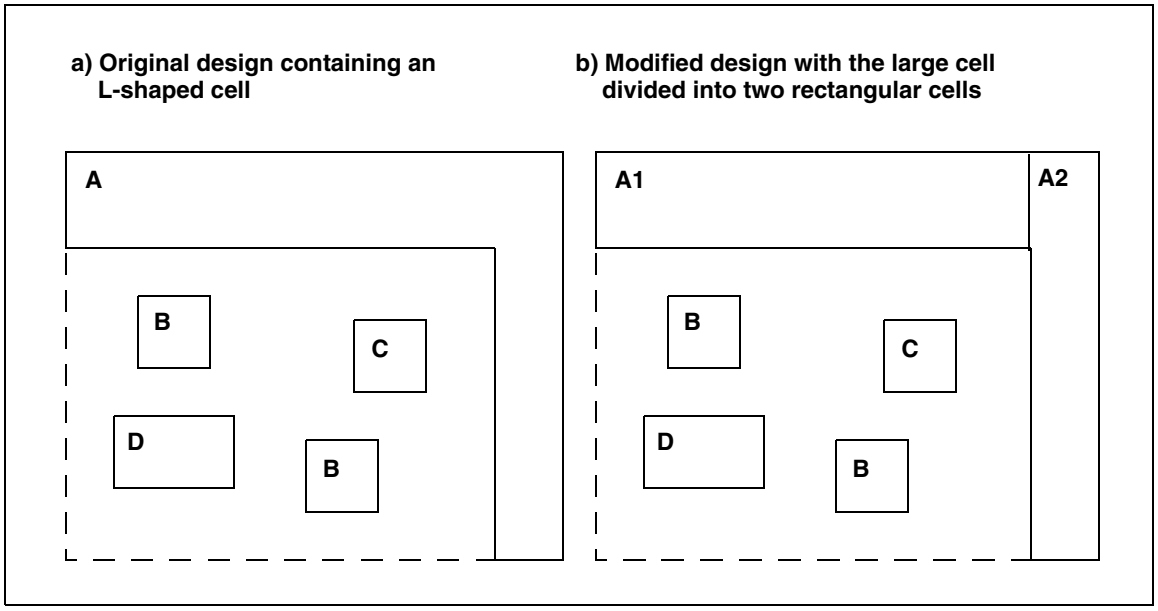
*Figure 6-6 Overlapping Cells that Cause Poor Performance*




The third principle of good hierarchical design is to minimize the number of cells that are only placed once in the design. The data in these cells is still going to be checked once regardless of whether it is in the top cell or a lower-level cell; however, placing it in the top cell allows Hercules to use special memory-management routines that are not used with lower-level cells. This principle can be relaxed when you have a cell containing a large functional block, such as a central processor or a memory block, where the large block makes more sense from a design standpoint.

One situation that can affect Hercules is the use of non-rectangular cells. Figure 6-7 displays several small cells nested in the crook of a large L-shaped cell. These cells do not interact with A, but Hercules treats them as if they do. This is because the interactions are determined not by A's boundaries (shown by the solid lines), but by its extents, or the edges of the smallest rectangle that completely contains the cell (shown by the dashed line). Hercules does not have any problems handling this situation, but its performance will improve if A is divided into two regular rectangular cells. It would still need to consider interactions between these two cells, but it would no longer need to check for interactions between either of them and the remaining cells.

*Figure 6-7 Hierarchical Interactions Resulting from Cell Geometry*




## How Hercules Processes Hierarchy

Whenever Hercules verifies a design, it tries to optimize the hierarchy of that design to achieve better performance. This optimization takes place during the preprocessing phase of the program's execution, and its behavior is controlled by the TECHNOLOGY_OPTIONS in the Hercules runset. The default values of these options have been set to give the best performance for the widest range of designs, and most users will not need to change them.

The verification process does not actually change the hierarchy of the input database. Hercules instead creates an internal copy of the database, and it is this copy that is modified by the optimization process.

Hercules tool actions during preprocessing can be examined in three places. In the *block*.sum file, you can see a summary of what Hercules has done and how long it took to do it. See Example 6-1. The *block*.tech file lists the TECHNOLOGY_OPTIONS settings in effect during the run, and the *block*.treeN file lists the design's hierarchy at different stages of the run. Information on these files is provided in the *Hercules Reference Manual*, Key Concepts chapter.

*Example 6-1 Excerpt from the Preprocessing Section of a block.sum File*

```
Preprocessing group files...
Warning #190: Adding default TECHNOLOGY_OPTIONS.
You can look in the
```

```
drcc.tech file to see what options were used.
     Preprocess Step 1 : Exploding
     Duplicate placements found, check "drcc.err" file.
       BLACK_BOX_TO_NO_EXPLODE changed 18 cells to NO_EXPLODE
       SCHEMATIC_TO_NO_EXPLODE changed 265 cells to NO_EXPLODE
       Preprocessing time = 0:03:28  User=195.71 Sys=9.47 Mem=242.006

     TECHNOLOGY_OPTIONS {
             VCELL_PASS   {
                     MIN_CELL_OVERLAP = 60
                     RATIO=10000.000
                     MIN_COUNT = 5000
                     MIN_LEAF = 40
                     STYLE = EXPLODE
             }
     }
     Preprocess Step 2 : Vcell_pass Explode Iteration 1
        Explode time = 0:00:03  User=3.02 Sys=0.00 Mem=131.057

     VCELL_PASS 1, no changes.

     TECHNOLOGY_OPTIONS {
             VCELL_PASS   {
                     EXPLODE_INTO_VCELL = TRUE
                     MIN_CELL_OVERLAP = 60
                     RATIO=10000.000
                     NO_CORNER_PAIRING = FALSE
                     STYLE = SETS
                     ITERATE_MAX = 2
             }
     }
     Preprocess Step 3 : Vcell_pass Sets Iteration 1
        Sets time = 0:00:01  User=1.15 Sys=0.00 Mem=131.041

     VCELL_PASS 1, no changes.

     TECHNOLOGY_OPTIONS {
             VCELL_PASS   {
                     EXPLODE_INTO_VCELL = FALSE
                     MIN_COUNT = 20
                     STYLE = PAIRS
                     ARRAY_ID = TRUE
                     ITERATE_MAX = 15
                     TOP_PERCENT_OF_VALUE = 40
             }
     }
     Preprocess Step 4 : Vcell_pass Arrays and Pairs Iteration 1
        Pairs time = 0:00:16  User=15.70 Sys=0.00 Mem=139.134

     Preprocess Step 5 : Vcell_pass Arrays and Pairs Iteration 2
        Pairs time = 0:00:16  User=16.05 Sys=0.00 Mem=140.134

     . . .
```

```
Preprocess Step 17 : Vcell_pass Arrays and Pairs Iteration 14
   Pairs time = 0:00:10  User=10.68 Sys=0.00 Mem=145.041

Preprocess Step 18 : Vcell_pass Arrays and Pairs Iteration 15
   Pairs time = 0:00:13  User=12.10 Sys=0.00 Mem=145.056

   Combined VCELL time = 0:03:37  User=210.70 Sys=2.16 Mem=240.991

Preprocess Step 19 : Post-VCell Explodes
   Post VCell time = 0:00:01  User=0.63 Sys=0.00 Mem=116.979
```

The following paragraphs describe the steps Hercules goes through during preprocessing in detail. Each section begins with an example listing some of the options that control how that step behaves and the messages generated in the *block*.sum file during that step. This is followed by a discussion of what Hercules is doing during that operation and how it affects the design's hierarchy. The discussion may comment on particular options, but it is not intended to provide an in-depth description of them. To learn more about a particular option, refer to the *Hercules Reference Manual*, Detailed Options chapter.

## General Explodes

*Example 6-2 General Explode Options and the Corresponding Entry in the block.sum File*

```
TECHNOLOGY_OPTIONS {
        SUBLEAF_AUTO_EXPLODE = 6
        EXPLODE_CELL_SIZE_PERCENT = 70
        CELL_SIZE_AUTO_EXPLODE <= 10
        EXPLODE_AREFS = FALSE
        EXPLODE_1XN_AREFS = TRUE
        EXPLODE_DATA_CELL_LIMIT = 4
        EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
        EXPLODE_BIG_SPARSE_CELL = TRUE
        EXPLODE_HOLDING_CELL_LIMIT = 1
        EXPLODE_PLACEMENT_LIMIT = 1

OPTIONS {
        EQUIV_TO_NO_EXPLODE=TRUE
        BLACK_BOX_TO_NO_EXPLODE=TRUE
}
. . .
Preprocess Step 1 : Exploding
Duplicate placements found, check "drcc.err" file.
  BLACK_BOX_TO_NO_EXPLODE changed 18 cells to NO_EXPLODE
  SCHEMATIC_TO_NO_EXPLODE changed 265 cells to NO_EXPLODE
  Preprocessing time = 0:03:28  User=195.71 Sys=9.47 Mem=242.006
```

The first thing Hercules does during preprocessing is remove inefficient hierarchy from the design. Inefficient hierarchy includes things like empty cells, duplicate placements of cells, cells that contain very little data, and cells that are only placed in the design a few times. In the case of empty cells, the hierarchy is improved by removing the cell from the design. In

other cases, the data in the cell is exploded into that cell's parent. This happens to cells with low data counts and big, sparse cells (large cells that contain very little data). As shown in Example 6-2, TECHNOLOGY_OPTIONS that control this process usually include the word *explode* in their name.

The general explode pass does not necessarily explode every cell that is eligible to be exploded according to the criteria specified by the appropriate TECHNOLOGY_OPTIONS. Cells that have been identified as potential equivalence points during an LVS run, for example, are rarely exploded. Cells that look like they might be base cells in gate array designs are also rarely exploded. You can control whether or not specific cells are exploded using one of the NO_EXPLODE options in the EXPLODE_OPTIONS section of the runset. The SELECT_CELL command can also cause cells to not be exploded if the OPTIONS section option, SELECT_CELL_TO_NO_EXPLODE, has been set. For additional information on these options, see the *Hercules Reference Manual*, Detailed Options chapter.

Hercules looks for duplicate cell placement throughout preprocessing, with the first check made during the general explode pass. The check is cell-level by default, but you can force it to be hierarchical by using the HIER_REMOVE_DUP_PLACEMENTS option. The former is faster, but hierarchical operation is more thorough. Duplicate placements can be removed only if both duplicate cell placements are under the same parent; or if a parent cell of a duplicate placement is placed only once throughout the design. Neither is guaranteed to find all duplicate cells, but those that remain will have a negligible effect on Hercules tool operation.

## VCELL Explodes

*Example 6-3 VCELL Explode Options and the Corresponding Entry in the block.sum File*

```
TECHNOLOGY_OPTIONS {
        VCELL_PASS   {
                MIN_CELL_OVERLAP = 60
                RATIO=10000.000
                MIN_COUNT = 5000
                MIN_LEAF = 40
                STYLE = EXPLODE
        }
}
. . .
Preprocess Step 2 : Vcell_pass Explode Iteration 1
   Explode time = 0:00:03  User=3.02 Sys=0.00 Mem=131.057

VCELL_PASS 1, no changes.
```

The second stage of preprocessing is known as the VCELL explode pass. The explodes that occur during this pass are intended to bring selected cells to the same level of hierarchy to facilitate later VCELL formation. (VCELLs and VCELL formation are described in the following sections of this user guide.) This is necessary because VCELL formation is a cell-level operation that requires the cells involved to have the same parent.

This requirement is shown most clearly by considering the typical hierarchy seen in gate array designs. Figure 6-8(a) illustrates that these designs frequently place the programming cells in one holding cell and the base cells in a different holding cell. Hercules is unable to combine the two types of cells into VCELLs because they do not have the same parent. However, when they are exploded into TOP, as shown in Figure 6-8(b), this changes. The two types of cells now have a common parent and can be successfully combined into VCELLs.

*Figure 6-8 VCELL Explode Pass: First Example*

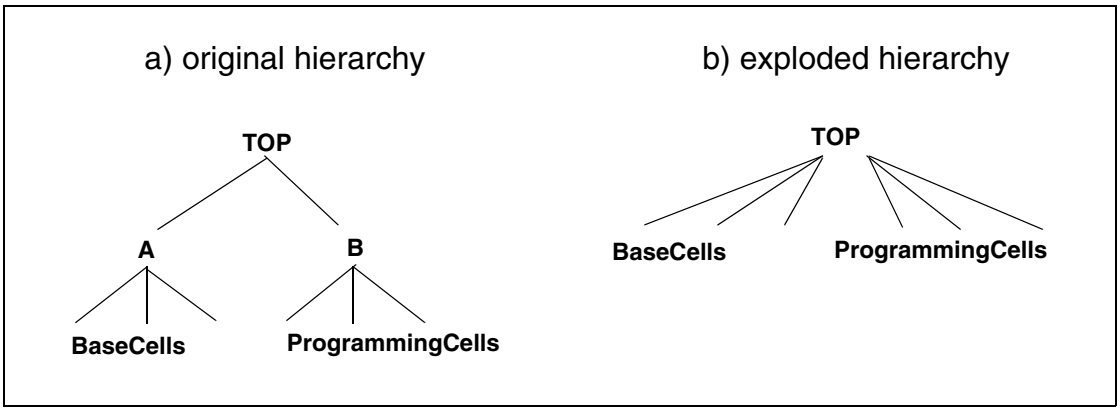
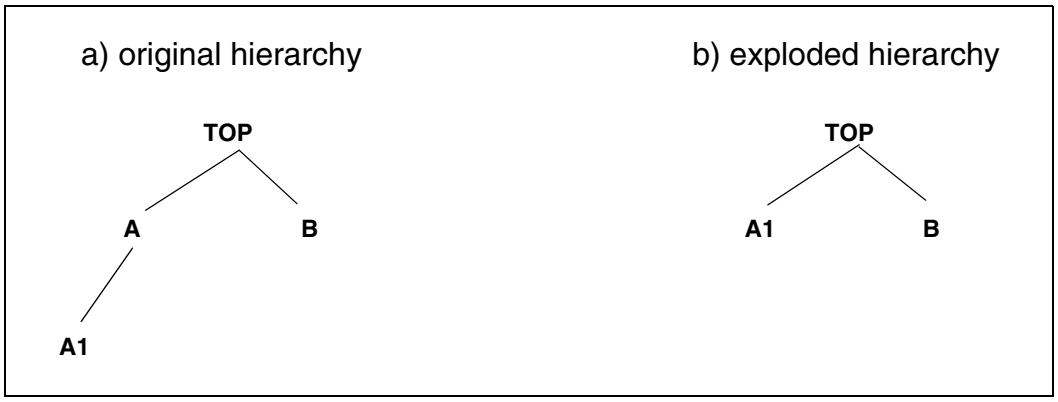



Figure 6-9 presents a different situation. In this case, cells A1 and B are often adjacent to each other in the design, so they are excellent candidates for VCELL creation. They cannot be combined, however, because they are at different hierarchical levels. The VCELL explode pass recognizes the fact that A1 and B form a likely VCELL pair, and explodes A into its parent in order to make A1 and B siblings. As before, the cells now have a common parent and can be combined to form a new VCELL.

*Figure 6-9 VCELL Explode Pass: Second Example*




## VCELL Sets Pass

*Example 6-4 VCELL Sets Pass Options and the Corresponding Entry in the block.sum File*

```
TECHNOLOGY_OPTIONS {
        VCELL_PASS   {
                EXPLODE_INTO_VCELL = TRUE
                MIN_CELL_OVERLAP = 60
                RATIO=10000.000
                NO_CORNER_PAIRING = FALSE
                STYLE = SETS
                ITERATE_MAX = 2
        }
}
Preprocess Step 3 : Vcell_pass Sets Iteration 1
   Sets time = 0:00:01  User=1.15 Sys=0.00 Mem=131.041

VCELL_PASS 1, no changes.
```

The VCELL sets pass is designed to combine base cells and programming cells in gate array designs, and it is only required when Hercules is verifying these designs. If your design is not a gate array, you may turn this pass off using the Hercules command-line option, -noSets. If your design is a gate array, the VCELL sets pass is required for two reasons:

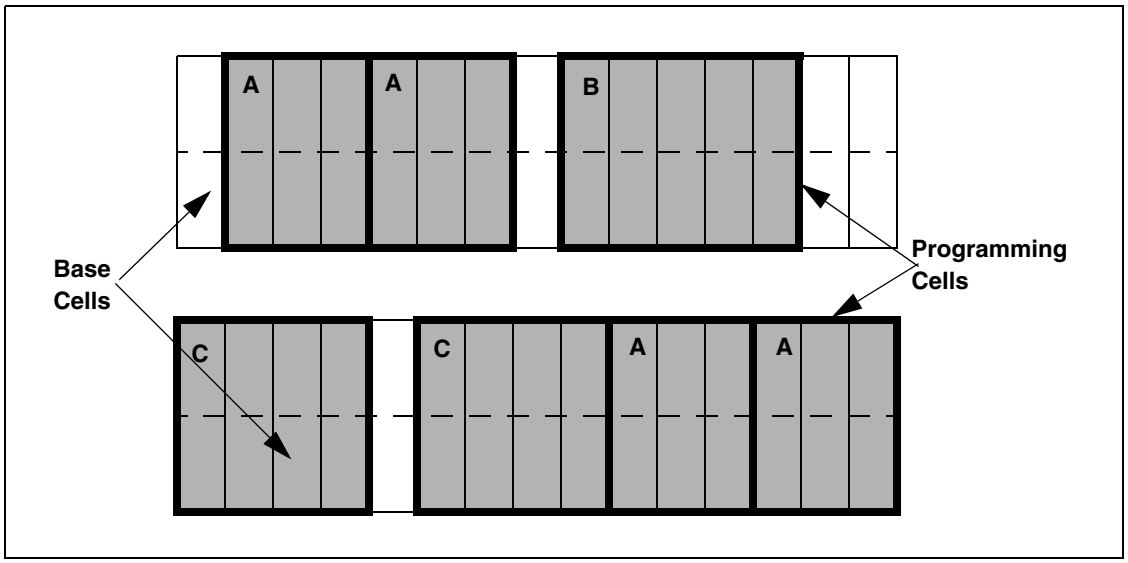
Gate array designs, by definition, involve different cells interacting flatly across the entire hierarchy. Combining the cells changes these to cell-level interactions and improves Hercules tool performance.

LVS runs cannot successfully compare the layout to the schematic unless the base and programming cells have been combined. If they have not been combined, Hercules sets equivalence points based on the programming cells, but these do not contain any primitive devices (MOSFETs, for example) because part of the data needed to define the device is only present in the base cells.

VCELLs created during the sets pass are based on the programming cells, and combine the data in the programming cell with the data in the base cell with which it interacts. This process is illustrated in Figure 6-10, which shows two rows of base cells interacting with three different types of programming cells. Programming cell A interacts with three base cells to form one VCELL SET placed four times. (This assumes the relative orientation of the base and programming cells remains unchanged. If the base cells are reflected but the programming cells are not, you would have two different VCELL sets, each of which would be placed in the design. See also the discussion related to Figure 6-14.) Programming cell B interacts with five base cells to form a VCELL SET that is placed only once, and programming cell C interacts with four base cells to form a VCELL SET that is placed twice.

The set names are derived from the name of the programming cell by adding the suffix _VCnn. A programming cell named INV, for example, might generate a set named INV_VC1. The sets are ordered based on the number of base cells they include, and smaller sets take priority over larger sets. By default, base cells are not shared between different sets; the set having priority will get the base cell. However, the SHARE_ BASE_CELLS option is available to allow different sets to share a common base cell.

*Figure 6-10 Gate Array Designs - Base Cells are Drawn as Unshaded Rectangles with Thin Edges; Programming Cells are Drawn as Large Shaded Shapes with Thick Edges*



## VCELL Pairs Pass

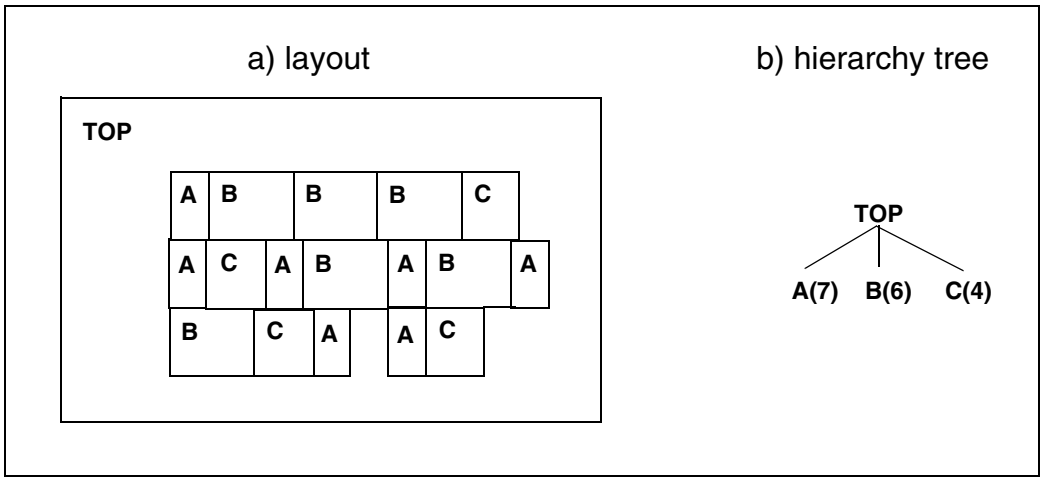
*Example 6-5 VCELL Pairs Pass Options and the Corresponding Entry in the block.sum File*

```
TECHNOLOGY_OPTIONS {
        VCELL_PASS   {
                EXPLODE_INTO_VCELL = FALSE
                MIN_COUNT = 20
                STYLE = PAIRS
                ARRAY_ID = TRUE
                ITERATE_MAX = 15
                TOP_PERCENT_OF_VALUE = 40
        }
}
Preprocess Step 4 : Vcell_pass Arrays and Pairs Iteration 1
   Pairs time = 0:00:16  User=15.70 Sys=0.00 Mem=139.134

Preprocess Step 5 : Vcell_pass Arrays and Pairs Iteration 2
   Pairs time = 0:00:16  User=16.05 Sys=0.00 Mem=140.134

. . .

Preprocess Step 18: Vcell_pass Arrays and Pairs Iteration 15
   Pairs time = 0:00:13  User=12.18 Sys=0.00 Mem=141.073
```

VCELL pairs are created wherever repeating cell patterns occur in the design and are independent of the nature of the cells (unlike the sets pass, which is intended for use with gate arrays). The sets pass takes place before the pairs pass, however, to ensure that programming and base cells are properly matched. The purpose of pairs formation is to add efficient hierarchy to the design. This is accomplished by encapsulating the data to increase the number of checks that can be carried out at the cell level and reduce the number of hierarchical checks that have to be performed (see the discussion on the principles of efficient hierarchical design in “Efficient vs. Inefficient Hierarchy” on page 6-6).

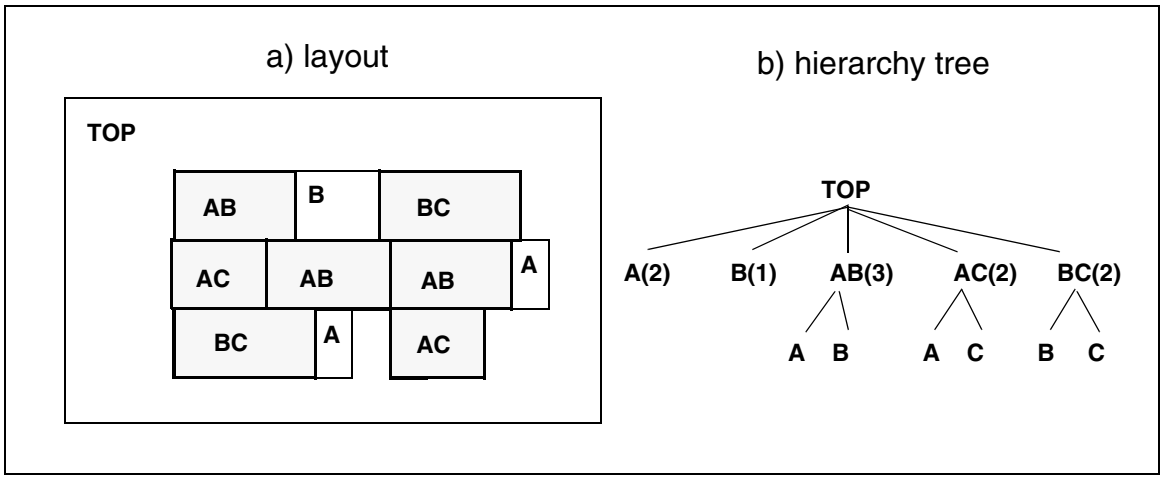
To understand the concept of a VCELL PAIR, consider Figure 6-16. This figure shows a design that contains seven placements of cell A, six placements of cell B, and four placements of cell C. The design’s hierarchy is inefficient, however, because it does not take advantage of any repeating cell patterns. There are three places, for example, where an instance of A is immediately adjacent to an instance of B. This is inefficient because the same interactions between these two cells would have to be checked three times.

*Figure 6-11 A Typical Design Showing Cell Placements Only*



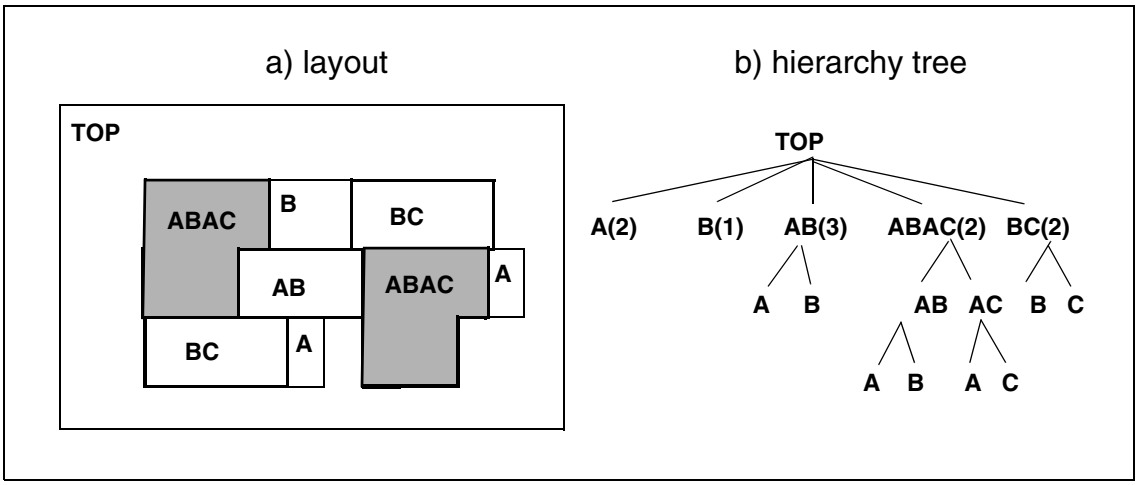

To improve the design, Hercules creates a new virtual cell, a VCELL PAIR, out of any cell pairings that occur multiple times in the design. Figure 6-12 shows the design in Figure 6-11after the first pairs pass. Three new cells have been created, and the design now contains three levels of hierarchy.

*Figure 6-12 Design After Forming the First Set of VCELLs*



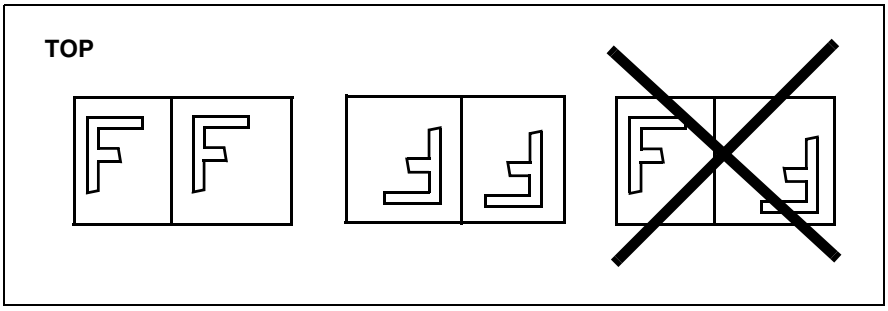

Once a VCELL has been created it is treated just like a regular cell. This applies both to the VCELL pairs currently being discussed, and to the VCELL sets that were described above. Figure 6-13, for example, shows the design in Figure 6-12 after a second pairs pass has been executed.

*Figure 6-13 Design After Forming the Second Iteration of VCELL Pairing*




In order to be considered for inclusion in a VCELL, cells must touch or overlap, and if they overlap they must do so by the same amount. In addition, they must have the same relative orientation. In Figure 6-14, the first two cell pairs can form a VCELL because the two cells have the same orientation *relative* to each other. The second pair is simply rotated 180 degrees with respect to the first, and the resulting VCELL will be rotated by the same amount. The third pair cannot be part of the same VCELL, however, because there is no way to transform these two cells into the same relative orientation seen in the other two pairs.

*Figure 6-14 Effect of Orientation on VCELL Formation*

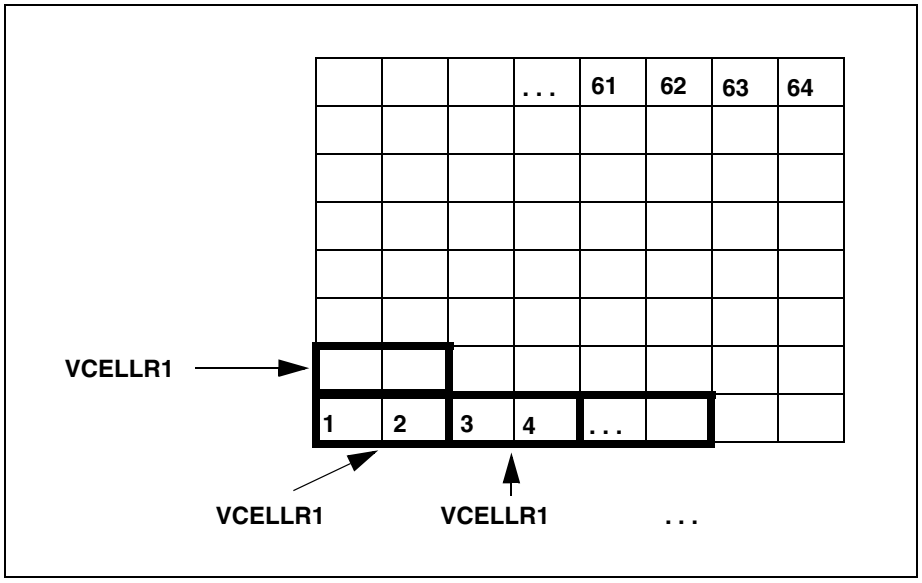



Pairs creation is carried out in Hercules in two stages:

- STAGE 1: In the first stage, Hercules looks for cells in arrays (AREFs) that can be combined to form pairs. This allows the tool to create the pairs more efficiently because it can process the array systematically. Figure 6-15, for example, shows how Hercules begins looking for pairs in the lower left corner of an array (CELL 1) and proceeds to create pairs in a horizontal direction (indicated by the heavy lines). When it finishes

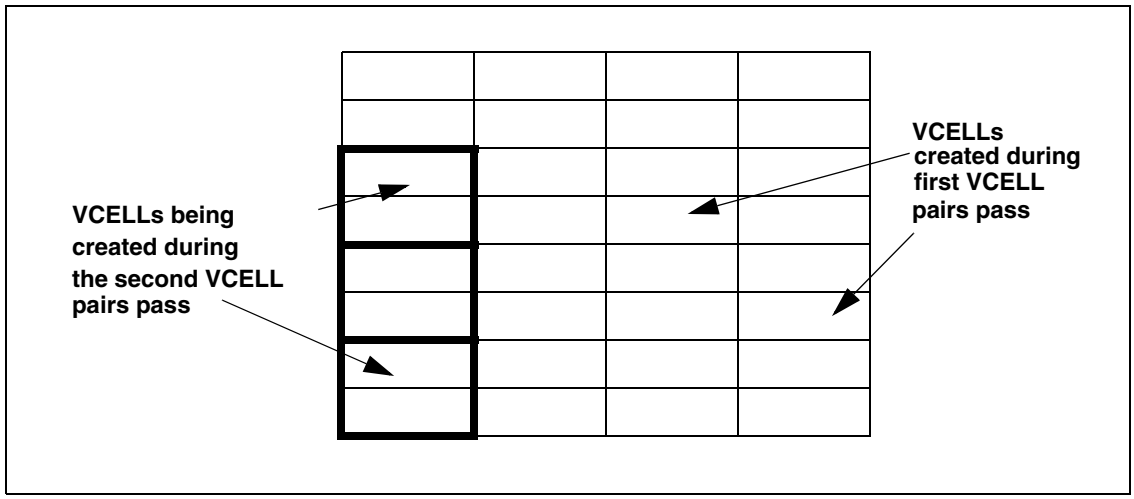
looking at one row, it moves up to the next and continues the process. (Because the cells being combined are identical, there will only be one VCELL created, but it will be placed in the array thirty-two times.)

*Figure 6-15 An 8x8 Array During the First VCELL Pairs Pass*




In the next pass, Hercules again starts at the lower left corner of the array. This time, though, it goes through the array by columns, finishing one before moving on to the next. This process is illustrated in Figure 6-16. A third iteration would again proceed horizontally, a fourth vertically, and so on, until no more VCELLs can be formed or the maximum number of iterations has been reached.

*Figure 6-16 An 8x8 Array During the Second VCELL Pairs Pass*




- STAGE 2: Once all array pairs have been created, the second stage of pairs processing looks for random pairs in the design (that is, isolated pairs that are not part of an array). This process was discussed earlier.

## Post VCELL Explodes

*Example 6-6 Post-VCELL Explode Options and the Corresponding Entry in the block.sum File*

```
TECHNOLOGY_OPTIONS {
        POST_VCELL_EXPLODE_CELL_SIZE <= 10
        POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
}
. . .
Preprocess Step 19 : Post-VCell Explodes
  Post VCell time = 0:00:01  User=0.63 Sys=0.00 Mem=116.979
```

After is finished creating VCELLs, it goes through a second, post-VCELL explode pass. This pass is similar to the original explode pass described in “General Explodes” on page 6-11. Because the tool does not have to worry about protecting potential sets or pairs, it can be more aggressive about how it carries out these explodes.

## Summary Listing of Cells Created by Hercules Preprocessing

In the preceding discussion, it was pointed out that some of the Hercules preprocessing operations (most notably the VCELL sets and pairs passes) result in the formation of new cells. The names Hercules assigns to these cells will tell you how they were created, and, except for VCELL pairs, they will also tell you the parent cell from which they were derived.

The tables below list the different types of cells created by Hercules during the different preprocess operations. Detailed information about how the cells are created and their nomenclature can be found in the *Hercules Reference Manual*, Detailed Options chapter.

Note:
MYCELL Refers to any Input Layout Cell Name)

*Table 6-1 Cells Created by Hercules During Preprocessing: SETS (VCELL sets pass)*

| Cell | Definition |
|---|---|
| MYCELL_VC1 | Based on name of the programming cell |

*Table 6-2 Cells Created by Hercules During Preprocessing: PAIRS (VCELL pairs pass)*

| Cell | Definition |
|---|---|
| VCELLRn | Array type |
| VCELLAn | General pairing type |

*Table 6-3 Cells Created by Hercules During Preprocessing: Prototyped Array*

| Cell | Definition |
|---|---|
| MYCELL_LB1 | Left bottom corner |
| MYCELL_L2 | Left edge |
| MYCELL_LT3 | Left top corner |
| MYCELL_T4 | Top edge |
| MYCELL_RT5 | Right top corner |
| MYCELL_R6 | Right edge |

*Table 6-3 Cells Created by Hercules During Preprocessing: Prototyped Array(Continued)*

| Cell | Definition |
|---|---|
| MYCELL_RB7 | Right bottom corner |
| MYCELL_B8 ----- | Bottom edge |
| MYCELL_C9 ----- | Center |

*Table 6-4 Cells Created by Hercules During Preprocessing: Prototyped Placements*

| Cell | Definition |
|---|---|
| MYCELL | 0 rotation, no reflection |
| MYCELL_Q2 | 90-degree rotation, no reflection |
| MYCELL_Q3 | 180-degree rotation, no reflection |
| MYCELL_Q4 | 270-degree rotation, no reflection |
| MYCELL_R1 | 0 rotation, reflected |
| MYCELL_R2 | 90-degree rotation, reflected |
| MYCELL_R3 | 180-degree rotation, reflected |
| MYCELL_R4 | 270-degree rotation, reflected |
| MYCELL_45D_0R | 45-degree rotation, no reflection |
| MYCELL_135D_1R | 135-degree rotation, reflected |

*Table 6-5 Cells Created by Hercules During Preprocessing: Array Trees*

| Cell | Definition |
|---|---|
| MYCELL_ATn | |

# Debugging Problems Related to Hierarchy

There may be times when Hercules performs poorly due to problems with the design's hierarchy. The following sections describe some of the more common problems you might encounter, and offer guidelines on how to resolve those problems. This is not always a simple process because it is not always possible to relate a specific performance issue to a particular technology option. It is recommended that when you are debugging these problems, you keep the output generated from one run to serve as a benchmark against which all other runs are evaluated. This way you can quickly judge which of your changes have a significant impact and which are negligible.

## Excessive Memory Use During Group File Creation and Sorting

*Problem:* Group file creation and sorting requires an excessive amount of memory, disk space and run time. Other commands in the runset also seem to require an excessive amount of memory.

*Solution:* These symptoms often indicate that too much data is being flattened into the top cell, or worse, into a lower-level cell. If only one or two commands require large amounts of memory, though, the problem probably lies elsewhere. The tree files and the *block*.tech file are the most useful files for debugging this type of problem. In addition, set the following option in your runset.

```
PREPROCESS_OPTIONS {
   CELL_PROFILE    = TRUE
}
```

This causes Hercules to print out the following statistics in each tree file:

```
- TOP 20 CELLS WITH LOWEST DATA COUNT
- TOP 20 CELLS WITH HIGHEST DATA COUNT
- TOP 20 CELLS WITH LOWEST FLAT PLACEMENT COUNT
- TOP 20 CELLS WITH HIGHEST FLAT PLACEMENT COUNT
```

Run Hercules and examine the tree files looking for cells whose data count increases drastically from one tree file to the next. See Chapter 7, "General Results and Analysis," for information on how to read a tree file. The tree file in which this increase occurs will give you an indication of which preprocess step is causing the problem. If it shows up in *block*.tree1, look for general explode options that might be causing too much data to be exploded. If it shows up in the last tree file, look for problems with the post-VCELL explode options. In either case, both the tree files and the *block*.tech file will list which options caused a particular cell to explode. You can determine which ones are having the largest impact and remove them from the runset to see if that solves your problem.

When going through this analysis, consider the number of times the cell is placed in the design. A cell that has only one placement is going to generate the same amount of data whether it is exploded or not. This might occur, for example, if routing cells are placed under the top cell.

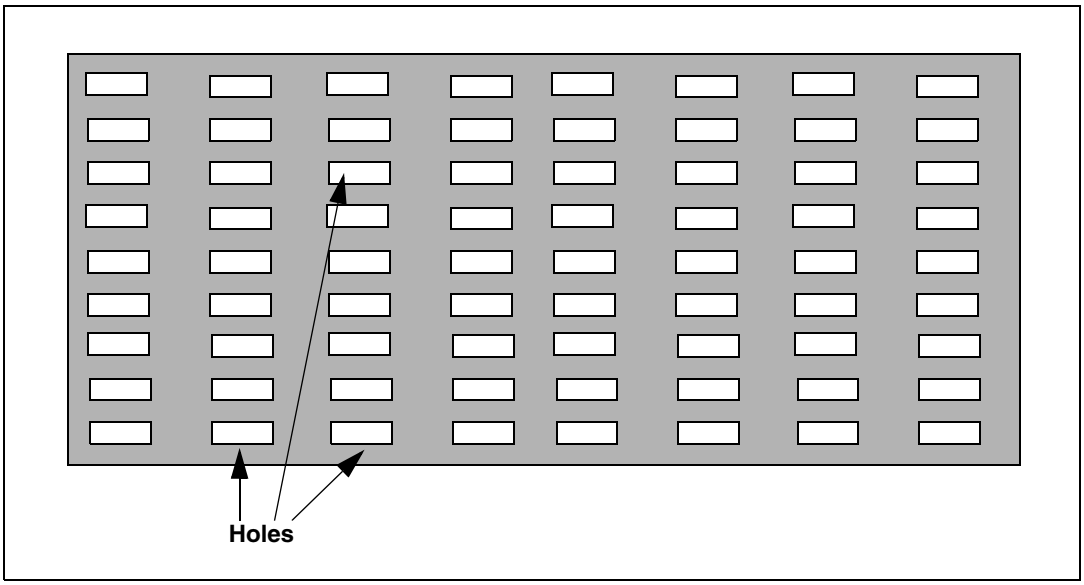
Also remember that the top cell will usually contain more data than lower-level cells. Hercules recognizes this and includes several special memory control algorithms that operate only on this cell. However, you may run into a problem if you use the SELECT_CELL feature. Data that would ordinarily be exploded into the top cell can be exploded by this command into a lower-level cell, where memory control features do not operate.

## Group File Creation Uses Too Much Memory

*Problem:* You are trying to debug the problem described in the previous section, but you are unable to identify any cells that are generating an excessive data count.

*Solution:* If you are unable to identify a hierarchical cause of the behavior described above, you need to consider other possible problems. One situation that can cause this problem is the inadvertent formation of large waffle-shaped polygons during group file creation. Waffle-shaped polygons are complex polygons that contain many holes and therefore have many coordinates that Hercules needs to monitor. An example is shown in Figure 6-17. These are formed when interacting polygons on the same layer are merged into a single layer during the group file sorting process. Normally, this process reduces the amount of data Hercules has to keep in memory, but in this case it increases the number of data points. The problem can occur when there is an error in the ASSIGN section of your runset and layers are inadvertently merged. However, other causes are possible. To solve this problem, look for situations where this is occurring and identify the cause.

*Figure 6-17 A Typical Waffle-Shaped Polygon*



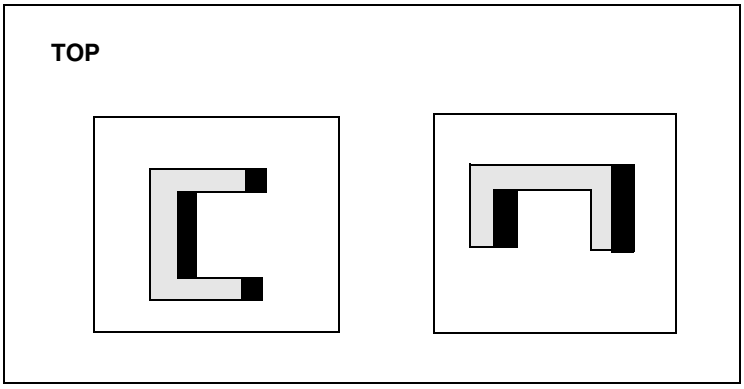

## Excessive Runtimes

*Problem:* Runtimes seem excessive, particularly for selected data creation commands. The problem seems particularly acute for commands that involve the resizing (growing or shrinking) of polygons.

*Solution:* This problem frequently occurs whenever you have a directional sizing command. A GROW command that only grows the polygon to the right while leaving the other three sides unchanged, for example, could easily demonstrate this behavior. The problem with this is that the sizing direction in the command is listed relative to the design as a whole, but the command implements the operation at the cell level.

To see how this works, look at Figure 6-18. The cell on the right is the same as the one on the left except that it has been rotated clockwise by 90 degrees. The original data in the cell is the C-shaped polygon shaded gray, and the black represents the material that would be added by a GROW command directed to the right. The only way Hercules can perform this operation and still keep the polygons in the lower-level cells is if there are two separate cells. To do this, Hercules turns on the PROTOTYPE_PLACEMENTS option, which essentially creates a new cell for each rotation and/or reflection of the existing cell. This can greatly increase the amount of data Hercules must process, and can therefore significantly increase runtimes for the relevant commands. see the *Hercules Reference Manual*, Detailed Options chapter, for more information on prototype placements and prototype cells.

*Figure 6-18 Directional Commands and Prototype Placements*




## Memory Pool Dump During Preprocessing

*Problem:* Hercules runs out of memory during preprocessing causing a memory dump to occur, and the top pool using the excessive memory is reported in the error message as *hlrc->hrchy_pool*.

*Solution:* This problem usually occurs when too many placements are being flattened, particularly when the design contains arrays of arrays and the intermediate arrays are being exploded. Hercules keeps track of both existing and exploded references, and this process significantly increases the number of references it has to follow. To resolve the problem, look in the tree files and the *block*.tech file to see which options are causing the arrays to be exploded. The most likely ones are the general explode options and the VCELL_EXPLODE options. You can then either turn those options off, or you can explicitly set the exploding cells not to explode using the NO_EXPLODE option.

## VCELL Passes Run Very Slowly

*Problem:* The VCELL explode pass runs very slowly or does not complete.

*Solution* When the VCELL explode pass appears to hang, it is usually caused by vias or other cells with a large number of placements having a high degree of individual hierarchical interaction not getting exploded. Hercules tries to recognize via cells and automatically explode them, but it may not recognize non-standard cells or cells containing multiple vias. Check the tree1 file for any names that look like vias and, if they are vias, explode them using the appropriate EXPLODE_OPTION. Also look for cells having a very high flat placement count that interact with other placements. If they do not overlap with other

placements, and are not in an array that overlaps with other placements, they are probably not a problem. If they do overlap with other placements, it may help to explode or flatten these cells into their parents.

# 7

# General Results and Analysis

*This chapter provides an overview of Hercules Design Rule Checking (DRC) and Layout versus Schematic (LVS) output files.*

## Description of Output Files

Hercules produces many output files. Some of the output files are specific to the type of run Hercules performs. If you are performing a Design Rule Checking (DRC) run, Hercules produces error and summary files. The error files list the DRC errors in your design. The summary files list information about your design and the completed run.

If you are performing a device extraction, Hercules produces error, summary, and layout netlist files. The layout netlist file is the file that is compared to a schematic netlist during LVS comparison. Both DRC and device extraction runs produce .tree, .vcell, and .tech files in the run_details directory.

If you are performing an LVS comparison, Hercules produces error, log, and equivalence summary files. The error file lists which nets and devices from the layout do not match the schematic.

The error, summary, and layout netlist files are written to the directory from which Hercules is executed. The detailed LVS equivalence summary files are written to the directory specified by the HEADER section COMPARE_DIR variable. The LVS log file is located in run_details directory.

All output files are named *block.abbreviation*, where *block* is the name specified by the HEADER section BLOCK variable, or the -b option of the command line. The extension following *block* is an abbreviation for the explanation of the file. For example, the Account file name is *block*.acct.

Only those files needed to debug the design are written directly into your working directory. These files are: *block*.RESULTS, *block*.LAYOUT_ERRORS, *block*.LVS_ERRORS, and *block*.waive. All other output files are placed inside a new directory, called run_details, created within the working directory.

## Account File

Hercules creates an Account file during netlist extraction. *block*.acct lists the number of devices and their types extracted from each block in the hierarchy. The cumulative count shown is the total count of all devices contained in that structure.

## Summary File

Hercules creates a Summary file during DRC or netlist extraction runs. *block*.sum contains a transcript of the information that was printed to the screen during the run.

## Results File

Hercules creates a Results file that provides the results for each step of the run. *block*.RESULTS reports DONE (if the step completed) or ERROR (if the step did not complete). If the step did not complete, the Results file refers you to a file that offers more details.

The Results file also reports errors in the layout and/or compare portion of the run. These errors are prefixed with pound signs (###) so that you can easily spot them. These are errors in your design that you must fix, rather than errors indicating an abnormal termination of the run. The Results file refers you to the appropriate file to debug the design.

The last line of the file reports "Hercules is done" if the run is completed, and "Hercules did not complete" if the run did not complete.

The Results file is the starting point for analyzing the results of a Hercules run.

## Error File

Hercules creates an Error file during DRC or netlist extraction runs. These types of errors are referred to as layout errors. *block*.LAYOUT_ERRORS contains the results of a run, a summary list of the errors, and a more detailed list of the errors.

The top of the file states either CLEAN or ERRORS so that you can quickly determine if there are layout errors in the design.

The next section is the summary section. This section provides the comment for a command, the command itself, and the number of errors for a command with errors. The comments are taken from the COMMENT field of each command. It will be reported as No Comment if no comment was provided for the command.

The last section is the details section. Each error report contains the following: an error type; an error description that identifies the check being performed; a structure name in which the error occurred; a position or a bounding box value where the error occurred; and other information specific to a check. The position coordinates are reported hierarchically (relative to the named structure and only once for each referenced structure). These errors can also be viewed graphically using the GUI debugging tool. Table 7-1 describes the error types reported in the *block*.LAYOUT_ERRORS file during a run.

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_AREA | AREA | Lists violations produced by the AREA command. |
| ERR_BOOLEAN_AND | BOOLEAN | Lists violations produced by the BOOLEAN command with the AND operator. |
| ERR_BOOLEAN_ANDOVERLAP | BOOLEAN | Lists violations produced by the BOOLEAN command with the ANDOVERLAP operator. |
| ERR_BOOLEAN_NOT | BOOLEAN | Lists violations produced by the BOOLEAN command with the NOT operator. |
| ERR_BOOLEAN_OR | BOOLEAN | Lists violations produced by the BOOLEAN command with the OR operator. |

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File(Continued)*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_BOOLEAN_ XOR | BOOLEAN | Lists violations produced by the BOOLEAN command with the XOR operator. |
| ERR_C_THRU | C_THRU | Flags the polygons of layer1 that do not belong to the same net and that are found inside of a layer2 polygon. |
| ERR_CELL_ EXTENT | CELL_EXTENT | Flags polygons the size of the extent of the cell. |
| ERR_CENTER_TO_ CENTER | CENTER_TO_ CENTER | Lists violations produced by the CENTER_TO_CENTER command. |
| ERR_CHECK_ DEVICE | RESCHECK<br>MOSCHECK | Flags extracted devices whose parameters meet the check criteria. |
| ERR_COMPARE_ GROUP | COMPARE_GROUP | Lists violations produced by the COMPARE_GROUP command. |
| ERR_CONNECT_ POINTS | CONNECTION_ POINTS | Flags rectangles produced by the CONNECTION_POINTS command. |
| ERR_COPY | COPY | Lists violations produced by the COPY command. |
| ERR_CREATE_ PORTS | CREATE_PORTS | Marker polygon was not used after all marker assignments. If verbose is TRUE in the CREATE_PORTS command, also writes out all new ports created. |
| ERR_CUT | CUT | Lists violations that meet the criteria in the CUT command. |
| ERR_DENSITY | DENSITY | Identifies the window where the area or ratio calculation does not satisfy the AREA or RATIO constraints in the DENSITY command. |

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File(Continued)*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_DEVICE | CAPACITOR<br>DIODE<br>DEVICE<br>NMOS, PMOS<br>NPN, PNP<br>RESISTOR | Error found while attempting to extract unrecognizable devices. Could be caused by a missing terminal or bulk layer, extra terminals, or from the device equations. |
| ERR_DEVICE_ COUNT | DEVICE_COUNT | Lists nets that satisfy the device count criteria. |
| ERR_DEV_ CONNECT_CHECK | DEV_CONNECT_ CHECK | Flags devices that meet the criteria when verbose is set to TRUE in the DEV_CONNECT_CHECK command. |
| ERR_DEV_NET_ COUNT | DEV_NET_COUNT | Flags devices that meet the DEV_NET_COUNT command criteria. Verbose is set to TRUE in the DEV_NET_COUNT command. |
| ERR_DUP_PLCMT | | During preprocessing of input data, duplicate cells or an array of duplicate cell references and their locations are flagged. |
| ERR_ENCLOSE | ENCLOSE | Lists violations produced by the ENCLOSE command. |
| ERR_EXTERNAL | EXTERNAL | Lists violations produced by EXTERNAL command. |
| ERR_GRID | GRID_CHECK | Lists boundary, path-center-line, rectangle, cell-reference, and array-of-cell references whose coordinates are off-grid. |
| ERR_INSIDE_EDGE | INSIDE_EDGE | Lists violations produced by INSIDE_EDGE command. |
| ERR_INTERNAL | INTERNAL | Lists violations that meet the criteria in the INTERNAL command. |

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File(Continued)*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_LENGTH | LENGTH | Lists violations that meet the criteria in the LENGTH command. |
| ERR_MASK_ALIGN | MASK_ALIGN | Flags area of intersecting polygons that fail the X, Y, and W criteria from a matrix table. The Y value in the table determines the minimums of X and W. Any parameter less than the minimum input value is an error. |
| ERR_MISSING_ CELL | | Each run checks for a cell that is referenced (by another cell) but that does not exist in the input database. The missing cell name and its reference location are flagged. |
| ERR_NEGATE | NEGATE | Lists violations produced by the NEGATE command. |
| ERR_NET_FILTER | NET_FILTER | Lists nets that satisfy all net_filter criteria. |
| ERR_NET_PATH_ CHECK | NET_PATH_CHECK | Lists nets that satisfy the path criteria. |
| ERR_NOTCH | NOTCH | Lists violations produced by the NOTCH command. |
| ERR_POLYGON_ FEATURES | POLYGON_ FEATURES | Lists violations produced by the POLYGON_FEATURES command. |
| ERR_PUSH | PUSH | Lists violations produced by the PUSH command. |
| ERR_RATIO | RATIO | Flags all nets that do not satisfy the RATIO criteria. |
| ERR_RELOCATE | RELOCATE | Lists violations produced by the RELOCATE command. |
| ERR_REMOVE_ OVERLAP | REMOVE_ OVERLAP | Produced by the REMOVE_OVERLAP command. |

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File(Continued)*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_SELECT | SELECT | Lists violations that meet the criteria in the SELECT command. |
| ERR_SELECT_ CELL | SELECT_CELL | Lists violations produced by the SELECT_CELL command. |
| ERR_SELF_ISECT | | Each Hercules run checks for input polygons whose edges form a path that intersects itself. The coordinates of the self-intersecting edges are flagged. |
| ERR_SIZE | SIZE | Lists violations that meet the criteria in the SIZE command. |
| ERR_SNAP | SNAP | Lists violations produced by the SNAP command. |
| ERR_TEXT_BAD | | Each Hercules run checks for a bad text string in the layout database. The bad text string might contain an unprintable character or characters such as a control character, or the string length exceeds 512 characters. |
| ERR_TEXT_OPEN | TEXT with NETLIST | A set of physical nets that have the same text but are not electrically connected; the netlist was modified as if the nets were connected. An open could be instance-specific or in all instances of the parent structure. If it is instance-specific, the origin of the instance is also given. |
| ERR_TEXT_OPEN_ RENAME | TEXT with NETLIST CONNECT_BY_ NAME under TEXT_OPTIONS | Text in named structures have been renamed due to opens. |
| ERR_TEXT_ OVERRIDE | TEXT with NETLIST | A text string in Edtext file is chosen over the text string in layout database. |

*Table 7-1 Error Types Reported in block.LAYOUT_ERRORS File(Continued)*

| Error Type | Command Generating Error | Output Description |
|---|---|---|
| ERR_TEXT_ POLYGON | TEXT_POLYGON | Flags polygons produced by the TEXT_POLYGON command. |
| ERR_TEXT_SHORT | TEXT with NETLIST | Text strings for net in named structures are shorted (more than one text string is assigned to a given net). |
| ERR_TEXT_SHORT_ DISCARD | TEXT with NETLIST REMOVE_TEXT_ FROM_SHORT under TEXT_OPTIONS | The text strings of a net in named structures are removed due to shorts. |
| ERR_TEXT_ SHORTEST_PATH | FIND_SHORTEST_ PATH_BETWEEN_ TEXT_ SHORTS | Lists pairs of text points for which shortest-path data has been written to the error hierarchy. |
| ERR_TEXT_ UNUSED | TEXT with NETLIST | Text string was not used after all text assignments. |

## Flat Error File

Hercules creates the *block*.toperr file when the OPTIONS section option FLAT_ERROR is set to TRUE. For an explanation of the FLAT_ERROR option, see the *Hercules Reference Manual*, Detailed Options chapter. The *block*.toperr file contains a list of one-line reports of the flat coordinates of all errors in the *block*.LAYOUT_ERRORS file and an identification (an output layer number, layer name, or cell name). The flat error coordinates in the *block*.toperr are always relative to the origin of the top-level block, as if it has been flattened. These coordinates are useful when looking at the graphical output from the top structure. The error coordinates in the *block*.LAYOUT_ERRORS file are relative to the origin of the structure in which the error occurred.

The output layer and data type are written for identification. For error types without an output layer and data type, such as ERR_DUP_PLCMT, ERR_MISSING_CELL, ERR_SELF_ISECT, and some ERR_GRID, the structure name that caused the error is printed. For temporary output (which may or may not have a layer number), the equated name is printed.

## LVS Debugging File

Hercules COMPARE produces a file called *block*.LVS_ERRORS, which indicates whether or not the schematic versus layout comparison was successful. If it was not successful, the file lists the PASS/FAIL information and layout error if any. The *block*.LVS_ERRORS file then lists the important failed equivalence points, sorted in high and low priorities, and provides diagnostics designed to assist you in troubleshooting the design. The *block*.LVS_ERRORS file is written to your working directory and you should considered it to be the starting point for debugging LVS issues.

## Extracted Layout Netlist File

Hercules creates *block*.net, which is a hierarchical description of the structures found in the layout database, in the working directory whenever a NETLIST extraction command is specified in the runset file.

## LPE Statistics File

Hercules creates *block*.lpe, an LPE Statistics file, that contains the extraction statistics, geometric parameters, and calculated variables for the LPE devices. This file is created whenever the LPE_STATS command is executed in the runset file and statistics printing is turned on for the extracted devices. The statistics printing for individual devices can be turned on and off using the appropriate option in the extraction command.

## Extracted Layout SPICE Netlist File

Hercules creates *block*.sp, a hierarchical SPICE description of the structures found in the layout database, whenever the SPICE command is specified in the runset file. See the *Hercules Reference Manual*, Detailed Commands chapter.

## Compare Log File

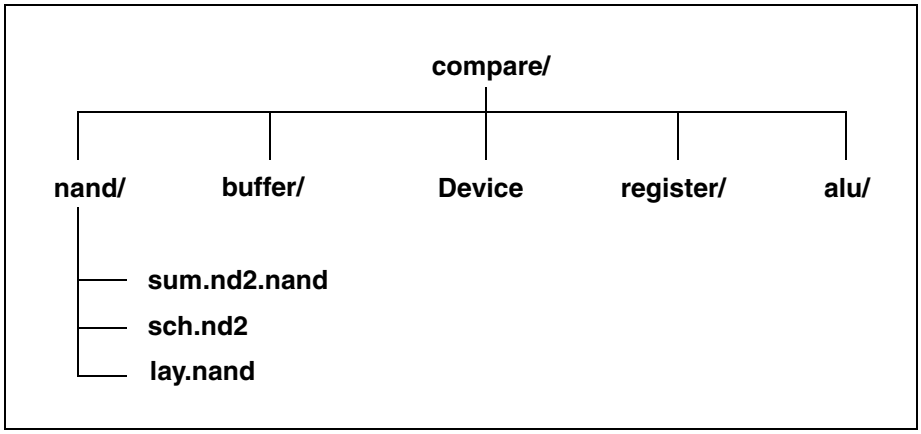
Hercules creates *block*_lvs.log, a log file in run_details directory, during an LVS run. This compare log file contains all the messages output to the screen during the run.

## Compare Directory Output Files

The detailed output files from an LVS comparison are placed under the directory specified by the HEADER section COMPARE_DIR variable. A subdirectory under this compare directory is created for each equivalence point in the Equivalence file as they are evaluated.

The COMPARE option RETAIN_NEW_DATA can be set to FALSE to prevent creating directories and printing compare information on equivalence points that compare without warnings or errors. These subdirectories are named after the layout structure name. The files shown in Figure 7-1 are subcategories of the COMPARE Directory Output Files.

*Figure 7-1 Compare Directory Structure Example*




## Individual Equivalence Summary File

The results of the comparison of each structure are written to a summary file named sum.*schematic_module_name.layout_structure_name* (generally referred to as the sum.block.block file). However, this file is not generated if the COMPARE option RETAIN_NEW_DATA is set to FALSE, and the particular equivalence point completes without errors or warnings. The information in the summary includes the following: the number of occurrences of each device found; the total number of nets that were found; and the number of devices and nets that were removed or created by merging. If all of the devices and nets could not be matched, the summary includes a listing of the discrepancies.

## Partitioned Schematic Netlist

The schematic data used for this portion of the comparison process is written to a file called sch.block. However, this file is not generated if the COMPARE option RETAIN_NEW_DATA is set to FALSE, and the particular equivalence point completes without error or warnings or if the COMPARE option WRITE_NETLISTS is set to FALSE. This file is the schematic netlist that results after exploding non-equivalent structures; it generally provides little or no useful information for analyzing comparison results.

## Partitioned Layout Netlist

The layout data used for this portion of the comparison process is written to a file called lay.block. However, this file is not generated if the COMPARE option RETAIN_NEW_DATA is set to FALSE, and the particular equivalence point completes without errors or warnings or if the COMPARE option WRITE_NETLISTS is set to FALSE. This file is the layout netlist that results after exploding non-equivalent structures; it generally provides little or no useful information for analyzing comparison results.

## Compare Tree Files

Hercules COMPARE produces two tree files. One is named lay.tree and provides the hierarchical structure of the layout netlist. The second is named sch.tree and provides the hierarchical structure of the schematic netlist. These files are in the run_details/lvsflow directory.

## Equivalence Files

Hercules COMPARE automatically produces an equivalence file to compare the layout and schematic netlists if you do not supply one. When producing an equivalence file, Hercules recognizes that some equivalence points are extraneous and do not aid in the comparison process. After all analysis is completed, Hercules generates an equivalence file, equiv.run, in the directory run_details, to be used during runtime. The file contains all the user-provided equivalence points and any extra ones that are found by the FIND_ADDITIONAL_EQUIVS option in COMPARE section.

## TECHNOLOGY_OPTIONS File

Hercules creates *block*.tech, a TECHNOLOGY_OPTIONS file, that displays the options that are set for the Hercules run. The list of options includes the default TECHNOLOGY_OPTIONS as well as user-specified TECHNOLOGY_OPTIONS. In addition, it lists all the cells that are exploded by the TECHNOLOGY_OPTIONS.

## Tree Structure Output File

Hercules creates Tree Structure files, named *block*.tree, during DRC or netlist extraction runs. Tree Structure files contain information about the design hierarchy tree and reference statistics for every cell in the design tree. The *block*.tree0 file represents the original data. The *block*.tree1 file is created after all of the exploding options are completed. Additional tree files are created when using technology options. The *block*.treeN file with the highest number (N) is the hierarchy that was processed during the run. All tree files can be created

if the PRINT_TREE_FILE option is set to TRUE. The PRINT_TREE_FILE option is set in the TECHNOLOGY_OPTIONS section of the runset. For more information about PRINT_TREE_FILE, see the *Hercules Reference Manual*, Detailed Options chapter.

Tree structure files can be generated without executing a complete Hercules run by using the -P switch on the command line.

## WAIVE File

Hercules creates, *block*.waive, a waive file for unwanted or waived dimensional check violations. Waived violations are reported in the *block*.waive file, which has the same format as the *block*.LAYOUT_ERRORS file. For more information on the WAIVER command, refer to the *Hercules Reference Manual*, Detailed Commands chapter.

## Milkyway View and Object Type Statistics

The cell statistics section of tree files contains information on which Milkyway View and Object Types were read in during the Hercules run.

The OBJECT_TYPE column is represented by abbreviations of OBJECT_TYPEs.

| Abbreviation | Represents |
|---|---|
| RE | rectangle |
| PO | polygon |
| PA | path |
| BO | boundary |
| PI | pin |
| HW | horizontalwire |
| VW | verticalwire |

A new report contains VIEW and its version and OBJECT_TYPEs. For example:

```
CELL = invbd2
```

Layer Usage of Original Cell:

| Name | Data/ Text | Layer:Dtype | Layer Extents (LL,UR) | Lyr Area | VIEW Ver. | OBJECT_ TYPE |
|---|---|---|---|---|---|---|
| nwell | 2/0 | 1:0 (-0.40, 3.70) (3.60, 8.40) | 18.80 | CEL | 2 | RE PO |
| nwell_fram | 2/0 | 1:0 (-0.20, 2.50) (0.22, 4.12) | 4.20 | FRAM | 2 | RE |
| m1_pin | 4/0 | 17:0(0.00, 0.00) (3.20, 8.00) | 9.27 | FRAM | 1 | PI |
| m1_pin.text | 0/10 | (0.42, 0.53) (1.82, 7.47) | | FRAM | 1 | |

In the following example, cell $$VIA1 is a contact cell. Therefore, VIEW Ver. and OBJECT_TYPE columns are automatically skipped.

```
CELL = $$VIA1
```

Layer Usage of Original Cell:

| Name | Data/ Text | Layer:Dtype | Layer Extents (LL,UR) | Lyr Area | VIEW Ver. | OBJECT_ TYPE |
|---|---|---|---|---|---|---|
| metal1_pin | 1/0 | 16:0 (-0.20, -0.20) (0.20, 0.20) | 0.16 | | | |
| via1 | 1/0 | 17:0 (-0.18, -0.18) (0.18, 0.18) | 0.13 | | | |
| metal2 | 1/0 | 18:0 (-0.20, -0.60) (0.20, 0.60) | 0.48 | | | |

In this example:

- layer (1:0) of TOP.CEL:1 contains RE and PO.
- layer (1:0) of TOP.FRAM:2 contains RE.
- layer m1_pin.text is m1_pin's text container, and its OBJECT_TYPE is always TEXT. The OBJECT_TYPE column is automatically skipped.

# Graphical Error Structures

In addition to the output files discussed in the preceding section, Hercules generates a graphical error output database that can be viewed in a layout editor. It contains error polygons and coordinates that are located in proximity to design rule violations. The shapes produced by different checks can be written to separate layers and data types in the error structure as specified in the runset file.

## Graphical Error Hierarchy

Hercules generates a hierarchy of error structures that mimics the database hierarchy that is being checked. Thus, any errors that are found in a particular structure in the database appear in a corresponding error structure in the error hierarchy. Error structures are generated only for the database structures that contain errors or that reference other structures with errors. Hercules will not create an error hierarchy if no errors were found.

Error structures are named with the default error prefix ERR_, which precedes the structure name. For example, the default error structure for the cell CLOCK is ERR_CLOCK. The error prefix can be redefined with the ERR_PREFIX option in the runset OPTIONS section.

For example:

| **Data Base Hierarchy** | **Error Hierarchy** |
|---|---|
| CHIP {<br>  ALU {<br>  BIT0<br>  BIT1 }<br>  CLOCK {<br>  CLK1<br>  CLK2<br>  CLK3 }<br>} | ERR_CHIP {<br>  ERR_ALU {<br>  ERR_BIT0<br>  ERR_BIT1 }<br>  ERR_CLOCK {<br>  ERR_CLK1<br>  ERR_CLK3 }<br>} |

Note:

In the error hierarchy, there is no occurrence of the error structure ERR_CLK2. There are several possible causes of this. Either the original CLK2 cells were exploded during preprocessing, or the structure CLK2 did not contain any errors or structures that contain errors. If a previous Hercules run with errors is corrected and rerun, previous error

hierarchy structures are retained; Hercules does not overwrite these structures if no errors are found, however the new error hierarchy will not have references to these previous error containing cells.

## Error and Permanent Output

Hercules creates a top holding structure in the Output Library or GDSII file for error and permanent output, see "Output Formats" on page 7-19. The name of this structure is defined by the OUTPUT_BLOCK option in the runset HEADER section. If not specified, OUTPUT_BLOCK defaults to EV_OUTPUT. This structure contains placements of the error hierarchy and all output structures resulting from PERM=layer_name assignments made in the runset file. This structure can be placed into the block that was checked to allow for quick overlaying of all output data. The top holding structure is created only when error data or PERM output data is generated.

## Placing the Error Structures and Permanent Output

The Error structures and permanent output created by Hercules are placed in the library defined by the OUTLIB option in the runset HEADER section. If the errors and permanent output are placed in a library other than the library that contains the original database, you can bind the error library to the database library as a permanent or a temporary reference library. You can then overlay the data not needed here in generic description to place an occurrence of the error structure. Notice that because Hercules mimics the database hierarchy, you can open an error structure at any level of the hierarchy and place the corresponding error structure. In the example in "Graphical Error Hierarchy" on page 7-14, you can open the structure CLOCK and place the error structure ERR_CLOCK. Alternatively, the GUI debugging tool can be used to view errors directly on original data without having to bind in a reference library. For more details on VUE, see the *Hercules VUE User Guide*, Running Hercules Within VUE chapter.

## Editing Errors

Once the error structure has been placed into the database, you can begin editing and correcting the errors. The error file, *block*.LAYOUT_ERRORS, generated by Hercules, lists the location and the structure name for each error encountered. The summary file, block.sum, lists the checks that have errors.

Because the output of Hercules is hierarchical, you need to correct an error only once, no matter how many times the structure is placed. This process continues until all errors are eliminated.

# HTML Interface

Hercules provides an HTML interface to its DRC, LVS Extract, and LVS COMPARE output data. This enables the information located in the block.sum, block.LAYOUT_ERRORS, block.LVS_ERRORS, and sum.block.block files for each equiv point to be accessed easily, and organizes that information for easy interpretation. This interface is also closely tied on the LVS COMPARE side to COMPARE's debugging hints to further assist in locating and fixing LVS errors.

## Invoking HTML Output

Hercules provides two command-line options related to HTML output:

- -html produces HTML data and invokes the browse script at the completion of the run.
- -html-nobrowse produces HTML data and the browse script but does not invoke the script at the completion of the run.

## Example of HTML Output

Figure 7-2 shows available HELP, Summary file, and Error Summary information. The Error Summary information is displayed by default. If HELP (which further describes the organization of HTML data) or summary file information is needed then just click HELP or Summary in the left frame of your web browser. Also, note that you can click the number of violations by each violation type and the web browser will automatically load the necessary error information in the web browser.

*Figure 7-2 HTML Interface: HELP, Summary File, Error Summary*

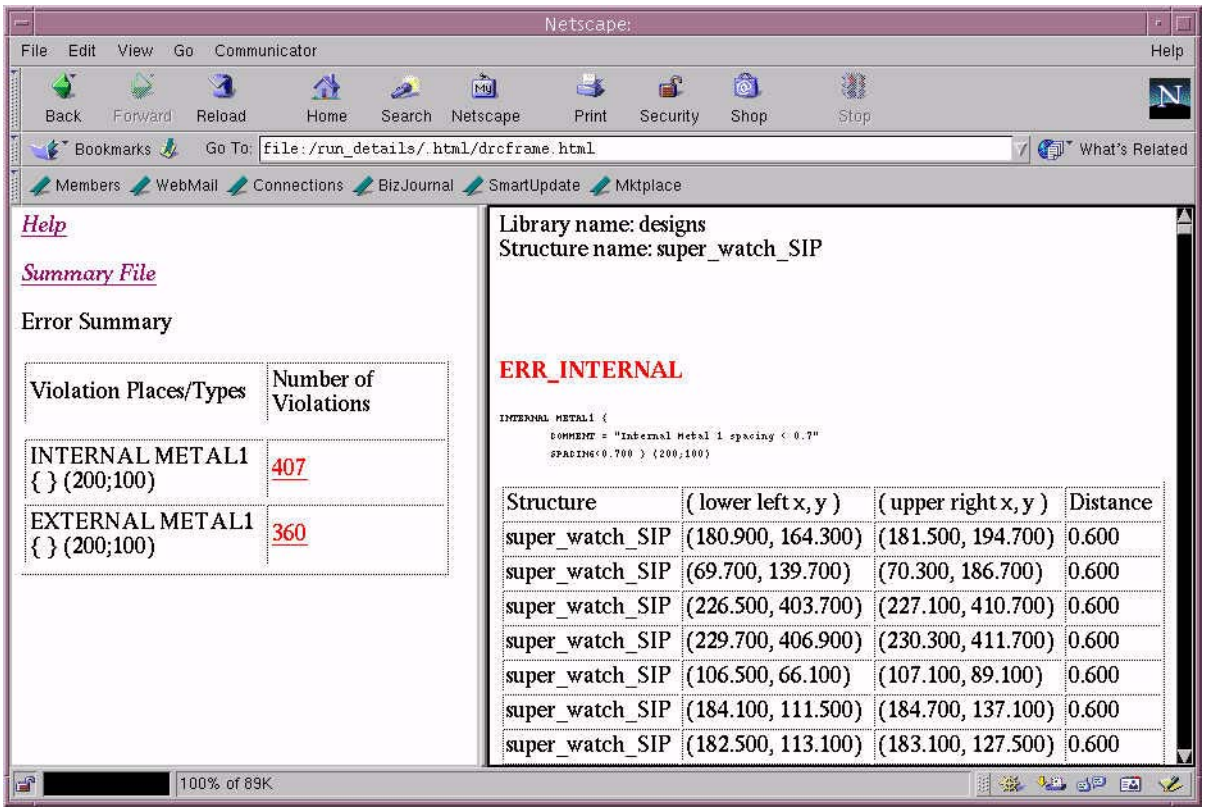
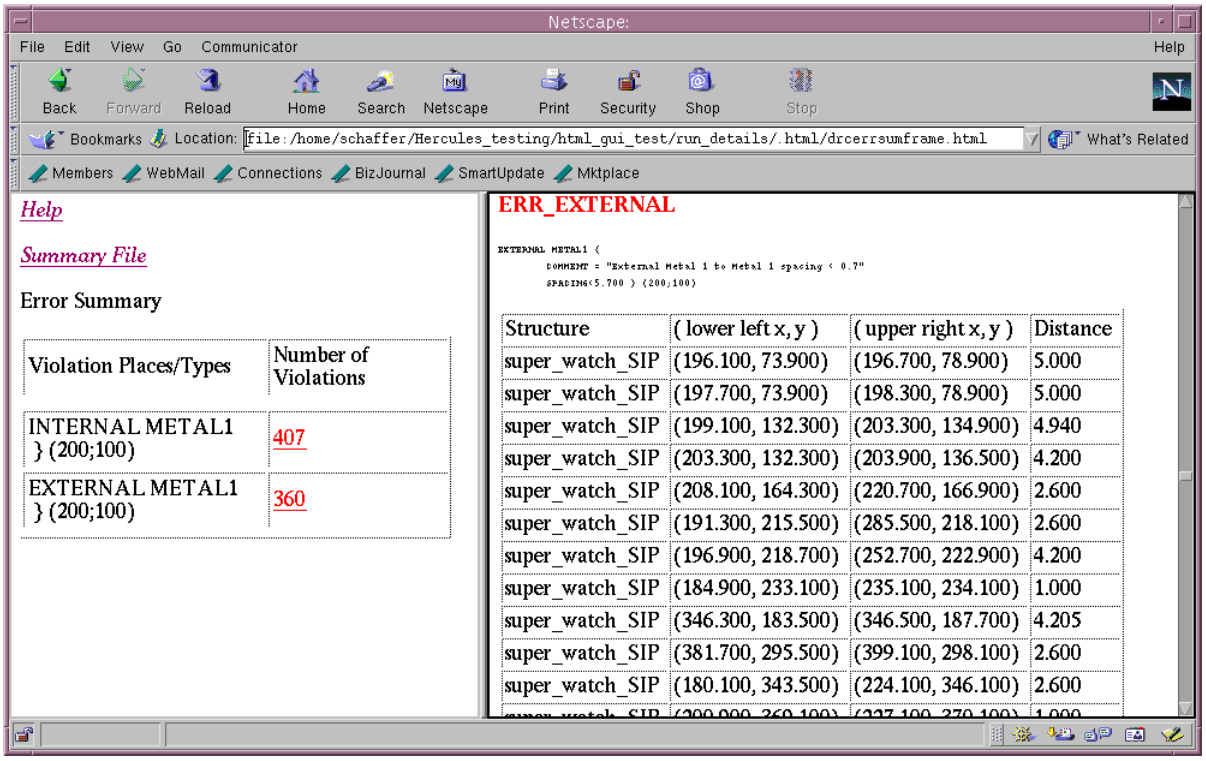



Figure 7-3 shows error information for the EXTERNAL command. After clicking 360 besides the EXTERNAL command, layout errors for the EXTERNAL command will be shown. You can thus view the error summary in the left frame and detailed error information in the right frame.

*Figure 7-3 HTML Interface: Error Information for EXTERNAL Command*




## Saving Output from Runset Commands

The results of runset commands can be saved as output upon completion of the run (permanent output); or, the resulting data can be stored as intermediate data to be deleted at the end of the run (temporary output). Command output data is stored hierarchically using the hierarchy of the block being checked.

Both permanent and temporary layers are given a label so that they can be referenced later in the run by other commands. The label-given data that is saved as a temporary or intermediate layer can be used again in the runset. Beware of using labels for layers that are equivalent to the original layer. If a Hercules command modifies a layer and the original layer name is uses for the temp layer, the original data cannot be retrieved in the same Hercules run. The format for defining command output is shown below.

## Output Formats

### Error Hierarchy Syntax

```
(label; datatype)
```

Results are output to the error hierarchy on the given layer and optional datatype. Default datatype is 0.

```
ERROR=(layer; datatype)
```

This verbose syntax is provided to allow for an error hierarchy output in addition to a TEMP output for the following commands (ENCLOSE, EXTERNAL, INSIDE_EDGE, INTERNAL, *LEVEL, NOTCH, *DEVICE commands).

Note:
You must have VERBOSE=TRUE for these errors to be written to the *block*.LAYOUT_ERRORS file.

For example:

```
INTERNAL poly { SPACING < 0.13 VERBOSE=TRUE } TEMP=err_int ERROR=(1;0)
```

### Temporary Result Syntax

```
TEMP=Label
```

Defines an intermediate result file named *Label*. This result can be referenced by subsequent commands using the name *Label*; *Label* can also be reused as an output by any other commands (except for the BUILDSUB command; see the *Hercules Reference Manual*, Detailed Commands chapter, for a discussion of the use of BUILDSUB).

### Permanent Result Syntax

```
PERM=Label (layer;datatype)
```

Defines a permanently saved result file named *Label*. This data is output on the given layer and optional data type. This data can be referenced by subsequent checks using the name *Label*. However, *Label* cannot be reused as an output by any other command; Hercules exits with an error message. (Structures created are named Label_*structure_name*.) Default datatype is 0.

Note:
The PERM file name cannot be designated as ERR_. This is the default for the error structure prefix name. (see the *Hercules Reference Manual*, Detailed Options chapter, the ERR_PREFIX option.) PERM cannot be set to any error structure prefix name or Hercules issues a warning.

Some examples are shown in Example 7-1.

*Example 7-1*

```
   /* All data resulting from this command is saved in the
   Error Hierarchy on layer 25.*/
BOOLEAN diff AND poly (25)

   /* The temporary layer GATE is deleted upon completion
   of the run.*/
BOOLEAN diff AND poly TEMP=GATE

   /*The output data is retained at the completion of the
   run and stored on layer 25 with a name GATE_structure.*/
BOOLEAN diff AND poly PERM=GATE (25)

   /*The temporary layer GATE is reused in a subsequent
   operation, and deleted upon completion of the run.*/
BOOLEAN diff AND poly TEMP=GATE
BOOLEAN diff NOT GATE PERM=SOURCEDRAIN (25)
```

# 8

# Scheme Interface

*This chapter discusses Hercules interfaces with the Scheme programming language and the different areas of Hercules used with Scheme. The Scheme programming language to provides flexible data manipulation.*

For details on how to write Scheme programs, see the SolvNet article “Scheme Language Reference.” Refer to “Accessing SolvNet” on page 2-xvii for instructions on accessing SolvNet.

# Introduction to Scheme Usage

A Hercules run using the Scheme language requires and runs off of a standard Hercules runset. The control flow and ordering of the commands are based on the design of that runset. These routines require a Scheme file in addition to the Hercules runset file. Only one Scheme file is allowed per runset.

In the runset, when a command is reached containing Scheme code, execution control switches from Hercules tool algorithms to the Scheme code. When the Scheme code completes its execution, control reverts back to Hercules algorithms, and the run continues normally.

Some of Hercules tool functionality requires Scheme code in the runset, as with the EXPLODE language and the GENDEV command. Other functionality has default Hercules algorithms in place that are executed if no Scheme code is written for them. For example, the default methods for Flexible Device Netlisting and LVS Property Comparison do not require Scheme, but Scheme is provided for users who want to tailor the default method. Hercules does not check the results of the user-written Scheme code, and assumes the code is logically and algorithmically correct.

When writing your scheme routines, you can access existing scheme routines from the Slib library, which is part of your Hercules installation (under ./etc/scheme/slib).

You will need to insert a require statement within your scheme file for each slib routine you want to call. For example, if you wish to call the slib format routine, you will need to add the following line to your scheme file:

```
(require 'format)
```

There are three steps involved in using the Scheme interface. Although the specifics vary with each application, the basic steps listed below remain the same.

*Step 1:* Create a Scheme file containing the code to perform the desired function. Various routines are available that allow you to write this code to perform a set of meaningful operations. The routines that pertain to each application are described later in this chapter.

*Step 2:* Call the Scheme file into your Hercules runset using the SCHEME_FILE option in the HEADER section of your runset, as shown in the example below. (see the *Hercules Reference Manual*, Detailed Options chapter, for more information on the SCHEME_FILE option.)

```
--- runset.ev ---

HEADER {
   SCHEME_FILE = netlist.scm
}
```

*Step 3:* Define which function in the Scheme file applies to a given command. The syntax varies based on the command, but, in every case, one or more functions must be listed in the Hercules runset, along with the corresponding command. This defines which section of the Scheme code from the Scheme file is to be executed.

## Flexible Device Netlisting

For each device creation command (NMOS and DIODE are examples) a Scheme procedure can be registered for use by Hercules during the creation of the SPICE or Hercules netlists.

*Step 1:* Create a Scheme file containing the procedures that will output the netlist entries. Each procedure takes in a single argument: *device-id*. This *device-id* may then be passed to the EVaccess DEVCON routines to retrieve information. A Scheme file is shown in Example 8-1.

*Example 8-1 Scheme File*

```
--- netlist.scm ---

;; Will output
;; M12 net1 net2 net3 n l=4e-3u
;;
(define spice-mos (lambda (dev-id)
   (string-append
     (ev-devcon-dev-name-get dev-id) " "
     (ev-devcon-net-name-get
         (ev-devcon-dev-conn-net-get dev-id "DRN")) " "
     (ev-devcon-net-name-get
         (ev-devcon-dev-conn-net-get dev-id "GATE")) " "
     (ev-devcon-net-name-get
         (ev-devcon-dev-conn-net-get dev-id "SRC")) " "
     (ev-devcon-dev-type-get dev-id) " "
          "l=" (number->string
         (ev-devcon-dev-prop-value-get dev-id "EV_LENGTH")) "u"
   )
))
```

*Step 2:* Register the Scheme file in the HEADER section of the Hercules runset, as shown here:

```
--- runset.ev ---

HEADER {
   SCHEME_FILE = netlist.scm
}
```

*Step 3:* Register the Scheme procedures with the device extraction commands to be custom-output, as shown in below. All devices that do not have an associate Scheme procedure use the default Hercules output format.

The two possible netlist types to register procedures are EV_NETLIST_FUNC and SPICE_NETLIST_FUNC.

```
--- runset.ev ---

NMOS n gate src drn {
   SPICE_NETLIST_FUNC = spice-mos;
} TEMP = ndev

PMOS p gate src drn {
   SPICE_NETLIST_FUNC = spice-mos;
} TEMP = pdev
```

## Device Routines

The device routines that can be called by user-defined Flexible Device Netlisting routines are described here.

- The information returned by the routine is shown with ==>.
- #t represents true (0) and #f represents false (1).

### ev-devcon-dev-type-get

Returns a string containing the device type. The syntax is:

`(ev-devcon-dev-type-get` *`dev-id`*`)`

==> string

### ev-devcon-dev-name-get

Returns a string containing the device name. The syntax is:

`(ev-devcon-dev-name-get` *`dev-id`*`)`

==> string

For example:

`(ev-devcon-dev-name-get` *`dev-id`*`)  ==> "M12"`

### ev-devcon-dev-class-get

Returns the device class. The syntax is:

`(ev-devcon-dev-class-get` *`dev-id`*`)`

==> ev-devcon-dev-class

For example:

```
(ev-devcon-dev-class-get dev-id)  ==> "ev-devcon-dev-class-nmos"
```

## ev-devcon-dev-is-shorted

Returns true if the device is shorted; returns false if not. The syntax is:

```
(ev-devcon-dev-is-shorted ev-devcon-dev-id)
```

==> #t | #f

## ev-devcon-dev-is-netlisted

Returns true if the device is netlisted; returns false if not. The syntax is:

```
ev-devcon-dev-is-netlisted dev-id)
```

==> #t | #f

## ev-devcon-dev-is-parasitic

Returns true if the device is parasitic; returns false if not. The syntax is:

```
(ev-devcon-dev-is-parasitic dev-id)
```

==> #t | #f

## ev-devcon-dev-prop-list

Returns a list of property name and value pairs. Value may be of type string, either exact or inexact. The syntax is:

```
(ev-devcon-dev-prop-list dev-id)
```

==> ("string1" value1 ... "stringN" valueN)

For example:

```
(ev-devcon-dev-prop-list dev-id)  ==> ("EV_DEVNUM" 1
   "EV_DEVTYPE" 1 EV_TERMNUM" 3 "EV_TERM_X1_COORD" 0.0
   "EV_TERM_Y1_COORD" 0.0 "EV_TERMTAG1" 4 "EV_IDEALNETID1" 0
   "EV_TERM_X2_COORD" -3.0 "EV_TERM_Y2_COORD" 0.0
   "EV_TERMTAG2" 5" EV_IDEALNETID2" 0 "EV_TERM_X3_COORD" 3.0
   "EV_TERM_Y3_COORD" 0.0 "EV_TERMTAG3" 6 "EV_IDEALNETID3" 0
   "EV_DARA" 4.0 "EV_DPERIM" 8.0 "EV_SARA" 4.0" EV_SPERIM" 8.0
   "EV_LENGTH" 4.0 "EV_WIDTH" 2.0 "EV_NRS" 12.500000317221e24
   "EV_NRD" 12.500000317221e24)
```

## ev-devcon-dev-prop-value-get

Returns the value for the specified property. Value may be of type string, exact, or inexact. The syntax is:

```
(ev-devcon-dev-prop-value-get dev-id "property_name")
```

==> value

For example:

```
(ev-devcon-dev-prop-value-get dev-id "EV_DAREA")   ==> 4.0
```

## ev-devcon-dev-conn-list

Returns a list of port name and net id pairs. The syntax is:

```
(ev-devcon-dev-conn-list dev-id)
```

==> ("string1" value1 ... "stringN" valueN)

For example:

```
(ev-devcon-dev-conn-list dev-id)
   ==> ("GATE" #<ev-devcon-net-id 0>
        "SRC" #<ev-devcon-net-id 1>
        "DRN" #<ev-devcon-net-id 2>)
```

## ev-devcon-dev-conn-net-get

Accepts a port name as an argument in the form of a string. Returns a net ID or true if no match is found and false if there is an error. The syntax is:

```
(ev-devcon-dev-conn-net-get dev-id string)
```

==> ev-devcon-net-id | #t | #f

For example:

```
(ev-devcon-dev-conn-net-get dev-id "GATE")
   ==> #<ev-devcon-net-id 0>
```

## ev-devcon-net-name-get

Returns a net name. The syntax is:

```
(ev-devcon-net-name-get net-id)
```

==> "string"

For example:

```
(ev-devcon-net-name-get (ev-devcon-dev-conn-net-get dev-id
   "GATE")) ==> "NET12"
```

## GENDEV Command

The GENDEV command defines special devices for extraction. Hercules provides an extensive programming capability through Scheme. It is similar to other extraction commands; it executes when it is found in the runset; it takes layers as input and can write data to an output layer.

When Hercules detects a GENDEV command in the runset during execution, execution control passes from Hercules to Scheme. Scheme executes the initialization function and then the extraction function. Once those two Scheme functions are completed, control reverts back to Hercules, which starts executing the remaining commands in the runset.

There are two modes for the GENDEV command operation. They are AUTO mode and MANUAL mode (refer to the *Hercules Reference Manual*, Detailed Commands chapter). There are some differences in the corresponding Scheme procedures and applicable Scheme functions, depending on whether you are using AUTO or MANUAL mode.

A Scheme procedure can be registered for use by the GENDEV command to define the extraction of the generic device. The generic steps of coding the scheme file for both modes are as follows (the differences between them and their coding examples will be shown specifically in their own categories).

*Step 1:* Create a Scheme file that contains the procedures to be used to extract the generic devices. The GENDEV command requires two scheme functions defined in this file: 1) an initialization function, and 2) an extraction function. (See the following examples for each mode.)

*Step 2:* Register the Scheme file in the HEADER section of the runset, as shown here:

```
--- runset.ev ---

HEADER {
    SCHEME_FILE = extract.scm
}
```

*Step 3:* Register the Scheme procedures with the GENDEV command in the runset. (See the examples for each mode, “Example for AUTO mode” on page 8-8 and “Example for MANUAL Mode” on page 8-9.)

## Example for AUTO mode

Both the initialization function and extraction function have no parameters. The polygon_list_id variable is directly available by the same name as the layer_name in the GENDEV command. The polygon_list variable represents the polygons from the layer which interacts with the polygon from the recognition_layer in the current cell.

*Step 1*

*Example 8-2*

```
--- extract.scm ---

;Initialization function for a simple resistor
;resistance (rval) is the only extracted property

(define res_init (lambda ()
(ev-dev-property-type-set "rval" 'DOUBLE)
))

;Extraction function for a simple resistor.  The device is made
;up by a resistor material layer and a resistor marker layer
;that defines the body of the device.
(define res_extract (lambda ()
   (let
      (
        (L #f)
        (W #f)
        (resistance #f)
        (sheet_res 1)
      )

   ;compute resistance
   (set! W (/ (ev-measure2 rbody 'COIN_PERIM layer1) 2))
   (set! L ( / (- (ev-measure1 'PERIM rbody) (* 2 W)) 2))
   (set! resistance (* sheet_res  (/ L W) ))

   ;assign properties to the device
   (ev-dev-property-value-set gen_res "rval" resistance)
)))
```

*Step 2*

```
--- runset.ev ---

HEADER {
   SCHEME_FILE = extract.scm
}
```

*Step 3*

If terminal layers are the same:

```
--- runset.ev ---

GENDEV gen_res terminal terminal {
   INITIALIZATION_FUNC = res_init
   EXTRACTION_FUNC = res_extract
   RECOGNITION_LAYER = {rbody}
   PROCESSING_LAYERS = {layer1}
} TEMP = gendev_res
```

Or, if terminal layers are not the same:

```
GENDEV gen_res terminal1 terminal2 {
   INITIALIZATION_FUNC = res_init
   EXTRACTION_FUNC = res_extract
   RECOGNITION_LAYER = {rbody}
   PROCESSING_LAYERS = {layer1}
} TEMP = gendev_res
```

## Example for MANUAL Mode

In MANUAL mode, there are some differences in the Scheme function which extract a device from two identical terminal layers versus from two different terminal layers.

For two identical terminal layers use these three steps:

*Step 1*

*Example 8-3*

```
;Initialization function for a simple resistor
;resistance (rval) is the only extracted property

(define res_init (lambda ()
        (ev-dev-property-type-set "rval" 'DOUBLE)
))

;Extraction function for a simple resistor.  The device is made up
;by a resistor material layer and a resistor marker layer that
;defines the body of the device.
(define res_extract (lambda (term term)
   (let
      (
        (term-p '())
        (term-len #f)
        (layer1-p #f)
        (body-p '())
        (resistance #f)
        (L #f)
        (W #f)
```

```
          (sheet_res 1)
          (dev-id #f)
          (proc_layers (ev-dev-processing-layers-get))
        )
      ;get the polygons for the resistor "marker" layer
      (set! body-p (ev-dev-polygons-get (cadr proc_layers)))

      (do () ((null? body-p) #t)
         (set! layer1-p (ev-dev-polygon-select (car proc_layers)
           'INTERACT (car body-p)))
         (set! term-p (ev-dev-polygon-select term 'INTERACT (car
           body-p)))
         (set! term-len (length term-p))
         ;Check if there are two terminals connected to the device.
         (cond
            ((> term-len 2)
               (ev-dev-error-message 'EXTRA_TERM
                   (car body-p) (- term-len 2)))
            ((< term-len 2)
               (ev-dev-error-message 'MISSING_TERM
                   (car body-p) (- 2 term-len)))
            ((= term-len 2)
                (begin
                ; calculate the length and width of the resistor,
                ; and find the resistance
                   (set! W (/ (ev-measure2 body-p 'COIN_PERIM
                      layer1-p) 2))
                   (set! L (/ (- (ev-measure1 'PERIM
                      (car body-p)) (* W 2)) 2))
                   (set! resistance (* sheet_res (/ L W)))

                ;create the device with two terminals
                   (set! dev-id (ev-dev-device-create (list
                      (car term-p) (cadr term-p))))
                   (ev-dev-property-value-set dev-id "rval"
                      resistance)
                   ;store the device
                   (ev-dev-device-write dev-id)
                  ));end begin and last condition
         );end conditional
      ;move on to next device
         (set! body-p (cdr body-p))
      );end do loop
(ev-dev-output-set (ev-dev-polygons-get (cadr proc_layers)))
(ev-dev-error-output-set (ev-dev-polygons-get (cadr proc_layers)))

))) ; End of define, lambda & let
```

*Step 2*

```
HEADER {
   SCHEME_FILE = extract.scm
}
```

*Step 3*

```
GENDEV gen_res1 term term {
     INITIALIZATION_FUNC = res_init
     EXTRACTION_FUNC = res_extract
     PROCESSING_LAYERS = {layer1 rbody}
} TEMP = gendev_res
```

For two different term layers, follow these three steps:

*Step 1*

*Example 8-4*

```
;Initialization function for a simple resistor
;resistance (rval) is the only extracted property

(define res_init (lambda ()
        (ev-dev-property-type-set "rval" 'DOUBLE)
))

;Extraction function for a simple resistor.  The device is made up
;by a resistor material layer and a resistor marker layer that
;defines the body of the device.
(define res_extract (lambda (term1 term2)
   (let
      (
        (term1-p '())
        (term2-p '())
        (term1-len #f)
        (term2-len #f)
        (layer1-p #f)
        (body-p '())
        (resistance #f)
        (L #f)
        (W #f)
        (sheet_res 1)
        (dev-id #f)
        (proc_layers (ev-dev-processing-layers-get))
      )
      ;get the polygons for the resistor "marker" layer
      (set! body-p (ev-dev-polygons-get (cadr proc_layers)))

      (do () ((null? body-p) #t)
         (set! layer1-p (ev-dev-polygon-select (car proc_layers)
           'INTERACT (car body-p)))
         (set! term1-p (ev-dev-polygon-select term1 'INTERACT
           (car body-p)))
         (set! term2-p (ev-dev-polygon-select term2 'INTERACT
           (car body-p)))
         (set! term1-len (length term1-p))
         (set! term2-len (length term2-p))
      ;Check if there are 2 terminals connected to the device.
```

```
          (cond
             ((or (> term1-len 1) (> term2-len 1))
                 (ev-dev-error-message 'EXTRA_TERM
                (car body-p) (- (+ term1-len term2-len) 2)))
             ((or (< term1-len 1) (< term2-len 1))
                 (ev-dev-error-message 'MISSING_TERM
                (car body-p) (- 2 (+ term1-len term2-len))))
             ((and (= term1-len 1) (= term2-len 1))
                 (begin
                  ;calculate the length and width of the resistor,
                and find the resistance
                     (set! W (/ (ev-measure2 body-p 'COIN_PERIM
                layer1-p) 2))
                     (set! L (/ (- (ev-measure1 'PERIM
                (car body-p)) (* W 2)) 2))
                     (set! resistance (* sheet_res (/ L W)))

                     ;create the device with two terminals
                     (set! dev-id (ev-dev-device-create (list
                (car term1-p) (car term2-p)))))
                     (ev-dev-property-value-set dev-id "rval"
                resistance)
                     ;store the device
                     (ev-dev-device-write dev-id)

                  ));end begin and last condition
           );end conditional
           ;move on to next device
           (set! body-p (cdr body-p))
        );end do loop
(ev-dev-output-set (ev-dev-polygons-get (cadr proc_layers)))
(ev-dev-error-output-set (ev-dev-polygons-get (cadr proc_layers)))

))) ; End of define, lambda & let
```

### *Step 2*

```
HEADER {
   SCHEME_FILE = extract.scm
}
```

### *Step 3*

```
GENDEV gen_res1 term1 term2 {
     INITIALIZATION_FUNC = res_init
     EXTRACTION_FUNC = res_extract
     PROCESSING_LAYERS = {layer1 rbody}
} TEMP = gendev_res
```

## Device Initialization Routine

The ev-dev-property-type-set routine is used in both AUTO and MANUAL modes.

### ev-dev-property-type-set

This routine is only useful in the context of the initialization function. All properties on devices that the current GENDEV command is going to extract are defined here. The property name is a string identifying the property, and the property type defines what type of value this property can receive. The currently supported types are DOUBLE and STRING. It is used in both AUTO and MANUAL modes. The syntax is:

```
GENDEV gfet poly nsd nsd {initialization_func = init
   extract_func = tgate } TEMP=ngates
```

For example:

```
(ev-dev-property-type-set "length" 'DOUBLE)   ==> #t

(ev-dev-property-type-set "id_tag" 'STRING)   ==> #t
```

## Device Definition Routines

Device definition routines form the core of what is done in a GENDEV command. By contrast, all other routines are used to obtain information that allows for calls of the following three routines in a meaningful way.

- The ev-dev-property-value-set routine is used in both AUTO and MANUAL modes.
- The ev-dev-device-create and ev-dev-device-write routines are MANUAL mode only.

### ev-dev-property-value-set

Once a device has been defined, its property values can be set using this routine. All properties whose types were defined in the initialization function are allowed to be used here. This routine is used in both AUTO and MANUAL modes.

In AUTO mode, the device-id is simply the device name on the GENDEV command line. In MANUAL mode, device-id should be created by the call of ev-dev-device-create first, signifying to which device to attach the property. The syntax is:

(ev-dev-property-value-set *device-id property-name property-value*)

==> #t | #f

For example:

```
(ev-dev-property-value-set mos-dev "width" w-val)       ==> #t

(ev-dev-property-value-set mos-dev "id-tag" id-string)  ==> #t
```

### ev-dev-device-create

This routine creates a device made up of the input polygons. The input polygons must be in a list, regardless of how many there are. A device-id is returned by this function, which is then used to pass into the following two routines. This routine is used in MANUAL mode only. The syntax is:

```
(ev-dev-device-create polgon-id-list)
```

==> device-id

### ev-dev-device-write

This is the final step in GENDEV device definition. It causes the device to be written into the database. Any device that is created and has properties set, but is not written, will not be saved. This routine requires the device-id created by ev-dev-device-create as input. It also requires all property values to be set prior to its call. This routine is used in MANUAL mode only. The syntax is:

```
(ev-dev-device-write device-id)
```

==> #t | #f

## Output Data Routines

Output data routines produce output from the GENDEV command other than the actual devices that are being executed. This includes writing data to the output layer, writing data to the error layer, and reporting errors in the *block*.err file.

The ev-dev-output-set, ev-dev-error-output-set, and ev-dev-error-message routines are used only in MANUAL mode.

### ev-dev-output-set

This routine is used to specify the polygons written to the output layer. The name and layer number of the output layer is defined in the runset, and this is the only way to get any data on that layer. This routine expects a list of polygon-ids, and that list must contain all the output polygons. If this routine is called more than once with different polygons, the polygons from the last call will be the only ones saved. It is used only in MANUAL mode. The syntax is:

```
(ev-dev-output-set polygon-id-list)
```

==> #t | #f

### ev-dev-error-output-set

This routine is used to specify the polygons written to the error layer. The name and layer number of the error layer is defined in the runset, and this is the only way to get any data on that layer. This routine expects a list of polygon-ids, and that list must contain all the output polygons. If this routine is called more than once with different polygons, the polygons from the last call will be the only ones saved. It is used only in MANUAL mode. The syntax is:

```
(ev-dev-error-output-set polygon-id-list)
```

==> #t | #f

### ev-dev-error-message

This routine provides the ability to report extraction errors from the GENDEV Scheme code. The first parameter sets the message type. This is restricted to MISSING_TERMS or EXTRA_TERMS. The second parameter is a polygon to identify the error. The third parameter is an integer indicating how many extra or missing terminals this device has. It is used only in MANUAL mode. The syntax is:

```
(ev-dev-error-message msg-type polygon-id term-num)
```

==> #t | #f

For example:

```
(ev-dev-error-message 'EXTRA_TERMS gate-polygon 4))  ==> #t
```

## Data Manipulation Routines

Calls to the data manipulation routines should precede calls to the device definition routines. Data manipulation routines control the design data so that the required information is found for defining devices. This information is used in the device definition routines. The data manipulation routines isolate the device polygons and obtain polygon parameters for the device property values.

- The ev-dev-edges-get, ev-dev-area-get, ev-dev-coordinates-get, ev-dev-polygon-select, ev-dev-polygon-boolean, ev-dev-spacing-get, and ev-dev-cell-name-get routines are used in both AUTO and MANUAL modes.
- The ev-dev-processing-layers-get and ev-dev-polygons-get routines are MANUAL mode only.

Note:
Be aware that in AUTO mode all variables named by the same layer_names in the GENDEV command lines are a polygon_list, with one exception: the recognition_layer_name variable denotes the current cell recognition polygon. The polygon_lists are the polygons which currently interact with the recognition polygon. Be sure to implement the right scheme object type to the Data Manipulation Routines.

## ev-dev-edges-get

Similar to the ev-dev-polygons-get routine, this routine gets edges off a polygon instead of polygons off a layer. All polygons will have at least three edges, with the notable exception of vector polygons. When this routine is called with a vector polygon passed in, a list with one edge is returned. That edge defines the vector. This routine is used in both AUTO and MANUAL modes. The syntax is:

```
(ev-dev-edges-get polygon-id)
```

==> (edge-id1 ... edge-idN)

For example:

```
(set! polygon-edges (ev-dev-edges-get (car gate-polygons)))
   ==> polygon-edges
```

## ev-dev-area-get

This routine returns the area of the specified polygon. This routine is used in both AUTO and MANUAL modes. The syntax is:

```
(ev-dev-area-get polygon-id)
```

==> exact

For example:

```
(set! gate-area (ev-dev-area-get (car gate-polygons))) ==> 16.0
```

## ev-dev-coordinates-get

An input edge-id is required for this routine, and the coordinates of the endpoints are returned. Those coordinates are returned as a list of exactly four numbers. Those numbers are organized as follows: (x1 y1 x2 y2). This routine is used in both AUTO and MANUAL modes. The syntax is:

```
(ev-dev-coordinates-get edge-id)
```

==> (x1 y1 x2 y2)

For example:

```
(set! edge-coord (ev-dev-coordinate-get (car polygon-edges)))
   ==> (0 0 0 10)
```

## ev-dev-polygon-select

This routine provides a mechanism to select polygons based on interaction relationships. This routine performs differently than the runset SELECT command. This is not a layer-to-layer command; it is a polygon-to-polygon, or layer-to-polygon, routine. Also, this operation takes place only for data within the same cell, not across the hierarchy.

The second parameter determines the select criteria. The choices are OVERLAP, INTERACT, EDGETOUCH, and ENCLOSED-BY. The third parameter must be a polygon-id, and becomes the polygon all interaction will be checked against. The first parameter can be either a single polygon (polygon-id) or a layer (layer-id).

This routine always returns a list of polygons, regardless of what the first parameter is. However, it is easier to understand if they are dealt with separately.

Polygon versus Polygon: The select routines return either a list of one polygon (the first parameter) or an empty list, depending on whether the specified relationship was satisfied.

Layer versus Polygon: The select routines return a list of all the polygons from the input layer which satisfy the specified relationship. This can be an empty list if none satisfy that condition. This routine is used in both AUTO and MANUAL modes. The syntax is:

(ev-dev-polygon-select *data-id operator polygon-id*)

==> (polygon-id1 ... polygon-idN)

For example:

```
(set! select-result (ev-dev-polygon-select (car gate-polygons)
   'OVERLAP (car bulk-polygons)))   ==> select-result

(set! select-result (ev-dev-polygon-select gate-layer
  'OVERLAP (car bulk-polygons)))   ==> select-result
```

## ev-dev-polygon-boolean

This routine is very similar to ev-dev-polygon-select. The first parameter is either a layer-id or a polygon-id. The second parameter is the operator, but the only two Boolean operations that are supported are AND and NOT. The third parameter is a single polygon-id. The resultant polygons from the Boolean operation will be returned in a list. This routine is used in both AUTO and MANUAL modes. The syntax is:

(ev-dev-polygon-boolean *data-id operator polygon-id*)

==> (polygon-id1 ... polygon-idN)

For example:

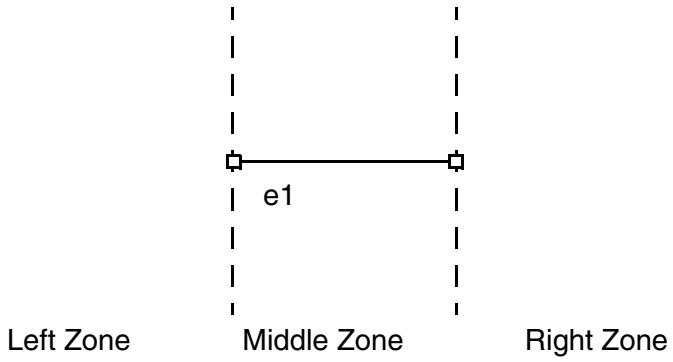
```
(set! boolean-result (ev-dev-polygon-boolean (car gate-polygons)
            'AND (car bulk-polygons)))   ==> boolean-result

(set! boolean-result (ev-dev-polygon-boolean gate-layer
            'NOT (car bulk-polygons)))   ==> boolean-result
```

## ev-dev-spacing-get

This routine finds the minimum distance between a given edge and an edgelist. This is accomplished by cycling through all the edges in the edgelist and comparing each one to the given edge. This routine is used in both AUTO and MANUAL modes.

For example, let two edges be called e1 and e2. The following procedure is used to find the distance between e1 and e2.

1. Make a projection perpendicular to e1 from each endpoint of e1.




   The projections extend infinitely on both sides of e1 and create three zones called the left, middle, and right zones. For each endpoint of e2 that lies within the middle zone of e1, find the distance perpendicular from e1 to that point.

2. Repeat the first step, this time projecting from e2 and looking for e1 endpoints.
3. If neither edge has an endpoint lying in the middle zone of the other, find the four point-to-point distances from each endpoint of e1 to the endpoints of e2.

After cycling through the edges of the given edgelist and comparing each one to the given edge, return the minimum distance found. The syntax is:

```
(ev-dev-edge-spacing-get edge-id edge-list)
```

==> exact

For example:

```
(set! edge-spacing (ev-dev-spacing-get current-edge
   (ev-dev-edges-get (car gate-polygons))))  ==> 28.25
```

## ev-dev-cell-name-get

This routine accepts no parameters, but returns a string identifying the current cell that is being operated on by the GENDEV command. This is useful for debugging if a problem has been isolated to a particular cell. This routine is used in both AUTO and MANUAL modes. The syntax is:

```
(ev-dev-cell-name-get)
```

==> string

For example:

```
(set! cell-name (ev-dev-cell-name-get)) ==> "CHIP"
```

## ev-dev-processing-layers-get

The GENDEV command allows specification of layers in addition to the layers that define the device terminals. These extra layers are called processing layers, and may be used to identify the input terminal layers which make up a device. A BOOLEAN or SELECT command could be used to find whether or not a terminal layer is interacted with a processing layer.

In order to get layer-ids for the processing layers into the Scheme function, this routine must be called. It returns a list of layer-ids, which can then be used just like the layer-ids from the terminal layer that were passed in as parameters. However, polygons from these layers can never be used to define a device in calls to ev-dev-device-create. It is used only in MANUAL mode. The syntax is:

```
(ev-dev-processing-layers-get)
```

==> (layer-id1 ... layer-idN)

For example:

```
(set! processing-layers (ev-dev-processing-layers-get))
   ==> processing-layers
```

### ev-dev-polygons-get

When called with a layer-id as the input parameter, this routine returns a list of polygon-ids. The polygons will be every polygon on this layer in the current cell. They will always be returned in a list, regardless of how many (or how few) there are. These polygons can then be analyzed individually to determine which are forming devices and which are not. It is used only in MANUAL mode. The syntax is:

```
(ev-dev-polygons-get layer-id)
```

==> (polygon-id1 ... polygon-idN)

For example:

```
(set! gate-polygons (ev-dev-polygons-get gate-layer))
   ==> gate-polygons
```

## Data Measurement Routines

### ev-measure

The ev-measure routines measure the required properties designated by the OPERATOR of inputs for layers, polygon lists, or polygons. These routines can be used in both the AUTO mode or MANUAL mode. If the property of only one input is desired, ev-measure1 would be used; *ev-measure2* is designed to get interaction measurements of two inputs. The syntax is:

```
(ev-measure1 'OPERATOR input) (ev-measure2 input1 'OPERATOR input2)
```

| Argument | Description |
|---|---|
| *input* | Could be a polygon, polygon list, or layer |
| *input1, input2* | Could be a polygon or polygon list. (Note: In AUTO mode, all the *layer_name* variables already represent a polygon list or polygon.) |

For ev-measure1, the allowed operators are:

| Operator | Description |
|---|---|
| 'PERIM | Returns the perimeter of input |
| 'AREA | Returns the area of input |

| Operator | Description |
|---|---|
| 'BENDS | Counts all convex bends of input in 90-degree units (45 bends count as 0.5 bends; convex here and the following means the corner that points to the inside of a polygon) |
| 'BENDS90 or 'CONVEX90 | Counts only 90-degree bends |
| 'BENDS45 or 'CONVEX135 | Counts only 45-degree bends (135-degree angle) |
| 'BENDS135 or 'CONVEX45 | Counts only 135-degree bends (45-degree angle) |
| 'COUNT | Returns the number of polygons in the input |

For ev-measure2, the allowed operators are:

| Operator | Description |
|---|---|
| 'INSIDE_PERIM | Returns perimeter of input1 inside input2, coincident edges excluded |
| 'OUTSIDE_PERIM | Returns perimeter of input1 outside input2, coincident edges excluded |
| 'COIN_PERIM | Returns perimeter of input1 coincident input2 |
| 'ABUT_PERIM | Returns perimeter of input1 coincident input2, where input1 is outside input2 |
| 'INSIDE_AREA | Returns area of input1 inside input2 |
| 'OUTSIDE_AREA | Returns area of input1 outside input2 |
| 'COUNT_INSIDE | Returns the number of polygons of input1 fully inside input2 |
| 'COUNT_OUTSIDE | Returns the number of polygons of input1 fully outside input2 |

# EXPLODE Language

The EXPLODE language writes general Scheme code that describes which cells and data to explode. This section refers to exploding cells, which can include EXPLODE, FLATTEN, DELETE, EXPLODE_ALL, and NO_EXPLODE cell processing. For further information, see the EXPLODE_OPTIONS section in the *Hercules Reference Manual*, Detailed Options chapter.

The runset-defined routine called to determine exploded cells is called once for every cell in the design. This runset-defined routine must be written to accept one parameter, which is the current cell (cell-id). EXPLODE operations can be performed on cells or on layers within a cell.

Within the runset-defined routine, there are many routines available to determine whether or not a cell or layer should be exploded. Several of those routines may accept either cell-ids or layer-ids as input, and are therefore generic routines that handle either type of data.

Follow the steps below to set up the EXPLODE language.

*Step 1:* Create a Scheme file that contains the procedure used to determine what to explode and how to explode it. This function must receive only one parameter, which is a cell-id (a data structure containing all the available information about the current cell). This function is called once for each cell in the design. See Example 8-5.

*Example 8-5*

```
--- explode.scm ---

;; Explode cells which contain no real data, and have
;; 2 or less placements in them (holding cells)

(define explode-holding-cells (lambda (cell-id)
    (let (
        (num-placements 0)
        (num-polygons 0)
    )
    (set! num-placements (+ (ev-exp-sref-count-get cell-id)
        (ev-exp-aref-count-get cell-id)))
    (set! num-polygons (ev-exp-data-count-get cell-id))

    (if (and (=? num-polygons 0) (<? num-placements 3))
        (ev-exp-explode cell-id)
    )
)))
```

*Step 2:* Register the Scheme file in the HEADER section of the runset, as shown here:

```
--- runset.ev ---
```

```
HEADER {
    SCHEME_FILE = explode.scm
}
```

*Step 3:* Register the Scheme procedure in the EXPLODE_OPTIONS section of the runset.

```
--- runset.ev ---

EXPLODE_OPTIONS {
   scheme_func = explode-holding-cells
}
```

## Data Information Routines

The following section lists the available Data Information Routines that can be called by user-defined Scheme Explode routines.

### ev-exp-is-name-equal

This routine is used to test the name of a cell or layer. The first parameter can be either a cell-id or a layer-id. The second parameter is a string against which the name of this cell or layer will be tested. The test string does accept the asterisk (*) and question mark (?) as wildcards. The routine returns #t if the test-string matches the data name, and #f otherwise. The syntax is:

```
(ev-exp-is-name-equal data-id test-string)
```

==> #t | #f

For example:

```
(if (ev-exp-is-name-equal cell-id "MEM_CELL") ...  ==> #t

(if (ev-exp-is-name-equal layer-id "met*") ...     ==> #f
```

### ev-exp-is-non-orthogonal

This routine is used to test whether a cell or layer has any non-orthogonal data. The routine returns #t if the cell or layer has non-orthogonal data, and #f otherwise. The syntax is:

```
(ev-exp-is-non-orthogonal data-id)
```

==> #t | #f

For example:

```
(if (ev-exp-is-non-orthogonal cell-id)    ==> #f
```

```
(if (ev-exp-is-non-orthogonal layer-id)   ==> #t
```

## ev-exp-name-get

The routine returns that name associated with the cell or layer that is passed in. The syntax is:

```
(ev-exp-name-get data-id)
```

==> string

For example:

```
(set! cell-name (ev-exp-name-get cell-id))      ==> "CHIP"

(set! layer-name (ev-exp-name-get layer-id))    ==> "metal1"
```

## ev-exp-area-get

This routine returns that area of the cell or layer that is passed in. For cells, it is the area of the bounding box created from all layers in the cell. For layers, it is the area of that layer within the current cell calculated by adding all individual areas of each polygon together. The syntax is:

```
(ev-exp-area-get data-id)
```

==> number

For example:

```
(set! cell-area (ev-exp-area-get cell-id))      ==> 128860.0

(set! layer-area (ev-exp-area-get layer-id))    ==> 458.2
```

## ev-exp-top-cell-area-get

This routine returns the area of the top cell in the design, as signified by the BLOCK field in the HEADER section of the runset. The syntax is:

```
(ev-exp-top-cell-area-get)
```

==> number

For example:

```
(set! top-cell-area (ev-exp-top-cell-area-get))  ==> 4486900.0
```

## ev-exp-data-count-get

This routine accepts either a cell-id or a layer-id as input. If passed a cell-id, it returns the total number of polygons, rectangles, and paths in that cell. If passed a layer-id, it returns the total number of polygons, rectangles, and paths on that layer in the current cell. The syntax is:

```
(ev-exp-data-count-get data-id)
```

==> number

For example:

```
(set! cell-data-count (ev-exp-data-count-get cell-id))  ==> 102

(set! layer-data-count (ev-exp-data-count-get layer-id)) ==> 18
```

## ev-exp-text-count-get

This routine accepts either a cell-id or a layer-id, and returns the total number of text placements in that cell, or on that layer in the current cell. The syntax is:

```
(ev-exp-text-count-get data-id)
```

==> number

For example:

```
(set! cell-text-count (ev-exp-text-count-get cell-id))   ==> 57

(set! layer-text-count (ev-exp-text-count-get layer-id)) ==> 6
```

## ev-exp-layer-list-get

This routine anticipates a cell-id and an optional test string, and will return a list of layer-ids. If no test string is given, the list returned will contain every layer that exists in this cell. If a test string is specified, only the layers in this cell which match the test string are returned. Wildcards are accepted in the test string. The syntax is:

```
(ev-exp-layer-list-get cell-id (test-string))
```

==> (layer-id1 ... layer-idN)

For example:

```
(set! layers (ev-exp-layer-list-get cell-id "met*"))
   ==> (layer-id)
```

## ev-exp-sref-count-get

This routine accepts a cell-id and returns the total number of SREFs in that cell. The syntax is:

```
(ev-exp-sref-count-get cell-id )
```

==> number

For example:

```
(set! cell-sref-count (ev-exp-sref-count-get cell-id))   ==> 5
```

## ev-exp-aref-count-get

This routine accepts a cell-id and returns the total number of AREFs in that cell. The syntax is:

```
(ev-exp-aref-count-get cell-id)
```

==> number

For example:

```
(set! cell-aref-count (ev-exp-aref-count-get cell-id))  ==> 1
```

## ev-exp-aref-as-sref-count-get

This routine accepts a cell-id and returns the total number of AREF placements in that cell, counting the AREFs as if they had been placed as SREFs. The syntax is:

```
(ev-exp-aref-as-sref-count-get cell-id)
```

==> number

For example:

```
(set! cell-aref-count (ev-exp-aref-as-srefcount-get cell-id))
   ==> 8
```

## ev-exp-hier-placement-count-get

This routine accepts a cell-id and returns the total number of times that cell is placed hierarchically throughout the design. The syntax is:

```
(ev-exp-hier-placement-count-get cell-id)
```

==> number

For example:

```
(set! cell-hier-place-count
   (ev-exp-hier-placement-count-get cell-id))   ==> 1
```

## ev-exp-flat-placement-count-get

This routine accepts a cell-id and returns the total number of times that cell would be placed if the entire design was flat. The syntax is:

```
(ev-exp-hier-placement-count-get cell-id)
```

==> number

For example:

```
(set! cell-hier-place-count
   (ev-exp-hier-placement-count-get cell-id))      ==> 1
```

## ev-exp-explode

This routine accepts a cell-id. The specified cell is exploded. It will return #t if successful, and #f otherwise.

Syntax:

```
(ev-exp-explode cell-id)
```

==> #t | #f

For example:

```
(ev-exp-explode cell-id)    ==> #t
```

## ev-exp-flatten

This routine accepts a cell-id. The specified cell is exploded. It will return #t if successful, and #f otherwise.

```
(ev-exp-flatten cell-id)
```

==> #t | #f

For example:

```
(ev-exp-flatten cell-id)    ==> #t
```

## ev-exp-delete

This routine accepts a cell- or layer-id, followed by an optional data type. If no data type is given, the specified cell or layer will be deleted. The optional data type is only valid for cells, and allows for the following types of data to be deleted in that cell: DATA, TEXT, SREF, and AREF. It will return #t if successful, and #f otherwise. The syntax is:

```
(ev-exp-delete data-id [data-type])
```

==> #t | #f

For example:

```
(ev-exp-delete cell-id 'TEXT)   ==> #t

(ev-exp-delete layer-id)        ==> #t
```

## ev-exp-explode-all

This routine accepts a cell- or layer-id, followed by an optional data type. If no data type is given, the specified cell or layer will have the EXPLODE_ALL operation performed on it. The optional data type is only valid for cells, and allows for the following types of data to be operated on in that cell: DATA, TEXT, SREF, and AREF. It will return #t if successful, and #f otherwise. The syntax is:

```
(ev-exp-explode-all  data-id [data-type])
```

==> #t | #f

For example:

```
(ev-exp-explode-all cell-id 'TEXT)   ==> #t

(ev-exp-explode-all layer-id)        ==> #t
```

## ev-exp-no-explode

This routine accepts a cell- or layer-id and that cell or layer will have the NO_EXPLODE operation performed on it. It will return #t if successful and #f otherwise. The syntax is:

```
(ev-exp-no-explode data-id)
```

==> #t | #f

For example:

```
(ev-exp-no-explode cell-id 'TEXT)    ==> #t

(ev-exp-no-explode layer-id)         ==> #t
```

# LVS User-Defined Property Comparison

LVS User-Defined Property Comparison allows you to use the Scheme interface to Hercules to alter the default mechanism for comparing device properties. You can use this feature to perform property comparisons that are not currently handled by default, such as comparing properties that are strings, or checking user-defined properties on merged devices.

For each EQUATE command, a Scheme procedure can be registered for use by Hercules to define comparison of device properties.

*Step 1:* Create a Scheme file containing the procedures to be used to compare device properties.The property comparison function must receive three parameters:

- An identifier for a schematic device (sch-id)
- An identifier for the corresponding layout device (lay-id)
- A string containing the property name

Unlike other Scheme interfaces to Hercules, these functions' return values are important to Hercules. If the function returns #t, the properties compared correctly. If the function returns #f, the properties were not compared by this function, and the default comparison should be used. If the function returns a string, that string is the error to be reported because this property does not match. See Example 8-6.

*Example 8-6*

```
;; Compare the string property "type" on MOS devices. Only the first
;; four characters must match (pmos or nmos), so that differing
;; suffixes are not reported.

(define prop_comp (lambda (sch-id lay-id prop)
    (call-with-current-continuation (lambda (return)
    (let (

   (sch-type #f) (lay-type #f)

        ;; Get the property values for the specified property
        (sch-prop (lvs-inst-prop-get sch-id prop))
        (lay-prop (lvs-inst-prop-get lay-id prop))
   (sch-name (lvs-inst-name-get sch-id))
   (lay-name (lvs-inst-name-get lay-id))
   (is-mos (or (lvs-inst-is-pmos sch-id)
   (lvs-inst-is-nmos sch-id)))
    )

    ;; Only compare "type" property for MOS devices with this routine.
    ;; Use the default for everything else.
    (if (not is-mos)
    (return #f)
```

```
    )

    ;; Make sure property values exist for this property on both devices
    (if (not sch-prop)
         (return (format "~s: No schematic property '~s' found"
          sch-name prop))
    )

    (if (not lay-prop)
         (return (format "~s: No layout property '~s' found"
          lay-name prop))
    )

    ;; Get the first 4 characters off the string property value
    (set! sch-type (substring (car sch-prop) 0 4))
    (set! lay-type (substring (car lay-prop) 0 4))

    ;; Test if those 4 characters are identical.
    (if (string=? sch-type lay-type)
    #t
    ;; else
        (return (format "~a=~a: Property ~s mismatch. ~s != ~s\n"
         sch-name lay-name prop (car sch-prop) (car lay-prop)))
    )
)))))
```

*Step 2:* Register the Scheme file in the HEADER section of the Hercules runset, as shown here:

```
--- runset.ev ---

HEADER {
   scheme_file = extract.scm
}
```

*Step 3:* Register the Scheme procedures with the EQUATE commands in the Hercules runset, as shown here:

```
--- runset.ev ---

EQUATE NMOS n=n G S D B {
   check_user_properties = {name}
   comp_prop_func[name] = name-compare
}
```

The available Hercules routines are explained in the following listings.

Note:

The parameter inst-id is passed into the Scheme function by Hercules, and prop-names are strings.

## LVS Routines

The following section lists the available LVS Routines that can be called by user-defined Property Comparison Scheme routines.

### lvs-connection-inst-id-get

Returns inst-id by accepting a conn-id. Returns #f if the specified id is not connected to any device. The syntax is:

```
(lvs-connection-inst-id-get conn-id)
```

==> instid | #f

For example:

```
(set! inst-name (lvs-inst-name-get (
   lvs-connection-inst-id-get conn-id)) )     ==> "M2"
```

### lvs-inst-name-get

Returns the instance name. The syntax is:

```
(lvs-inst-name-get inst-id)
```

==> string

For example:

```
(set! inst-name (lvs-inst-name-get inst-id))   ==> "M1"
```

### lvs-inst-primitive-name-get

Returns the instance primitive name. The syntax is:

```
(lvs-inst-primitive-name-get inst-id)
```

==> string

For example:

```
(set! prim-name (lvs-inst-primitive-name-get inst-id))   ==> "nmos"
```

### lvs-inst-prop-get

Returns a list of property values, as exact numbers, associated with the given property name. The syntax is:

```
(lvs-inst-prop-getinst-id prop-name)
```

==> (value1 ... valueN)

For example:

```
(set! prop-list (lvs-inst-prop-get inst-id "w"))     ==> (4.0)
```

## lvs-inst-member-get

Returns a list of member instance ids for merged devices. Returns #f if specified id is not a merged device. The syntax is:

```
(lvs-inst-member-get inst-id)
```

==> (inst-id1 ... inst-idN)

For example:

```
(set! member-list (lvs-inst-member-get inst-id))
   ==> (inst-id1 ... inst-idN)
```

## lvs-inst-tolerance-get

Returns the plus and minus tolerance for a given property on the specified instance. The first element on the list is the plus tolerance, the second is the minus. The syntax is:

```
(lvs-inst-tolerance-get inst-id prop-name)
```

==> (value value)

For example:

```
(set! toler (lvs-inst-tolerance-get inst-id "w"))    ==> (5 10)
```

## lvs-inst-tolerance-is-absolute

Returns #t if the property tolerance is absolute, and #f if it is relative. The syntax is:

```
(lvs-inst-tolerance-is-absolute inst-id prop-name)
```

==> #t | #f

For example:

```
(if (lvs-inst-tolerance-is-absolute inst-id "w") ...   ==> #t
```

## lvs-inst-tolerance-is-relative

Returns #t if the property tolerance is relative, and #f if it is absolute. The syntax is:

```
(lvs-inst-tolerance-is-relative inst-id prop-name)
```

==> #t | #f

For example:

```
(if (lvs-inst-tolerance-is-relative inst-id "w") ...   ==> #f
```

## lvs-inst-is-parallel

Returns #t if the instance is a parallel merged device and #f if it is not. The syntax is:

```
(lvs-inst-is-parallel inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-parallel inst-id) ...   ==> #t
```

## lvs-inst-is-series

Returns #t if the instance is a series merged device and #f if it is not. The syntax is:

```
(lvs-inst-is-series inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-series inst-id) ...      ==> #t
```

## lvs-inst-is-parallel-chain

Returns #t if the instance is a parallel chain merged device and #f if it is not. The syntax is:

```
(lvs-inst-is-parallel-chain inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-parallel-chain inst-id) ...   ==> #t
```

## lvs-inst-is-extracted

Returns #t if the instance is an extracted device (not a merged composite) and #f if it is not. The syntax is:

```
(lvs-inst-is-extracted inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-extracted inst-id) ...   ==> #t
```

## lvs-inst-is-member

Returns #t if the instance is a member of a merged composite device and #f if it is not. This routine is not intended to indicate whether a device is a merged composite device itself; rather, it should be used to indicate whether the device is part of a merged composite device (such as the device instances returned by the lvs-inst-member-get routine). The syntax is:

```
(lvs-inst-is-member inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-member inst-id) ...   ==> #t
```

## lvs-inst-is-nmos

Returns #t if the instance is an NMOS device and #f if it is not. The syntax is:

```
(lvs-inst-is-nmos inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-nmos inst-id) ...   ==> #t
```

## lvs-inst-is-pmos

Returns #t if the instance is a PMOS device and #f if it is not. The syntax is:

```
(lvs-inst-is-pmos inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-pmos inst-id) ...   ==> #t
```

## lvs-inst-is-cap

Returns #t if the instance is a CAP device and #f if it is not. The syntax is:

```
(lvs-inst-is-cap inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-cap inst-id) ...    ==> #t
```

## lvs-inst-is-res

Returns #t if the instance is a RES device and #f if it is not. The syntax is:

```
(lvs-inst-is-res inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-res inst-id) ...   ==> #t
```

## lvs-inst-is-npn

Returns #t if the instance is an NPN device and #f if it is not. The syntax is:

```
(lvs-inst-is-npn inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-npn inst-id) ...   ==> #t
```

## lvs-inst-is-pnp

Returns #t if the instance is a PNP device and #f if it is not. The syntax is:

```
(lvs-inst-is-pnp inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-pnp inst-id) ...   ==> #t
```

## lvs-inst-is-np

Returns #t if the instance is a DIODE device and #f if it is not. The syntax is:

```
(lvs-inst-is-np inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-np inst-id) ...   ==> #t
```

### lvs-inst-is-pn

Returns #t if the instance is a DIODE device and #f if it is not. The syntax is:

```
(lvs-inst-is-pn inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-pn inst-id) ...   ==> #t
```

### lvs-inst-is-generic

Returns #t if the instance is a generic device and #f if it is not. The syntax is:

```
(lvs-inst-is-generic inst-id)
```

==> #t | #f

For example:

```
(if (lvs-inst-is-generic inst-id) ...   ==> #t
```

## Advanced Filtering

Scheme advanced filtering allows the specification of complex filtering criteria for both primitive devices and equivalent cells. LVS COMPARE supports many filtering options for primitives devices, but the Scheme advanced filtering extensions provide further flexibility and capability for non-standard filtering needs.

For each EQUATE command, a Scheme procedure must be registered for use by Hercules to define the filtering criteria.

*Step 1:* Create a Scheme file containing the procedures that will filter devices. The filtering function must receive one parameter: an identifier for the device being considered for filtering. The Scheme function must also return a Boolean value: #t indicating that the device will be filtered, and #f indicating a non-filtered device. See Example 8-7.

*Example 8-7*

```
--- filter.scm ---
```

```
;; Filter all MOS devices whose gate connection
;; is only connected to other gates; no source or drain
;; connections on the net.
(define mos-filter (lambda (inst-id)
    (let (
        (pin-list (lvs-pins-get inst-id 'GATE))
        (filter #t)
        (conn-list #f)
        (conn-type #f)
    )
    (set! conn-list (lvs-connections-get (car pin-list)))

;; Drop out of the loop if the list is done or
;; have already determined that something is not
;; connected to a GATE.
    (do () ((or (null? conn-list) (not filter)) #t)
    (set! conn-type (lvs-connection-type-get (car conn-list)))

        (if (not (equal? conn-type 'GATE))
            (set! filter #f)
        )
        (set! conn-list (cdr conn-list))
    )

    filter
)))

;; Filter all equivalent cells for which all the output pins
;; are shorted together
(define equiv-filter (lambda (inst-id)
    (let (
        (pin-list (lvs-pins-get inst-id))
        (filter #t)
        (first-pin #f)
    )
    (set! first-pin (car pin-list))
    (set! pin-list (cdr pin-list))

    ;; Drop out of the loop if the list is
    ;; done or have already determined
    ;; that something is not shorted.
    (do () ((or (null? pin-list) (not filter)) #t)
      (if (not (lvs-are-pins-shorted first-pin (car pin-list)))
            (set! filter #f)
      )
        (set! pin-list (cdr pin-list))
    )

    filter
)))
```

*Step 2:* Register the Scheme file in the HEADER section of the Hercules runset, as shown here.

```
--- runset.ev ---

HEADER {
    SCHEME_FILE = filter.scm
}
```

*Step 3:* Register the Scheme procedures with the EQUATE commands in the runset and with the EQUIV commands in the equivalence file.

```
--- runset.ev ---

EQUATE NMOS n=n G S D B {
   filter_FUNC = mos-filter
   filter_options = {NMOS-1}
}

--- equiv_file ---

EQUIV inv=INV {
   filter_FUNC = equiv-filter
}
```

## Filtering Routines

The following section lists the available Filtering Routines that can be called by user-defined Filtering Scheme routines.

### lvs-pins-get

This routine accepts a device ID and returns the list of pin IDs connected to this device. Those pin IDs are then used in subsequent routines. An optional pin type is allowed as the second parameter, to limit the pin IDs returned to only those of the desired type. The types allowed are shown in Table 8-1. The syntax is:

```
(lvs-pins-get dev-id [pin-type])
```

==> (pin-id ... )

For example:

```
(set! pin-list (lvs-pins-get dev-id 'S/D))  ==> (pin-id pin-id)
```

## lvs-connection-count-get

This routine accepts a pin ID and returns the number of other connections made by the net connected to that pin. No connection is counted for the connection to the pin in question, and, if the net is a port, one additional connection is added. For example, the GATE pin on an NMOS device is connected to net IN, which has 3 other connections, and is a port from the cell (implying more connections). The count returned by this routine would be 4. The syntax is:

```
(lvs-connection-count-get pin-id)
```

==> number

For example:

```
(set! conn-count (lvs-connection-count-get (car pin-list)))
      ==> 4
```

## lvs-is-pin-net-ground

This routine accepts a pin ID, and returns #t if that pin is connected to ground, and #f otherwise. The syntax is:

```
(lvs-is-pin-net-ground pin-id)
```

==> #t | #f

## lvs-is-pin-net-power

This routine accepts a pin ID, and returns #t if that pin is connected to power, and #f otherwise. The syntax is:

```
(lvs-is-pin-net-power pin-id)
```

==> #t | #f

For example:

```
(if (lvs-is-pin-net-power pin-id) ...   ==> #t
```

## lvs-is-pin-net-port

This routine accepts a pin ID, and returns #t if that pin is connected to a port net, and #f otherwise. The syntax is:

```
(lvs-is-pin-net-port pin-id)
```

==> #t | #f

For example:

```
(if (lvs-is-pin-net-port pin-id) ...   ==> #t
```

## lvs-are-pins-shorted

This routine accepts two pin IDs, and returns #t if those pins are connected together (shorted), and #f otherwise. The syntax is:

```
(lvs-are-pins-shorted pin-id1 pin-id2)
```

==> #t | #f

For example:

```
(if (lvs-are-pins-shorted first-pin (car pin-list)) ...  ==> #t
```

## lvs-connections-get

This routine accepts a pin ID, and returns a list of identifiers for every other connection that pin net makes. The syntax is:

```
(lvs-connections-get pin-id)
```

==> conn-id1 ... conn-idN

For example:

```
(set! conn-list (lvs-connections-get pin-id))
   ==> (conn-id conn-id)
```

## lvs-connection-type-get

This routine accepts a connector ID, and returns its type. The types returned will be among those shown in Table 8-1. The syntax is:

```
(lvs-connection-type-get conn-id)
```

==> conn-type

For example:

```
(set! conn-type (lvs-connection-type-get conn-id))   ==> 'GATE
```

*Table 8-1 Optional Pin Types*

| Device type | Pin types |
|---|---|
| NMOS, PMOS | GATE, S/D, BULK |

*Table 8-1 Optional Pin Types(Continued)*

| Device type | Pin types |
|---|---|
| NPN, PNP | BASE, EMIT, COLL |
| NP, PN | ANODE, CATHODE |
| RES, CAP | A, B |
| DEV | T1, T2, ..., T10 |

## Available Scheme Routines

All the available Scheme routines are listed here.

- Device routines: See “Device Routines” on page 8-4.

  ev-devcon-dev-class-get
  ev-devcon-dev-conn-list
  ev-devcon-dev-conn-net-get
  ev-devcon-dev-is-netlisted
  ev-devcon-dev-is-parasitic
  ev-devcon-dev-is-shorted
  ev-devcon-dev-name-get
  ev-devcon-dev-prop-list
  ev-devcon-dev-prop-value-get
  ev-devcon-dev-type-get
  ev-devcon-net-name-get

- Device Initialization routine: See “Device Initialization Routine” on page 8-13.

  ev-dev-property-type-set

- Device Definition routines: See “Device Definition Routines” on page 8-13.

  ev-dev-device-create
  ev-dev-device-write
  ev-dev-property-value-set

- Output Data routines: See “Output Data Routines” on page 8-14.

  ev-dev-error-message
  ev-dev-error-output-set
  ev-dev-output-set

- Data Manipulation routines: See “Data Manipulation Routines” on page 8-15.

ev-dev-area-get
ev-dev-cell-name-get
ev-dev-coordinate-get
ev-dev-polygon-boolean
ev-dev-polygon-select
ev-dev-polygons-get
ev-dev-processing-layers-get
ev-dev-spacing-get

- Data Information routines: See "Data Information Routines" on page 8-23.

  ev-exp-area-get
  ev-exp-aref-as-sref-count-get
  ev-exp-aref-count-get
  ev-exp-data-count-get
  ev-exp-delete
  ev-exp-explode
  ev-exp-explode-all
  ev-exp-flat-placement-count-get
  ev-exp-flatten
  ev-exp-hier-placement-count-get
  ev-exp-is-name-equal
  ev-exp-is-non-orthogonal
  ev-exp-layer-list-get
  ev-exp-name-get
  ev-exp-no-explode
  ev-exp-text-count-get
  ev-exp-top-cell-area-get

- LVS routines: See "LVS Routines" on page 8-31.

  lvs-inst-is-cap
  lvs-inst-is-extracted
  lvs-inst-is-generic
  lvs-inst-is-member
  lvs-inst-is-nmos
  lvs-inst-is-np
  lvs-inst-is-npn
  lvs-inst-is-parallel
  lvs-inst-is-parallel-chain
  lvs-inst-is-pmos
  lvs-inst-is-pn
  lvs-inst-is-pnp
  lvs-inst-is-res
  lvs-inst-is-series
  lvs-inst-member-get

lvs-inst-name-get
lvs-inst-primitive-name-get
lvs-inst-prop-get
lvs-inst-tolerance-get
lvs-inst-tolerance-is-absolute
lvs-inst-tolerance-is-relative

- Advanced Filtering routines: See “Filtering Routines” on page 8-38.

lvs-pins-get
lvs-connection-count-get
lvs-is-pin-net-ground
lvs-is-pin-net-power
lvs-is-pin-net-port
lvs-are-pins-shorted
lvs-connections-get
lvs-connection-type-get

# Index


## A

A2drc 4-30
A2lvs 4-32
account file 7-2
aliasing keywords 3-28
ASSIGN section
  group file creation 2-6
Astro to Hercules
  DRC 4-30
  LVS 4-32

## B

block_lvs.log 7-9
block.acct 7-2
block.err file 2-6
block.LAYOUT_ERRORS 7-3
block.lpe 7-9
block.LVS_ERRORS 7-9
block.net 7-9
block.RESULTS 7-2
block.sp 7-9
block.sum 7-2
block.sum file 2-6
block.tech 7-11
block.toperr 7-8
block.tree 7-11
block.waive 7-12
BOOLEAN expressions, IF-ELSE rules 3-11

## C

CDL Map File 4-53
CELL LIST file 3-5
cell statistics, Milkyway 7-12
cells created by preprocessing, summary 6-21
command line syntax 5-1
compare error file 7-9
Creating
  skeletal equivalence file 4-54
  wire resolution log file 4-55

## D

.dprc 5-12
Data Measurement Routines 8-20
debugging problems due to hierarchy 6-23
Description Block Translation Rules 4-27
Design Databases 4-1
design flow 1-3
design methodology, flow 1-2
Direct GDSII Output from Hercules 4-14
distributed components, removing 5-12

distributed log file 5-19
distributed output 5-19
distributed processing 5-11
  auto start 5-13
  combining multithreading and 5-21
  components 5-12
  distributed log file 5-19
  distributed runset file 5-19
  distributed statistics file 5-19
  distributed work directory 5-20
  dp_master 5-12
  dp_portmap 5-12
  dp_slave 5-12
  dp_slave daemon 5-12
  .dprc 5-12
  .evdp 5-12
  host 5-12
  quick start method 5-16
  slave process 5-12
  subcommands 5-20
  troubleshooting 5-21
distributed run, monitoring 5-17
distributed runset file 5-19
distributed statistics file 5-19
distributed work directory 5-20
dp_master 5-12
dp_portmap 5-12
dp_slave 5-12
dp_slave daemon 5-12
DRAC2HE
  error and warning messages 4-26
  syntax 4-24
  text map file 4-26
  utility
    description block translation rules 4-27
    special translation operation 4-30
DRAC2HE utility
  DRACULA_DEFAULTS for dimensional checks 4-29
  INPUT_LAYER Block Translation Rules 4-27
  Translated Operation Block Commands 4-28
DRACULA_DEFAULTS for dimensional checks 4-29

# E

.evdp 5-12
EDIF 2 0 0
  translation 4-46
EDIF 3 0 0
  translation 4-47
Edtext file 3-2
efficient vs. inefficient hierarchy 6-6
Equivalence file 3-3
  BLACK_BOX syntax 3-3
  EQUIV syntax 3-3
error file 7-3
error messages 5-24
  suppressing 5-26
EVP utility 3-5
  syntax 3-5
exit codes 5-25
EXPLODE Language
  Scheme Interface 8-22
Explode List file 3-4
  syntax 3-4
extracted layout netlist file 7-9
extracted layout SPICE netlist file 7-9

# F

features and benefits 1-2
flat database
  group file system requirements 2-6
  produces larger output database 2-6
  reports more errors 2-6
flat error file 7-8
Flexible Device Netlisting
  Scheme Interface 8-3

# G

gcpp Preprocessor 3-20
GDSII 4-2
  syntax for multiple files 4-11
  using GZIPPED files 4-13
  using multiple files 4-11
GDSII file
  input database format 2-6
  stores graphic error data 2-6
GDSIN
  syntax 4-6
  use to stream in the GDSII database 4-6
GDSOUT Utility 4-15
GENDEV Command
  Scheme Interface 8-7
general explodes 6-11
graphic output database 2-6
  smaller with hierarchical input database 2-6
graphical error structures 7-14
  hierarchy 7-14
  placing
    error structures and permanent output 7-15
graphichal error structures
  editing errors 7-15
group file
  definition of 2-6
  flat database system requirements 2-6
  hierarchical database system requirements 2-6
  using ASSIGN 2-6
  using PERM layers 2-6
  using TEMP layers 2-6

# H

Hercules
  command line syntax 5-1
  Netlist Format 4-35
  translate into the Hercules Netlist Format 4-37
Hercules command line syntax 5-1
Hercules high performance 5-9
HERCULES output files
  block.err 2-6
  block.sum 2-6
HERCULES_CELL 2-5
HERCULES_DEVICE 2-2
HERCULES_DRC 2-3
HERCULES_ERC 2-3
HERCULES_LVS 2-3
HERCULES_MASK 2-4
HERCULES-DP_MT 2-5
HERCULES-NETLIST 2-4
HERCULES-RUN_TRAN 2-5
hierarchical database
  group file system requirements 2-6
  produces smaller graphic output database 2-6
hierarchical design 6-2
hierarchical design, principals 6-1
hierarchical verification 6-3
host 5-12
HTML interface 7-16

# I

IC Compiler, running Hercules with 1-4
input files 3-1
  CELL LIST file 3-5
  Edtext file 3-2
  Equivalence file 3-3
    BLACK_BOX Syntax 3-3
    EQUIV Syntax 3-3
  Explode List file 3-4
    syntax 3-4
  runset file 3-1
  Schematic Netlist file 3-5
input files, parsing 3-5
INPUT_LAYER Block Translation Rules 4-27
installing Hercules 2-1
invoking HTML output 7-16

# L

layer entry, syntax rules 4-12
LAYER_EMPTY 3-15
licensing 2-1
  determining correct one 2-2
  product 2-2
    HERCULES_DEVICE 2-2
    HERCULES_DRC 2-3
    HERCULES_ERC 2-3
    HERCULES_LVS 2-3
    HERCULES_MASK 2-4
    HERCULES-DP_MT 2-5
    HERCULES-NETLIST 2-4
    HERCULES-RUN_TRAN 2-5
    NET-TRAN 2-4
  software 2-1
Linux Kernel Version 2-10
LPE statistics file 7-9
LVS debugging file 7-9
LVS Routines 8-31
LVS User-Defined Property Comparison 8-29

# M

Macro Preprocessing 3-20
macro preprocessing
  directives 3-21
macro preprocessing runsets 3-20
macro preproecssing
  predefined symbols 3-22
    @error and @warning 3-27
    @if @else @endif 3-22
    @if expressions 3-25
    @undef 3-26
MACROS control line 3-27
MAPPING_FILE
  syntax rules 4-5
Milkyway 4-18
  flows
    application specific 4-22
    DRC black box 4-19
    full 4-19
    LVS black box 4-21
  flows with Hercules 4-19
  objects 4-19
  views 4-18
Missing Files 3-18
multithreading 5-10
  combined with distributed processing 5-21

# N

Netlist Format
  Understanding 4-35
netlist translation
  two steps to complete 4-55
netlists 4-34
NET-TRAN 2-4
NetTran
  Command Line Syntax 4-39
  error and warning messages 4-56
  use to translate into the Hercules Netlist Format 4-38
  Utility 4-38
NFS Settings 2-10

# O

OASIS 4-18
operating system
  requirements
    HP 2-8
    SUN 2-8
operating systems 2-8
OUTLIB output file specification 2-6
output files
  account file 7-2
  compare directory 7-9
    compare tree files 7-11
    equivalence files 7-11
    individual equivalence summary file 7-10

partitioned layout netlist 7-11
partitioned schematic netlist 7-10
compare error file 7-9
description of 7-1
error file 7-3
extracted layout netlist file 7-9
extracted layout SPICE netlist file 7-9
flat error file 7-8
LPE statistics file 7-9
LVS debugging file 7-9
results file 7-2
runset commands 7-18
summary file 7-2
TECHNOLOGY_OPTIONS file 7-11
WAIVE file 7-12
output formats
error hierarchy syntax 7-19
permanent result syntax 7-19
temporary result syntax 7-19


# P


page faults 2-7
parsing
input files 3-5
passing parameters, include files 3-28
performance tips
hierarchical database produces smaller error output 2-6
keep files on local machine for faster execution 2-8
more disk and swap space for flat designs 2-5
provide enough memory to avoid accessing swap space 2-7
smaller group files with hierarchical database 2-6
PERM layers
creating group file for 2-6
permanent output 7-18
post VCELL explodes 6-20


# Q


quick start distributed processing 5-16


# R


registering 2-5
results file 7-2
runset
customize 3-8
environment variables
declaring 3-9
flow control constructs 3-11
flow variables
declaring 3-8
defining check values 3-10
resetting 3-10
user-defined device syntax equations 3-10
using 3-10
internal variables, declaring 3-9
version variable 3-13
runset file 3-1
runsets 4-24
DRAC2HE 4-24
Macro Preprocessing
gcpp Preprocessor 3-20
macro preprocessing 3-20
directives 3-21
runtime
flow control 3-14
checking LAYER_EMPTY 3-15
testing TEXT_SHORT 3-19
warning and error messages 3-17


# S


Schematic Netlist file 3-5
Scheme Interface
device routines 8-4
EXPLODE Language 8-22
Flexible Device Netlisting 8-3
GENDEV Command 8-7

Data Manipulation Routines 8-15
Device Definition Routines 8-13
Device Initialization Routine 8-13
Output Data Routines 8-14
LVS Routines 8-31
Usage 8-2
signoff_drc, IC Compiler command 1-4
signoff_metal_fill, IC Compiler command 1-4
Silos translation 4-49
slave process 5-12
SPICE/CDL translation 4-49
subcommands 5-20
summary file 7-2
swap space 2-5, 2-7
system requirements 2-5
determining need 2-5
flat database group files 2-6
hierarchical database group files 2-6
page faults 2-7
reducing memory with bottom up error fixing 2-7
swap space 2-5, 2-7

## T

TECHNOLOGY_OPTIONS
modifying 6-23
TECHNOLOGY_OPTIONS file 7-11
TEMP layers
creating group file for 2-6
temporary output 7-18
TEXT_SHORT 3-19
Translated Operation Block Commands 4-28
Translating
standard format netlists to Hercules format 4-44
Translation Utilities
A2lvs 4-32
GDSOUT 4-15
NetTran 4-38
translation utilities
A2drc 4-30

## U

Using Hercules to Stream in the GDSII Database 4-2

## V

VCELL explodes 6-12
VCELL pairs pass 6-16
VCELL sets pass 6-14
verify_drc, IC Compiler command 1-4
Verilog
translation 4-48

## W

WAIVE file 7-12