Hercules™
LVS
User Guide

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

3. Using HLVS Device Extraction

Preface

This preface includes the following sections:

What's New in This Release

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

To see the *Hercules Release Notes*,

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

   https://solvnet.synopsys.com/DownloadCenter

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

About This Guide

The *Hercules LVS User Guide* assists the designer in running layout-versus-schematic (LVS) using Hercules software. Use this user guide in conjunction with the *Hercules Reference Manual* and *Hercules Getting Started Tutorial*.

This preface provides an overview of the chapters included in this *Hercules LVS User Guide*, as well as information on how you can contact Customer Support and access SolvNet.

Audience

This user guide takes beginning users through the Hercules LVS flow, further enhancing their knowledge of the Hercules tool and its functions.

This user guide includes information on

- Executing LVS extract and compare
- Using Hercules LVS device extraction
- Netlisting
- Analyzing and debugging

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 LVS

This chapter gives a brief overview of Hercules LVS tool, telling how it fits into your design methodology.

What Is LVS?

Hercules, a hierarchical physical verification tool, allows you to perform layout-versus-schematic (LVS) on your integrated circuit design. Hercules LVS performs a comparison process that verifies whether the geometric or layout implementation of a circuit matches the schematic representation. This process is performed by extracting the designed devices and then assessing the connectivity between them. Following this, a netlist representation is built. This netlist will show the hierarchy in layout. Hercules will then look for matching blocks in the schematic and layout netlist. These points of comparison are known as equivalence points. Pairs of these points indicate where the underlying circuitry between the schematic and layout blocks are equivalent.

Hercules LVS Features and Benefits

Instead of flattening to the device level, Hercules LVS performs hierarchical verification, which is more efficient in processing and locating errors.

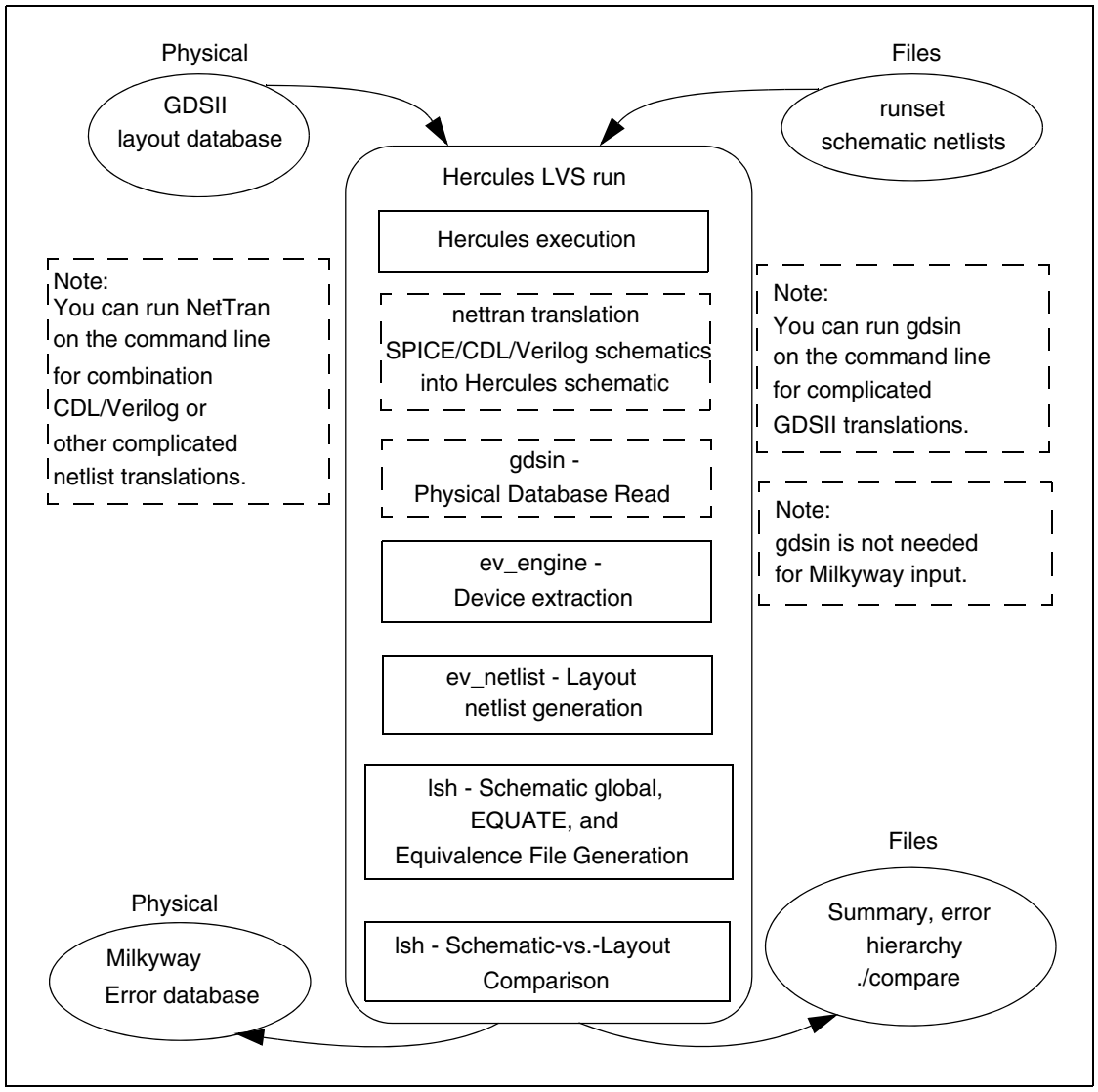
Using equivalence points requires that the schematic and layout netlists are checked just once for each block. Therefore, it is only necessary to check the connectivity to the lower-level blocks.

Physical connectivity can be checked with or without checking the properties. If you do check the properties, you can still check the connectivity and not have the parameters cause false errors. Parameter errors can be flagged either as errors or warnings.

Using the Hercules LVS Design Flow

The flow diagram in Figure 1-1 shows the process that automatically takes place during the Hercules LVS extraction.

Figure 1-1 LVS Design Flow



2

Executing LVS Extract and Compare

This chapter describes the execution and analysis of a Hercules LVS runset, as well as information on advanced Hercules LVS usage models.

The Hercules Command Line

Note:
For details on the Hercules command line, see the *Hercules Reference Manual*, Running Hercules chapter. For details on executing Hercules from VUE, see the *Hercules VUE User Guide*.

Many Hercules LVS options can be set from the command line. For example:

```
hercules -C -e equiv.file -s top.sch -stb cellA
   -c compare1 lvs_runset.ev
```

The above command will run Compare only (because of the -C option) using the equivalence file equiv.file on cellA from the schematic file top.sch. Also, detailed output files will be stored under the new name compare1/, instead of having the default name compare/.

For a complete list of Hercules command-line syntax, see *Hercules General Usage Information*, General Hercules Execution chapter.

Hercules High Performance

Hercules high performance capabilities 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, although currently only the extraction phase of LVS can take advantage of this.

With distributing processing, different commands in a runset are processed by different CPUs at the same time (in parallel). With multithreading, on the other hand, the work within each command is separated among multiple CPUs (on a single host only).

Although using distributed processing has certain advantages, multithreading is more effective with multiple CPUs. This is because most commands in an LVS runset depend on the results of previous commands, thus making parallelism within the command more effective than at the runset level. The most effective use of multiple CPUs is to run Hercules with a combination of distributed processing and multithreading.

For detailed information on the Hercules high performance capabilities, *Hercules General Usage Information*, General Hercules Execution chapter.

Advanced Hercules LVS Use Models

This section describes more advanced usage models. (see *Hercules General Usage Information*, General Hercules Execution chapter, for basic usage models.) It includes settings required in the Hercules runset to create Extract-view and cross-reference information used as input for StarRC. It also describes the comparisons of Layout vs. Layout and Schematic vs. Schematic using Hercules.

LVS and StarRC

You can use Hercules for StarRC transistor-level extraction. There are requirements for an LVS runset that are specific to StarRC. These requirements are reviewed in this chapter.

LVS Runset Command Requirements for Use with StarRC

WRITE_EXTRACT_VIEW Command

The WRITE_EXTRACT_VIEW command is required for StarRC extraction. It is used to generate a Milkyway™ view called XTR that contains a graphical representation of connectivity and of extracted devices along with their properties. The extract view is created as input to the StarRC tool. For example:

```
WRITE_EXTRACT_VIEW {
```

```
   LIBRARY_NAME = LIB_NAME
   LIBRARY_PATH = .
}
```

After Hercules is done, the XTR view will be written into a directory named by LIBRARY_NAME which will be placed in directory defined by LIBRARY_PATH. Both LIBRARY_NAME and LIBRARY_PATH are required variables without default value.

Notice that StarRC uses Milkyway XTR views for input of the layout database, instead of the CHECK_POINT directory which had been used for STAR RC.

In writing the XTR view, Hercules makes its own Milkyway technology file called hx2mw.tf. This file has all the layer information of the XTR view, which is different from that in ASSIGN section of the runset. This is because there are more layers in the CONNECT sequence, which have been derived from the original layers in the ASSIGN section.

If your design comes from a Milkyway library, you might want to write the XTR view into your existing library. However, this is not recommended, because the technology file of the original Milkyway database will be different from the one made by Hercules.

TECHNOLOGY_OPTIONS Command

TECHNOLOGY_OPTIONS can be used to optimize the hierarchy in a Hercules run, thereby speeding up the run and reducing memory usage. In addition to these performance advantages, TECHNOLOGY_OPTIONS optimizes the output hierarchy in the XTR library, which will be used as an input to StarRC. This will enhance StarRC performance and the effect can be significant (greater than 50 percent) if the design is very hierarchical.

However, in XREF flows, some of the default options (such as `VCELL_PASS {STYLE = SETS}`) might cause unwanted explodes of equivalence points and possible mismatches. As a result, you should not use all of the default options for LVS enabled Hercules runs. This is because XREF will cross-reference down to the equivalence points, and all the equivalence points should be preserved.

Use the Hercules command-line option -rcxt to set TECHNOLOGY_OPTIONS automatically, which is most optimal for StarRC.

Note:

If -rcxt will be used at Hercules runtime, make sure that the runset does not have any TECHNOLOGY_OPTIONS, because -rcxt will not work if INCLUDE_TECHNOLOGY_DEFAULTS=FALSE is set in the Hercules runset.

Example 2-1 is an example of using the TECHNOLOGY_OPTIONS command.

Example 2-1 TECHNOLOGY_OPTIONS

```
TECHNOLOGY_OPTIONS {
   INCLUDE_TECHNOLOGY_DEFAULTS = FALSE
   ALLOW_EXPLODE_WITH_TEXT = TRUE
```

```
    EXPLODE_AREFS = TRUE
    EXPLODE_1XN_AREFS = FALSE
    EXPLODE_CELL_SIZE_PERCENT = 70
    EXPLODE_CELL_SIZE_PERCENT_OF_TOP = 70
    EXPLODE_BIG_SPARSE_CELL = TRUE
    POST_VCELL_EXPLODE_CELL_SIZE <= 10
    VIA_AUTO_EXPLODE = TRUE
    SUBLEAF_AUTO_EXPLODE = 6
    CELL_SIZE_AUTO_EXPLODE <= 10
    EXPLODE_SPARSE_CELL = 1
    EXPLODE_DATA_CELL_LIMIT = 4
    POST_VCELL_EXPLODE_DATA_CELL_LIMIT = 12
    EXPLODE_HOLDING_CELL_LIMIT = 1
    EXPLODE_PLACEMENT_LIMIT = 1
    VCELL_PASS  {
       MIN_CELL_OVERLAP = 60
       RATIO=10000.000
       MIN_COUNT = 40
       MIN_LEAF = 40
       STYLE = EXPLODE
    }
    VCELL_PASS  {
       ITERATE_MAX = 15
       ARRAY_ID = TRUE
       EXPLODE_INTO_VCELL = SMART
       MIN_COUNT = 20
       TOP_PERCENT_OF_VALUE = 40
       STYLE = PAIRS
    }
}
```

EVACCESS_OPTIONS {CREATE_VIEWS} Command

StarRC uses the Hercules EVaccess netlist view and requires that the view be generated during the Hercules run. Therefore, retain the setting of CREATE_VIEWS to TRUE (the default value), so that the netlist view is generated by Hercules.

TEXT_POLYGON Command

GDSII databases have no concept of input and output ports or PIN shapes. However, StarRC requires such shapes to define instance ports, particularly top-level ports. Thus, Hercules runsets intended for use with StarRC should define a methodology for identifying ports based on text placement on metal layers. This methodology uses the Hercules TEXT_POLYGON command to create a polygon everywhere that text is placed on a metal layer. Thus, every metal layer defined in the Hercules ASSIGN section should have a corresponding TEXT_POLYGON command. The new marker layers created by these TEXT_POLYGON commands should also be connected to their corresponding metal layers.

For example, for metal1:

```
TEXT_POLYGON metal1.text {
```

```
   SIZE = 0.1
   CELL_LIST = { * }
   TEXT_LIST = { * } } temp = metal1_mark
   CONNECT {
   metal1 by metal1_mark
}
```

Note that the TEXT_POLYGON command and its corresponding CONNECT statements should be inserted prior to the TEXT command in the runset.

COMPARE Command

Some Hercules COMPARE command options cause difficulty for StarRC back-annotation flows. These options include:

MEMORY_ARRAY_COMPARISON
DETECT_PERMUTABLE_PORTS
MERGE_PATHS
PUSH_DOWN_DEVICES
PUSH_DOWN_PINS

When cross-referencing with the StarRC XREF command, it is recommended that the above COMPARE options be set to FALSE to ensure successful back-annotation in StarRC.

Runset Rules

In preparing the Hercules runset, there are rules that users have to follow in data manipulation:

Terminology used in these rules:

- *Runset layer*: Any layer specified in the CONNECT section of the Hercules runset.
- *Physical layer*: Any layer specified as a CONDUCTOR or VIA in the StarRC ITF process description file.

Rule I: A runset layer via may only connect two physical layers.

Incorrect runset: In this incorrect example, runset layer m1 is mapped to the physical layer metal1, poly is to poly, and nsd psd are to SUSBTRATE. Therefore, cont would be connecting *three* physical layers, which is not allowed.

```
CONNECT m1 poly BY cont
CONNECT m1 nsd psd BY cont
```

Correct runset: For most runsets, following the correct convention is simple. In general, the runset should have specific contacts for connecting the following physical layers.

If the physical layers are:

metal2 - metal1
metal1 - poly
metal1 - SUBSTRATE

the runset would be:

```
BOOLEAN cont AND poly { } TEMP = polyCont
BOOLEAN cont NOT polyCont { } TEMP = subCont
CONNECT m1 poly  BY polyCont
CONNECT m1 nsd psd BY subCont
```

Rule II: All device runset layers should be NOTed from the routing runset layers.

This is required to account for the removal of gate capacitance and duplicated designed capacitance. This is only an issue with MOSFET gates and the terminals of designed capacitors.

Incorrect: Example 2-2 shows an incorrect runset.

In Example 2-2, StarRC will not ignore the gate capacitance correctly and the designed capacitance will be double counted in this example. Other devices may not have these issues.

Example 2-2 Incorrect Runset (Do not use. For example only.)

```
BOOLEAN poly AND ndiff { } TEMP = ngate
BOOLEAN ndiff NOT poly { } TEMP = nsd
NMOS n ngate nsd nsd SUSBTRATE { } TEMP = ndev

BOOLEAN m2 AND cap_recog  { } TEMP = cap_top
BOOLEAN m1 AND cap_recog  { } TEMP = cap_bot
BOOLEAN cap_top AND cap_bot {} TEMP = cap_dev
CAP m1m2cap cap_dev cap_top cap_bot { } TEMP = cdev

CONNECT poly BY ngate
CONNECT cap_top BY m2
CONNECT cap_bot BY m1
```

Correct: The correct runset to use for this situation is shown in Example 2-3.

Inserting the BOOLEAN NOT for gate and field poly is required for both planar processes (where gate poly and field poly layers are both mapped to a single POLY layer in the StarRC .nxtgrd file) and non-planar processes (where gate poly and field poly layers are mapped to separate co-vertical layers in the StarRC .nxtgrd file), since IGNORE_CAPACITANCE works in both cases.

Example 2-3 Correct Runset

```
BOOLEAN poly     AND ndiff      { } TEMP = ngate
BOOLEAN ndiff    NOT poly       { } TEMP = nsd
BOOLEAN poly     NOT ngate      { } TEMP = poly
NMOS n ngate nsd nsd SUBSTRATE  { } TEMP = ndev
```

```
BOOLEAN m2       AND cap_recog  { } TEMP = cap_top
BOOLEAN m1       AND cap_recog  { } TEMP = cap_bot
BOOLEAN cap_top  AND cap_bot    { } TEMP = cap_dev
BOOLEAN m2       NOT cap_top    { } TEMP = m2
BOOLEAN m1       NOT cap_bot    { } TEMP = m1
CAP m1m2cap cap_dev cap_top cap_bot { } TEMP = cdev

CONNECT poly BY ngate
CONNECT cap_top BY m2
CONNECT cap_bot BY m1
```

Rule III: A physical layer cannot connect another physical layer unless they are co-vertical.

As shown in the Rule II example, connecting poly and ngate layers is allowed, since they are co-vertical. Connecting cap_top and m2 does not matter at all, since they are mapped to the same physical layer (metal2). This also applies for connecting cap_bot and m1.

Additional notes:

- EQUATE statements do not have to be included in the original Hercules runset; Hercules-generated EQUATE statements in lvsflow/lvs_include.ev are sufficient.
- Command-line overrides may be used in the Hercules layout extraction/LVS compare.

Example 2-4 shows a typical Hercules runset for StarRC transistor level extraction.

Example 2-4 StarRC Transistor Level Extraction Runset

```
HEADER {
   LAYOUT_PATH = .
   INLIB = ADD4_CUSTOM.DB
   OUTLIB = EV_OUT
   FORMAT = MILKYWAY
   BLOCK = add4
   SCHEMATIC = ADD4.sch
}
OPTIONS {
   SCHEMATIC_GLOBAL = {VDD GND}
   SCHEMATIC_GROUND = {GND}
   SCHEMATIC_POWER = {VDD}
   LAYOUT_GLOBAL = {VDD GND}
   LAYOUT_POWER = {VDD}
   LAYOUT_GROUND = {GND}
   EXPLODE = { VIA *con *tie DEV_* }
   NET_PREFIX = N_
   EDTEXT= ADD4.text
}
TEXT_OPTIONS {
   ATTACH_TEXT = ALL
   CONNECT_BY_NAME = MIXED_MODE
}
ASSIGN {
```

```
   Ndiff(1)
   Pdiff(2)
   Nwell(3)
   Poly (5)
   Cont (6)
   Metal1 (8) TEXT (28)
   Via1 (9)
   Metal2 (10) TEXT (30)
   PnAnode (51)
   PnCathode (52)
   NpCathode (61)
   NpAnode (62)
   NpnCollector (71)
   NpnBase (72)
   NpnEmitter (73)
   Cap1 (11)
   Cap2 (12)
   PolyRes (20)
}
TEXT_POLYGON Metal2.text {
   SIZE = 0.01
   CELL_LIST = { add4 }
   TEXT_LIST = { * !*VDD* !*VSS* }
} TEMP = Metal2_Mark

/*** NPN BJT ***/
BOOLEAN Cont AND NpnEmitter { } TEMP = NpnEmitterCont
BOOLEAN Cont NOT NpnEmitterCont { } TEMP = Cont
BOOLEAN Cont AND NpnBase { } TEMP = NpnBaseCont
BOOLEAN Cont NOT NpnBaseCont { } TEMP = Cont
BOOLEAN Cont AND NpnCollector{ } TEMP = NpnCollectorCont
BOOLEAN Cont NOT NpnCollectorCont{ } TEMP = Cont

/*** Without these, all the three terminals of this Npn device will be
shorted and the device will be filtered out. ***/
NPN NpnDev NpnEmitter NpnBase NpnCollector SUBSTRATE { }
TEMP = NpnDev

/*** PN DIODE ***/
BOOLEAN PnCathode NOT PnAnode { } TEMP = PnCathodeTerm
BOOLEAN PnAnode AND PnCathode { } TEMP = PnDevice
COPY PnAnode  { } TEMP = PnAnodeTerm
DIODE PnDev PnDevice PnAnodeTerm PnCathodeTerm {
   DIODE_TYPE = PN
   DIODE_RECOGNITION_LAYER_USED = TRUE
}TEMP = PnDev

/*** MOS ***/
BOOLEAN Pdiff AND Nwell { } TEMP = pdev
BOOLEAN Ndiff NOT Nwell { } TEMP = ndev
BOOLEAN Pdiff NOT Nwell { } TEMP = subtie
BOOLEAN Ndiff AND Nwell { } TEMP = welltie
BOOLEAN Poly  AND ndev  { } TEMP = ngate
```

```
BOOLEAN Poly  AND pdev  { } TEMP = pgate
BOOLEAN ndev  NOT ngate { } TEMP = nsd
BOOLEAN pdev  NOT pgate { } TEMP = psd
BOOLEAN Poly  NOT ngate { } TEMP = Poly
BOOLEAN Poly  NOT pgate { } TEMP = Poly
/*** These 2 BOOLEANs are mandatory for StarRC. ***/

NMOS n ngate nsd nsd SUBSTRATE { } TEMP = ndevice
PMOS p pgate psd psd Nwell   { } TEMP = pdevice

/*** Contact separation ***/
BOOLEAN Cont AND Poly     { } TEMP = PolyCont
BOOLEAN Cont NOT PolyCont { } TEMP = SubCont
/*** Contact separation is a must for StarRC. ***/

/*** CAPACITOR ***/
BOOLEAN Cap1   AND Metal1 { } TEMP = Cap1
BOOLEAN Cap2   AND Poly   { } TEMP = Cap2
BOOLEAN Metal1 NOT Cap1   { } TEMP = Metal1
BOOLEAN Poly   NOT Cap2   { } TEMP = Poly
BOOLEAN Cap1   AND Cap2   { } TEMP = CapRec

CAP CapDev CapRec Cap1 Cap2 {
   EV_AREA_CAPVAL = 1.0e-15;
   CAP_RECOGNITION_LAYER_USED = TRUE;
} TEMP = CapDev

/*** Poly RESISTOR ***/
BOOLEAN Poly AND PolyRes { } TEMP = pres_body
BOOLEAN Poly NOT pres_body { } TEMP = field_poly
/*** Routing layers can be used as resistor terminals. ***/

RES ResDev pres_body field_poly field_poly {
   EV_RESVAL = 3.5;
} TEMP = ResDev

CONNECT {
   Metal2 Metal1 BY Via1
   Metal1 BY Cap1
   Metal1 field_poly BY PolyCont
   field_poly BY Cap2
   Metal1 nsd psd welltie subtie BY SubCont
   Metal1 PnCathodeTerm BY SubCont
   Metal1 PnAnodeTerm BY SubCont
   Metal1 NpnCollector BY NpnCollectorCont
   Metal1 NpnBase BY NpnBaseCont
   Metal1 NpnEmitter BY NpnEmitterCont
   ngate pgate BY field_poly
   Nwell BY welltie
   SUBSTRATE BY subtie
   pres_body BY pres_body
   /*** Resistor body layer should be connected to itself. ***/
   Metal2_Mark BY Metal2
```

```
}
TEXT {
   Metal1 BY Metal1.text
   Metal2  BY Metal2.text
}
NETLIST { }

WRITE_EXTRACT_VIEW {
   LIBRARY_NAME = ADDER
   LIBRARY_PATH = .
}
COMPARE {
   MEMORY_ARRAY_COMPARISON = FALSE
   DETECT_PERMUTABLE_PORTS = FALSE
   MERGE_PATHS = FALSE
   PUSH_DOWN_DEVICES = FALSE
   PUSH_DOWN_PINS = FALSE
}
```

Layout Versus Layout

You can use Hercules to compare (perform a Boolean XOR) layers of two different libraries. The libraries do not need to have the same hierarchy or cell names, except for the top cell name.

Note:

If the two libraries compared have different database resolutions, the database resolution of the output library will be the same as the second library resolution (INLIB in the second sample runset).

The Layout vs. Layout process is done in two steps.

1. Run Hercules to load the layers of the first library into the GROUP directory.

   The first run of Hercules loads the FIRST_LIB library. This run creates only the group files, placing them into the check_point directory using the CHECK_POINT command. Example 2-5 shows a runset.

Example 2-5 Loading First Library for Layout Versus Layout Comparison

```
HEADER {
   BLOCK = TOP
   INLIB = FIRST_LIB
   GROUP = group
}
ASSIGN {
   POLY    (10)
   PPLUSD   (4)
   NPLUSD   (5)
   DIFF     (4) (5)
}
```

```
CHECK_POINT {
   PATH = check_point
   MOVE_GROUP_FILES = TRUE
}
```

2. Run Hercules to load the layers of the second library and perform a Boolean XOR with the first library layers (stored in GROUP).

   The second run of Hercules uses the SECOND_LIB library and the group directory specified in the runset. This run compares the second layout against the first backup. Layout vs. Layout comparison is done with the COMPARE_GROUP command for each layer listed in it. Example 2-6 shows a runset for loading the second library and comparing the two layouts.

Example 2-6 Loading Second Library and Performing Layout Versus Layout Comparison

```
HEADER {
   BLOCK = top
   INLIB = SECOND_LIB
   GROUP = group
}
ASSIGN {
   POLY (10)
   PPLUSD (4)
   NPLUSD (5)
   DIFF (4) (5)
}
COMPARE_GROUP {
   group_dir = check_point
   group_prefix = old
   POLY (10)  /* these layer numbers are ignored     */
   PPLUSD (4) /* but this format makes it easy to    */
   NPLUSD (5) /* copy and paste from the ASSIGN list */
 }
```

Schematic Versus Schematic

Hercules can be used to compare two schematic netlists. Because the LVS engine is used to compare two schematic netlists, one netlist must have the name *block*.net (where *block* is value of the HEADER variable), and will be referred to throughout the generated output as the layout netlist.

Because Hercules handles the schematic and layout netlists differently, the option schematic_vs_schematic=TRUE needs to be set in the COMPARE command. This option causes the OPTIONS section variables LAYOUT_GLOBAL and SCHEMATIC_GLOBAL to operate identically; that is, global nets in the layout netlist do not have to be explicitly connected, and artificial connections are made where necessary. In addition,

schematic_vs_schematic=TRUE causes the pin lists in all the EQUATE commands to refer to both netlists, instead of only the schematic netlist. Device names on both sides of the EQUATE statement must reflect the device names in the two schematic netlists.

Hercules Run Analysis for LVS

When a Hercules LVS run is executed, what sequence of events transpires? How do you know the exact status of the run in progress?

There are two phases of an LVS run: the layout netlist extraction phase and the comparison phase.

The layout netlist extraction phase invokes the DRC engine, thus the format and analysis are the same as for a DRC and ERC run. (For details, see the *Hercules DRC and ERC User Guide*, Executing Hercules DRC and ERC chapter.)

Analysis During the Run

During a Hercules LVS run, you can see a brief description of the run's progress in the *block*.RESULTS file. For more detailed analysis of the run's progress, look at the screen or the *block*_lvs.log file in the run_details directory. Also, if the job stops before finishing, this file gives a more detailed description on what caused the run to stop.

Analysis After the Run

Hercules LVS outputs three primary files: *block*.RESULTS, *block*.LAYOUT_ERRORS and *block*.LVS_ERRORS.

The *block*.RESULTS file is the place to start your analysis of the run. This file indicates whether or not the run completed and whether there are any layout extraction or LVS comparison errors.

If there are LVS comparison errors, you should first analyze the *block*.LAYOUT_ERRORS file to identify layout extraction errors. These errors include text shorts, text opens, and device extraction and other ERC errors. It is important to understand and fix those errors before analyzing LVS errors, because comparison results are dependent on clean extraction.

After all layout errors are cleaned, you can analyze the *block*.LVS_ERRORS file. This file first identifies whether or not the top block is LVS-clean. It then lists all lower-level cells that are not clean. Those cells are listed in two categories of errors, first and second priority. The

cells listed as first-priority errors are those most likely to be the root causes of the non-comparison and, therefore, should be investigated first. Hercules also gives a detailed diagnosis and potential reason for each cell in the first-priority errors category.

To get more detailed diagnosis for comparison results of each cell, go to the run_details/compare directory. There, each cell has its own directory with a detailed LVS log (referred to as the sum.*block*.*block* file or as the equivalence summary file) for that particular cell.

3

Using HLVS Device Extraction

This chapter describes how devices are extracted in Hercules. The first section provides an overview of hierarchical device extraction with information on the golden rule of device extraction. This section is followed by an example of a simple resistor device extraction.

Often, devices are designed in such a way that it is difficult to apply the simple rules of device extraction. For these cases, you may be required to set additional options, or to use the Scheme programming utility to achieve the proper results. This chapter includes a few sample device extractions, illustrating the use of the Scheme interface where necessary.

Overview of Hierarchical Device Extraction

Before LVS can be completed, a netlist must be extracted from the layout database. Hercules uses DEVICE commands such as NMOS and RESISTOR to define the devices that are extracted from the layout and written to the layout netlist. These DEVICE commands have several arguments that consist of a device name followed by a device layer and terminal layers. The layers are defined before the DEVICE command in the runset through Data Creation commands to uniquely identify the device elements. For DEVICE commands syntax, refer to the *Hercules Reference Manual*, Detailed Commands chapter.

The Golden Rule of Device Extraction

The most important concept to remember when extracting devices is that for any device there is:

- A device recognition polygon—this polygon must interact in some way with all the other device polygons.
- A device definition polygon—for each device definition polygon, Hercules attempts to extract one device.

For most devices, the device recognition polygon and the device definition polygon are the same. If you keep this concept in mind, device extraction is fairly simple. Hercules extracts one device for every device recognition polygon that interacts with all the other device polygons. The only exceptions to this rule are the lateral NPN and PNP commands. For the lateral BJTs, the base is the device recognition polygon and the emitter/collector is the definition polygon. Table 3-1 displays the device recognition and definition layers for all Hercules devices.

Table 3-1 Hercules Device Recognition and Definition Layers

Device	Recognition Layer	Definition Layer
NMOS	gate	gate
PMOS	gate	gate
NPN (vertical)	emitter	emitter
PNP (vertical)	emitter	emitter
NPN (lateral)	base	emitter
PNP (lateral)	base	collector

Table 3-1 Hercules Device Recognition and Definition Layers(Continued)

Device	Recognition Layer	Definition Layer
CAPACITOR	device_layer	device_layer
DIODE	device_layer	device_layer
RESISTOR	device_layer	device_layer

How to Write Optimal Device Extraction Commands

Although following the golden rule by itself will result in devices being extracted from your layout, it is better to follow stricter rules when defining devices. The advantage to following stricter rules is that the properties associated with devices will be far more likely to be accurate. The following section offers descriptions of the different devices Hercules provides, and suggestions on how to define the devices for optimal device extraction.

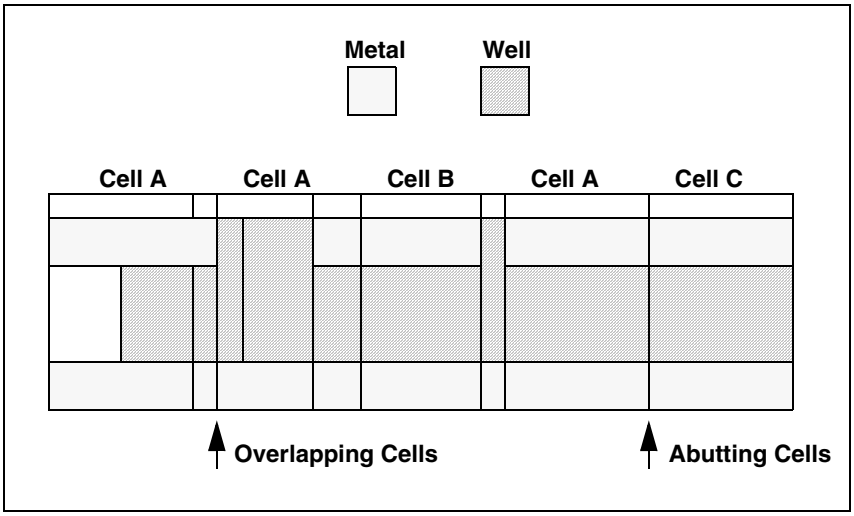
Hierarchical Device Extraction

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

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

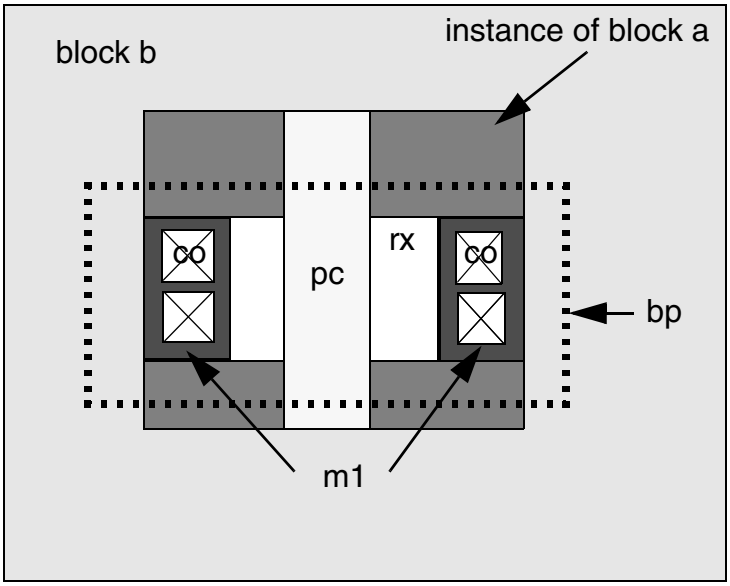
Some interaction of data across the hierarchy is to be expected. For example, power rails and nwells from blocks placed side by side will abut against each other or slightly overlap and interact during verification. Another normal situation is when metal and via polygons connect blocks together. Figure 3-1 shows an example of these situations. Notice that the polygons either abut or only slightly overlap.

Figure 3-1 Interaction of Data Across the Hierarchy



However, layers in a device should be kept in proper hierarchies. There may be exceptions, such as diffusion-programmable ROMs. Consider Figure 3-2, for example.

Figure 3-2 Structure Crossing Hierarchies in a p-device



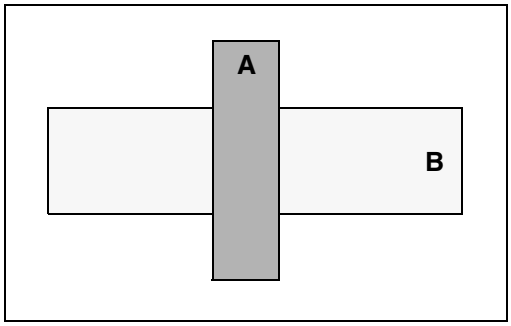
Note that this p-device has an instance of block *a* within block *b*. Block *a*, in turn, has layers *pc, rx, co,* and *m1*, each layer having one or more geometries. However, *bp* is in block *b*. Because *bp* is not in block *a*, there will be some data interaction across the hierarchy. Preferably,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

Generating Mosfets

A MOSFET transistor consists of a gate polygon, two source/drain polygons, and at most four bulk terminals. The gate polygon is generated from a BOOLEAN AND between layer A and layer B (see Figure 3-3 below). The source and drain polygons result from performing a BOOLEAN B NOT A operation. The gate layer is both the recognition and definition layer. For optimal Hercules extraction:

- The gate polygon should exactly edge-touch each of the source/drain polygons.
- The gate polygon and the source/drain polygons should be enclosed in the bulk polygon, if it is specified.

Figure 3-3 MOSFET



Specialty MOSFET Extraction

MOSCAP Extraction

Netlist extraction of the MOSCAP device is accomplished using the CAPACITOR command along with the necessary derived layers. The CAPACITOR command has the following format. (see the *Hercules Reference Manual*, Detailed Commands chapter.)

```
CAP device_name device_layer terminal_layer1 terminal_layer2 [bulk]
{Dev_Input Dev_Equations [Dev_Options]} Output_Definition [Error_Output]
```

Figure 3-4 and Figure 3-5 show cross section images of the PMOS and NMOS CAPs.

Figure 3-4 Cross Section Image of (PMOS) MOSCAP Device

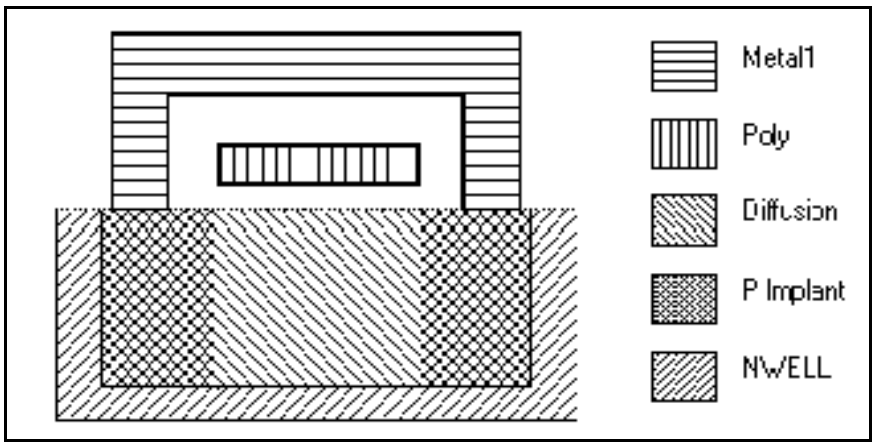
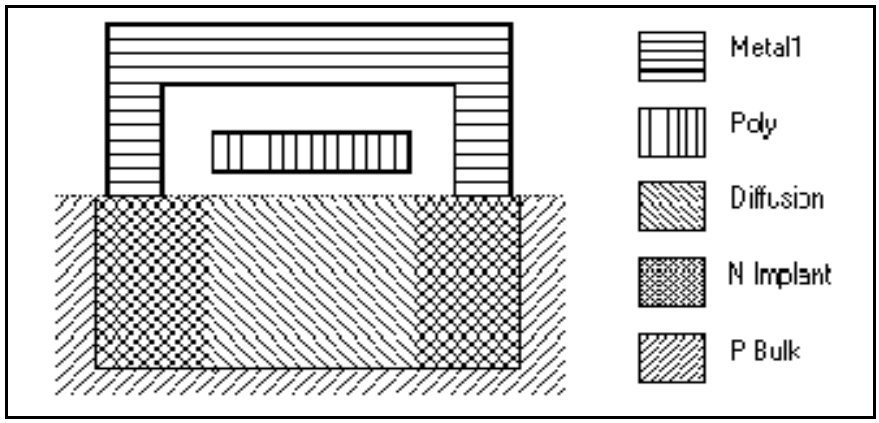


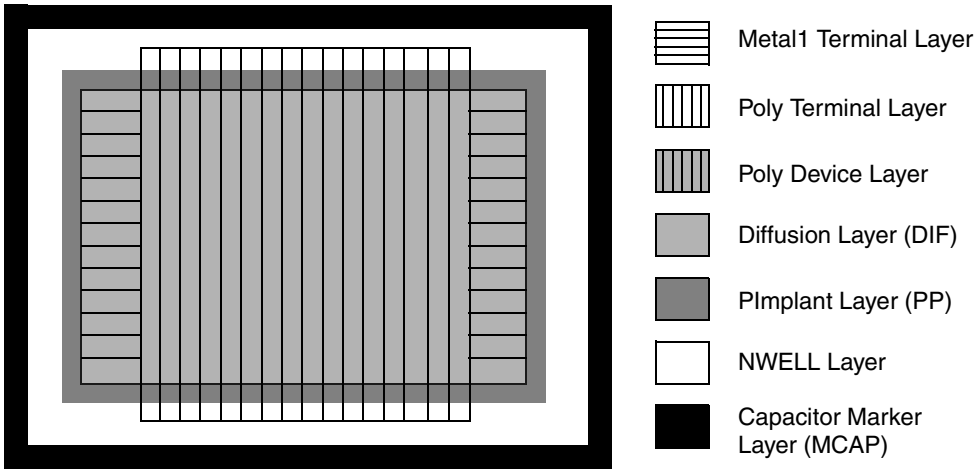
Figure 3-5 Cross Section Image of (NMOS) MOSCAP Device



To successfully extract MOSCAP devices, the Poly device layer, Poly terminal layer, and Metal1 terminal layer must be isolated using BOOLEAN and SELECT operations. Figure 3-6 shows an example of layer isolation used to extract the PMOS Cap.

- The metal terminal layer is the same as the source & drain region of the extracted PMOS and NMOS devices.
- The poly terminal layer is poly that is within the capacitor mask region and interacts with the active region.
- The poly device layer is poly that is within the capacitor mask and the active region.
- MOSCAP terminal and device layers should not interact with resistor mask layers.

Figure 3-6 Layout of a p-Type MOSCAP



Device Extraction

```
BOOLEAN   POLY     NOT  RES_MASK          {} TEMP = CPOLY
BOOLEAN   NWELL    AND  DIF               {} TEMP = ACTIVE
BOOLEAN   CPOLY    AND  MCAP              {} TEMP = CAP_POLY
SELECT    CAP_     POLY INTERACT ACTIVE   {} TEMP = PTERM
BOOLEAN   PTERM    AND  ACTIVE            {} TEMP = CBODY
BOOLEAN   PP       AND  ACTIVE            {} TEMP = PSD1
BOOLEAN   PSD1     NOT  CAP_POLY          {} TEMP = PSD0
BOOLEAN   PSD0     NOT  RES_MASK          {} TEMP = PSD
CONNECT   PSD      METAL1 BY DIFFUSION_CONTACT
CONNECT   PTERM    METAL1 BY POLY_CONTACT
          CAP PMOS_MOSCAP CBODY PTERM PSD {} TEMP = moscap_output
```

Circular MOSFET

To extract circular MOSFETs, use the PMOS/NMOS MOS_MULTITERM_EXTRACT option. When this option is set to TRUE, MOS devices are extracted when a single gate polygon forms multiple devices. When set to FALSE, MOS devices are not extracted. This option is used primarily when there are waffle-shaped or donut-shaped gates.

Think of this extraction as performing an EXTERNAL check with spacing less than infinity between all source/drain polygons. Take the output of this EXTERNAL check and perform a BOOLEAN AND with the gate polygon. The output polygons from the BOOLEAN AND will be the gate polygons used for multiterm extract. Several MOS devices will be extracted for each waffle/donut shaped gate, and device extraction errors are likely to occur, especially for devices with angled gates. As long as something is being extracted for each of these gates, it is unlikely that you need to worry about the reported errors. Two examples of circular MOSFETs are shown in Figure 3-7 and Figure 3-8.

In Figure 3-7, you can see that if you project between the outside PSD polygons as the EXTERNAL command would, the resulting error polygons in these regions would touch three PSD polygons, which of course is a device error. Example 3-1 shows a device error file, *block*.LAYOUT_ERRORS.

Example 3-1 Circular MOSFET Error File

```
Library name:  final
Structure name:  donut
#####--- ERR_DEVICE ----------------------------------

PMOS P pgate psd psd NW {
        MOS_PRINT_XY_POSITION = FALSE;
        MOS_PRINT_STATS = FALSE;
        MOS_SAVE_ALL_PROPS = FALSE;
        MOS_HIERARCHICAL = TRUE;
        MOS_CALC_NRS_NRD = FALSE;
        MOS_MULTITERM_EXTRACT = TRUE;
        MOS_COMBINE_SOURCE = TRUE;
        MOS_SINGLE_SD=BOTH;} TEMP = dev

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure     Error Type   Layer Value  ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
donut    TOO_MANY_TERMINALS   psd  3    (-2.040, -4.960)
donut    TOO_MANY_TERMINALS   psd  3    (-2.040, -4.110)
donut    TOO_MANY_TERMINALS   psd  3    (-2.040, -1.990)
```

Figure 3-7 Example of Circular MOSFET

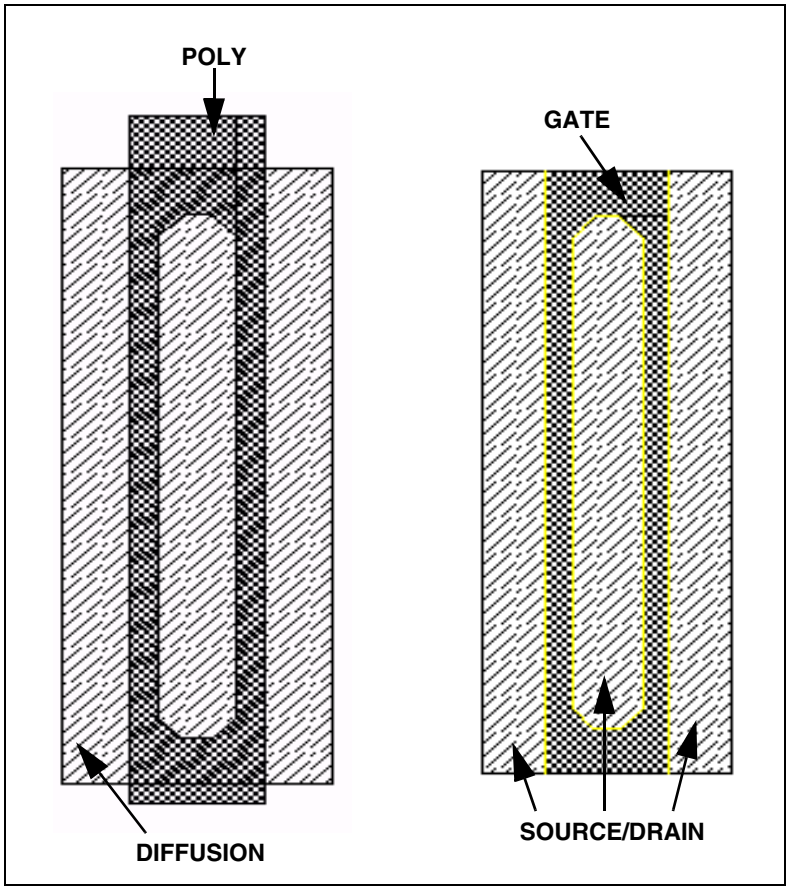
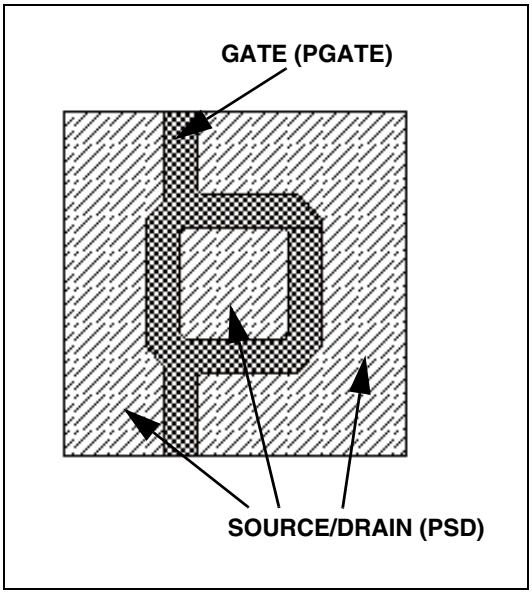


Figure 3-8 Example of Circular MOSFET



7-Terminal MOSFET

The MOS_EXTRA_BULK_LAYER option is designed for 7-terminal MOSFET extraction. When this option is assigned, the bulk layer must be specified in the device command, and the extra bulk layers are recognized as bulk2, bulk3, and bulk4 layers. The syntax for this option is:

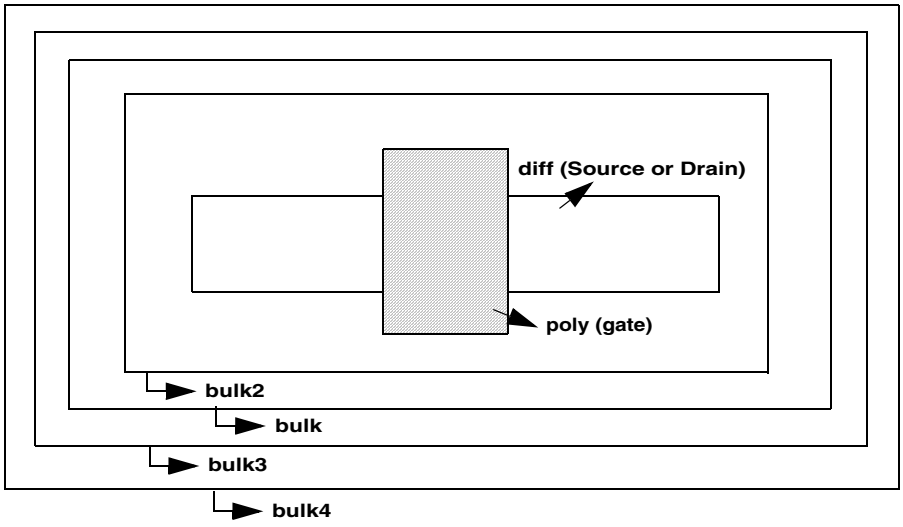
```
MOS_EXTRA_BULK_LAYER = {bulk2, ..., bulk4}
```

For example:

```
NMOS nmos_dev ngate nsd nsd pwell {
   MOS_EXTRA_BULK_LAYER = {substr, substr2}
}
```

The 7-terminal MOSFET extraction checks the *bulk2*, *bulk3*, and *bulk4*layers (shown in Figure 3-9) the same as the *bulk* layer.

Figure 3-9 7-Terminal MOSFET



When setting this option, Hercules netlists the *bulk2*, *bulk3*, and *bulk4* connections as well as *bulk* connection in both the *block*.net and *block*.sp files.

An example of a Netlist output .net file is:

```
{INST M0=nmos_dev {PIN 2=GATE 3=SRC 3=DRN
                   3=BULK 1=BULK2 4=BULK3 5=BULK4}}
```

An example of a SPICE output .sp file is:

```
M0 3 2 3 3 1 nmos_dev $BULK3= 4 $BULK4= 5
```

Generating Bipolar Devices

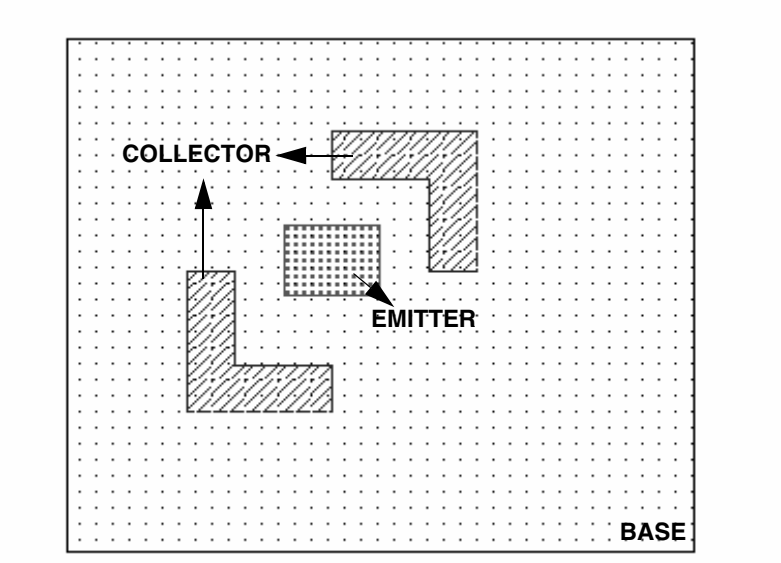
There are two basic types of bipolar transistors, vertical and lateral. To determine what kind of device you have, think of the bipolar transistor as a three layer sandwich. If the sandwich is lying flat on the tabletop, it is a vertical sandwich. If it is lying on its edge (crust on the tabletop), it is a lateral sandwich.

Lateral BJT

A lateral BJT consists of a base polygon which interacts with collector and emitter polygons. All of these polygons may be enclosed by an optional bulk polygon. Since the base polygon is the only device polygon to interact with all the other layers, the base layer is the device recognition polygon. The collector and emitter polygons do not interact with each other. If they touch or overlap, then it is not considered a lateral BJT. The device definition layer can

be either the emitter or collector. Hercules extraction allows the user to specify which layer is the definition layer with the BJT_MULTI_TERM_LAYER option. In Figure 3-10, two devices will be extracted, provided BJT_MULTI_TERM_LAYER=COLLECTOR is set.

Figure 3-10 Lateral BJT

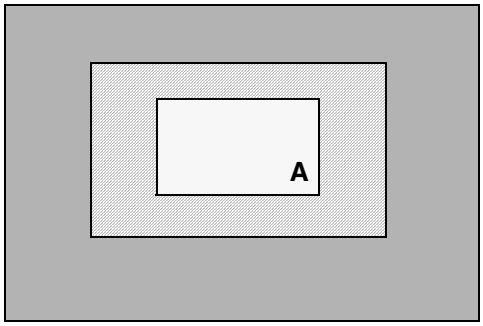


For optimal extraction of lateral BJTs, the base layer should completely enclose or exactly edge-touch the emitter and collector polygons. All of these polygons should be completely enclosed by the bulk layer if it exists.

Vertical BJT

The vertical BJT comes in two forms, normal and inverted. The normal BJT is defined by an emitter polygon and is enclosed by a base polygon, collector polygon, and an optional bulk polygon. The inverted BJT is defined by a collector polygon and is enclosed by a base polygon, an emitter polygon, and an optional bulk polygon. The normal BJT uses the emitter as the device recognition and definition layer while the inverted BJT uses the collector. The device shown in Figure 3-11 has a normal topology if layer A is the emitter. It has an inverted topology if layer A is the collector.

Figure 3-11 Vertical BJT



For optimal extraction of vertical BJTs, the recognition polygon should be completely enclosed in the base polygon, which in turn should be completely enclosed in the collector polygon (emitter for inverted BJT). All of these polygons should be enclosed in the bulk layer, if it exists.

Generating Capacitors, Diodes, Resistors, and Inductors

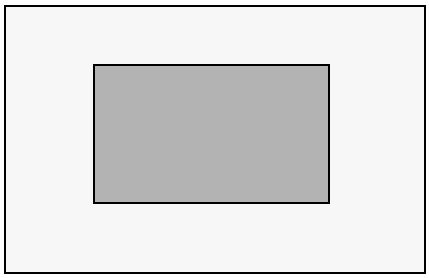
CAPACITOR Device Extraction

Capacitor devices have the same topology as diodes. The capacitor device consists of two overlapping layers, where the overlapping region is the device area. The BOOLEAN AND of the two overlapping layers is frequently used as the definition/recognition layer.

DIODE Device Extraction

The diode device consists of two overlapping layers. The overlapping region is the active device area and must be enclosed by the device recognition/definition layer. Frequently a BOOLEAN AND between the two overlapping layers is performed with the output being used as the recognition/definition layer. The layout of a diode is illustrated in Figure 3-12.

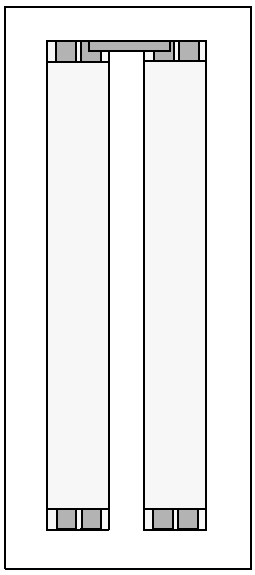
Figure 3-12 Diode Layout



RESISTOR Device Extraction

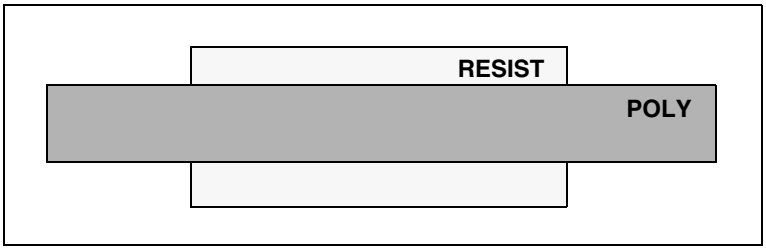
A resistor device consists of a resistive body polygon, usually path-like in shape, which has terminals at each end. An additional bulk terminal may be specified which encloses the device. Two resistors in series are shown in Figure 3-13.

Figure 3-13 Two Resistors in Series



There are many different types of resistors, but most consist of a poly layer covered by a resist (see Figure 3-14).

Figure 3-14 Resistor



A resistor typically has two bulk terminals, although sometimes it also has a third bulk terminal. The terminals are the two end connections, referred to as A and B. The RESISTOR command has the following syntax:

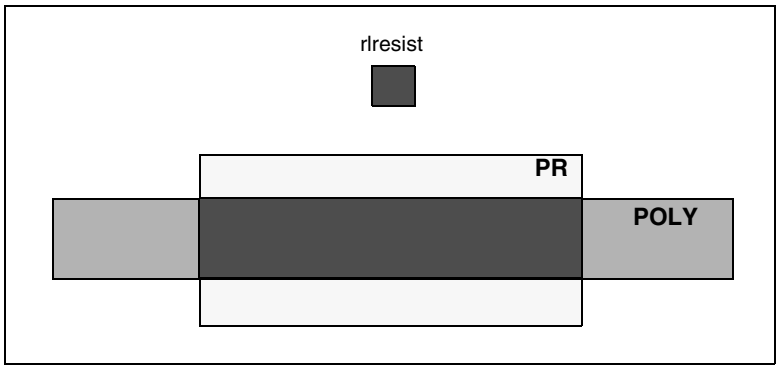
```
RESISTOR device_name device_layer term_layer term_layer
```

Argument	Description
device_name	A unique name given to the particular resistor being defined. This name cannot be the name of a layer.
device_layer	The layer in which the resistor parameters are calculated. The *device_layer* must interact with exactly one polygon from each terminal layer.
term_layer	The layer that forms the connection points to the *device_layer*. For this reason the *term_layer* must touch the device_layer.

In the example below, a poly resistor is created by a poly resist layer over a poly path (see Figure 3-15). To create the device layer, a BOOLEAN AND is performed on the poly and resist:

```
BOOLEAN POLY AND PR {} temp = r1resist
```

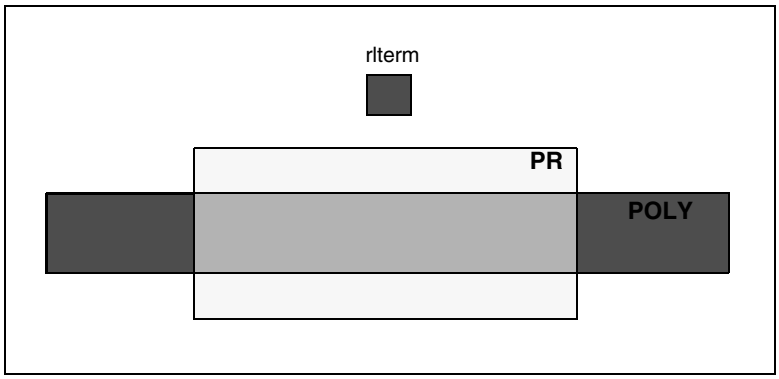
Figure 3-15 Resistor Extraction



Now that the device layer is present, the terminals of the resistor need to be defined. Since the terminal must be touching, the device layer can be NOTed away from the poly to produce terminals (see Figure 3-16). For example:

```
BOOLEAN POLY NOT r1resist {} temp = r1term
```

Figure 3-16 Resistor Extraction



With a device layer and term layers, the resist can now be defined in the runset. For example:

```
RESISTOR r1 r1resist r1term r1term {
   EV_RESVAL = 220
} temp = extract_r1
```

The input variable EV_RESVAL is required for resistors. The value is set to the number of ohms per square of unit length for the resistor type, and is used in calculating the final resistance value. For more information, see information about device equations for the RESISTOR command in the *Hercules Reference Manual*, Detailed Commands chapter.

Hercules recognizes a resistor wherever there is an rlresist layer with two rlterm layers touching it. Hercules flags device extraction errors if it finds different combinations of these layers. For example, if an rlresist layer is found with only one rlterm layer touching it, an error will be flagged.

Once the devices are defined in the runset, Hercules writes out the layout extracted netlist. This is executed with the NETLIST command. Typically this command does not need any options. For more information on the NETLIST command, see the *Hercules Reference Manual*, Detailed Commands chapter.

Now that Hercules knows how to recognize the device, it must be told how to compare the layout device with a schematic device. This is done with the EQUATE command. The syntax is:

```
EQUATE device_type schematic_name = layout_name schematic_pin1, ...,
schematic_pinN Process_Definitions}
```

Argument	Description
device_type	The type of devices, such as RES, NMOS, CAP, and others.
schematic_name	The schematic name each netlist uses for the specified type of device.
layout_name	The layout name each netlist uses for the specified type of device.
schematic_pin	
Process_Definitions	

In the following example of the EQUATE command for the resistor, the layout name is r1 and the schematic name is R.

```
EQUATE RES R=r1 A B {}
```

The two schematic pin names are from the schematic netlist, which can be obtained from the output of NetTran. When looking at a Hercules schematic netlist, a resistor may look like:

```
{inst R34=R
   {prop R=840}
   {pin VCC=A 43=B}}
```

In the pin section, pins named A and B are included. These are the schematic pins for the device type R. These pin names need to be specified in the Hercules EQUATE command so that Hercules will know which pins in the schematic map go with which pins in the layout netlist.

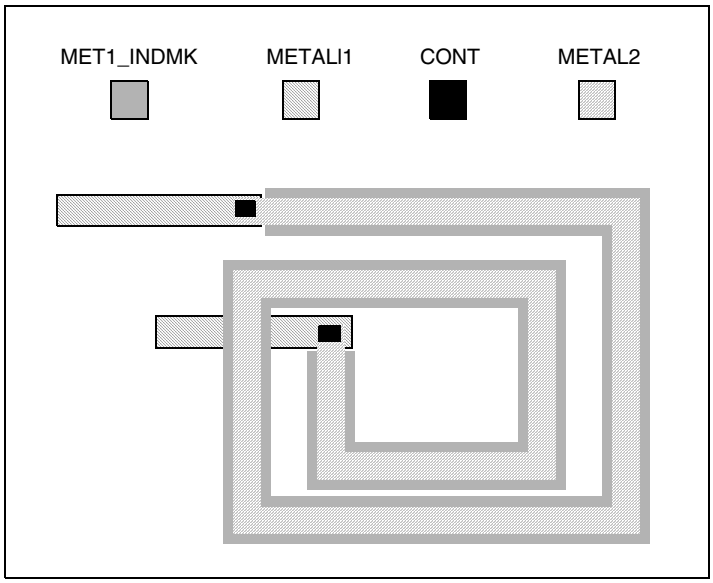
INDUCTOR Device Extraction

The INDUCTOR command extracts single-layer designed inductors that have a device layer, two terminal layers, and an optional bulk layer.

Note:
For information about device extraction for differential inductors, see the *Hercules Reference Manual*, Detailed Commands chapter, “Layer Functions for Differential Inductors” section.

A simple example of an inductor is shown in Figure 3-17.

Figure 3-17 Inductor



The device layer polygons should line interact (should not overlap or share any other relation) with exactly one polygon from each terminal layer. The inductor device layer is usually generated by an AND operation between an inductor recognition layer and the inductor material. The device layer is used to cut the inductor material using a NOT operation to form the inductor terminals.

The Hercules INDUCTOR command has the following syntax:

```
IND device_name device_layer term_layer term_layer [bulk]
```

Argument	Description
device_name	A unique name given to the particular inductor being defined. This cannot be a layer name.
device_layer	The layer of the resistor in which the inductor parameters will be calculated. The device layer must line touch one edge of each terminal layer.
term_layer	The layer that forms the connection points to the device layer.
bulk	A layer that completely encloses the entire inductor (optional).

For our example, we want a BOOLEAN AND operation between the inductor marker layer and Metal1.

```
BOOLEAN METAL1 AND MET1_INDMK {} TEMP = ind_met
```

We then define the terminal layers with a BOOLEAN NOT operation. This gives us the line touching between the device layer and the terminal layers that Hercules requires.

```
BOOLEAN METAL1 NOT ind_met {} TEMP = ind_term
```

After the device creation steps are completed, the INDUCTOR command extracts an inductor device called ind1 wherever Hercules finds the device layer ind_met line-touching two ind_term layers (for this example, there is no bulk layer):

```
INDUCTOR ind1 ind_met ind_term ind_term { } TEMP = inddev
```

All correctly formed devices will have the device layer output to the layer inddev. The *block*.sum file contains the following message:

```
Extracted 10 unique 2 terminal devices.
```

If the user specifies a bulk layer in the INDUCTOR extraction command, the message would read:

```
Extracted 20 unique 3 terminal devices.
```

If Hercules finds a device layer that does not correctly line-touch two terminal layers, then it cannot extract that device and in turn prints the following message in the *block*.sum file:

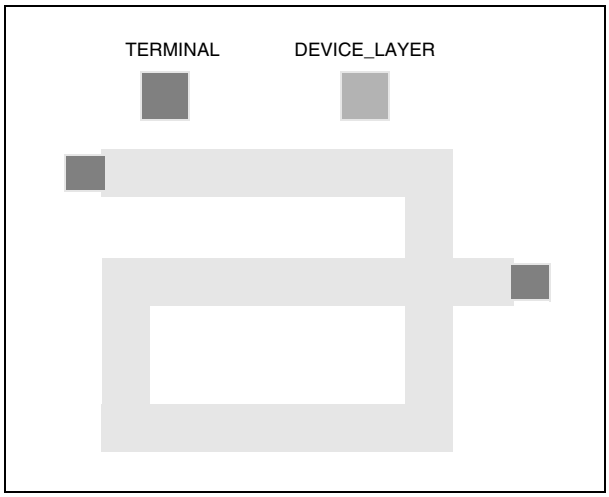
```
WARNING: 1 unrecognized device.
```

The *block*.LAYOUT_ERRORS file contains explanations on why the device was not extracted.

Three of these explanations are:

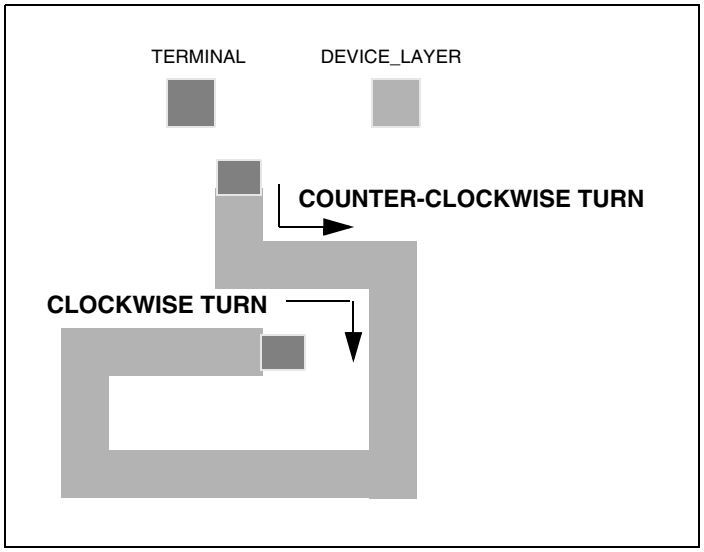
- *Too few terminals*—If Hercules finds that there are fewer terminals connected to the body than the extraction command has listed, the device will not be extracted.
- *Too many terminals*—If Hercules determines that there are more terminals connected to the body than the extraction command has listed, the device will not be extracted.
- *Bad Geometry*—The inductor command can only extract single layer inductors. That is, you cannot extract an inductor that has more than one device layer. Because of this, the inductor body cannot cross itself. An example is shown in Figure 3-18.

Figure 3-18 Crossing Device Layer



An inductor cannot change its turning direction. If the inductor curls clockwise, then there can be no point in the inductor body where it turns counter-clockwise. An example is shown in Figure 3-19.

Figure 3-19 Multi-Directional Device Layer



There are several messages that will not stop the inductor from being extracted, but give warning that the extracted device might have problems with it.

- *WARNING: Polygon of device_layer has bad graph*—This occurs mainly when the inductor is a non-orthogonal spiral, and the terminal layers cut the edges in such a way that one side of the terminal is touching an edge that is not parallel to the opposite side.
- *Polygon traces are not a complete loop*—This occurs when the number of turns in the inductor is not an integer.

Properties

The inductance is calculated as the number of squares times the EV_RESVAL.

In checking properties, there is a user-controlled tolerance for the EV_WIDTH and EV_LENGTH parameters. The default is +/- 5 percent.

The inductance can be defined using values that Hercules extracts from the device.

Table 3-2 Inductance Values Hercules Extracts from Device

Value	Definition
EV_BBAREA	Area of the bounding box of the inductor
BBHEIGHT	Height of the bounding box. It is the longer edge length of the bounding box and can use BBHEIGHT or BBH at EQUATE, CHECK_PROPERTIES section.

Table 3-2 Inductance Values Hercules Extracts from Device(Continued)

Value	Definition
BBWIDTH	Width of the bounding box. It is the shorter edge length of the bounding box and can use BBWIDTH or BBW at EQUATE, CHECK_PROPERTIES section.
EV_AREA	Area of the *device_layer* polygon
EV_EDGE	The number of edges in a loop. For a square inductor, equals 4. For an octagonal inductor, equals 8.
EV_LENGTH	Total Length of the *device_layer* polygon
EV_SPACE	Spacing between loops of the inductor
EV_TURN	Number of turns in the inductor
EV_WIDTH	Width of the *device_layer* polygon

User-Defined Device Equations

User-defined device equations modify calculated variables for all extracted devices. These equations follow standard C syntax, making use of the predefined parameters and variables corresponding to the device being extracted. Equations can be defined specifically for each extraction command. Or, they can be defined generically using the SET_PARAM command and applying that parameter set to the appropriate device extraction command. For more information on the SET_PARAM command, see the *Hercules Reference Manual*, Detailed Commands chapter.

Device Equation Syntax

The Hercules equation language was defined to allow users a method of redefining device equations. The Hercules language generally follows the C programming language in construction and syntax. However, the following two notable variations exist:

- No variable definition or type casting is required. Variables can only be doubles or strings (except the FOR loop indices, which are always integers).
- Arithmetic function names are not case-sensitive. Therefore, sqrt is equivalent to SQRT.

The Hercules equation language implements a small subset of the C language, which includes only those constructs necessary for Hercules device manipulation. These constructs are as follows:

- All standard arithmetic expressions are supported, including: - * / % (the mod operator).
- Arithmetic expressions using C math library functions are allowed. These functions follow the C syntax. The following math functions are supported:

SIN	**COS**	**TAN**
ASIN	ACOS	ATAN
EXP	POW	ROUND
SQRT	ABS	SINH
COSH	TANH	ASINH
ACOSH	ATANH	
LOG (natural log)	LOG10 (log base 10)	

- The following operators are used to handle conditional IF-ELSE expressions:

==	**conditional equal**
!=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&&	logical AND
||	logical OR
!	logical negation

Here is an example of using some of the operators:

```
if ((AREA >= 5.0) && (process == 0.8)) {
   length = pow(w1,2);
   width = w1;
}
else {
   length = pow(w1,3);
```

```
   width = w2;
}
```

- FOR loops operate on an integer loop index variable that does not need to be previously declared. Loops require the specification of the starting value of the loop index, a conditional test, and a step value. For example:

```
for (i = 1; i <= EV_TNUM; i++) {
   if (EV_AREA[i] > EV_GAREA)
       EV_PERIM[i] = process * EV_PERIM[i];
   else
       EV_AREA[i] = process * EV_AREA[i];
}
```

- The BREAK construct serves two purposes. Inside a FOR loop it causes the loop to exit prematurely. Outside a FOR loop it causes equation execution to stop at that point and all equations following the BREAK to be ignored. For example:

```
for (i = 1; i <= EV_TNUM; i++) {
   if (EV_AREA[i] > EV_GAREA)
       BREAK;
   else
       EV_AREA[i] = process * EV_AREA[i];
}
```

- The CLOSE function compares two real numbers, returning TRUE or FALSE depending on whether they are within a user-supplied tolerance value of each other. It may be used within the BOOLEAN IF statement. The syntax is:

  close (*value1, value2, tolerance*)

  If the user-supplied tolerance is >= | a-b |, the result is TRUE.

  If the tolerance is < | a-b |, the result is FALSE.

Layer-Based Property Functions

Hercules provides a list of layer-based functions for property calculations. These functions are grouped into five categories: AREA, PERIM, COUNT, BEND, and DEVICE.

Table 3-3 Property Calculation: AREA Functions

Function	Description
AREA (*pin_or_layer_name*)	Returns the area of *pin_or_layer_name*.
AREA_COIN (*pin_or_layer_name1*, *pin_or_layer_name2*)	Returns the area of *pin_or_layer_name1* coincident *pin_or_layer_name2*.

Table 3-3 Property Calculation: AREA Functions(Continued)

Function	Description
AREA_OUT (*pin_or_layer_name1*, *pin_or_layer_name2*)	Returns the area of *pin_or_layer_name1* outside *pin_or_layer_name2*.

Table 3-4 Property Calculation: PERIM Functions

Function	Description
PERIM (*pin_or_layer_name*)	Returns the perimeter of *pin_or_layer_name.*
PERIM_COIN (*pin_or_layer_name1*, *pin_or_layer_name2*)	Returns the perimeter of *pin_or_layer_name1* coincident *pin_or_layer_name2*.
PERIM_IN (*pin_or_layer_name1*, *pin_or_layer_name2*)	Returns the perimeter of *pin_or_layer_name1* inside *pin_or_layer_name2*, coincident edge excluded.
PERIM_OUT(*pin_or_layer_name1*, *pin_or_layer_name2*)	Returns the perimeter of *pin_or_layer_name1* outside *pin_or_layer_name2*, coincident edge excluded.

Table 3-5 Property Calculation: COUNT Functions

Function	Description
COUNT_EDGE (*pin_or_layer_name*)	Calculates the number of edges in the device.
COUNT_POLYGON (*pin_or_layer_name*)	Calculates the number of polygons in the device.
COUNT_CELL_DEV (*pin_or_layer_name*)	Calculates the number of merged devices enclosed by a recognition layer. When this function is called, devices with identical connections are merged into one. COUNT_CELL_DEV only merges devices if all devices within the recognition layer have identical connections. *Note:* The value of 1 is given if the recognition layer is not declared or no merging occurred.
COUNT_COIN_EDGE (*pin_or_layer_name1*, *pin_or_layer_name2*)	Calculates the number of edges that *pin_or_layer1* coincidents *pin_or_layer2*.

Table 3-5 Property Calculation: COUNT Functions(Continued)

Function	Description
MERGE_DEV_COUNT (*pin_or_layer_name*)	Counts the number of merged devices within a merged group (devices within a group have identical connections) enclosed by a recognition layer. When this function is called, devices with identical connections are merged into one. MERGE_DEV_COUNT merges parallel devices within multiple groups enclosed by a recognition layer. COUNT_CELL_DEV is different in that it requires ALL devices within a recognition layer to have the same connections.

Table 3-6 Property Calculation: BEND Functions

Function	Description
BEND (*pin_or_layer_name*)	Calculates the total number of bends in the polygon of *pin_or_layer_name1*. 45-degree bends count as 0.5 bend, 90-degree bends count as 1 bend, and 135-degree bends count as 1.5 bends.
BEND_45 (*pin_or_layer_name*)	Calculates the total number of 45-degree bends
BEND_90 (*pin_or_layer_name*)	Calculates the total number of 90-degree bends
BEND_135 (*pin_or_layer_name*)	Calculates the total number of 135-degree bends

Table 3-7 Property Calculation: DEVICE Functions

Function	Description
DEV_IACT_NUM (*layer_name*)	Returns the number of devices that share the given layer polygon. It is similar to COUNT_CELL_DEV() but it does not cause the devices to merge.

Example 3-2 shows the usage of several of the property calculation functions.

Example 3-2 Usage of Property Calculation Functions

```
DEV CCX_dev cpoly_nsink_cap cpoly<A> nsink_comp<B> _gen_buildsub<BULK> {
        PROPERTY = { new_length new_width };
        DEV_PRINT_STATS = TRUE;
        PROCESSING_LAYERS = "CAPDEF";
        DEV_NONPIN_BODY = TRUE;
        /* EV_AREA_CAPVAL = 4.93e-15; */
```

```
        /* set property value */
        new_length = PERIM_COIN(cpoly_nsink_cap,CAPDEF)/2;
        new_width = AREA(cpoly_nsink_cap)/new_length;
} TEMP = _generated_output_layer
```

Multiple Model Naming (1-to-N)

Hercules can generate devices with different model names from a single device extraction command. This is referred to as 1-to-N multiple model name support, where one device command produces devices of several different names. This is implemented through the device option EV_MODEL_NAME. The extracted device will get the default name given as part of the device command, until it is overwritten by the EV_MODEL_NAME option.

In the following example, extracted devices will have the model name of pfet unless the area of the gate is greater than ten, in which case the extracted device will have the model name of bigP.

```
PMOS pfet pgate psd psd NWELL {
   IF (EV_GAREA > 10) {
       EV_MODEL_NAME = "bigP";
   }
} TEMP = pdevice
```

Compare Implications: In the example above

- Hercules requires an EQUATE for pfet because it is the default model name.
- If an EQUATE for bigP exists, it will be used; if not, one will be generated for it using the EQUATE for pfet as a template and changing only the device name.

See “Dynamic Model Naming” on page 3-27.

Parametric device name assignment occurs when the device name can be assigned based on one or more parameter values.

Dynamic Model Naming

Hercules has the ability to build a model name for a device from parameters of the device being extracted. This is referred to as Dynamic Model Naming.

This keys off of the 1-to-N multiple model name feature, and is implemented through the SPRINTF function. This function is used to combine multiple variables of different types into a string that is used as the model name for the device. The basic syntax is:

```
SPRINTF(variable_name, format, arguments);
```

Argument	Description
variable_name	Name of the variable that will be set to the results of the SPRINTF function.
format	Definition of how to format the arguments given. Plain text entries are simply typed in, variables to be included are identified with %s for string type variables and %d for variables of type double.
arguments	A list of the variables to be formatted and included as part of the string created by SPRINTF. Their order, and type, need to match the declarations in the format section.

Example 3-3 shows dynamic model naming. See "Layer-Based Property Functions" on page 3-24 for a complete listing of available functions.

Example 3-3 Dynamic Model Naming

```
NPN layNPN bi_emit bi_base bi_col  {
   property = {b_area};
   b_area = area(bi_base);
   BJT_TOPOLOGY=NORMAL
   /* 1st character: 'N' */
   /* 2nd character: count of base */
   base_count = COUNT_POLYGON(bi_col);

   w_emit = <user equation for emitter width> ;
   l_emit = <user equation for emitter length> ;

   r_w_emit = ROUND(w_emit - 0.5); /* truncate integer part of value */
   f_w_emit = w_emit - r_w_emit;   /* truncate decimal part of value */
   tenx_f_w_emit = ROUND(10 * f_w_emit);

   /* 3rd -- 5th character: width of emitter */
   sprintf(NAME35, "%dp%d", r_w_emit, tenx_f_w_emit);

   /* 6th character: 'x' */

   r_l_emit = ROUND(l_emit - 0.5); /* truncate integer part of value */
   f_l_emit = l_emit - r_l_emit;   /* truncate decimal part of value */
   tenx_f_l_emit = ROUND(10 * f_l_emit);

   /* 7th -- 9th character: length of emitter */
   sprintf(NAME79, "%dp%d", r_l_emit, tenx_f_l_emit);
                                          /* So 4.8 looks like 4p8 */

   /* assign the model name dynamically by SPRINTF and EV_MODEL_NAME */
```

```
    SPRINTF(EV_MODEL_NAME, "N%d%sx%s", base_count, NAME35, NAME79);

    /* final model name will look like N21p3x4p0 */

} TEMP = output_layer
```

Note the use of the ROUND function in Example 3-3. This function returns the result of rounding the given value to the nearest whole integer. For example:

ROUND(3.2) = 3
ROUND(26.8) = 27

The ROUND function can be used as a truncate function by subtracting 0.5 from the input value.

For example:

to truncate 3.2: ROUND(3.2-0.5) = ROUND(2.7) = 3
to truncate 26.8: ROUND(26.8-0.5) = ROUND(26.3) = 26

Compare Implications:

For the COMPARE portion of Example 3-3, Hercules requires an EQUATE for the default name, layNPN. Thus, the runset requires:

```
EQUATE NPN schNPN = layNPN E B C {
   Option1 = true
   Option2 = false
}
```

Hercules keeps track of all the device names it creates, and generates additional EQUATEs for each one using this template EQUATE. For example, if you created a device named N21p3x4p0, Hercules would create the EQUATE:

```
EQUATE NPN N21p3x4p0 = N21p3x4p0 E B C {
   Option1 = true
   Option2 = false
}
```

Both the schematic name and layout name are replaced with the created device name.

Proximity

CALC_COP_ARRAY

The CALC_COP_ARRAY function fractures diffusion polygons into regions based on the placement of diffusion contacts, and returns the area of the diffusion regions that are split by these contacts. Contacts that are fully or partially enclosed by the source and drain layers

are considered. This function is invoked separately for source and drain pins, meaning that separate calls to the function are necessary for source area fracturing and for drain area fracturing.

The diffusion polygon is fractured according to the placement of the contact polygon in two ways:

- The diffusion polygon is fractured along lines between the gate layer edge and the center point of each contact layer polygon.
- The diffusion polygon is fractured along lines running exactly mid-way between and parallel to fracture lines created by contact polygon center points.

```
center_x = (lt.x + rb.x) / 2
center_y = (lt.y + rb.y) / 2
```

Figure 3-20 shows an example of contact center point fracture lines.

Figure 3-20 Contact Center Point Fracture Lines

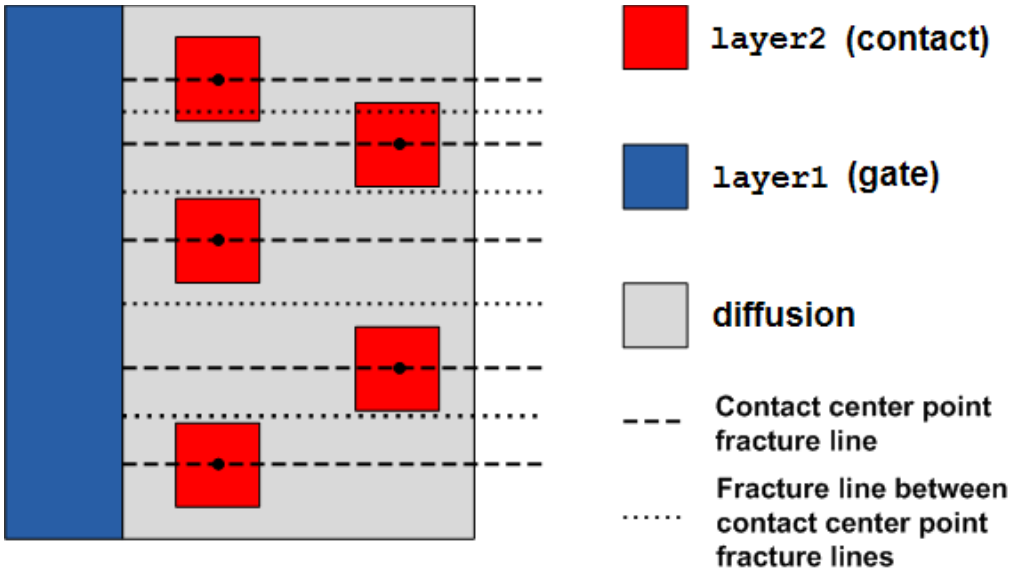
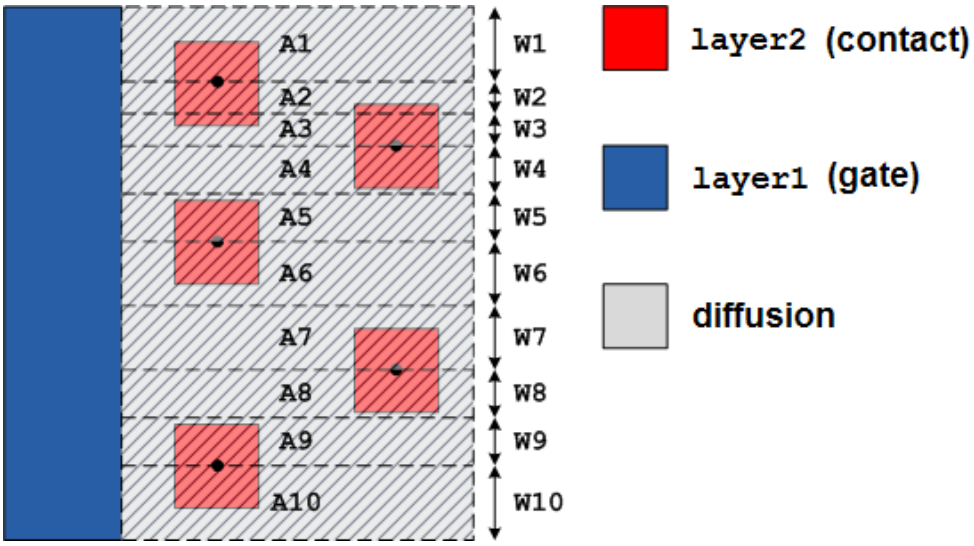


Figure 3-21 shows an example of the areas and widths into which diffusion is fractured.

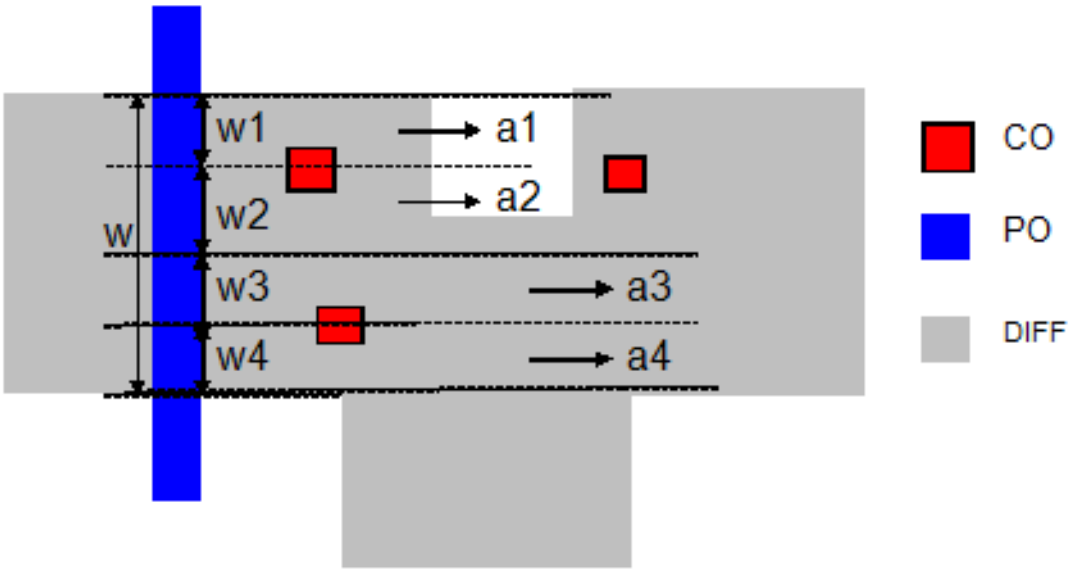
Figure 3-21 Area and Widths of Fractured Regions



Note:

No fracture line is created for contacts whose center points fall beyond the edge of the body layer. All fracture lines must be perpendicular to the body layer edge and must touch both the body layer edge and the contact center point.

The following example shows how you can generate user arrays containing the areas of diffusion regions fractured according to contact position, as well as sum the diffusion areas and widths.



w2 = w3
w1 + w2 + w3 + w4 = w
ai : area of each OD segment

```
NMOS n ngate nsd nsd SUBSTRATE {
   PROPERTY = { src_area[], src_width[],src_area_sum, src_width_sum }
   MOS_SOURCE_DRAIN_CONTACT = {diff_con};
   MOS_CALC_NRS_NRD = FALSE;
   src_cop_num = 0;
   calc_cop_array(ngate, SRC, diff_con, src_area, src_width,
   src_cop_num);
   src_area_sum = 0.0;
   src_width_sum = 0.0;
   for( i=0; i < src_cop_num; i=i+1) {
      src_area_sum = src_area_sum + src_area[i];
      src_width_sum = src_width_sum + src_width[i];
   }
} temp = ndev
```

The resulting layout netlist for the device would be:

```
{NETLIST top
{VERSION 1 0 0}
/* Level 0 */
{CELL top
   {PORT}
   {PROP top=0.670000 bottom=0.040000 left=0.200000 right=1.078000}
```

```
   {INST M1=n {PROP x=0.320 y=0.372  l=0.070 w=0.445 src_area_sum=0.181
      src_width_sum=0.445 src_area0=0.018 src_area1=0.024 src_area2=0.047
src_area3=0.090
      src_width0=0.088 src_width1=0.117 src_width2=0.116
      src_width3=0.125}
      {PIN N_4=GATE N_2=SRC N_3=DRN N_1=BULK }}
   }
 }
```

Notice the following details from the example:

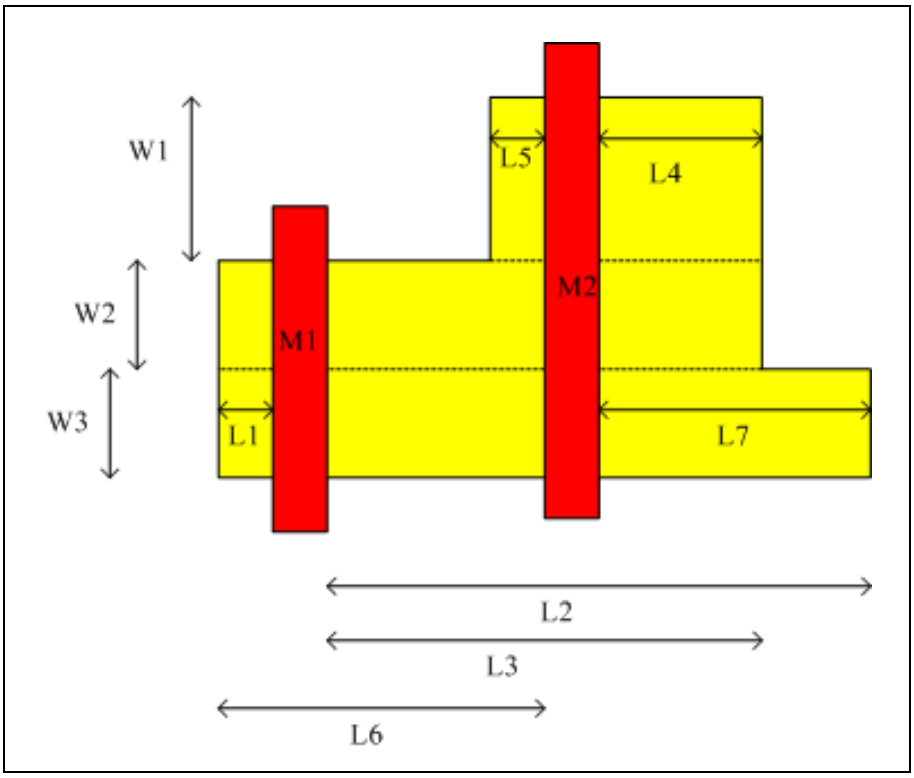
- When specifying a user array in the PROPERTY line, the array name must be followed by square brackets [] to indicate an array of values.
- In order to retain the contact layer (diff_con in the example) as an input to CALC_COP_ARRAY, the layer must be specified using MOS_SOURCE_DRAIN_CONTACT. If NRS and NRD property values based on MOS_SOURCE_DRAIN_CONTACT are not desired, also specify MOS_CALC_NRS_NRD=FALSE.
- In the preceding example, CALC_COP_ARRAY calculates only properties for the SRC pin of each NMOS/PMOS device. To calculate properties for the DRN pin, CALC_COP_ARRAY should be called a second time by specifying DRN for the second argument.

For more information on syntax for the CALC_COP_ARRAY function, see the *Hercules Reference Manual*, Detailed Commands chapter.

CALC_LOD_ARRAY

You can extract an array of values representing the distance from the gate layer to the edge of the LOD (Length of Oxide Definition) layer using the CALC_LOD_ARRAY function. This value is measured perpendicularly to the gate layer (in the l direction) and goes through any other gates that share that same LOD layer.

Note:
The maximum index value of the array (i) is 10 if you want to save data in the LPE_DATABASE and use the value as a property.



Thus, when processing,

For M1: For M2:

The diagram below illustrates this:

```
num=2       num=3

                    SW[0]=W1
                    SA[0]=L5
                    SB[0]=L4

SW[0]=W2    SW[1]=W2
SA[0]=L1    SA[1]=L6
SB[0]=L3    SB[1]=L4

SW[1]=W3    SW[2]=W3
SA[1]=L1    SA[2]=L6
SB[1]=L2    SB[2]=L7
```

The following example shows how you can generate the average SA and SB values for each device, and total SW value:

```
PMOS pfet pgate psd psd NWELL {
   PROPERTY = {SA, SB}
   PROCESSING_LAYERS = {PDIFF}
   CALC_LOD_ARRAY (pgate, pdiff, SAARRAY, SBARRAY, SWARRAY
                     snum);
   A = 0;
   B = 0;
   SW = 0;
   for (i = 0; i < snum; i++) { /* note index of
                              * first value in array
                              * is 0 (not 1) */
      A = A + SAARAY[i];
      B = B + SBARRAY[i];
      SW = SW + SWARRAY[i];
   }
   SA = A/snum;
   SB = B/snum;
} TEMP = pdevice
```

If you wanted to keep all of the LOD parameters separate and simply alter the names from the default names (EV_LOD_SA, EV_LOD_SB, EV_LOD_SW) to some other name (SA, SB, SW), you could do it this way:

```
PMOS pfet pgate psd psd NWELL {
   PROCESSING_LAYERS = {PDIFF}
   MOS_LOD_LAYER = {PDIFF}
   PROPERTY = {SW[], SA[], SB[] }
   CALC_LOD_ARRAY (pgate, PDIFF, SA, SB, SW, snum);
} TEMP = pdevice
```

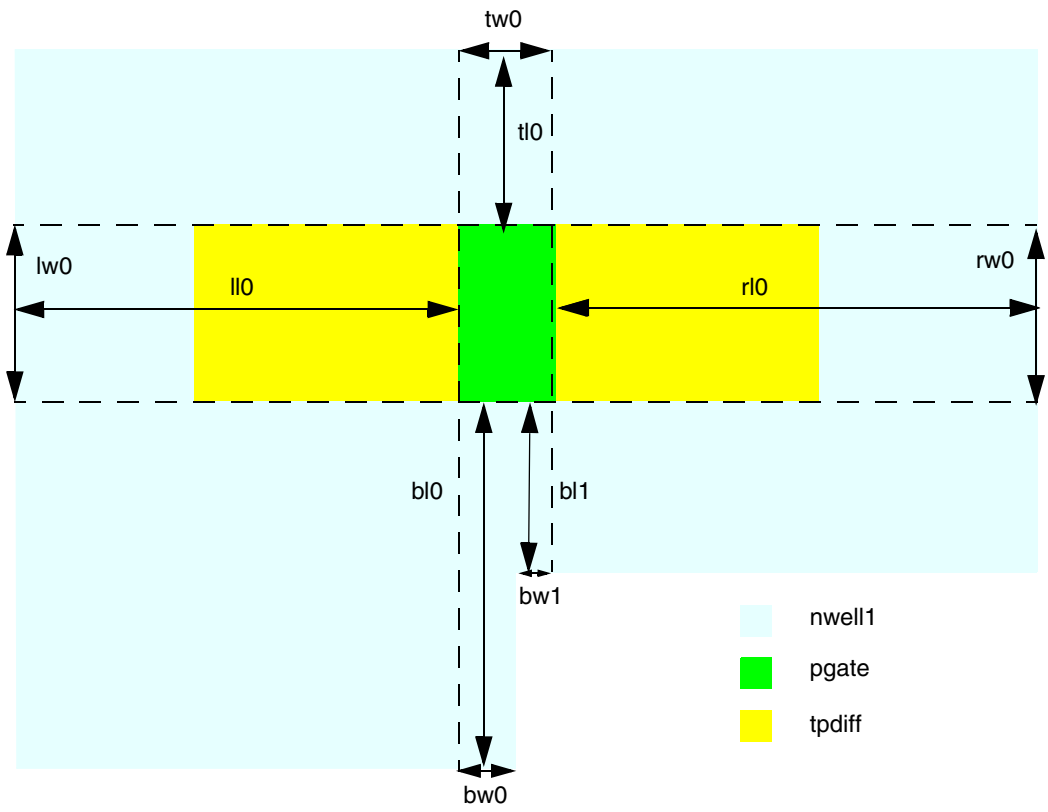
Notice the square brackets [] in the PROPERTY line. This indicates that the variable is an array of values.

For more information on syntax for the CALC_LOD_ARRAY function, see the *Hercules Reference Manual*, Detailed Commands chapter.

CALC_PROJ_DIST_ARRAY

You can calculate nwell proximity effects using the CALC_PROJ_DIST_ARRAY function.

For example, consider the pmos device:



To output the values of nwell proximity for the above pmos, we need to declare the user arrays tl, tw, bl, bw, rl, rw, ll, lw. For example:

```
PMOS LOD_PMOS pgate tpdiff tpdiff nxwell {
   PROPERTY = {tl[](U),tw[](U),bl[](U),bw[](U),rl[](U),rw[](U),
          ll[](U),lw[](U)} */ Note the use of square brackets. */
   PROCESSING_LAYERS = { nwell1 }
   calc_proj_dist_array       (pgate,nwell1,tpdiff,tl,tw,tnum,EV_TOP);

   calc_proj_dist_array       (pgate,nwell1,tpdiff,bl,bw,bnum,EV_BOTTOM);

   calc_proj_dist_array       (pgate,nwell1,tpdiff,rl,rw,rnum,EV_RIGHT);

   calc_proj_dist_array       (pgate,nwell1,tpdiff,ll,lw,lnum,EV_LEFT);
} TEMP=_generated_output_layer
```

The layout netlist for this device would be:

```
{INST M17=LOD_PMOS {PROP x=1.420 y=2.255  l=0.130 w=0.230
tl0=2.395 tw0=0.130 bl0=7.675  bl1=3.215 bw0=0.130 bw1=0.170
rl0=7.680 rw0=0.060 ll0=2.250 lw0=0.230}
```

Example 3-4 shows combining properties as output. This example shows defining the average of vertical-length and summation of horizontal-width as output. Therefore, the output netlist will only contain vert_l_avg, horz_w_sum, and default properties.

Example 3-4 Nwell Proximity with Combined Output Properties

```
PMOS LOD_PMOS pgate tpdiff tpdiff nxwell {
   PROPERTY = {vert_l_avg, horz_w_sum}
   PROCESSING_LAYERS = { nwell1 }
   calc_proj_dist_array (pgate, nwell1, tpdiff, tl, tw, tnum, EV_TOP);
   calc_proj_dist_array (pgate, nwell1, tpdiff, bl, bw, bnum, EV_BOTTOM);
   calc_proj_dist_array (pgate, nwell1, tpdiff, rl, rw, rnum, EV_RIGHT);
   calc_proj_dist_array (pgate, nwell1, tpdiff, ll, lw, lnum, EV_LEFT);
   vert_l_sum=0
   for (I=0;I<rnum;I=I+1)
      vert_l_sum=vert_l_sum+rl[I];
   for (I=0;I<lnum;I=I+1)
      vert_l_sum=vert_l_sum+ll[I];
   vert_l_avg = vert_l_sum/(rnum+lnum);
   horz_w_sum=0
   for (I=0;I<tnum;I=I+1)
      horz_w_sum=horz_w_sum+tw[I];
   for (I=0;I<bnum;I=I+1)
      horz_w_sum=horz_w_sum+bw[I];
} TEMP =_generated_output_layer
```

For more information on syntax for the CALC_PROJ_DIST_ARRAY function, see the *Hercules Reference Manual*, Detailed Commands chapter.

CALC_STRESS_ARRAY

Strained silicon effects are extracted using the CALC_STRESS_ARRAY function. This function extracts distance projecting from the gate width and length to nearby layers within a user specified search window range for MOS devices. Nearby layers are usually polysilicon, diffusion and well layers. The nearby layers do not have interaction requirements with gate layer.

This function is available only for the NMOS and PMOS extraction commands and processes straight and bent gates with vertical or horizontal orientations.

The following example shows how the CALC_STRESS_ARRAY function extracts stress effect properties for NMOS devices. This function checks the first and second nearby PO and NWEL_COPY edges in the left and right directions and only the first and second nearby NWEL_COPY edges in the top and bottom directions. Equation code is also included as an example of looping through the se_r user array in the right direction.

```
NMOS nch nmos_device n_src_drn n_src_drn {
   PROCESSING_LAYERS = { NWEL_COPY, PO, ptox }
   CALC_STRESS_ARRAY (nmos_device, PO, NWEL_COPY, n_src_drn,
      sp_r, se_r, pw_r, num_r, 2, 2, 8, EV_RIGHT);
   CALC_STRESS_ARRAY (nmos_device, PO, NWELL_COPY, n_src_drn,
```

```
      sp_l, se_l, pw_l, num_l, 2, 2, 8, EV_LEFT);
   CALC_STRESS_ARRAY (nmos_device, PO, NWELL_COPY, n_src_drn,
      sp_t, se_t, pw_t, num_t, 0, 2, 8, EV_TOP);
   CALC_STRESS_ARRAY (nmos_device, PO, NWELL_COPY, n_src_drn,
      sp_b, se_b, pw_b, num_b, 0, 2, 8, EV_BOTTOM);
   PROPERTY = {sp_r[], sp_l[], sp_t[], sp_b[], se_r[],
      se_right_sum = 0;
      sp_right_sum = 0;
      for(i=0; i<num_r*2; i=i+1){ /*nearby_layer3_num is 2*/
        se_right_sum = se_right_sum + se_r[i]; /* This will sum
      all gate to layer3 projections to the right of the
      layer1.*/
        }
} TEMP=_generated_output_layer
```

See Figure 3-22 for information on extracting stress effect properties for NMOS devices.

Note:
The se_r user array size is equal to the num_r value only for a nearby_layer3_num value of 1. For a nearby_layer3_num larger than 1, the layer3 array index is defined between 0 and num*nearby_layer3_num - 1. (The for loop in the NMOS command example above shows how to access the se_r array for nearby_layer3_num values > 1.)

Note:
User arrays and variables are accessible through Hercules device equation syntax for user-defined mathematical processing. For more information on device equation syntax, see “Device Equation Grammar” on page 3-50

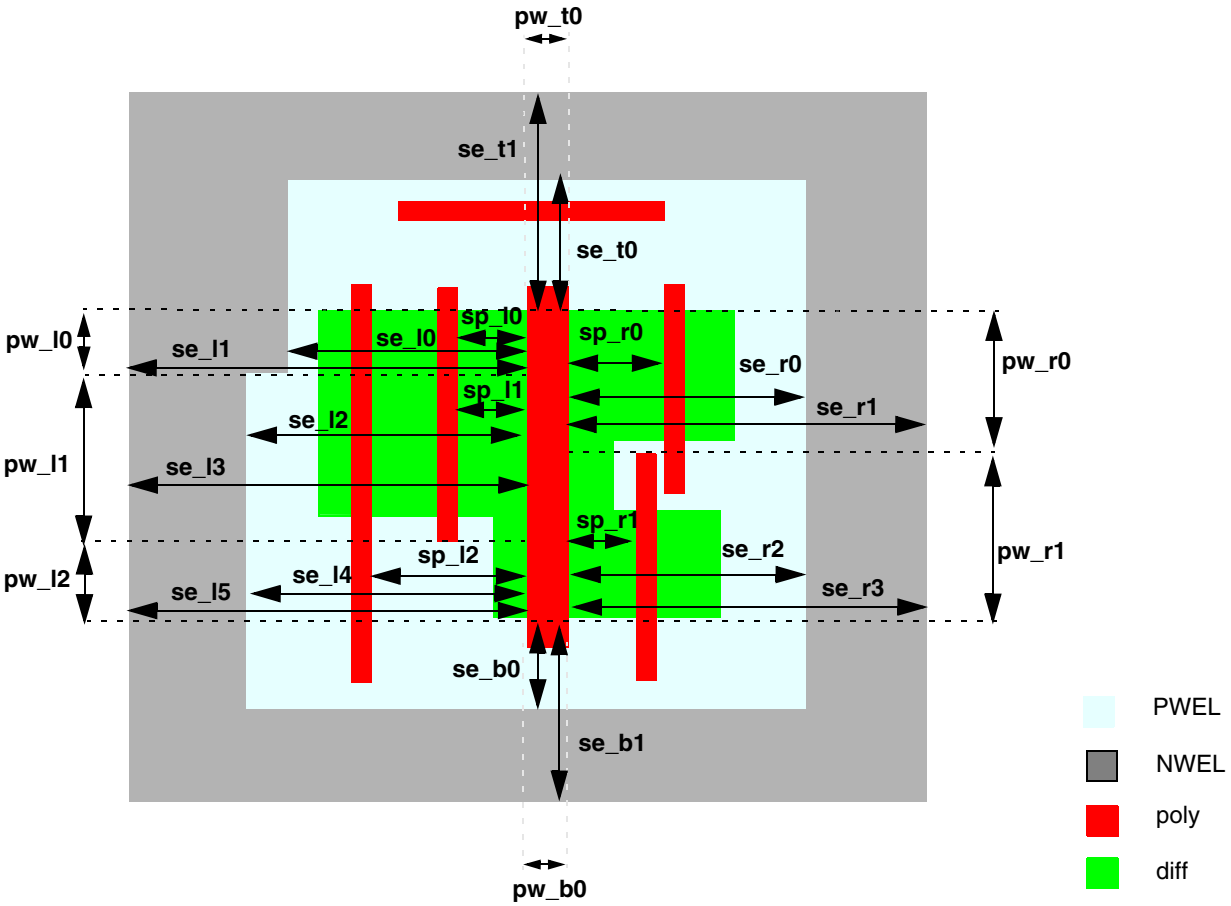
Note:
The bulk terminal layer for the PMOS and NMOS extraction commands cannot be used as a processing layer within the CALC_STRESS_ARRAY function. To do this, add a renamed equivalent from the bulk terminal layer to the PROCESSING_LAYER option.

Example 1: Extracted Netlist

```
{INST M3=nch {PROP x=5.950 y=5.600 l=0.500 w=8.000
sp_r0=2.600 sp_r1=8.000 sp_r2=1.200 sp_r3=2.600 sp_r4=1.200
sp_r5=8.000 sp_l0=1.600 sp_l1=3.400 sp_l2=1.600 sp_l3=3.400
sp_l4=3.400 sp_l5=8.000 se_r0=5.146 se_r1=8.000 se_r2=5.146
se_r3=8.000 se_r4=5.146 se_r5=8.000 se_right_sum=39.438
se_l0=5.203 se_l1=8.000 se_l2=5.878 se_l3=8.000 se_l4=5.878
se_l5=8.000 se_t0=2.973 se_t1=4.450 se_b0=1.691 se_b1=4.055}
```

Figure 3-22 shows that distance projections are extracted from the NMOS gate to the edge entering the poly layer, and from the NMOS gate layer to the edge entering and leaving the NWEL layer.

Figure 3-22 Extracting Stress Effect Properties with NMOS Devices



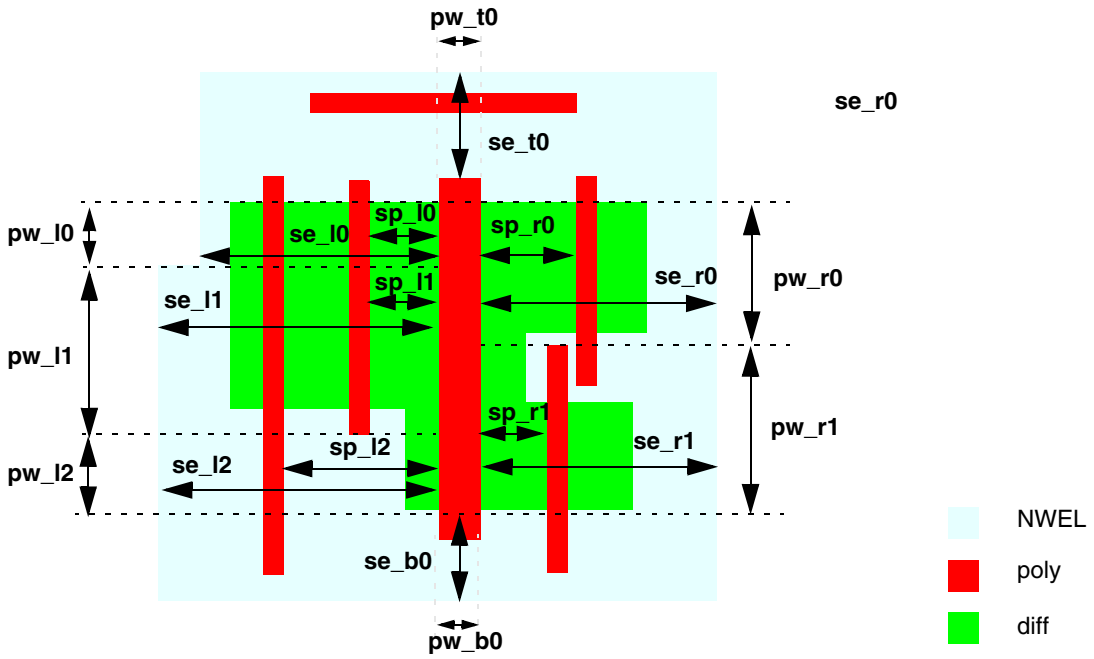
Example 2: Extracted Netlist and Runset Code

```
{INST M3=nch {PROP x=5.950 y=5.600 l=0.500 w=8.000 se0=0.400 se1=1.000}
```

```
CALC_STRESS_ARRAY (NWEL, poly, diff, align_layer, sp, se, pw, num,
1, 1, 1, EV_TOP , 0.5 );
```

Figure 3-23 shows that distance projections are extracted from the PMOS gate to the edge entering the poly layer, and from the PMOS gate layer to the edge leaving the NWEL layer.

Figure 3-23 Extracting Stress Effect Properties with PMOS Devices



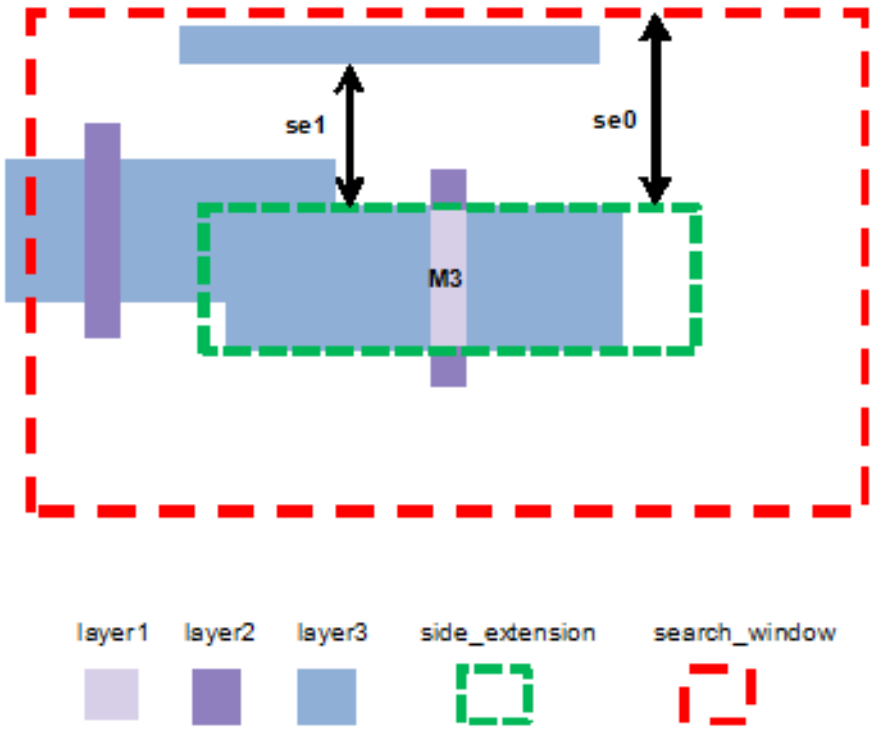
Example 3: Extracted Netlist and Runset Code

```
{INST M3=nch {PROP x=5.950 y=5.600 l=0.500 w=8.000 se0=0.400 se1=1.000}
```

```
CALC_STRESS_ARRAY (layer1, layer2, layer3, align_layer, sp, se, pw, num,
1, 1, 1, EV_TOP , 0.5 );
```

Figure 3-24 illustrates how the CALC_STRESS_ARRAY function can be used to accurately extend the projection region in the left and right directions when extracting parameters in the EV_TOP direction. Notice that by extending the projection region via CALC_STRESS_ARRAY for a device, merging of projections regions from neighboring devices is avoided.

Figure 3-24 Example of side_extension Option



For more information on the CALC_STRESS_ARRAY function syntax, see the *Hercules Reference Manual*, Detailed Commands chapter.

CALC_WPE_CORNER

The CALC_WPE_CORNER property-calculation function allows you to locate the pointed concave corners for the WPE (well proximity effect) corner effect. The function extracts projection distances between the gate vertices (on the body_layer) and the corners of the NWELL layer (proj_layer). The CALC_WPE_CORNER function is available only for the NMOS and PMOS extraction commands.

The syntax of the function is:

```
CALC_WPE_CORNER( body_layer, proj_layer, align_layer,
   scv, sch, num, Smax, direction);
```

Example of using CALC_WPE_CORNER:

```
NMOS norm_gate Gate Diff Diff {
  PROCESSING_LAYERS = { NWASP, Diff }
  MOS_MULTITERM_EXTRACT = TRUE
  CALC_WPE_CORNER(Gate, NWASP, Diff, ltv, lth, ltnum, 5, EV_LEFT_TOP);
```

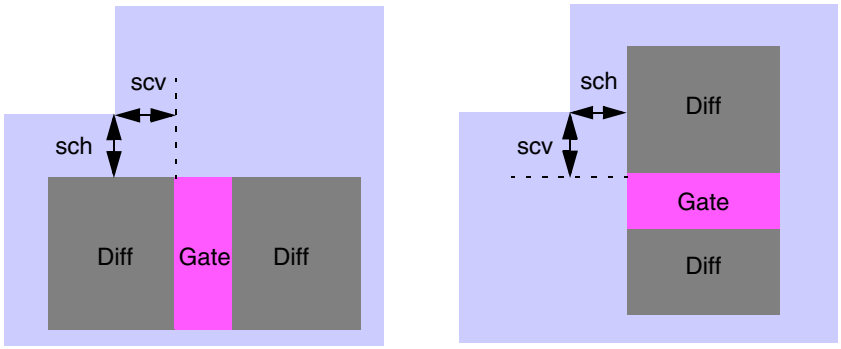
```
  CALC_WPE_CORNER(Gate, NWASP, Diff, lbv, lbh, lbnum, 5, EV_LEFT_BOTTOM
);
  CALC_WPE_CORNER(Gate, NWASP, Diff, rtv, rth, rtnum, 5, EV_RIGHT_TOP);
  CALC_WPE_CORNER(Gate, NWASP, Diff, rbv, rbh, rbnum, 5,
EV_RIGHT_BOTTOM);
  PROPERTY = {ltnum,ltv[],lth[],lbnum,lbv[],lbh[],
              rtnum,rtv[],rth[],rbnum,rbv[],rbh[] }
} TEMP = nbody
```

Notice the square brackets [] in the PROPERTY line. This indicates that the variable is an array of values.

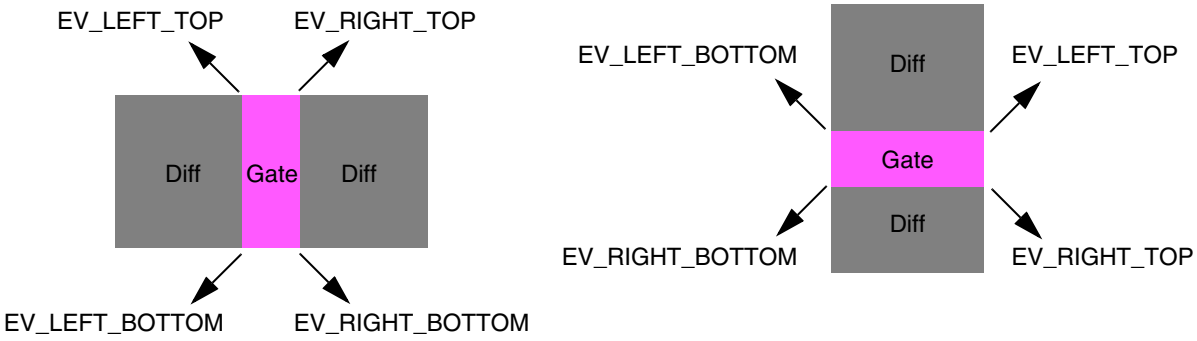
The resulting layout netlist for this device would be:

```
{NETLIST TOP_CELL_FINAL
{VERSION 1 0 0}
/* Level 0 */
{CELL TOP_CELL_FINAL
   {PORT}
   {PROP top=-192.700000 bottom=-196.500000 left=253.000000
      right=257.200000}
   {INST M1=norm_gate {PROP x=255.130 y=-194.600  l=0.220 w=1.240
      ltnum=3.000 ltv0=1.090 ltv1=0.890 ltv2=0.130 lth0=0.150 lth1=0.450
      lth2=0.830 lbnum=3.000 lbv0=0.790 lbv1=0.990 lbv2=0.230 lbh0=0.710
      lbh1=0.310 lbh2=0.790 rtnum=3.000 rtv0=0.790 rtv1=1.190 rtv2=0.170
      rth0=0.750 rth1=0.250 rth2=0.290 rbnum=3.000 rbv0=0.990 rbv1=0.990
      rbv2=0.150 rbh0=0.110 rbh1=1.010 rbh2=0.310}
     {PIN N_2=GATE N_3=SRC N_3=DRN }}
}
}
```

X- and Y-direction Double Arrays

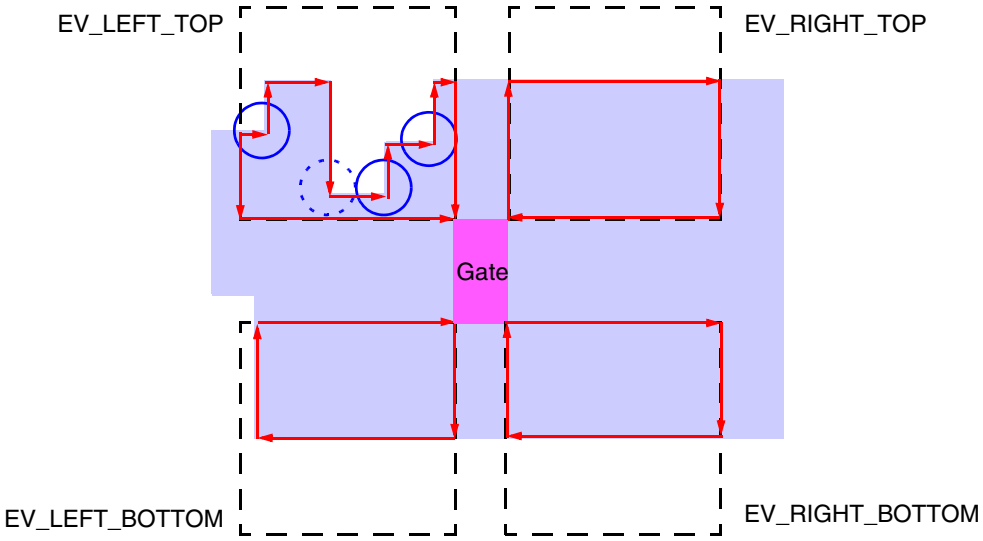


Direction Determination



Polygon Selection

The selected pointed concave corners are marked with solid blue circles in this example. The concave corner marked with the dotted blue circle is not selected because it is not pointed.



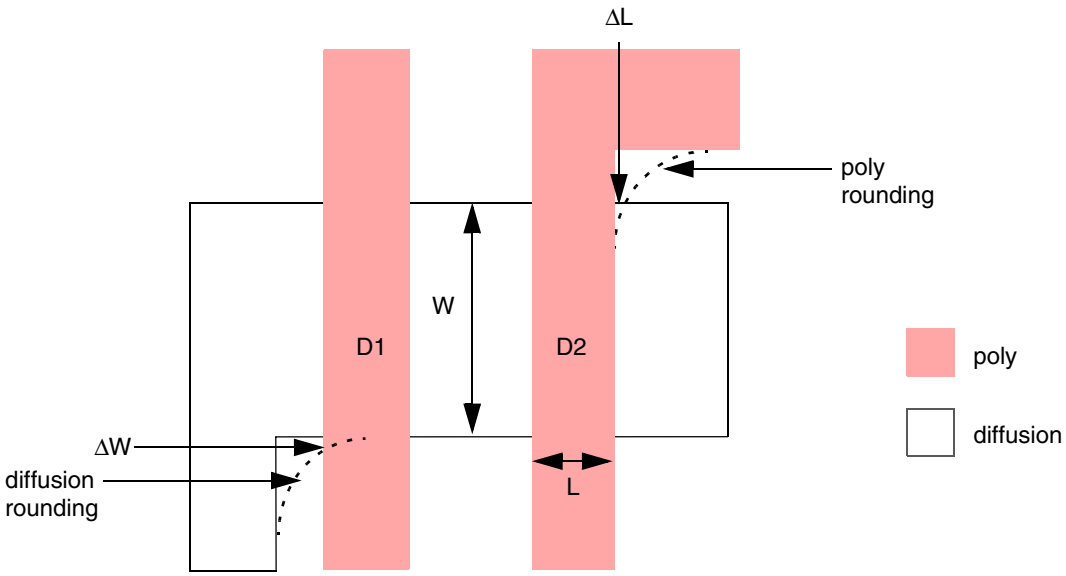
For more information on the syntax for the CALC_WPE_CORNER function, see the *Hercules Reference Manual*, Detailed Commands chapter. For a full description of well proximity effect model, see "BSIM4.5.0 MOSFET Model - User's Manual," Chapter 14 (Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA).

Table-Based Lookup Functionality

Table-based lookup functionality is a mechanism by which layout engineers can import post-manufacturing effects into the design cycle for better simulation results. These post-manufacturing effects can be sampled, then populated in lookup tables. The information in the lookup tables can be imported into the Hercules flow by any device extraction command and the MOSCHECK command.

For example, one use of the table-based lookup functionality in DFM applications is to model the effective changes in a MOS device channel length and width due to post-manufacturing contour inaccuracies introduced by L-shaped poly or diffusion layers. The dotted line in the figure below shows how the effective channel length and width can change after manufacturing from L-shaped poly and diffusion contours.

- W is the drawn channel width. L is the drawn channel length.
- ΔW and ΔL are the change in W and L after manufacturing.
- Effective channel width is W' = W + ΔW. Effective channel length is L' = L + ΔL



You can save the width and length changes via a lookup table, then use the get_dfm_lookup_value() function in a device extraction command, or the MOSCHECK command, to return the width and length changes at the runset level. You can then use these values to change the effective width and length for better simulation accuracy.

Using Table-Based Lookup

To use a lookup table:

1. You need a lookup table that is already defined. See Lookup Table Structure.
2. Define the path to the file that has the lookup table. See DFM_TABLE_LOOKUP_FILE Option.
3. Use the table-lookup extraction function (get_dfm_lookup_value) within the device extraction commands. See Table-Lookup Extraction Function.

DFM_TABLE_LOOKUP_FILE Option

To define the path to the file that has the lookup tables, use the HEADER option:

```
HEADER { DFM_TABLE_LOOKUP_FILE =
            table_lookup_file; ... ; table_lookup_file }
```

Hercules searches the files, in order, to find the table specified in the table-lookup extraction function. The *table_lookup_file* can be either a relative of full path.

Table-Lookup Extraction Function

The table-lookup extraction function returns a delta length (ΔL) or delta width (ΔW) based on the input table name and specified array values. You may use more than one extraction function in a device extraction command.

The table-lookup extraction function syntax is:

```
get_dfm_lookup_value(table_name, input_array)
```

table_name

Name used to locate the lookup table within the files specified by the DFM_TABLE_LOOKUP_FILE option. Table names are case-sensitive.

input_array

Values used in referencing the table to determine the return value. The input array size must match the table dimension. For example, an error message is reported if the lookup table is 5-D but the input array size is 6. The input array can contain numeric variables created within the device extraction command.

Use the returned value to compute the effective length (L') and width (W') within a device extraction command and netlist. In the example below, the DFM input array contains the numeric values created from the PERIM_COIN layer-based functions. It is then used in the get_dfm_lookup_value function to find the delta width.

```
NMOS nfet gate src_drn src_drn {
   DFM[0] =  PERIM_COIN ( ox, src_drn);
   DFM[1] =  PERIM_COIN ( ox, src_drn);
   DFM[2] = 1;
   delta_width = get_dfm_lookup_value(rounding_effects_3d,DFM);
} TEMP = nbody
```

Lookup Table Structure

Within a table-lookup file:

- The `begin` statement is the first line in the table. The table name is defined on this line.
- The next line contains the table dimension following the number of data points for each dimension.
- Then, there is one input data row for each dimension. The data in each row must increase in value from left to right. The values are separated by a space.

  Note:
  Adjacent input values cannot be equivalent as this would cause divide by zero errors when interpolating to determine the return value.

- The return data follows the input data.

  The number of data points in each row of return values must equal the number of data points for the first dimension. In the example below, the first dimension (p) is 5; therefore, each row of return values must have five data points.

  The number of rows containing return values must equal the product of the other dimensions. In the example below, the number of required rows of return values is q*r or 4*2=8. If a fourth dimension containing 3 input values was added, the number of required rows would be q*r*s or 4*2*3=24.

- The table can be of any dimension. The format requirements apply to all dimensions.
- Lines that start with # are comments. They are allowed anywhere.
- Empty lines are allowed anywhere.
- An `end` statement follows the last row of return values for any given table.
- Multiple tables can be defined, each with a unique name and delimited with `begin` and `end` statements.
- The table file can be encrypted using the Hercules vpcrypt executable.

  ```
  %vpcrypt plain_text.txt encrypted.txt
  ```

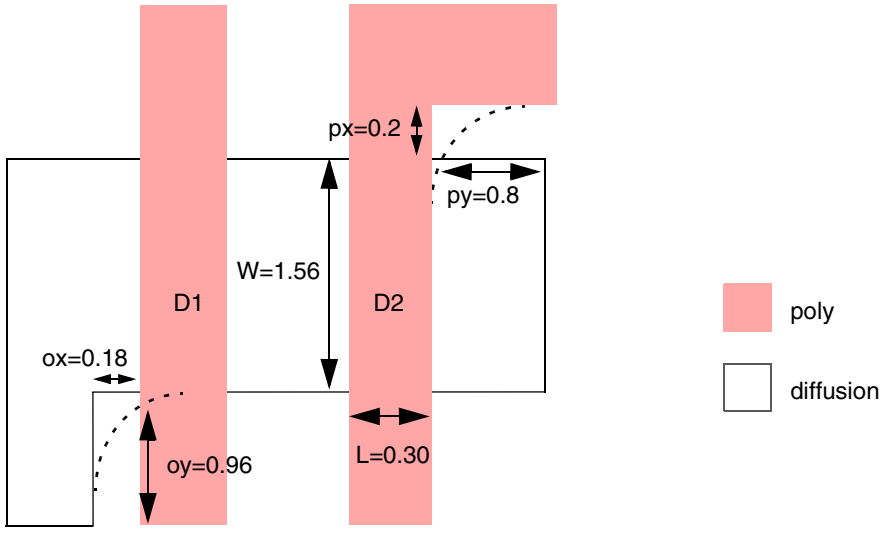
Here is an example 3-D table structure:

```
# 3-D table example
begin rounding_effects_3d
3 5 4 2                                  ← dimension p q r
0.00 0.12 0.16 0.18 0.22                 ← x1, x2, x3, x4, x5
0.00 0.80 0.90 0.96                      ← y1, y2, y3, y4
1 2                                      ← z1, z2
0.00 0.00 0.00 0.00 0.00                 ← vy1
0.00 0.01 0.02 0.04 0.08                 ← vy2
0.00 0.02 0.04 0.08 0.10                 ← vy3
0.00 0.04 0.08 0.10 0.16                 ← vy4   } vz1
0.00 0.00 0.00 0.00 0.00                 ← vy1
0.00 0.02 0.03 0.05 0.09                 ← vy2
0.00 0.03 0.05 0.09 0.11                 ← vy3
0.00 0.05 0.09 0.11 0.17                 ← vy4   } vz2
end
```

vx1 vx2 vx3 vx4 vx5

Table-Based Lookup Example

In this example, Hercules extracts the effective width and length of devices D1 and D2. Hercules data creation commands extract polygons representing ox and oy edges (diffusion rounding effects), and px and py edges (poly rounding effects). These layers are collected by device extraction and processed by layer-based functions to give numeric input values to the get_dfm_lookup_value function. Hercules returns a user-accessible numeric value from the table.



Hercules uses the 3-D table shown below to cross-reference input values x, y, and z for a return value V. The table contains five x input values, four y input values, and two z input values. The x- and y-dimension values are the extracted length measurements ox, oy, px, and py. The z-dimension value represents a flag to return values from either poly rounding or diffusion rounding.

```
# 3-D table example
begin rounding_effects_3d
3 5 4 2
0.00 0.12 0.16 0.18 0.22
0.00 0.80 0.90 0.96
1 2
0.00 0.00 0.00 0.00 0.00
0.00 0.01 0.02 0.04 0.08
0.00 0.02 0.04 0.08 0.10
0.00 0.04 0.08 0.10 0.16
0.00 0.00 0.00 0.00 0.00
0.00 0.02 0.03 0.05 0.09
0.00 0.03 0.05 0.09 0.11
0.00 0.05 0.09 0.11 0.17
end
```

For device D1, ox and oy are 0.18 µm and 0.96 µm, respectively. The z flag is set to 1 for diffusion rounding. From the input table data shown above, 0.18 µm is x4, 0.96 is y4 and 1 is z1. Hercules returns the value intersecting v_{x4}, v_{y4}, and v_{z1}, which is 0.1 µm.

$$V(v_x \cap v_{y2} \cap v_{z2}) = 0.1$$

For device D2, px and py are 0.2 µm and 0.8 µm, respectively. The z flag is set to 2 for poly rounding. From the input table data, interpolation is required because 0.2 µm is between the input values x4 and x5. The value 0.8 µm, however, is y2. The z value is noted as z2. Hercules returns a value based on this interpolation equation:

$$V(v_x \cap v_{y2} \cap v_{z2}) = (v_x \cap v_{y2} \cap v_{z2}) + \frac{x - x_4}{x_5 - x_4}(v_{x5} - v_{x4})$$

$$V(v_x \cap v_{y2} \cap v_{z2}) = 0.05 + \frac{(0.20 - 0.18)}{(0.22 - 0.18)}(0.09 - 0.05)$$

$$V(v_x \cap v_{y2} \cap v_{z2}) = 0.07$$

In this example, a 3-D table is used because different return values exist for diffusion and poly rounding. If a higher degree of table data hierarchy is needed, increase the table dimension. For example, a 4-D table may be needed if the return value differs based on whether the chip block was analog or digital.

The NMOS extraction command example, below, demonstrates the runset usage of table-based extraction with a resulting layout netlist.

```
NMOS table_base_nmos gate src_drn src_drn {
   PROCESSING_LAYERS = { py oy poly_route px ox }  # Collect
                                                   # runset data
                                                   # creation
   PROPERTY = { effective_width effective_length  }
   RECOGNITION_LAYER =  { gate_size }
   POx = PERIM_COIN ( px, poly_route); # Use layer-based
                                       # functionality to
                                       # extract table-based
                                       # input values
   POy = PERIM_COIN ( py, src_drn);
   DIFFx = PERIM_COIN ( ox, src_drn);
   DIFFy = PERIM_COIN ( oy, poly_route);
   delta_length = 0.0;
   delta_width = 0.0;
   DFM[0] = DIFFx;   # table input value for diffusion rounding
   DFM[1] = DIFFy;
   DFM[2] = 1;
   delta_width = get_dfm_lookup_value(
                    rounding_effects_3d, DFM );
   effective_width = (( EV_W1 + EV_W2 ) / 2 ) + delta_width ;
   DFM[0] = POx;
   DFM[1] = POy;
   DFM[2] = 2;
   delta_length = get_dfm_lookup_value(
                     rounding_effects_3d, DFM );
   effective_length = (( EV_L1 + EV_L2 ) / 2) + delta_length ;
} TEMP = nbody
```

The effective_length in the netlist:

```
{CELL TABLE_BASED_EXTRACT
   {PORT}
   {PROP top=60.200000 bottom=57.000000 left=3.120000
         right=7.380000}
   {INST M1=table_base_nmos {PROP x=4.430 y=58.840 l=0.380
         w=1.560 effective_width=1.660 effective_length=0.380}
      {PIN N_3=GATE N_4=SRC N_6=DRN }}
   {INST M2=table_base_nmos {PROP x=6.030 y=58.840 l=0.300
         w=1.560 effective_width=1.560 effective_length=0.370}
      {PIN N_2=GATE N_4=SRC N_5=DRN }}
```

Device Equation Grammar

The information in this section describes the grammar of the Hercules equation language. This grammar applies to all extraction commands and the SET_PARAM command.

Comments can be mixed with the equations using the /* and */ characters. Nested comments are allowed.

For additional information on ternary operators, see the NET_FILTER example in the *Hercules Reference Manual*, Detailed Commands chapter.

An equation can be of three forms:

```
assign_variable = expression;
```

or

```
IF ( bool_expression ) { equation } ELSE { equation }
   set_param_variable ;
```

or

```
FOR ( start_val ; bool_expression ; step_val )
   {equation } BREAK ;
```

`set_param_variable` (the SET_PARAM variable) is the name of a predefined parameter set:

```
string
```

`assign_variable` can be any of these forms:

```
variable
variable { number }
variable { variable }
```

`start_val` is of the form:

```
variable = expression
```

`step_val` can be of the form:

```
variable = expression
variable increment_op
variable decrement_op
```

`bool_expression` can be ant of the forms:

```
bool_expression && bool_expression
```

bool_expression || *bool_expression*
(*bool_expression*)
expression
expression relational_op expression
! *expression*

`expression` can be of the form:

expression binary_op expression
unary_op expression
function (*expression*)
MAX (*expression*, *expression*)
MIN (*expression*, *expression*)
POW (*expression* , *expression*)
expression ? *expression* : *expression*
(ternary op)
(*expression*)
variable
number
close (*expression*, *expression*, *expression*)

`relational_op` can be of the form:

==
!=
>
<
>=
<=
? :
(ternary operation, a ? b : c which sets a = b if a 0.25 0, or a = c if a = 0)

`binary_op` can be:

+
-
*
/
%

`unary_op` can be:

-
+

`increment_op` is:

++

`decrement_op` is:

```
--
```

`function` can be:

```
SIN
COS
TAN
ASIN
ACOS
ATAN
EXP
SQRT
ABS
SINH
COSH
TANH
ASINH
ACOSH
ATANH
LOG
LOG10
ROUND
SPRINTF
```

`number` is of the form:

```
double
```

Generating Generic Devices

The GENDEV command allows the user to create a generic device and extract it during LVS Extraction. For more information on GENDEV Command Syntax, refer to the *Hercules Reference Manual*, Detailed Commands chapter.

Hercules provides an extensive programming capability through Scheme. To extract the generic devices, the Scheme interface is used by the GENDEV command. For details on how to write Scheme routines for AUTO and MANUAL mode of GENDEV Command, refer to *Hercules General Usage Information*, Scheme Interface chapter.

Standard MOSFET Extraction

This example GENDEV extraction finds a standard MOS device with four terminals. The properties extracted are gate length and gate width.

An example of the GENDEV command for standard MOSFET extraction is:

```
GENDEV gfet ngate nsd nsd pwell{ \
   initialization_func = mos-init \
   extract_func = mos-extract \
} TEMP = ngates
```

Example 3-5 MOSFET Extraction

```
(define mos-init (lambda ()
    (ev-dev-property-type-set "length" 'DOUBLE)
    (ev-dev-property-type-set "width" 'DOUBLE)
))

(define mos-extract (lambda (gate source drain bulk)
   (let (
        (gate-p (ev-dev-polygons-get gate))
        (dev-id #f)
        (sd-list #f)
        (bulk-list #f)
        (sedge #f)
        (dedge #f)
        (slen #f)
        (dlen #f)
        (touchvects #f)
   )

   ;; Loop through all the gate polygons
   (do () ((null? gate-p) #t)

   ;; Get all the polygons on the source layer that edgetouch this gate.
   (set! sd-list (ev-dev-polygon-select source 'EDGETOUCH (car gate-p)))

   ;; Get all the polygons on the bulk layer that interact with this
gate.
   (set! bulk-list (ev-dev-polygon-select bulk 'INTERACT (car gate-p)))

   ;; Make sure there are exactly 2 sd polygons, 1 bulk polygon, and
   ;; that the bulk polygon encloses the gate polygon
   (if (and (and (= (length sd-list) 2) (= (length bulk-list) 1))
            (length (ev-dev-polygon-select (car gate-p)
                 'ENCLOSED-BY (car bulk-list))))
         (begin
            (set! touchvects (ev-dev-polygon-boolean source 'and
                 (car gate-p)))
            (set! sedge (ev-dev-edges-get (car touchvects)))
            (set! slen (edge-length (car sedge)))
```

```
            (set! dedge (ev-dev-edges-get (cadr touchvects)))
            (set! dlen (edge-length (car dedge)))

            ;; define the device with 4 polygons
            (set! dev-id (ev-dev-device-create
                     (list (car gate-p) (car sd-list)
                           (cadr sd-list) (car bulk-list))))

          ;; Calculate the length
        (ev-dev-property-value-set dev-id "width" (/ (+ slen dlen) 2))

          ;; Calculate the width
          (ev-dev-property-value-set dev-id "length"
                   (ev-dev-edge-spacing-get (car sedge) dedge))

          ;; store the device(identified by the gate polygon)
          (ev-dev-device-write dev-id)
            (set! outlist (append outlist (list (car gate-p))))
        )
      )

      ;; go to the next polygon
      (set! gate-p (cdr gate-p))
    )

    ; write to output layer
    (ev-dev-output-set outlist)
    ; re-initialize outlist
    (set! outlist #f)
)))
```

Vertical BJT Extraction

The function bipolar extracts a bjt when it finds an emitter polygon enclosed by a base polygon, which is in turn enclosed by a collector polygon. Here the device gets a different name depending on the relative position of the base and emitter contacts. Note that the contact layer is passed in as a processing layer.

An example of the GENDEV command for BJT extraction is:

```
GENDEV bip pdiff ndiff elayer {
   processing_layers = {cont}
   initialization_func = init
   extract_func = bipolar
} TEMP = bipolar
```

Example 3-6 BJT Extraction

```
(define init (lambda ()
```

```
    (ev-dev-property-type-set "carea" 'DOUBLE)
    (ev-dev-property-type-set "barea" 'DOUBLE)
    (ev-dev-property-type-set "earea" 'DOUBLE)
    (ev-dev-property-type-set "cperim" 'DOUBLE)
    (ev-dev-property-type-set "bperim" 'DOUBLE)
    (ev-dev-property-type-set "eperim" 'DOUBLE)
    (ev-dev-property-type-set "name" 'STRING)
))

(define bipolar (lambda (base collector emitter)
   (let (
        (collp (ev-dev-polygons-get collector))
        (dev-id #f)
        (basep #f)
        (emitp #f)
        (bcont #f)
        (ccont #f)
        (econt #f)
        (bcenter #f)
        (ccenter #f)
        (ecenter #f)
        (collonly #f)
        (baseonly #f)
        (clayer #f)
   )
(set! clayer (ev-dev-processing-layers-get))

    ; cycle through all the collector polygons
    (do () ((null? collp) #t)

     (set! basep (ev-dev-polygon-select base 'enclosed-by (car collp)))
     (set! emitp (ev-dev-polygon-select emitter 'enclosed-by
        (car basep)))
     (set! collonly (car (ev-dev-polygon-boolean (car collp) 'not
        (car basep))))
     (set! baseonly (car (ev-dev-polygon-boolean (car basep) 'not
        (car emitp))))
     (set! ccont (ev-dev-polygon-select (car clayer) 'enclosed-by
        collonly))
     (set! bcont (ev-dev-polygon-select (car clayer) 'enclosed-by
        baseonly))
     (set! econt (ev-dev-polygon-select (car clayer) 'enclosed-by
        (car emitp)))
     (set! bcenter (polygon-centerpt (car bcont)))
     (set! ccenter (polygon-centerpt (car ccont)))
     (set! ecenter (polygon-centerpt (car econt)))

     ;should find one base and one emitter per device
     (if (and (and (= (length basep) 1) (= 1 (length emitp)))
         (< (car bcenter) (car ecenter)))
       (begin
       ;; define one device per collector polygon
       (set! dev-id (ev-dev-device-create (list (car basep) (car collp)
```

```
      (car emitp))))

    ;; type of device
    (ev-dev-property-value-set dev-id "name" "cbe")

    ;; areas
    (ev-dev-property-value-set dev-id "barea" (ev-dev-area-get
       (car basep)))
    (ev-dev-property-value-set dev-id "carea" (ev-dev-area-get
       (car collp)))
    (ev-dev-property-value-set dev-id "earea" (ev-dev-area-get
       (car emitp)))

    ;; perimeters
    (ev-dev-property-value-set dev-id "bperim"
       (polygon-perimeter-get (car basep)))
    (ev-dev-property-value-set dev-id "cperim"
       (polygon-perimeter-get (car collp)))
    (ev-dev-property-value-set dev-id "eperim"
       (polygon-perimeter-get (car emitp)))

    ;; store the device
    (ev-dev-device-write dev-id)
  ))
 (if (and (and (= (length basep) 1) (= 1 (length emitp))) (>
     (car bcenter) (car ecenter)))
   (begin
   ;; define one device per collector polygon
    (set! dev-id (ev-dev-device-create (list (car basep) (car collp)
       (car emitp))))

    ;; type of device
    (ev-dev-property-value-set dev-id "name" "ceb")

    ;; areas
    (ev-dev-property-value-set dev-id "barea" (ev-dev-area-get
       (car basep)))
    (ev-dev-property-value-set dev-id "carea" (ev-dev-area-get
       (car collp)))
    (ev-dev-property-value-set dev-id "earea" (ev-dev-area-get
       (car emitp)))

    ;; perimeters
    (ev-dev-property-value-set dev-id "bperim"
       (polygon-perimeter-get (car basep)))
    (ev-dev-property-value-set dev-id "cperim"
       (polygon-perimeter-get (car collp)))
    (ev-dev-property-value-set dev-id "eperim"
       (polygon-perimeter-get (car emitp)))

    ;; store the device
    (ev-dev-device-write dev-id)
  ))
```

```
        ;; got to the next polygon
        (set! collp (cdr collp))
   )

        (ev-dev-output-set a)
)))
```

T-Shaped Gate MOSFET Extraction

This extraction finds three devices for every T-shaped gate polygon that edge-touches the source/drain layer in the three regions surrounding the gate. The MOSFETs are three terminal devices.

An example of the GENDEV command for T-shaped gate MOSFET extraction is:

```
GENDEV gfet poly nsd nsd {
   initialization_func = init
   extract_func = tgate
} TEMP = ngates
```

Example 3-7 T-Shaped Gate MOSFET Extraction

```
(define init (lambda ()
   (ev-dev-property-type-set "w" 'DOUBLE)
   (ev-dev-property-type-set "l" 'DOUBLE)
))

(define tgate (lambda (gate source drain)
   (let (
      (gatep (ev-dev-polygons-get gate))
      (dev-id1 #f)
      (dev-id2 #f)
      (dev-id3 #f)
      (sdlst #f)
      (sedge1 #f)
      (sedge2 #f)
      (sedge3 #f)
      (slen #f)
      (dlen #f)
      (touchvects #f)
   )

   (do () ((null? gatep) #t)

      (set! sdlst (ev-dev-polygon-select source 'edgetouch (car gatep)))

      (if (= (length sdlst) 3)
      (begin
      (set! touchvects (ev-dev-polygon-boolean source 'and (car gatep)))
      (set! sedge1 (ev-dev-edges-get (car touchvects)))
```

```
(set! sedge2 (ev-dev-edges-get (cadr touchvects)))
(set! sedge3 (ev-dev-edges-get (cadr (cdr touchvects))))

(if (= (length sedge1) 2)
   (begin
      (if (= (length sedge2) 2)
       (begin
           (set! dev-id1 (ev-dev-device-create (list (car gatep)
                 (car sdlst) (cadr sdlst))))
           (set! dev-id2 (ev-dev-device-create (list (car gatep)
                 (car sdlst) (cadr (cdr sdlst)))))
           (set! dev-id3 (ev-dev-device-create (list (car gatep)
                 (cadr sdlst) (cadr (cdr sdlst)))))
       (if (point-oppose-edge (edge-midpt (car sedge1))
                 (car sedge2))
             (begin
               (ev-dev-property-value-set dev-id1 "l"
     (ev-dev-edge-spacing-get (car sedge1) (list (car sedge2))))
               (ev-dev-property-value-set dev-id1 "w"
     (edge-length (car sedge1)))
               (ev-dev-property-value-set dev-id2 "l"
     (ev-dev-edge-spacing-get (cadr sedge1) (list (car sedge3))))
               (ev-dev-property-value-set dev-id2 "w"
     (edge-length (cadr sedge1)))
               (ev-dev-property-value-set dev-id3 "l"
     (ev-dev-edge-spacing-get (cadr sedge2) (list (car sedge3))))
               (ev-dev-property-value-set dev-id3 "w"
     (edge-length (cadr sedge2)))
              )
            )
       (if (point-oppose-edge (edge-midpt (car sedge1))
           (cadr sedge2))
       (begin
     (ev-dev-property-value-set dev-id1 "l"
     (ev-dev-edge-spacing-get (car sedge1) (list (cadr sedge2))))
     (ev-dev-property-value-set dev-id1 "w" (edge-length
         (car sedge1)))
     (ev-dev-property-value-set dev-id2 "l"
     (ev-dev-edge-spacing-get (cadr sedge1) (list (car sedge3))))
     (ev-dev-property-value-set dev-id2 "w" (edge-length
         (cadr sedge1)))
     (ev-dev-property-value-set dev-id3 "l"
     (ev-dev-edge-spacing-get (car sedge2) (list (car sedge3))))
     (ev-dev-property-value-set dev-id3 "w" (edge-length
         (car sedge2)))
       )
      )
       (if (point-oppose-edge (edge-midpt (cadr sedge1))
           (car sedge2))
       (begin
     (ev-dev-property-value-set dev-id1 "l"
     (ev-dev-edge-spacing-get (cadr sedge1) (list (car sedge2))))
     (ev-dev-property-value-set dev-id1 "w" (edge-length
```

```
      (cadr sedge1)))
   (ev-dev-property-value-set dev-id2 "l"
   (ev-dev-edge-spacing-get (car sedge1) (list (car sedge3))))
   (ev-dev-property-value-set dev-id2 "w" (edge-length
      (car sedge1)))
   (ev-dev-property-value-set dev-id3 "l"
   (ev-dev-edge-spacing-get (cadr sedge2) (list (car sedge3))))
   (ev-dev-property-value-set dev-id3 "w" (edge-length
      (cadr sedge2)))
     )
   )
     (if (point-oppose-edge (edge-midpt (cadr sedge1))
       (cadr sedge2))
     (begin
   (ev-dev-property-value-set dev-id1 "w" (edge-length
      (cadr sedge1)))
   (ev-dev-property-value-set dev-id1 "l"
   (ev-dev-edge-spacing-get (cadr sedge1) (list (cadr sedge2))))
   (ev-dev-property-value-set dev-id2 "l"
   (ev-dev-edge-spacing-get (car sedge1) (list (car sedge3))))
   (ev-dev-property-value-set dev-id2 "w"
      (edge-length (car sedge1)))
   (ev-dev-property-value-set dev-id3 "l"
   (ev-dev-edge-spacing-get (car sedge2) (list (car sedge3))))
   (ev-dev-property-value-set dev-id3 "w" (edge-length
      (car sedge2)))
      )
     )
   )
;; else sedge1 has 2 edges and sedge2 has 1 edge
     (begin
   (set! dev-id1 (ev-dev-device-create (list (car gatep)
      (car sdlst) (cadr (cdr sdlst)))))
   (set! dev-id2 (ev-dev-device-create (list (car gatep)
      (car sdlst) (cadr sdlst))))
   (set! dev-id3 (ev-dev-device-create (list (car gatep)
      (cadr sdlst) (cadr (cdr sdlst)))))
      )
     )
   )
;; else sedge1 has only one edge
     (if (and (= (length sedge2) 2) (= (length sedge3) 2))
     (begin
   (set! dev-id1 (ev-dev-device-create (list (car gatep)
      (cadr sdlst) (cadr (cdr sdlst)))))
   (set! dev-id2 (ev-dev-device-create (list (car gatep)
      (car sdlst) (cadr sdlst))))
   (set! dev-id3 (ev-dev-device-create (list (car gatep)
      (car sdlst) (cadr (cdr sdlst)))))
     (if (point-oppose-edge (edge-midpt (car sedge2))
       (car sedge3))
     (begin
   (ev-dev-property-value-set dev-id1 "l"
```

```
              (ev-dev-edge-spacing-get (car sedge2) (list (car sedge3))))
              (ev-dev-property-value-set dev-id1 "w" (edge-length
                  (car sedge2)))
              (ev-dev-property-value-set dev-id2 "l"
              (ev-dev-edge-spacing-get (cadr sedge2) (list (car sedge1))))
              (ev-dev-property-value-set dev-id2 "w" (edge-length
                  (cadr sedge2)))
              (ev-dev-property-value-set dev-id3 "l"
              (ev-dev-edge-spacing-get (cadr sedge3) (list (car sedge1))))
              (ev-dev-property-value-set dev-id3 "w" (edge-length
                  (cadr sedge3)))
                )
              )
                (if (point-oppose-edge (edge-midpt (car sedge2))
                  (cadr sedge3))
                (begin
              (ev-dev-property-value-set dev-id1 "l"
              (ev-dev-edge-spacing-get (car sedge2) (list (cadr sedge3))))
              (ev-dev-property-value-set dev-id1 "w" (edge-length
                  (car sedge2)))
              (ev-dev-property-value-set dev-id2 "l"
              (ev-dev-edge-spacing-get (cadr sedge2) (list (car sedge1))))
              (ev-dev-property-value-set dev-id2 "w" (edge-length
                  (cadr sedge2)))
              (ev-dev-property-value-set dev-id3 "l"
              (ev-dev-edge-spacing-get (car sedge3) (list (car sedge1))))
              (ev-dev-property-value-set dev-id3 "w" (edge-length
                  (car sedge3)))
                )
              )
                (if (point-oppose-edge (edge-midpt (cadr sedge2))
                  (car sedge3))
                (begin
              (ev-dev-property-value-set dev-id1 "l"
              (ev-dev-edge-spacing-get (cadr sedge2) (list (car sedge3))))
              (ev-dev-property-value-set dev-id1 "w" (edge-length
                  (cadr sedge2)))
              (ev-dev-property-value-set dev-id2 "l"
              (ev-dev-edge-spacing-get (car sedge2) (list (car sedge1))))
              (ev-dev-property-value-set dev-id2 "w" (edge-length
                  (cadr sedge2)))
              (ev-dev-property-value-set dev-id3 "l"
              (ev-dev-edge-spacing-get (cadr sedge3) (list (car sedge1))))
              (ev-dev-property-value-set dev-id3 "w" (edge-length
                  (cadr sedge3)))
                )
              )
                (if (point-oppose-edge (edge-midpt (cadr sedge2))
    (cadr sedge3))
                (begin
              (ev-dev-property-value-set dev-id1 "l"
              (ev-dev-edge-spacing-get (cadr sedge2) (list (cadr sedge3))))
              (ev-dev-property-value-set dev-id1 "w" (edge-length
```

```
                (cadr sedge2)))
            (ev-dev-property-value-set dev-id2 "l"
            (ev-dev-edge-spacing-get (car sedge2) (list (car sedge1))))
            (ev-dev-property-value-set dev-id2 "w" (edge-length
                (car sedge2)))
            (ev-dev-property-value-set dev-id3 "l"
            (ev-dev-edge-spacing-get (car sedge3) (list (car sedge1))))
            (ev-dev-property-value-set dev-id3 "w" (edge-length
                (car sedge3)))
                )
              )
            ))
          )
          (ev-dev-device-write dev-id1)
          (ev-dev-device-write dev-id2)
          (ev-dev-device-write dev-id3)

          ;; store the device
;          (append a (car gatep))
       ))

          ;; got to the next polygon
          (set! gatep (cdr gatep))
     )

;          (ev-dev-output-set a)
)))
```

SAVE_PROPERTY Option

The SAVE_PROPERTY option provides information from a device extraction. SAVE_PROPERTY specifies exactly the set of properties to be stored, including any properties defined by the user in runset equations, and polygon properties attached to the device layer using the ATTACH_PROPERTY command. (See the *Hercules Reference Manual*, Detailed Commands chapter, for further description.)

The SAVE_PROPERTY option should be placed after the optional property variable definition in the device equation section for user-defined variables. They cannot be defined in the LPE_OPTIONS section or used inside an IF construct or a loop. SAVE_PROPERTY may be included in SET_PARAM definitions. There should be one SAVE_PROPERTY for each property to be stored in the LPE database for a particular extraction command. The current minimum set of properties required for netlisting is stored in addition to any SAVE_PROPERTY properties. An exception to this is the GENDEV command (see the *Hercules Reference Manual*, Detailed Commands chapter), which has no predefined properties. These properties are stored by name in the database and accessed by name during netlist generation.

Note:
The SAVE_PROPERTY option overrides the _PRINT_STATS option, which saves all properties. This override includes CAP_PRINT_STATS, DEV_PRINT_STATS, MOS_PRINT_STATS, DIODE_PRINT_STATS, BJT_PRINT_STATS, and RES_PRINT_STATS. This means that if the option is set to TRUE *and* the SAVE_PROPERTY option is specified within the extraction command, then only the minimum properties, plus any user-defined properties created in the SAVE_PROPERTY option, are printed to the *block*.lpe file.

The syntax is:

```
SAVE_PROPERTY("prop_name", prop_value);
```

Argument	Description
prop_name	A user-defined name for the property. This name is used to reference the property during netlisting.
prop_value	The value assigned to the *prop_name*. This value may be assigned in the device equations section of the extraction command. In the case of polygon properties, the GET_PROPERTY option is used to obtain the property value. See Example 3-8 for details.

Note:
Only variables of type DOUBLE can be used as the *prop_value* for variable or geometry type properties defined within the *dev_equations*. The GET_PROPERTY option is used to assign the string values of polygon properties.

Example 3-8 Using SET_PROPERTY

```
ATTACH_PROPERTY ngate interact recog1 {
   SET_PROPERTY ("_dev_type", "rc1")
} TEMP = ngate

NMOS n ngate nsd nsd {
   new_area = EV_GAREA + 2
   SAVE_PROPERTY ("area", new_area);
   SAVE_PROPERTY ("num45", EV_WNUM45);
   SAVE_PROPERTY ("dev_type", GET_PROPERTY("dev_type"));
} TEMP = nr_gate
```

4

Netlisting

Hercules generates three basic types of netlists through three commands. The NETLIST command generates an internal format netlist used for layout-versus-schematic (LVS) comparison. The SPICE command generates a SPICE file for circuit simulation. And the GRAPHICS command generates a graphics netlist used for layout probing.

Understanding the Hercules Netlist Format

Note:

Before using any of the netlisting commands, it is necessary that all connectivity for the design has been established with the CONNECT command. After defining connectivity, text should be applied to the input data with the TEXT command. Applying text before netlisting allows meaningful net names to be written into the layout netlist.

The Hercules netlist format is the required syntax for layout-versus-schematic comparison. This format is automatically generated for the layout database by the NETLIST runset command. If the schematic netlist is not in the Hercules format, then it must be translated prior to LVS comparison.

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.

The cell instance and pin information are 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 this Hercules format netlist example, cell A contains one instance and two pins. The line `inst 1=B` identifies 1 as the name of the instance and identifies 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.

Property Values

Properties may be attached to the primitive devices in a netlist. Property values may be set explicitly, or may be set with parameters. By setting a property equal to a parameter name (as opposed to an explicit numeric value), the value of the property can be set hierarchically by placing the parameter on an instance of the cell which contains the primitive device. More than one level of hierarchy can be traversed by setting a parameter equal to another parameter name.

The precedence (highest to lowest) for setting property values is as follows:

global parameter value	Defined by *param_global*
instance parameter value	Defined by *prop* / Attached to inst
default parameter value	Defined by *param_init* / Attached to cell
default property value	Defined by *prop_default*

Additional Syntax Features

Comments in the Hercules Netlist Format are identical to those in the C language. A comment begins with /* and ends with */. All intervening text is ignored. Nested comments are not allowed. Delimeters in the netlist may be any type of white space, including spaces, tabs, and new lines.

Keywords within the Hercules format are not case sensitive. The following are the Hercules specific keywords:

CELL
INST
SYNOPSYS
NET_GLOBAL
PARAM
PARAM_GLOBAL
PARAM_INIT
PIN
PORT
PROP_DEFAULT
VERSION
WIRE

Note:
Property names (l, w, rval, and so forth) are also not case sensitive.

For the keywords listed above, the values that are associated with each keyword (net names, instance names, parameter names, etc.) may not contain any of the following characters: " , { } =

Keywords for the Hercules Netlist Format are:

Keyword	Definition
CELL	Precedes a cell name at the start of each cell level netlist
INST	Precedes each instance in the netlist
NET_GLOBAL	Precedes a list of any global nets in the netlist
PARAM	Precedes assignment of hierarchical (parameterized) property values
PARAM_GLOBAL	Precedes assignment of global values for parameterized property values

Keyword	Definition
PARAM_INIT	Precedes assignment of default values for parameterized properties
PIN	Precedes pin name assignments
PORT	Precedes a list of port nets for a cell
PROP	Precedes a list of properties and values for an instance
PROP_DEFAULT	Precedes assignment of default values for properties
WIRE	Precedes a list of wires in the netlist

Netlist Example

There are a total of four transistors in the example design shown in Example 4-1: 2 NMOS and 2 PMOS devices. A flattened version of this netlist, with the resulting property values, is shown in Example 4-2.

Example 4-1 Netlist for Example Design

```
{synopsys sample.sch
      /* 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}
   }
}
}
```

Example 4-2 Flattened Equivalent of Example 4-1

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

   {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}
    }
}
}
```

Customizing Netlist Output

Customizing the netlists that Hercules creates is accomplished by a scheme programming language function that is supplied by the user. To customize the Hercules format netlist, the EV_NETLIST_FUNC option must be added to the device extraction commands. To modify SPICE output, the SPICE_NETLIST_FUNC option must be added to the device extraction commands. Each of these options specifies the name of a scheme function that performs the netlist modification. See *Hercules General Usage Information*, Generating Input for Hercules chapter, for more details about customizing netlists, and how the scheme routines should be written.

In Example 4-3, two device types are extracted. The device type N uses standard netlisting, and its netlist entry is not altered when the netlist is written. However, the Ncustom device type does make use of netlist customization. For this device type, both SPICE and Hercules format netlists will be modified. For the Hercules netlist, the scheme function herc-formatting-routine contains the code for modifying the netlist entry for the Ncustom device type. Likewise, the scheme function spice-formatting-routine performs modification of the SPICE netlist.

Example 4-3

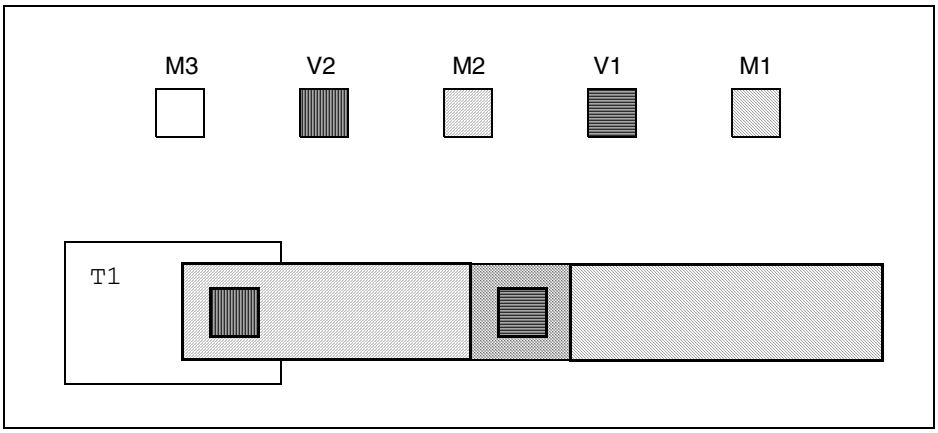
```
NMOS N ngate nsd nsd psub {
} temp = Nout
NMOS Ncustom cngate nsd nsd psub {
   EV_NETLIST_FUNC = herc-formatting-routine
   SPICE_NETLIST_FUNC = spice-formatting-routine
} temp = Ncustomout

NETLIST {}
```

Generating Graphical Netlists

Graphical netlists are generated by the GRAPHICS command. The graphical netlist contains polygons that have instance and net names attached to them. If connectivity has been established before the graphics command, then all polygons on each net will have text attached to them if they connect to another polygon with text. All layers that should be written to the graphical netlist must be specified in the GRAPHICS command.

The VUE_LAYERS option can be used to save layers automatically to the graphical netlist.

In Example 4-4, text is located on M3, but connections exist all the way from M3 to M1. Since all of the polygons are connected together, they are all part of the T1 net. When GRAPHICS saves these polygons, each polygon will have the net name T1 attached as a property.



Example 4-4

```
CONNECT {
   M3 M2 BY [TOUCH OVERLAP] V2
   M2 M1 BY [TOUCH OVERLAP] V1
}

TEXT { M3 by M3.TEXT }

GRAPHICS {
   M3 (1)
   V2 (2)
   M2 (3)
   V1 (4)
   M1 (5)
}
```

5

Layout Versus Schematic Comparison

The LVS COMPARE process verifies that the geometric or layout implementation of a circuit matches the schematic specification. The Hercules hierarchical verification process has distinct advantages for efficiency of processing and localization of errors. This chapter discusses concepts that are important for a detailed understanding of the LVS COMPARE process, and how these concepts are implemented in the Hercules software.

COMPARE Process

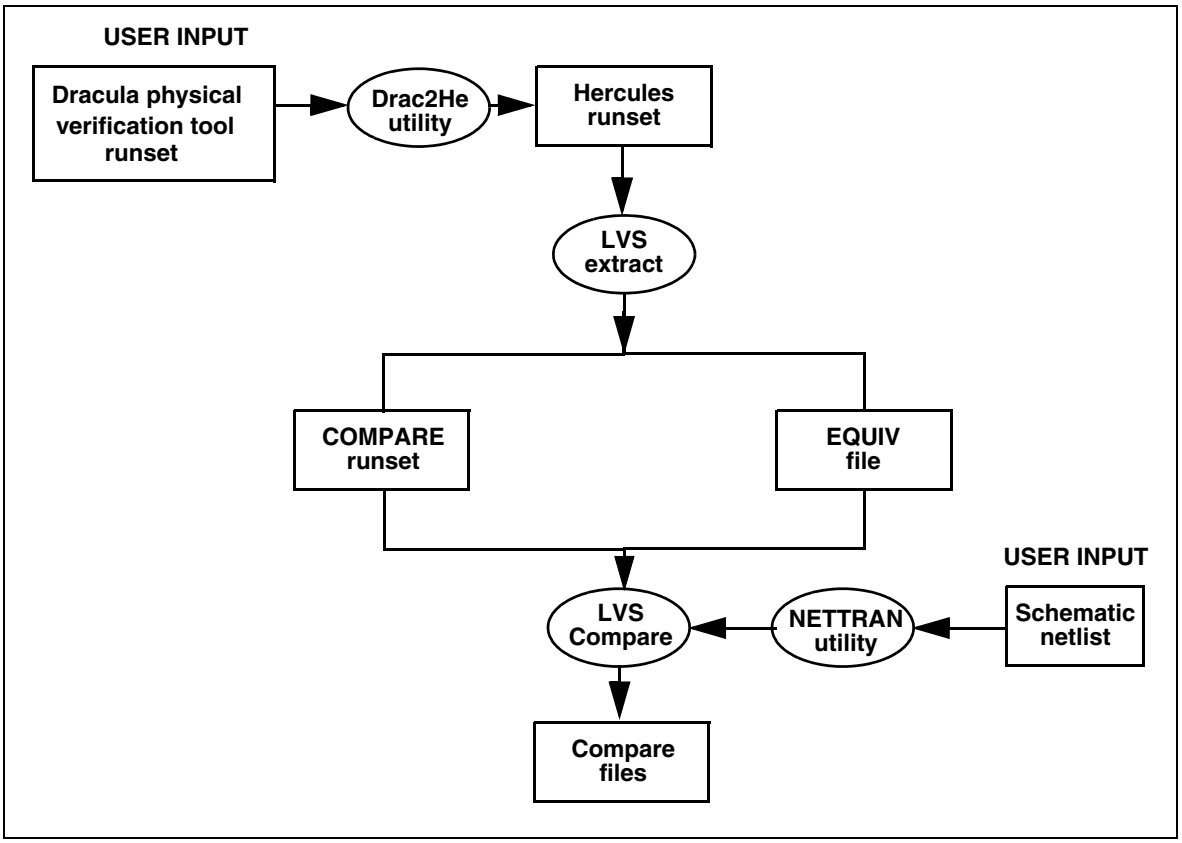
Hercules LVS consists of two phases involving hierarchy:

1. Extraction of a hierarchical layout netlist.
2. Comparison of the hierarchical layout netlist to a hierarchical schematic netlist. Throughout this chapter we will often refer to the comparison phase as COMPARE.

The COMPARE process verifies that the geometric or layout implementation of a circuit matches the schematic specification. Hercules performs this verification hierarchically, as opposed to flattening down to the device level, in order to maximize efficiency of processing and localization of errors.

Hercules LVS operation is illustrated in Figure 5-1 and described below.

Figure 5-1 Hercules LVS Flow Chart



Flow Description

If beginning with a Cadence® Dracula® physical verification runset, the Drac2He utility translates this runset into a Hercules DRC/Extraction runset. If not, this process begins with a Hercules runset. The Hercules Flow Manager converts the database to an internal format and processes the converted runset and database. The output is a netlist database. LVS extraction then outputs the layout netlist. If your schematic netlist is in CDL, NetTran will translate the netlist to Hercules format. Hercules automatically takes the input schematic netlist and the layout netlist and outputs the equivalence file and COMPARE runset. Hercules then performs LVS COMPARE and automatically outputs compared files. The whole process is recorded in the *block*.RESULTS file.

Hercules requires an LVS runset and a schematic netlist in its original format.

Equivalence Points

One of the primary requirements of a hierarchical LVS system is to specify points in the design hierarchy between the schematic and layout databases which are known as equivalence points. These equivalence pairs should be points where the underlying circuitry between the schematic and layout blocks are the same.

For a flat LVS system, there is only one equivalence point: the root block of the design. By specifying additional equivalence points, you partition the design into smaller sub-designs.

Instead of matching the contents of a block repeatedly for each of its references, equivalence points allow the circuitry to be matched only once. Subsequently, it is necessary to check only the connectivity to each of the block's references, because it is already known that contents of the block will always match.

Specify as many valid equivalence points as possible between the schematic and layout designs. Increased processing efficiency and finer error resolution are directly related to the number of equivalence points you specify. There are no restrictions on the number of required equivalence points.

Multiple Equivalences for a Single Block

It is possible to have more than one equivalence point for a schematic or layout block. These points are used when there is more than one implementation for the same circuitry. For example, it is common to have more than one layout implementation of a block to accommodate different space constraints.

The only restriction on multiple equivalencies is that they must be 1-to-N or N-to-1. In other words, either the schematic or layout block must remain the same for multiple equivalencies.

The following example equivalence file is valid:

```
EQUIV schematic = layout_1 { }
EQUIV schematic = layout_2 { }
EQUIV schematic = layout_3 { }
EQUIV schematic_1 = layout { }
EQUIV schematic_2 = layout { }
```

The next example is not valid because layout_2 is involved in both 1-to-N and N-to-1 relationships:

```
EQUIV schematic_1 = layout_1 { }
EQUIV schematic_1 = layout_2 { }
EQUIV schematic_2 = layout_2 { }
```

Note:
Blocks that have the same logical implementation but different device properties do not qualify as multiple equivalencies, unless property values are not being checked.

Using the Scheme Function to Filter Equivalence Points

It is possible to use Scheme functions in the equivalence file to filter the equivalence points.

The following example shows Scheme filtering on equivalence points. It checks to see whether the output pin Y on the cell is connected or floating. If the output pin is floating, the cells can be filtered.

This Scheme function is registered in the equivalence file like this:

```
equiv ND3D2=ND3D2 { filter_func = equiv-filter}
```

When doing cell-based filtering, it is often necessary to add `auto_exclude_equiv = FALSE` to the runset so that the instances of this equivalence do not get exploded before the Scheme filtering code is executed.

```
(define equiv-filter (lambda (inst-id)
   (let (
      (A-pin (lvs-pins-get inst-id "Y"))
      (filter #t)
   )
 ;; Check if output pin has 0 connections
   (if (not (equal? (lvs-connection-count-get (car A-pin)) 0))
      (set! filter #f)
)
   filter
)))
```

Generating the Best Equivalence File

A specified matching relation between a schematic netlist cell and a layout netlist cell is called an equivalence point. The equivalence file stores equivalence points, non-equivalence points, and options that describe how to match the equivalence points.

This file can be generated automatically by using Hercules, or you can specify this file to optimize the compare result.

If you do not specify the equivalence file, Hercules finds the corresponding relation and generates the equivalence file. The file is saved in run_details/equiv.run. You can also use the FIND_ADDITIONAL_EQUIVS command in the COMPARE section to specify how Hercules should automatically detect the equivalence points from your design.

EQUATE Command

The EQUATE command is used to associate schematic primitive devices with their corresponding extracted layout devices. These are single device types with no underlying hierarchy. They can be viewed as equivalence points for which you specify the correct matching.

Multiple EQUATEs for a Single Device

It is possible to have more than one EQUATE for a schematic or layout device. Both are used when the same device type has more than one representation in either the schematic or the layout.

The only restriction on multiple EQUATEs is that they must be 1-to-N or N-to-1. In other words, either the schematic or layout device must remain the same for multiple EQUATEs.

The following example EQUATEs are valid:

```
EQUATE NMOS schematic = layout_1 Gate Source Drain { }
EQUATE NMOS schematic = layout_2 Gate Source Drain { }
EQUATE NMOS schematic = layout_3 Gate Source Drain { }
EQUATE NMOS schematic_1 = layout Gate_1 Source_1 Drain_1 { }
EQUATE NMOS schematic_2 = layout Gate_2 Source_2 Drain_2 { }
```

The following example is not valid because layout 2 is involved in both 1-to-N and N-to-1 relationships:

```
EQUATE NMOS schematic_1 = layout_1 Gate_1 Source_1 Drain_1 { }
EQUATE NMOS schematic_1 = layout_2 Gate_1 Source_1 Drain_1 { }
EQUATE NMOS schematic_2 = layout_2 Gate_2 Source_2 Drain_2 { }
```

BLACK_BOX Definitions

The equivalence file can also contain definitions of BLACK_BOX structures. BLACK_BOX structures are portions of the design that are treated as if they COMPARE, even though no comparison verification has been done on them. Any device and connection data contained within the BLACK_BOX structure is ignored; only the port connections of the black box structures will be checked. A correspondence must be established between the BLACK_BOX ports in the schematic and in the layout using the EQUATE_PORTS construct. (For more information on LVS black box flow, see “LVS Black Box Flow” on page 5-50.)

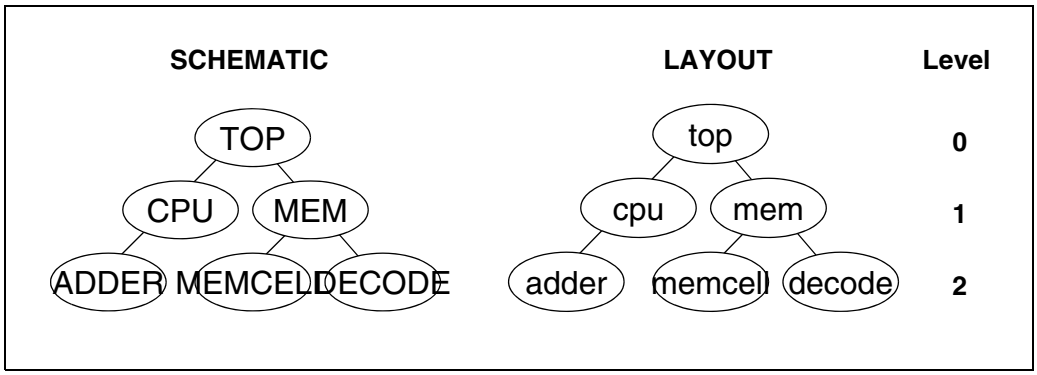
For example, the following black box structure is valid:

```
BLACK_BOX schematic = layout {
   equate_ports { sch_port=lay_port ... sch_port=lay_port }
}
```

COMPARE Operation/Flow

The COMPARE operation proceeds hierarchically, starting with the equivalence points which are the lowest in the design hierarchy. The top-level equivalence point of the LVS verification is level 0, and increases for dependent equivalence points. Equivalence points with the highest level number are compared first, followed by the next lowest level number, and so forth. Figure 5-2 demonstrates an example equivalence tree. The COMPARE process continues verifying decreasing level numbers until a stop condition occurs. You can control the stop condition, but generally it occurs at the completion of a level where errors were detected.

Figure 5-2 Equivalence Tree



Notice that equivalence points are always assigned to the highest possible level, while still maintaining the dependency order. This results in the most efficient processing during verification.

The EXPLODE_ON_ERROR, STOP_ON_ERROR, and TOP_BLOCK COMPARE options can be used to control the exact manner in which equivalence points are processed during LVS verification.

Merging

For some IC technologies, it is possible to create logically equivalent circuits that have different physical implementations. This is especially true for CMOS circuits. Various merging procedures have been incorporated in the COMPARE process to allow matching of logically equivalent structures. These procedures replace a group of devices with a single merged device. Thus, physically dissimilar structures are replaced by a common merged device in both the schematic and layout blocks.

Merging during the COMPARE process is a repetitive procedure. Merging stops only after there was a pass through all enabled merging and filtering options without any new additions or deletions. This allows complex physical structures to be gradually reduced to more generic merged devices.

The COMPARE process uses only the replacement merged devices in the matching process, but maintains a cross-reference to the original physical devices. The COMPARE summary files for the individual equivalence points list a merged device and its physical components for any merged device referenced in the report.

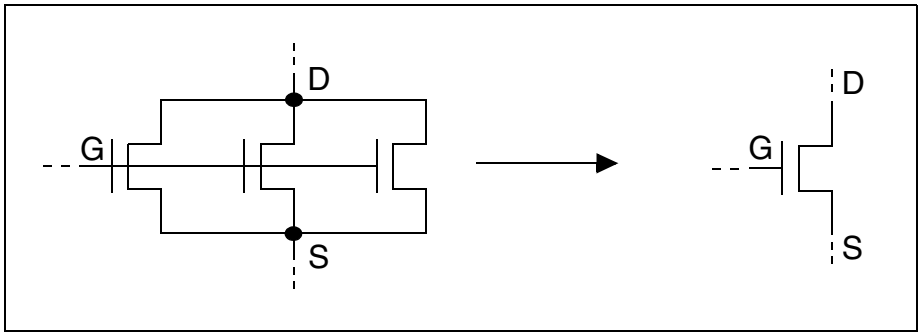
Parallel Merging

Parallel merging is turned on by setting MERGE_PARALLEL=TRUE in the COMPARE command. Devices are in parallel if every corresponding pin pair between the devices is connected to the same net. Two or more parallel devices are combined into a single merged device with the same device type and net connections.

- Bulk or substrate pin connections must also correspond in order for devices to be classified as parallel devices.
- Parallel merging only applies to devices specified in the EQUATE commands. Higher level blocks which satisfy the criteria for parallel devices will not be parallel merged.
- Newly-created parallel devices are assigned instance names of Parallel1, Parallel2, and so forth.

An example of parallel merging is shown in Figure 5-3.

Figure 5-3 Parallel Merging



Series Merging

To enable series merging, the COMPARE option MERGE_SERIES must be set to TRUE. Capacitors and Resistors in series can be merged into a single device which has the same net connections as the terminal pins on the chain.

NMOS and PMOS devices are in a series chain when their drain and source pins are connected consecutively. All devices in the chain must be of the same device type. In addition, the nets which connect the drain/source pins of neighboring devices cannot have any other connections or be a port of the block. A single merged device is created with drain/source pin connections to the terminal drain/source pins of the chain, and multiple Gate pin connections corresponding to the number of devices in the chain. The Gate pins on the merged device are automatically designated as interchangeable. As a result, net connections to the Gate pins are allowed to be in any order.

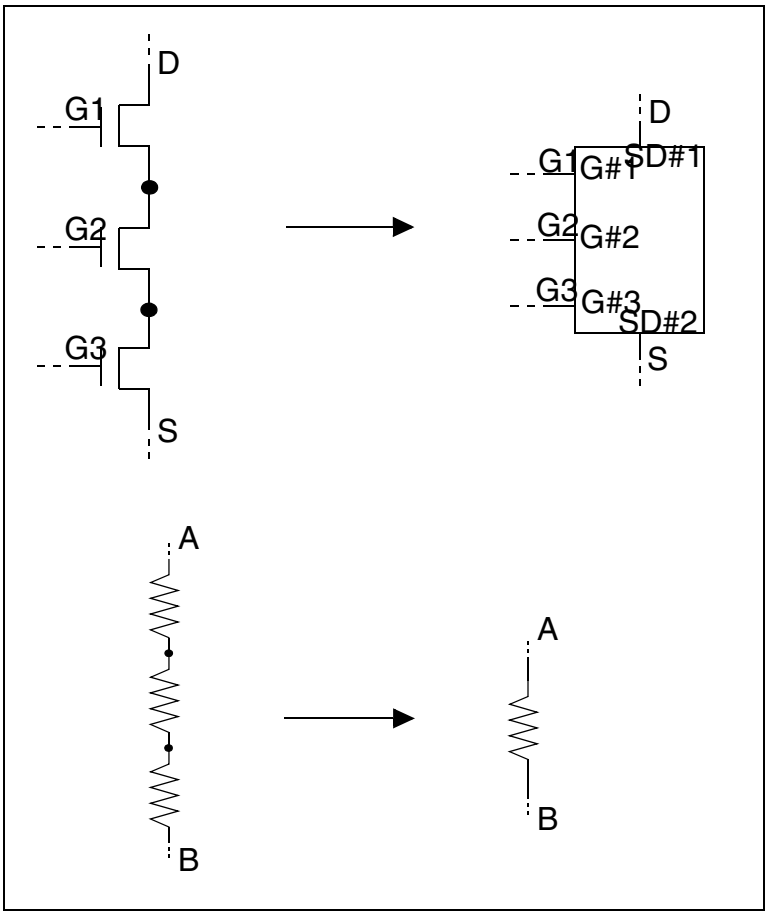
Bulk or substrate pin connections must all be tied to the same point in order for devices to be classified as series devices.

For Capacitors and Resistors, the merged device type is the same as that of the physical device members. However, a new device type must be created for NMOS and PMOS devices, since the merged device has multiple Gate pins. The name for the new device type is the name of the member device type, followed by -- and ending with the number of Gate pins in the chain. For example, a series chain of four devices with type name nmos would result in a merged device type of nmos--4.

Newly created series chain devices are assigned instance names of SerChain1, SerChain2, and so forth.

An example of series merging is shown in Figure 5-4.

Figure 5-4 Series Merging



Path Merging

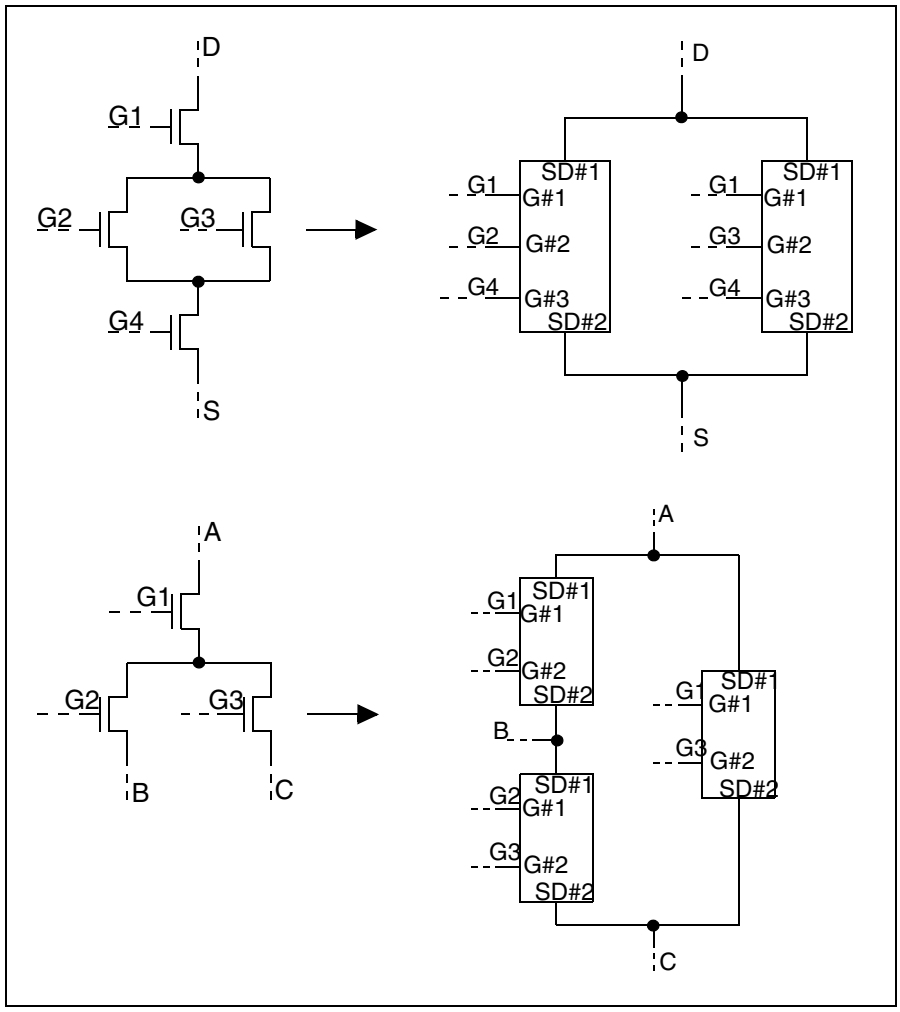
Paths of NMOS or PMOS devices are extensions of series chains. Path merging can be used for more complex structures where groups of devices are stacked. This type of structure is commonly found in AND-OR-INVERT logic. The series components of the stacked structure provide the AND function, while the parallel components result in an OR operation.

While series chains have only two terminating drain/source pins, path structures can have two or more terminating drain or source pins. The stacked structure is expanded into non-duplicate series chains between the terminating drain or source pins. This results in more than one merged device created for each stacked structure. Note that this differs from all other merging operations that replace multiple physical devices with a single merged device. In addition, a single physical device can be a member of many merged devices created from a path structure.

Bulk or Substrate pin connections must all be tied to the same point in order for devices to be classified as series devices. Since path merging is an extended version of series chain merging, the naming conventions are as described previously. In order to enable path merging in Hercules, you must set the COMPARE command MERGE_PATHS to TRUE.

An example of path merging is shown in Figure 5-5.

Figure 5-5 Path Merging



Equate

By default, all device types can be parallel merged. Capacitor and Resistor device types have the added default capability of series merging. Finally, path merging and series merging are all enabled by default for NMOS and PMOS device types.

The EQUATE runset command can be used to disable merging capabilities for individual device types. For example, path merging is disabled in the following:

```
EQUATE NMOS N=N gate source drain bulk { merge_paths = off }
```

The EQUATE command controls only the capability for merging of a particular device type. The actual process of merging is controlled by options in the EQUIV command. Consequently, merging of a device type only occurs if merging is enabled for the device type and within the equivalence options.

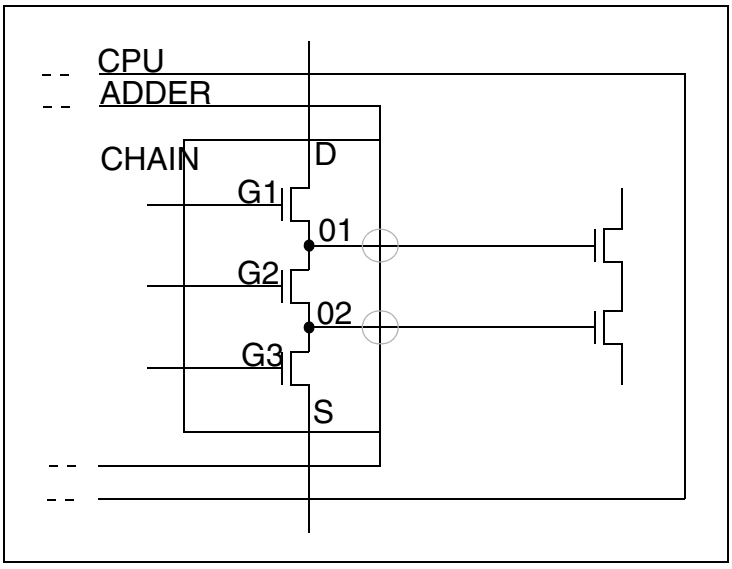
Ports

When groups of devices are merged during Series and Path merging, all intermediate nets not connected to the pins of the merged device are effectively removed from the circuit. However, if a net is also a port of the block, it cannot be removed. Thus, a net which is a port of a block can have merged devices attached to it, but it cannot be incorporated into a merged device.

Ports in the schematic block are determined by the way in which the schematics were captured. Ports for layout blocks are created based on interactions between physical geometric data. If there is a hierarchical interaction between data specified as attached by a CONNECT command, then that net becomes a port of the layout block. In some cases, this interaction does not exist.

For example, consider the block shown in Figure 5-6:

Figure 5-6 Layout Ports



In Figure 5-6, the block CHAIN and ADDER share CPU as a common parent and have a common bounding box border on the right hand side. The hierarchical interaction which creates ports O1 and O2 on the block CHAIN occurs by abutment of data at the level of block CPU. If the LVS process is restricted to run on the ADDER block, the hierarchical interaction does not exist. Consequently, the nets O1 and O2 in CHAIN would not be extracted as ports.

For blocks that are missing ports, the merging process arrives at different results. This causes the matching process to fail, and the COMPARE process will report that the schematic and layout blocks are not equivalent. Once the port problems are corrected, the merging process will be consistent between the schematic and layout blocks. For port nets which are named in the layout, the LAYOUT_STATIC_NET equivalence process definition can be used to prevent a net from being incorporated into a merged device.

The top level of the physical design never has ports designated in the netlist. Since there is no data above the top level, there is no way to determine ports from hierarchical interactions. To avoid problems with inconsistent merging, the COMPARE process ignores the port declarations for the highest level comparison point. This is the only case where it is acceptable to incorporate a port net into a merged device.

Properties

Special consideration is made for merged devices with respect to property comparison.

The Length or Width property values for MOS devices are combined as single values for parallel merging. However, for series and path merging, the Length and Width property values are maintained as a list for the merged device.

The Resistance and Capacitance property values are always combined into a single value for the merged device.

See “Properties” on page 5-21 for a more detailed explanation of merged property values.

LAYOUT_GROUND / LAYOUT_POWER /SCHEMATIC_GROUND / SCHEMATIC_POWER

Similar to block ports, nets specified as Ground or Power nets are not incorporated into merged devices.

Recall that port declarations are ignored for the top level of a design. However, Power and Ground nets are still considered to be off-limits for merging. This implies that if the Power and Ground nets are labelled at the top level in the schematic, they should also be labelled in the layout.

SCHEMATIC_GLOBAL

Nets listed as global in the schematic are assumed to be connected throughout the design. Consequently, they are implied ports of all blocks which use these nets. Since port nets must be excluded from merged devices, then all schematic globals are also excluded.

MERGE_NET_RANGE

As previously discussed, certain intermediate nets can be removed during device merging. For path merging, these intermediate nets are effectively split amongst a number of merged devices. In certain cases, it is possible to create a large number of merged devices during path merging. For example, global Ground and Power nets not correctly identified in the schematic can result in many merged devices, since these nets are often shared by many physical devices.

To safeguard against these unusual cases, the MERGE_NET_RANGE COMPARE Option can be used to control the amount of path merging by specifying an integer value to start warning and stop executing based on the number of merged nets.

LAYOUT_STATIC_DEVICE / LAYOUT_STATIC_NET / SCHEMATIC_STATIC_DEVICE / SCHEMATIC_STATIC_NET

For a specific equivalence point, individual devices and nets can be tagged as static. Static devices and nets are never incorporated into a merged device, and are never replaced. As in the case of block ports, nets which are static can have connections to merged devices, but they are never intermediate nets of a merged device.

These equivalence file process definitions expect fully hierarchical device and net identifiers. If design hierarchy is exploded, then an instance name separated by slashes (/) is required.

Filtering

For certain designs, blocks contain devices that do not have a functional effect on the circuit; they are unused. It may be necessary to remove these unused devices to ensure consistency between the schematic and layout blocks. The filtering operation removes unused or disabled devices from the block prior to device and net matching. Filter options allow you to control under what circumstances a device is to be filtered.

Equate

The filtering operation is controlled on a device-by-device basis by the FILTER_OPTIONS, FILTER_SCHEMATIC_OPTIONS, and FILTER_LAYOUT_ OPTIONS Process Definitions of the EQUATE runset command. If no filtering options are specified for a device, then that device type is considered unfilterable, regardless of whether filtering is enabled within the COMPARE and/or EQUIV options.

Note:

All filtering options are OFF by default.

The FILTER_SCHEMATIC_OPTIONS and FILTER_LAYOUT_OPTIONS Process Definitions allow different filtering to take place between the schematic and the layout. Anything specified by the FILTER_OPTIONS Process Definition applies to both the schematic and the layout.

The available filtering options are detailed in the following sections, and can be specified in an EQUATE runset command as shown in this example:

```
EQUATE NMOS n = n gate source drain bulk {
   filter_options = { NMOS-1, NMOS-3, NMOS-4 }
}
```

All references to power and ground nets in the filtering options descriptions refer to nets specified in the SCHEMATIC_GROUND, SCHEMATIC_POWER, LAYOUT_GROUND, and LAYOUT_POWER Options in the OPTIONS Section of the runset.

NMOS Devices

The following filtering options are available for NMOS devices.

Table 5-1 Filtering Options Available for NMOS Devices

Option	Description
NMOS-1	Filters devices when gate, source, and drain pins are shorted.
NMOS-2	Filters devices when gate, source, and drain pins are floating.
NMOS-3	Filters devices when gate pin is tied to ground.
NMOS-4	Filters devices when source and drain pins are tied to ground.
NMOS-5	Filters devices when source and drain pins are shorted.
NMOS-6	Filters devices when source or drain pins are floating.
NMOS-7	Filters devices when gate pin and either source or drain pin are floating.
NMOS-8	Filters devices when gate, source or drain pin is floating.
NMOS-9	Filters devices when gate pin is tied to power and source and drain pins are tied to ground.
NMOS-10	Filters devices when source or drain is unconnected and gate net is only tied to gates.
NMOS-11	Filters devices when gate is tied to ground and source and drain are tied together.
NMOS-12	Filters devices when gate is tied to ground and either source or drain is floating.
NMOS-13	Filters devices when gate, source, drain, and the first bulk pin are shorted.

Table 5-1 Filtering Options Available for NMOS Devices(Continued)

Option	Description
NMOS-14	Filters devices when gate pin is floating.
NMOS-15	Filters devices when gate and either source or drain pin are floating and the other source or drain pin is tied to a power or ground net.
NMOS-16	Filters devices when gate and the first bulk pin are tied to the same net.
NMOS-17	Filters devices when the gate pin is floating and either the source or drain pin have a path to ground.
NMOS-19	Filters devices when the gate pin is floating and the source and drain pins are shorted.
NMOS-20	Filters devices when source and drain pins are floating.
NMOS-21	Filters devices when gate and the first bulk pin are tied to the same ground.

PMOS Devices

The following filtering options are available for PMOS devices.

Table 5-2 Filtering Options Available for PMOS Devices

Option	Description
PMOS-1	Filters devices when gate, source, and drain pins are shorted.
PMOS-2	Filters devices when gate, source, and drain pins are floating.
PMOS-3	Filters devices when gate pin is tied to power.
PMOS-4	Filters devices when source and drain pins are tied to power.
PMOS-5	Filters devices when source and drain pins are shorted.
PMOS-6	Filters devices when source or drain pins are floating.
PMOS-7	Filters devices when gate pin and either source or drain pin are floating.
PMOS-8	Filters devices when gate, source or drain pin is floating.
PMOS-9	Filters devices when gate pin is tied to ground and source and drain pins are tied to power.

Table 5-2 Filtering Options Available for PMOS Devices(Continued)

Option	Description
PMOS-10	Filters devices when source or drain is unconnected and gate net is only tied to gates.
PMOS-11	Filters devices when gate is tied to power and source and drain are tied together.
PMOS-12	Filters devices when gate is tied to power and either source or drain is floating.
PMOS-13	Filters devices when gate, source, drain, and the first bulk pin are shorted.
PMOS-14	Filters devices when gate pin is floating.
PMOS-15	Filters devices when gate and either source or drain pin are floating and the other source or drain pin is tied to a power or ground net.
PMOS-16	Filters devices when gate and the first bulk pin are tied to the same net.
PMOS-17	Filters devices when the gate pin is floating and either the source or drain pin have the path to power.
PMOS-19	Filters devices when the gate pin is floating and the source and drain pins are shorted.
PMOS-20	Filters devices when source and drain pins are floating.
PMOS-21	Filters devices when gate and the first bulk pin are tied to the same power.

RES Devices

The following filtering options are available for RES devices.

Table 5-3 Filtering Options Available for RES Devices

Option	Description
RES-1	Filters devices when both pins are shorted.
RES-2	Filters devices when both pins are floating.
RES-3	Filters devices when either pin is floating.

CAP Devices

The following filtering options are available for CAP devices.

Table 5-4 Filtering Options Available for CAP Devices

Option	Description
CAP-1	Filters devices when both pins are shorted
CAP-2	Filters devices when both pins are floating
CAP-3	Filters devices when either pin is floating

NPN Devices

The following filtering options are available for NPN devices.

Table 5-5 Filtering Options Available for NPN Devices

Option	Description
NPN-1	Filters devices when emitter, base, and collector pins are shorted
NPN-2	Filters devices when emitter, base, and collector pins are floating
NPN-4	Filters devices when base pin is tied to ground
NPN-5	Filters multi-emitter devices when base pin is floating and collector is tied to power
NPN-6	Filters devices when emitter and base pins are floating
NPN-7	Filters multi-emitter devices when all emitters are floating

PNP Devices

The following filtering options are available for PNP devices.

Table 5-6 Filtering Options Available for PNP Devices

Option	Description
PNP-1	Filters devices when emitter, base, and collector pins are shorted
PNP-2	Filters devices when emitter, base, and collector pins are floating

Table 5-6 Filtering Options Available for PNP Devices

Option	Description
PNP-3	Filters devices when base pin is tied to power
PNP-4	Filters multi-collector devices when all pins are floating
PNP-5	Filters multi-collector devices when all collectors are floating

NP Devices

The following filtering options are available for NP devices.

Table 5-7 Filtering Options Available for NP Devices

Option	Description
NP-1	Filters devices when both pins are shorted
NP-2	Filters devices when both pins are floating
NP-3	Filters devices when either pin is floating

PN Devices

The following filtering options are available for PN devices.

Table 5-8 Filtering Options Available for PN Devices

Option	Description
PN-1	Filters devices when both pins are shorted
PN-2	Filters devices when both pins are floating
PN-3	Filters devices when either pin is floating

Layout_Ground/Layout_Power/Schematic_Ground/Schematic_Power

Many of the filtering criteria depend on a knowledge of which nets are power or ground signals. currently, because filtering is done on a per equivalence basis, the nets must be named locally in the block for filtering to operate correctly.

Layout_Static_Device/Schematic_Static_Device

For a specific equivalence point, individual devices can be tagged as static. Static devices are never filtered from a block.

These equivalence file process definitions require fully hierarchical device identifiers. If design hierarchy is exploded, then an instance name separated by slashes (/) is required.

Pin Class

Each port (pin) of a device or block is assigned a class for use by the COMPARE process. This class is simply a unique identifier which is used to differentiate ports on a block. Ports which share the same class can have their net connections interchanged without changing the functional operation of the block.

The concept of pin class is very important to the COMPARE matching process. It directly effects how nets and devices are matched between the schematic and layout blocks.

There are a number of reserved pin class identifiers used by COMPARE for extracted devices. These reserved identifiers provide port swapping information, as well as the type of a pin during merging and filtering operations. The drain and source pin classes actually represent the same pin class; they are synonymous.

Table 5-9 Reserved Pin Class Identifiers Used by COMPARE

Class Name	Terminal	Device Type
anode	Positive	CAP, DIODE, RES
base	Base	NPN, PNP
bulk	Substrate	CAP, DIODE, NMOS, NPN, PMOS, PNP, RES
bulk2	Substrate	NMOS, PMOS
bulk3	Substrate	NMOS, PMOS
bulk4	Substrate	NMOS, PMOS
cathode	Negative	CAP, DIODE, RES
collector	Collector	NPN, PNP
drain	Drain	NMOS, PMOS
emitter	Emitter	NPN, PNP
gate	Gate	NMOS, PMOS

Table 5-9 Reserved Pin Class Identifiers Used by COMPARE(Continued)

Class Name	Terminal	Device Type
node	Unsigned	CAP, RES
source	Source	NMOS, PMOS

Equating Nets

Symmetry always allows more than one possible matching of devices and/or nets. By controlling how nets are matched, it is possible to force a particular matching from the set of allowable results.

The COMPARE process uses a net equate as an initial point of reference. These initial reference points are often enough to control the outcome of the matching process.

Attempts to equate nets with different numbers of pin connections after filtering and merging operations are completed will be ignored. These nets cannot possibly match, so the COMPARE process automatically assumes they are in error.

EQUATE_BY_NET_NAME: This compare and equivalence option automatically equates nets between the schematic and layout blocks which have the same names. If the runset option IGNORE_CASE is set to TRUE, COMPARE performs a case-insensitive name EQUATE.

EQUATE_NETS: This equivalence option can be used to equate nets between the schematic and layout blocks. Because both the schematic and layout net names must be specified, this option can be used to equate nets with different names. If the runset option IGNORE_CASE is set to TRUE, the EQUATE_NETS Equivalence Option will be case-insensitive.

STATIC_EQUATED_NETS: This compare and equivalence option causes all equated nets (resulting from EQUATE_BY_NET_NAME and EQUATE_NETS) to be set static, and will therefore not be removed during merging or filtering operations. This option defaults to TRUE, and must be set FALSE if equated nets are not to be set as static.

SCHEMATIC_SWAPPABLE_PORTS: For cases where the COMPARE process is unable to identify ports that are swappable, the SCHEMATIC_SWAPPABLE_PORTS equivalence option can be used to identify schematic port nets which are logically interchangeable.

This option does not support dependent swapping (described previously). Multiple groups of swappable I/O are always considered to be mutually exclusive.

Note:
This option should be used with caution, since COMPARE makes no attempt to verify that the specified groups of ports can in fact be swapped without changing the functional behavior of the block.

Also refer to "SCHEMATIC_PERMUTABLE_PORTS" on page 5-39.

MATCH_BY_PROPERTY: For cases where a match must be assumed, the COMPARE process normally makes a random matching from identical groups between the schematic and layout blocks. Devices in the group appear to be the same because they are only differentiated based on the connectivity within the block. However, it is sometimes possible to further distinguish devices by their property values. When MATCH_BY_PROPERTY is enabled, the property values are used to make an initial ordering of the devices. In most cases, this will provide the proper matching between devices.

Note:
Only the properties specified in the CHECK_PROPERTIES Equate Process Definitions are used in this property matching procedure.

Properties

By default, only the types and connectivity of devices and nets are checked during the COMPARE process. However, it is also possible to check certain attributes of matched schematic and layout device pairs. These properties include length, width, resistance, capacitance, and area. Make sure that COMPARE_PROPERTIES=TRUE in the COMPARE section. Note that COMPARE will not compare properties if the cell has other LVS problems like opens or shorts.

Comparison of Device Properties

The EQUATE Process Definition CHECK_PROPERTIES specifies the list of properties to check for the device type. To specify that a certain device has no properties to be checked, set CHECK_PROPERTIES=FALSE in the EQUATE for that particular device.

The EQUATE Process Definition TOLERANCE can be used to control the percentage difference allowed between schematic and layout property values before an error or warning message is generated. Property tolerance defaults to 10 percent.

For example, if you want to check an NMOS device type for length values to a tolerance of 5 percent and width values to a tolerance of 10 percent, then the following EQUATE would be appropriate:

```
EQUATE NMOS n=n gate source drain {
   check_properties = { l, w }
   tolerance[l] = { 5 }
   tolerance[w] = { 10 }
}
```

Turning on the COMPARE option COMPARE_PROPERTIES will enable the checking of properties on matched devices for each equivalence point. Note that without the CHECK_PROPERTIES process definition, property values will not be checked.

Severity of Property Differences

There are three levels of property checking: no checking, warnings for differences, and errors for differences. By default, if property checking is enabled and a tolerance is exceeded, an error message is generated. However, differences can be downgraded to a warning message by turning on the COMPARE and/or EQUIV option PROPERTY_WARNING. Checking is completely disabled if the COMPARE option COMPARE_PROPERTIES is set to FALSE.

Merging Device Properties

For parallel MOS devices, the average length of the parallel devices is used as the equivalent length. The width values are added to provide an equivalent width.

The resistance values of parallel and series resistors are calculated as shown here:

Parallel

$$\frac{1}{R_{EQ}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_N}$$

Series

$$R_{EQ} = R_1 + R_2 + \ldots + R_N$$

The capacitance values of parallel and series capacitors are calculated as shown here:

Parallel

$$C_{EQ} = C_1 + C_2 + \ldots + C_N$$

Series

$$\frac{1}{C_{EQ}} = \frac{1}{C_1} + \frac{1}{C_2} + \ldots + \frac{1}{C_N}$$

Certain properties cannot be combined into a single value for a merged device. For example, the length and width properties are maintained as separate values for MOS devices merged into series and path devices. The number of values in the property list corresponds to the number of MOS devices which are members of the merged element.

HTML Output

The HTML output provides much greater ease in debugging LVS runs, but does involve some data creation. In the COMPARE section the default of the option HTML_OUTPUT is FALSE; thus, if it is not explicitly set in the runset, or given on the command line as -html, then the information is not created.

Hercules provides two command line options related to HTML output:

- `-html` produces HTML data and the browse script, 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.

The browse script first checks to see if a copy of Netscape® is currently running on your terminal. If there is, the script brings that copy to the foreground and activates the link to the HTML output generated by the Hercules run. If there is not, the script launches a copy of Netscape first.

Both DRC and LVS errors are displayed in HTML format. The main HTML file that is created is pwd/Compare_Results.html.

The DRC extraction error file is located at pwd/run_details/.html/drcframe.html.

Equivalence File

The equivalence file contains a list of EQUIV and BLACK_BOX command entries which associate schematic module names and layout structure names. These entries determine which schematic module to compare to which layout structure. The equivalence file also allows other equivalence files to be included using the include construct followed by the file name.

Note:
Comments enclosed by /* */ are allowed in this file, but nested comments are not allowed. Also, multiple comments on a line (such as, /* */ /* */) are not allowed. Blank lines are allowed.

The EQUIV syntax is:

```
EQUIV schematic_name = layout_name {
                  Option_Definitions Process_Definitions }
```

The BLACK_BOX syntax is:

```
BLACK_BOX schematic_name = layout_name {
```

Process_Definitions }

The IGNORE_EQUIV syntax is:

```
IGNORE_EQUIV schematic_name = layout_name { }
```

Argument	Description
schematic_name	The schematic module name
layout_name	The layout structure name
Option_Definitions	The specified options can be set to TRUE or FALSE. The syntax is shown below, and the allowed options are defined in Table 5-10.
Process_Definitions	The processing control for each individual equivalence point or black box.

EQUIV Option_Definitions

The EQUIV command specifies structures in the schematic and layout that should be equivalent and must be verified using LVS. The BLACK_BOX command specifies equivalent schematic and layout structures that are to be treated as if they have already been successfully compared.

The EQUIV command options are specified on separate lines. *Option_Definitions* set optional features for comparison in an individual equivalence point.

- Most options are available with the COMPARE command.
- The values in the equivalence file override the corresponding option values defined in the COMPARE command in the runset.

The options are described in detail in the COMPARE Section of the Hercules Reference Manual, Detailed Options chapter. The equivalence file options that are not available with the COMPARE command are described below, after Table 5-10.

The *Option_Definitions* syntax is:

```
option = TRUE | FALSE
```

or

```
option = value
```

The allowed options are defined in Table 5-10. The default for these options is the same as in the COMPARE section of the runset.

Table 5-10 EQUIV Option Definitions

Option	Description	Default
all_ports_texted	Requires text on all layout ports.	FALSE
auto_exclude_equiv	Excludes invalid equivalence points.	TRUE
compare_properties	Includes properties in comparison.	FALSE
detect_permutable_ports	Extracts and apply swappability rules.	FALSE
equate_by_device_name	Equates devices with matching names.	FALSE
equate_by_net_name	Equates nets with matching names.	FALSE
filter	Removes unused devices and nets.	FALSE
match_by_property	Uses property values to match devices.	FALSE
matched_ports_continue	Requires only matching ports.	FALSE
memory_array_comparison	Allows many arrayed memories to be compared faster, and with less memory.	TRUE
merge_net_range	Flags excessive merges through a net.	100 1000
merge_parallel	Merges parallel devices.	FALSE
merge_paths	Merges device stacks along paths.	FALSE
merge_paths_device_limit	Limits device merging along paths.	0 (no limit)
merge_series	Merges series devices.	FALSE
net_print_limit	Limits detail for unmatched nets.	10
one_connection_warning	Flags nets with only one connection.	FALSE
optional_pins	Controls checking of optional pins.	TRUE
parallel_merge_ratio	Ratios properties when parallel merging.	FALSE
proceed_after_port_swap_problem	Continues compare process after finding symmetry error.	FALSE

Table 5-10 EQUIV Option Definitions

Option	Description	Default
property_errors_continue	Continues execution in spite of property errors.	FALSE
property_tolerance	Sets tolerance of property mismatch.	10
property_warning	Reduces property mismatches to warning.	FALSE
push_down_pins	Pushes down hierarchically connected pins.	FALSE
require_texted_ports_match	Requires all design port text to match.	FALSE
require_texted_nets_match	Requires all design text to match.	FALSE
restrict_merge_series	Considers device properties during series merging.	FALSE
short_equivalent_nodes	Creates shorts between electrically equivalent nodes.	FALSE
static_equated_nets	Declares equated nets as static.	TRUE
text_resolves_port_swap	Determines whether text can be relied upon to resolve blocks with complex port swappability.	TRUE
tolerance_device_count	Checks device count tolerances.	NONE
tolerance_net_count	Checks net count tolerances.	NONE
tolerance_type	Specifies relative or absolute property checking.	RELATIVE
use_total_width	Controls width value on fingered devices.	FALSE
zero_connection_warning	Flags nets with zero connections.	FALSE

PROCEED_AFTER_PORT_SWAP_PROBLEM

This option allows the compare process to continues after finding a symmetry error. When this option is set to FALSE, blocks with symmetry errors stop processing at the moment of the symmetry detection. This reduces significant and usually unnecessary processing time spent trying to match a block that has already been determined to have symmetry problems. Setting this option to TRUE causes the normal matching process to proceed even after the symmetry error.

```
proceed_after_port_swap_problem = TRUE|FALSE
```

The default value is FALSE.

FILTER_FUNC

This option allows you to register a Scheme function for filtering instead of the default routine used in the equivalence file.

```
filter_func = equiv-filter
```

An example of FILTER_FUNC is shown in "Using the Scheme Function to Filter Equivalence Points" on page 5-4.

EQUIV Process_Definitions

EQUIV Process_Definitions provide processing control of a specific module. Each definition is specified on a separate line.

The general syntax of the definitions is:

Keyword { *name_pair_list* }

or

Keyword { *name_list* }

Argument	Description
name_pair_list	List of device name pairs or net name pairs.
name_list	List of device name or net names.

The EQUIV Process_Definition keywords are described in the following sections.

ABS_TOLERANCE, REL_TOLERANCE

These keywords allow the specification of:

- different tolerances for different members of multi-EQUATEs.
- tolerances as either absolute or relative values.

- property type (length, width, etc.) tolerances set uniquely for a particular equivalence point in the EQUIV file.

These specifications allow more complex tolerance conditions. Commented examples of runset tolerance settings and related EQUIV file tolerance settings are shown in Example 5-1 and Example 5-2.

Example 5-1 shows the runset settings.

Example 5-1 Runset Settings

```
EQUATE N=N g s d b {
   check_properties = { l, w }
   tolerance[l] = 5              /* Length tolerance of 5% */
   abs_tolerance[w] = { 0.5 }    /* Width tolerance of 0.5 um */
}

EQUATE P = P1 g s d b { /* Schematic device P is in multi-equate */
   check_properties = { l, w }
   rel_tolerance[l] = { 5,10 }  /* rel_tolerance is the same as
                                 * specifying tolerance. */
   rel_tolerance[w] = { 5 }
}

EQUATE P=P2 g s d b {
   check_properties = { l, w }
   abs_tolerance[l] = { 0.75 }   /* Length absolute tolerance
                                  * of 0.75 um. Width tolerance
                                  * will be the global value.*/
}

COMPARE {
   compare_properties = true
   property_tolerance = 15       /* Global property tolerance
                                  * set to 15%. */
   tolerance_type = relative     /* The tolerance type defaults
                        * to relative, making this redundant.*/
}
```

Example 5-2 shows the EQUIV file settings.

Example 5-2 EQUIV File Settings

```
EQUIV inv=INV {
   abs_tolerance { N=N[w] = { 0.5, 0.4 } /* Width tolerance for
                  * equate N=N is overridden to an absolute
                  * value of -0.5 um and +0.4 um. */

   P=P2[w] = { 0.5, 0.4 } /* Width tolerance for equate P=P2 is
                  * specified for the first time as having an
                  * absolute tolerance of -0.5 um and +0.4 um.
                  * NOTE: this is not overriding a previous
                  * EQUATE specification, but the global value
                  * that would have been used for
```

```
                 * this property. */
   }
   rel_tolerance { N=N[l]={ 5, 10 }
                /* The length tolerance for equates N=N
                 * P=P1[l]={ 15 } and P=P1 equate are
                 * overridden to their respective relative
                 * values. */
   }
     /* Without the following lines, the tolerances of the
      * remaining properties would be as follows:
      * P=P1[w] = 5% relative - from the EQUATE
      * P=P2[l] = 15% relative - from COMPARE section */

   property_tolerance = 5
   tolerance_type = absolute
     /* These two lines only change P=P2[l] tolerance because
      * it is the only one receiving its value from global
      * specification. It now becomes a 5 um absolute
      * tolerance. */
}

EQUIV ad4ful = AD4FUL { }   /* No changes to tolerance given
                             * in runset */

EQUIV tgate = TGATE {
   property_tolerance = 10  /* Global tolerance is changed but
               * the tolerance type is not.  The only property
               * affected is P=P2[l] which goes from a global
               * tolerance of 15% to 10%. */
}
```

DELETE_LAYOUT_INSTANCE

This keyword specifies which layout instances to ignore when COMPARE is operating. It is specific to each EQUIV entry individually, and affects only those EQUIV entries which include the DELETE_LAYOUT_INSTANCE keyword. The instance and all data beneath it hierarchically are ignored. Specify the instances to ignore using an instance path with / separating hierarchical levels. The syntax is:

```
delete_layout_instance = { layout_inst1, ..., layout_instN}
```

The default setting is NONE.

DELETE_SCHEMATIC_INSTANCE

This keyword specifies which schematic instances to ignore when COMPARE is operating. It is specific to each EQUIV entry individually, and affects only those EQUIV entries which include the DELETE_SCHEMATIC_INSTANCE keyword. The instance and all data beneath it hierarchically are ignored. In order to delete or ignore hierarchical instances, specify the instance path using / to separate hierarchical levels. The syntax is:

```
delete_schematic_instance = { schematic_instance1, ...,
                              schematic_instanceN }
```

The default setting is NONE.

EQUATE_DEVICES

This keyword equates device names in the schematic module to device names in the layout module. Equating devices reduces the amount of required processing. This process matches the devices being compared. Devices in symmetrical designs can often be matched in a number of different ways. In such cases the equate_devices Process_Definition can be used to prematch devices that might otherwise be interchanged during the comparison. The syntax is:

```
equate_devices{ schematic_device1=layout_device1, ...,
                schematic_deviceN=layout_deviceN }
```

The default setting is NONE.

EQUATE_NETS

This keyword equates net names in the schematic module to net names in the layout module. Equating nets reduces the amount of required processing. This process matches the nets being compared. Nets in symmetrical designs can often be matched in a number of different ways. In such cases the equate_nets Process_Definition can be used to prematch nets that might otherwise be interchanged during the comparison. If the Hercules runset option IGNORE_CASE is set to TRUE, the specified nets are case-insensitive. The syntax is:

```
equate_nets{ schematic_net1=layout_net1, ...,
             schematic_netN=layout_netN }
```

The default setting is NONE.

EXCLUDE_EQUIV

This keyword explodes specified submodules within the module being compared. All occurrences of the named schematic module and layout structure are exploded within the module being compared. Excluding equivalence points helps to compensate when the hierarchy in a schematic module does not exactly match the hierarchy in the layout structure. The syntax is:

```
exclude_equiv { schematic_module1=layout_structure1, ...,
                schematic_moduleN=layout_structureN }
```

The default setting is NONE.

FILTER_OPTIONS

FILTER_OPTIONS, FILTER_LAYPOUT_OPTIONS, and FILTER_SCHEMATIC_OPTIONS allow you to specify different filtering options for different equivalence points. The EQUATE FILTER_OPTIONS are considered the initial default options. All equivalence points use these options unless they are overridden in the equivalence file.

```
filter_options {
  schematic_device1=layout_device1 = { option1, option2 }
  ...
  schematic_deviceN=layout_deviceN = { option1, option2 }}
```

The default setting is NONE.

Example 5-3 shows a runset that uses filtering options in EQUATE.

Example 5-3 Runset

```
EQUATE N=N g s d b {
   filter_options = { NMOS-1 NMOS-3 NMOS-4 }
}

EQUATE P=P g s d b {
   filter_schematic_options = { PMOS-1 }
   filter_layout_options = { PMOS-1 PMOS-3 PMOS-6 }
}

COMPARE {
   filter = TRUE
}
```

Example 5-4 shows additional equivalence file settings to control the filtering down to cells.

Example 5-4 Equivalence File Settings

```
EQUIV inv=INV {
   filter_options { N=N = { NMOS-8 }
            /* N devices in both the schematic and
             * layout now filter using only option
             * NMOS-8; all the EQUATE options are
             * ignored. */
      P=P = { PMOS-1 PMOS-7 }
           /* Similarly for the P devices;
            * filtered using PMOS-1 and PMOS-7 in both the
            * schematic and layout regardless of
            * what the EQUATE says. */
   }
}

EQUIV ad4ful = AD4FUL { }
           /* No changes to filtering in runset. */

EQUIV tgate = TGATE {
```

```
   filter_layout_options { N=N = { NMOS-6 }
           /* N devices filtered using NMOS-6 in
            * the layout, and not filtered at all
            * in the schematic.*/
   }
    /* Since no alternate specification is given for
     * P devices in this equiv, the default value is taken from
     * the EQUATE in the runset. */
}
```

LAYOUT_STATIC_DEVICE

This keyword prevents special processing of specific device instances in the layout module during comparison. Devices included in the list are not allowed to be merged or filtered. The syntax is:

```
layout_static_device = { layout_device1, ...,
                         layout_deviceN }
```

The default setting is NONE.

LAYOUT_STATIC_NET

This keyword prevents special processing of specific nets in the layout module during comparison. No merging or filtering operation is performed that would cause the specified layout nets to be removed. The syntax is:

```
layout_static_net = { layout_net1, ..., layout_netN }
```

The default setting is NONE.

SCHEMATIC_PERMUTABLE_PORTS

The SCHEMATIC_PERMUTABLE_PORTS option allows permutability rules in the equivalence file to be defined. A combination of fixed port groups (f), permutable port groups (p), and cyclic port groups (c) can be defined. Defining permutability rules allows LVS to correctly identify equivalences resulting from dependent swappability. The syntax is:

```
schematic_permutable_ports = { perm_string }
```

The default setting is NONE. Refer to “SCHEMATIC_PERMUTABLE_PORTS” on page 5-39 for more information.

Note:

Defining permutability rules that do not reflect the actual, correct dependent swappability relationships can cause unmatched circuits to pass LVS comparison undetected.

SCHEMATIC_STATIC_DEVICE

This keyword prevents special processing of specific device instances in the schematic module during comparison. Devices included in the list are not allowed to be merged or filtered. The syntax is:

```
schematic_static_device = { schematic_device1, ...,
                            schematic_deviceN }
```

The default setting is NONE.

SCHEMATIC_STATIC_NET

This keyword prevents special processing of specific nets in the schematic submodule during comparison. No merging or filtering operation is performed that would cause the specified schematic nets to be removed. The syntax is:

```
schematic_static_net = { schematic_net1, ...,
                         schematic_netN }
```

The default setting is NONE.

SCHEMATIC_SWAPPABLE_PORTS

This keyword specifies a group of interchangeable schematic ports. The port names in the list are treated as being logically equivalent for comparison purposes at higher levels of the design hierarchy.

```
schematic_swappable_ports = { schematic_port1, ...,
                              schematic_portN }
```

The default setting is NONE.

BLACK_BOX Process_Definition

The BLACK_BOX Process_Definitions provide processing control of a specific module. Each definition is specified on a separate line.

The general syntax of the definitions is the same as for EQUIV Process_Definition:

```
Keyword { name_pair_list }
```

or

```
Keyword = { name_list }
```

Argument	Description
name_pair_list	List of device name pairs or net name pairs.
name_list	List of device name or net names.

The BLACK_BOX Process_Definition keywords are described in the following sections.

EQUATE_PORTS

The EQUATE_PORTS keyword specifies the port correspondence between the schematic and layout structures. Every port in both the schematic and the layout must be given in the EQUATE_PORTS keyword. The syntax is:

```
equate_ports { schematic_port=layout_port, ...,
               schematic_port=layout_port }
```

The default setting is NONE.

REMOVE_LAYOUT_PORTS, REMOVE_SCHEMATIC_PORTS

These keywords specify any ports in one netlist without an equivalent port in the other netlist (for example, dangling ports and feed-through nets). Every port on a cell that is black boxed must be specified within the BLACK_BOX entry using these keywords. Each port not listed within the BLACK_BOX creates an error.

```
remove_layout_ports = { layout_port, ..., layout_port2 }
```

and

```
remove_schematic_ports = { schematic_port, ...,
                           schematic_port }
```

The default setting is NONE for both of these.

SCHEMATIC_PERMUTABLE_PORTS

The SCHEMATIC_PERMUTABLE_PORTS option allows permutability rules in the equivalence file to be defined. A combination of fixed port groups (f), permutable port groups (p), and cyclic port groups (c) can be defined. Defining permutability rules allows LVS to correctly identify equivalences resulting from dependent swappability. The syntax is:

```
schematic_permutable_ports = { perm_string }
```

The default setting is NONE. Refer to "SCHEMATIC_PERMUTABLE_PORTS" on page 5-39 for more information.

Note:
Defining permutability rules that do not reflect the actual, correct, dependent swappability relationships can cause unmatched circuits to pass LVS comparison undetected.

SCHEMATIC_SWAPPABLE_PORTS

This keyword specifies a group of interchangeable schematic ports. The port names in the list are treated as being logically equivalent for comparison purposes at higher levels of the design hierarchy.

```
schematic_swappable_ports = { schematic_port, ...,
                              schematic_port }
```

The default setting is NONE.

IGNORE_EQUIV

The IGNORE_EQUIV syntax is consistent with that of an EQUIV or BLACK_BOX. There are no options for IGNORE_EQUIV and the braces ({ }) in the syntax must be empty. The syntax is:

```
IGNORE_EQUIV sch=lay { }
```

where sch is the name of a schematic cell and lay is the name of a layout cell.

The sch=lay pair on an IGNORE_EQUIV must correspond exactly to a sch=lay pair on an EQUIV in the same equivalence file. Its function is to ignore a previously declared EQUIV. It has the same effect as commenting out an EQUIV; the EQUIV is treated as if it never existed. It is most commonly used in conjunction with PRINT_IGNORE_EQUIV COMPARE option, so that unwanted EQUIV entries can be discarded automatically, avoiding the tedious effort of finding them in the file and commenting them out manually.

EQUIV and BLACK_BOX Syntax Examples

Example 5-5 shows several examples of EQUIV and BLACK_BOX in an equivalence file.

Example 5-5 EQUIV and BLACK_BOX Syntax Examples

```
EQUIV CORE=core {
   equate_nets { gnd=VSS }
   exclude_equiv { inv=inv }
}
EQUIV tgate=TGATE {
   schematic_swappable_ports={ SEL SELB }
```

```
   merge_series=FALSE
}
EQUIV inv=inv {
   compare_properties=FALSE
}
INCLUDE extra_equiv.ev
BLACK_BOX NAND2=NAND2 {
   EQUATE_PORTS { in1=in_1 in2=in_2 out=out VSS=VSS VDD=VDD }
   SCHEMATIC_SWAPPABLE_PORTS={ in1 in2 }
}
BLACK_BOX logic=L_LOGIG {
   equate_ports { A=A IN=IN CLK=Q CLKBAR=QBAR }
}
```

Schematic Netlist Translators

The schematic netlist translator host creates a skeletal equivalence file. The skeletal equivalence file contains an entry for each schematic module in the design. For these entries, the layout structure name is assumed to be the same as the schematic module name. You must change the layout structure name entry to the correct name for the cases where the schematic module name and layout structure name are not identical. The path to the equivalence input file must be specified in the runset file HEADER section by setting the EQUIVALENCE variable.

Permutable Ports for LVS COMPARE

The COMPARE process begins by matching all of the unique devices and nets in a block. The remaining devices and/or nets (assuming that the blocks are identical) appear to be symmetrical. In other words, there are groups of devices and nets in both blocks which look identical. Rather than stop at this point, the COMPARE process attempts to make further progress by assuming a match. A match is assumed by choosing a single device or net from a group in the schematic block and pairing it to a corresponding element in the group from the layout block. Then the matching procedure is continued, pairing any elements which now appear to be unique as a result of the assumed match. This process continues until all elements have been matched.

The COMPARE process provides a flexible and effective means of dealing with circuit symmetry. These options are discussed in the following sections.

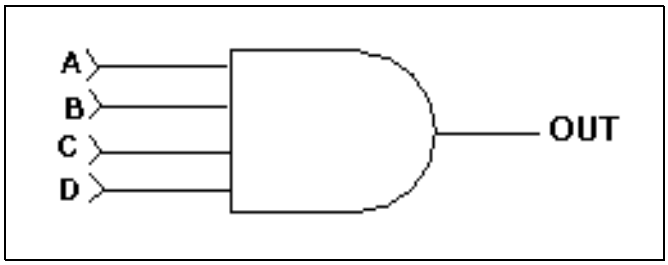
Ports which can be swapped are defined as those ports on a device or block which are logically equivalent. References of the block can have the net connections to these ports interchanged without changing the logical function of the circuit.

Independently swappable ports are logically equivalent ports that can be interchanged without affecting their function within the block. For example, any inputs of a NAND gate can be swapped without changing the function of the gate. To match circuits that contain independently swappable ports, identify the ports with one of the following options:

- SCHEMATIC_SWAPPABLE_PORTS—Define ports in the Equivalence file using this option.

In Figure 5-7, the input ports A, B, C, and D of the NAND gate are independently swappable.

Figure 5-7 Independent Swappability



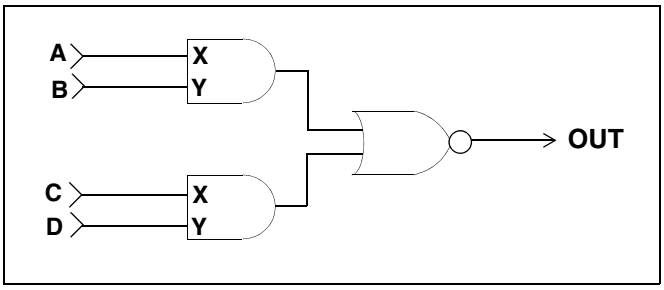
A special case of port swapping occurs for circuits which contain dependencies between groups of swappable ports. A block which functions the same if ports A and B are swapped and ports C and D are swapped is said to contain dependent swapping. In Figure 5-8, the input pairs A & B and C & D are independently swappable.

Dependently swappable ports rely on their functional relationship within the block and, therefore, cannot be separated from one another. LVS COMPARE offers the following two options for handling dependent swappability:

- SCHEMATIC_PERMUTABLE_PORTS—Define swappability rules in the equivalence file using this option.
- DETECT_PERMUTABLE_PORTS—Place this command in the runset to have swappability rules applied and extracted automatically.

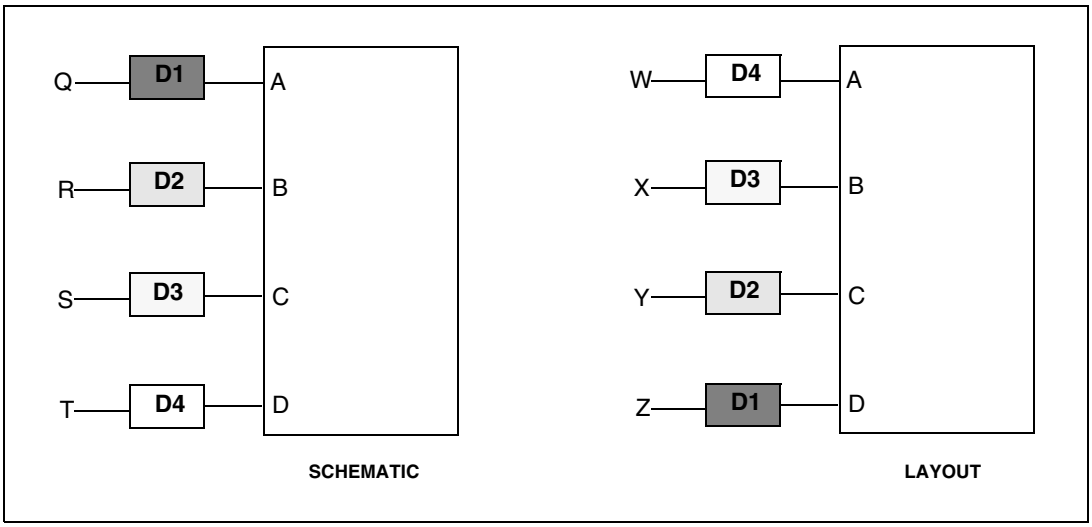
In the AND-OR-INVERT cell of Figure 5-8, A cannot be swapped with C unless B is swapped with D, thus demonstrating dependent swappability. In contrast, the independently swappable ports, A and B, are interchangeable, as are C and D.

Figure 5-8 AND-OR-INVERT



Equivalent properties must be specified for LVS COMPARE to recognize equivalent designs during an LVS COMPARE. For example, in Figure 5-9, the schematic and layout cells each connect with a set of circuits D1-D4. The symmetry of the circuits is the same despite the difference in arrangement. Unless the equivalence properties for these cells were specified, parts of the hierarchy would need to be flattened and some equivalence points removed for LVS COMPARE to recognize the equivalence.

Figure 5-9 COMPARE of AND-OR-INVERT



In the equivalence file in the runset, the commands EQUIV and BLACK_BOX can both be used with the independent swappability option SCHEMATIC_SWAPPABLE_PORTS and the dependent swappability option SCHEMATIC_PERMUTABLE_PORTS. Only the EQUIV command can be used with the dependent swappability option DETECT_PERMUTABLE_PORTS.

The effects of the following COMPARE option, from the COMPARE command of the runset (see the *Hercules Reference Manual*, Detailed Options chapter) are shown when the option is used in conjunction with either SCHEMATIC_SWAPPABLE_PORTS or SCHEMATIC_PERMUTABLE_PORTS:

- EQUATE_BY_NET_NAME and SCHEMATIC_PERMUTABLE_PORTS are compatible.

SCHEMATIC_PERMUTABLE_PORTS

You can manually define your own swappability rules in the equivalence file with the option SCHEMATIC_PERMUTABLE_PORTS. This method lets LVS COMPARE automatically identify the defined equivalences resulting from dependent swappability, without the need for further intervention. The syntax is:

```
schematic_permutable_ports = { perm_string }
```

where *perm_string* is an ordered sequence of expressions beginning with F, P, or C. The keywords are defined in Table 5-11.

```
(f|p|c perm_string)
```

or

```
(f|p|c port_list)
```

or

```
perm_string perm_string
```

where *port_list* is:

```
port_name port_name
```

or

```
port_name port_list
```

where *port_name* is:

```
ascii_string
```

Table 5-11 Ordering Sequence

Keyword	Definition
f	fixed: Indicates that the group elements cannot be swapped (non-permutable).

Table 5-11 Ordering Sequence

Keyword	Definition
p	permutable: Indicates that any of the group elements can be swapped (independently swappable).
c	cyclic: Indicates that the set of group elements may be rotated (cyclically permutable). For example, `c A B C` indicates that the permissible permutations are A B C, B C A, and C A B.

In Figure 5-8 on page 5-38, the correct syntax for the AND-OR-INVERT cell is

```
SCHEMATIC_PERMUTABLE_PORTS = {(p (p A B) (p C D))}
```

Other circuit examples are provided in the "Error Messages" on page 5-45. Table 5-12 illustrates the interpretation of some sample rules.

Table 5-12 Example Permutations

Rule	Valid Permutations
(p (f W X) (f Y Z))	W X Y Z Y Z W X
(p W X) (p Y Z)	W X Y Z X W Y Z W X Z Y X W Z Y
(p (p X Y Z) (f A B C))	A B C X Y Z X Y Z A B C A B C X Z Y X Z Y A B C A B C Y X Z Y X Z A B C A B C Y Z X Y Z X A B C A B C Z X Y Z X Y A B C A B C Z Y X Z Y X A B C
(p (c X Y Z) (f A B C)	A B C X Y Z X Y Z A B C A B C Y Z X Y Z X A B C A B C Z X Y Z X Y A B C

Any one combination of several rules implies a variety of valid permutations. For example,

```
(p (f A B) (f C D))
   (p B D)
```

implies the permutation

A B C D -> C B A D

The first rule permits A B C D -> C D A B. The second rule allows the swapping of B and D.

Also, SCHEMATIC_PERMUTABLE_PORTS can be combined in any fashion with the independent swappability option SCHEMATIC_SWAPPABLE_PORTS. Both options can express independent swappability, since SCHEMATIC_PERMUTABLE_PORTS is a broader case of SCHEMATIC_SWAPPABLE_PORTS. For example:

```
SCHEMATIC_SWAPPABLE_PORTS = {A B C D}
```

is equivalent to

```
SCHEMATIC_PERMUTABLE_PORTS = {(p A B C D)}
```

DETECT_PERMUTABLE_PORTS

DETECT_PERMUTABLE_PORTS automatically extracts and applies swappability rules. It handles independent or dependent swappability without the manual entry required by SCHEMATIC_SWAPPABLE_PORTS or SCHEMATIC_PERMUTABLE_PORTS.

DETECT_PERMUTABLE_PORTS is a Boolean option that is either TRUE or FALSE. The default is FALSE.

Apply DETECT_PERMUTABLE_PORTS globally by specifying it in the COMPARE section of the runset. Apply it to individual blocks by specifying it in the EQUIV file. Finally, apply the option globally with exception to selected blocks by specifying it both in the runset and setting it to FALSE in the EQUIV file for the appropriate blocks.

DETECT_PERMUTABLE_PORTS extracts symmetries (dependent swappability rules) for the block to which it is applied. This option does not affect criteria for determining the block's equivalence, but instead affects the criteria of the block's parent cell. Consequently, if the option is specified globally, it does not operate on the top-level block unless set to TRUE for the top-level block in the EQUIV file.

Two files, *block*_lvs.log and sum.*block*.*block*, are associated with the execution of both SCHEMATIC_PERMUTABLE_PORTS and DETECT_PERMUTABLE_PORTS.

*block*_lvs.log File

The *block*_lvs.log file lists messages that appear after SCHEMATIC_PERMUTABLE_PORTS or DETECT_PERMUTABLE_PORTS is invoked.

Example 5-6 is a section of an example *block*_lvs.log file. The messages that are highlighted are explained below.

Example 5-6 Example block_lvs.log File

```
**** LEVEL 1 ****

Schematic = AOI, Layout = AOIX, Level = 1
   Partitioning schematic ... OK
   Partitioning layout ... OK
   Comparing ...
      Number of dependently swappable ports = 4
      Building swappability representation ... WARNINGS
      WARNING: Matches were assumed
      WARNING: Symmetry warning
   Summary File: compare/AOIX/sum.AOI.AOIX
   Elapsed time = 0:00:02  User=0.07 Sys=0.17 Mem=2.358

Equivalent blocks:  level 1
    AOI == AOIX

**** LEVEL 0 ****

Schematic = CELL, Layout = CELL, Level = 0
   Partitioning schematic ... OK
   Partitioning layout ... OK
   Comparing ... WARNINGS
      WARNING: Port permutations were required
   Summary File: compare/CELL/sum.CELL.CELL
   Elapsed time = 0:00:00  User=0.07 Sys=0.10 Mem=2.374

Equivalent blocks:  level 0
   CELL == CELL

**** SUMMARY COMPARE RESULTS ****
Compare messages printed:
   WARNING: Matches were assumed
        [AOI, AOIX]
   WARNING: Symmetry warning
        [AOI, AOIX]
   WARNING: Port permutations were required
        [CELL, CELL]
```

The messages that are highlighted in Example 5-6 are explained here.

- Number of dependently swappable ports = 4

  This is the total number of different ports that appear in all swappability rules for the device.

- Building swappability representation

  This step creates a data structure which encodes the rules you have specified. The time for this step depends on the number of rules, and the total number of ports appearing in the rules. If you find that this step is slow, simplify or restrict the rules specified in the equivalence file.

- WARNING: Port permutations were required

  This message is printed for each block that required a swappability rule applied to establish equivalence, both in the individual block report and the summary at the end.

sum.*block.block* File

If message CMP-140 is enabled, the sum.*schematic_cell_name.layout_cell_name* file (referred to as the sum.*block.block* file) displays a port permutation map that lists permutations necessary for equivalence.

The following example shows a port permutation map. This output indicates the required port permutations for AOI instance $1I1 to establish equivalence in the parent cell.

```
WARNING: Port permutations were required.
Post-Compare Port Permutation Map:
   Cell AOI:
       Instance $1I1:
         Q --> T
         R --> S
         S --> R
         T --> Q
```

Any output that appears with SCHEMATIC_PERMUTABLE_PORTS can also appear with DETECT_PERMUTABLE_PORTS with minor additions. The highlighted lines in Example 5-7 indicate the additions.

Example 5-7 Permutable Port Output

```
**** LEVEL 1 ****
Schematic = AOI, Layout = AOIX, Level = 1
   Partitioning schematic ... OK
   Partitioning layout ... OK
   Checking for swappable ports ...
      Number of dependently swappable ports = 4
      Building swappability representation ...
   Comparing ... WARNINGS
      WARNING: Matches were assumed
      WARNING: Symmetry warning
   Summary File: compare/AOIX/sum.AOI.AOIX
   Elapsed time = 0:00:01 User=0.15 Sys=0.12 Mem=2.374
```

Extracted symmetries are also reported in the sum.block.block file, as shown in Example 5-8.

Example 5-8 Extracted Symmetries

```
Swappable Port Groups:

   A -> B
   B -> A
```

```
C -> D
D -> C

A -> C
B -> D
C -> A
D -> B
```

The output of the sample describes three permutations: swapping A with B, swapping C with D, and simultaneously swapping A with C and B with D. Any combination of these permutations is also a symmetry of the block.

Performance

There are two procedures in which dependent swappability can slow down performance: during the extraction of dependent swappability rules in large designs with DETECT_PERMUTABLE_PORTS, or during the application of dependent swappability rules while matching occurs with either DETECT_PERMUTABLE_PORTS or SCHEMATIC_PERMUTABLE_PORTS.

Extraction of Dependent Swappability Rules

The performance of DETECT_PERMUTABLE_PORTS depends on the total number of devices and nets in the block. To accelerate efficiency for large designs, DETECT_PERMUTABLE_PORTS is suppressed above its threshold default value of 512. When the total number of nets and devices is greater than 512, LVS does not extract symmetries of the block. At its default value, DETECT_PERMUTABLE_PORTS barely affects performance. You can alter the threshold, either in the COMPARE section of the runset, or in individual blocks in the EQUIV file, with the option setting

```
DETECT_PERMUTABLE_PORTS_LIMIT = n
```

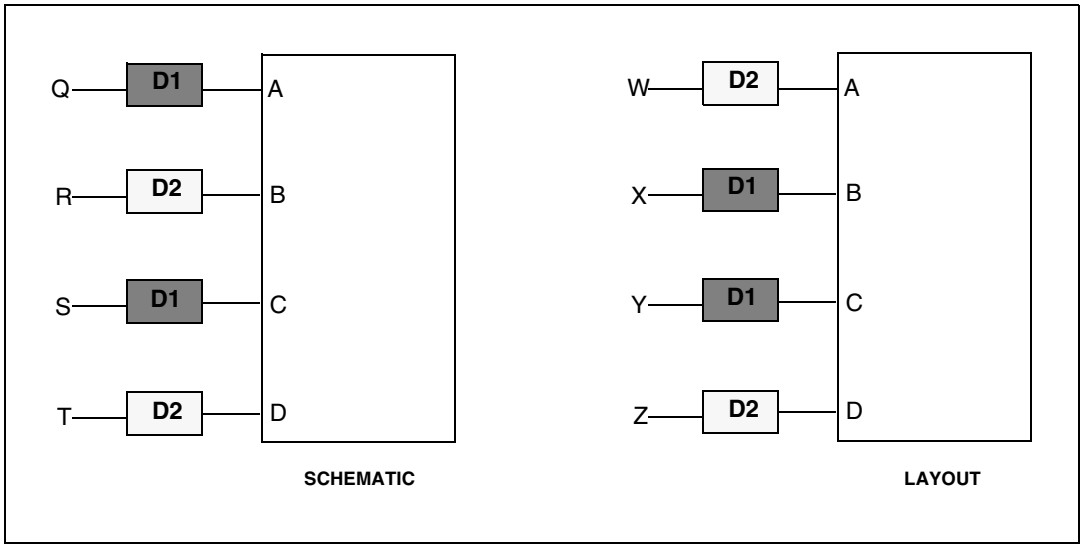
where *n* is the threshold. If n=0, no limit is applied, which results in the application of DETECT_PERMUTABLE_PORTS to blocks regardless of size. An estimate of the number of nets and devices in a block is sufficient for effective use of this option.

Application of Dependent Swappability Rules

The performance of SCHEMATIC_PERMUTABLE_PORTS depends on the complexity of the rules. More permutable (that is, not fixed) ports cause greater complexity. Generally, performance degrades rapidly as the number of swappable ports exceeds 100. To improve performance, simply identify and eliminate nonessential ports.

For example, in Figure 5-10, because there are two rather than four distinct devices (in contrast to Figure 5-9), dependent swappability is not necessary to establish equivalence.

Figure 5-10 COMPARE Not Requiring Dependent Swappability



For instance, Q in the schematic may be correctly (and arbitrarily) matched with X or Y in the layout. Only the independent swappability rules (A with B, and C with D) are necessary to compare the designs. Although the number of ports were not reduced in Figure 5-10, in general this strategy allows users to reduce both the number of ports and the complexity of the rules in more complex schematics. Also, restricting the rules to purely independent swappability yields better performance than mixing independent with dependent swappability.

Users can also improve the performance of SCHEMATIC_PERMUTABLE_PORTS by expressing the rules as compactly as possible. For example, the following rules are logically equivalent:

```
SCHEMATIC_PERMUTABLE_PORTS = {(p (f A B)(f C D)) (p A B)}

SCHEMATIC_PERMUTABLE_PORTS = {(p (p A B)(p C D))}
```

Avoid specifying a port more than once, either within a rule or among several rules, or a significant penalty in performance can occur. For this reason, the second, more compact rule in the example produces faster results.

Error Messages

Error messages are sent to standard output, as well as the *block*_lvs.log file. The possible error messages are explained below:

- ERROR: SCHEMATIC_PERMUTABLE_PORTS inconsistent rule.

  An inconsistent or illogical swappability rule was specified. One of the following two messages is displayed in the sum.block.block file:

  - ERROR: Multiple occurrences of a port in SCHEMATIC_PERMUTABLE_PORTS rule.

    This indicates that a port appeared more than once in the same rule, such as in this rule: `(p (p A B C)(p A D E))`

  - ERROR: Port name is extraneous in SCHEMATIC_PERMUTABLE_PORTS rule.

    This indicates an attempt to swap groups of different sizes. For example, in the rule: `(p (f A B C) (f W X Y Z))`, Z would be reported as extraneous.

- ERROR: Port permutation error

  Due to this error, the following message is displayed in the sum.*schematic_cell_name.layout_cell_name* file, which refers to the parent of the block exhibiting dependent swappability.

  - ERROR: Unable to verify permutation for block/instance.

    In this error message, block and instance refer to the cell and instance names of the device (a subcell of parent) that exhibits dependent swappability. LVS COMPARE is unable to confirm from the swappability rules that the port mapping required for equivalence is a valid permutation. Three strategies for coping with this infrequent error are listed below:

    (1) Remove nonessential swappability rules.

    (2) Add some EQUATE_NETS commands to parent to restrict the possible permutations used to attempt a match.

    (3) Remove the equivalence point for block.

Limitations

SCHEMATIC_PERMUTABLE_PORTS and DETECT_PERMUTABLE_PORTS cannot propagate symmetry rules through the hierarchy. The options only apply symmetries to the first-order subcells of the current block. Subcells of subcells do not affect the parent. For example, in the cell IAOI of Figure 5-11, if a rule for the AND gate is:

```
(p X Y)
```

and if a partial symmetry for the IAOI cell is:

```
(p (f A B) (f C D))
```

then the complete symmetry below would not apply inside the cell containing the IAOI:

```
(p (p A B) (p C D))
```

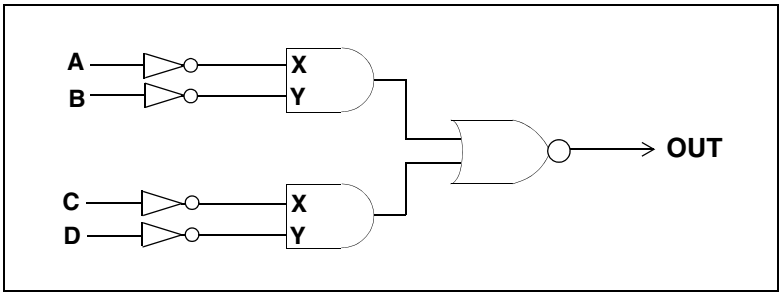
For example, this is not a legitimate permutation:

```
A B C D  -->  D C B A
```

If you suspect that this limitation is affecting your results, you can either specify a SCHEMATIC_PERMUTABLE_PORTS rule for the block in question, or remove that equivalence point.

Note:
Symmetries are inferred for the parent when the swappable ports of the subcell are connected directly to the ports of the parent with no intervening devices, and the subcell exhibits greater symmetry than the parent (as in Figure 5-8 on page 5-38).

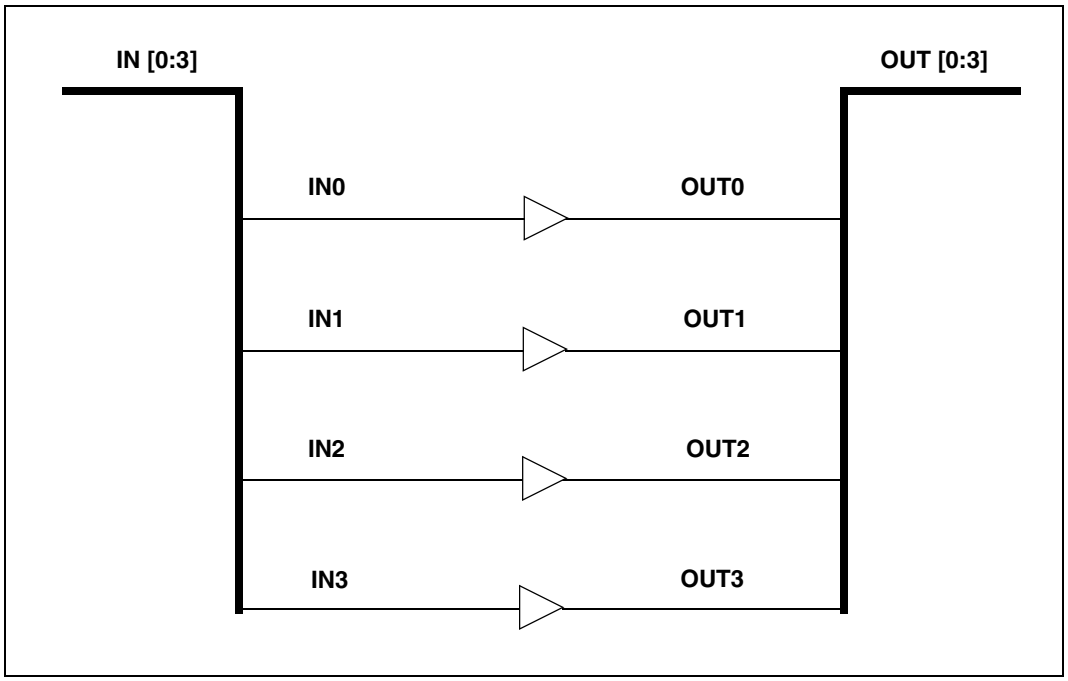
Figure 5-11 Cell IOAI



SCHEMATIC_PERMUTABLE_PORTS Examples

The example in Figure 5-12 shows the input and output buses, which are permutable between each bits.

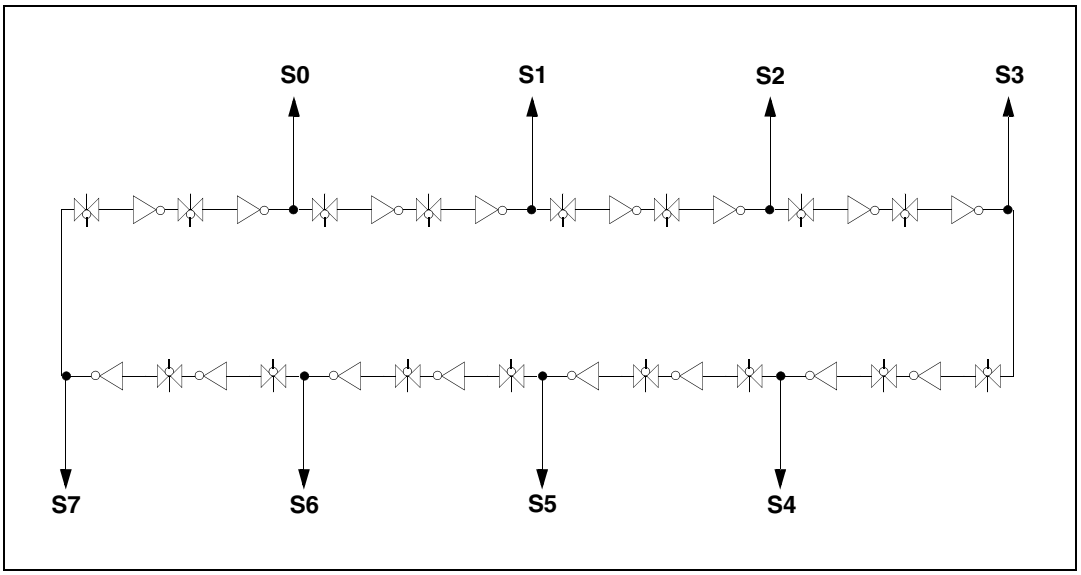
Figure 5-12 Buffer (BUF4X4)




```
SCHEMATIC_PERMUTABLE_PORTS = {
   (p (f IN0 OUT0) (f IN1 OUT1) (f IN2 OUT2) (f IN3 OUT3))
}
```

The example in Figure 5-13 shows all bits in shift register, which are cyclically permutable.

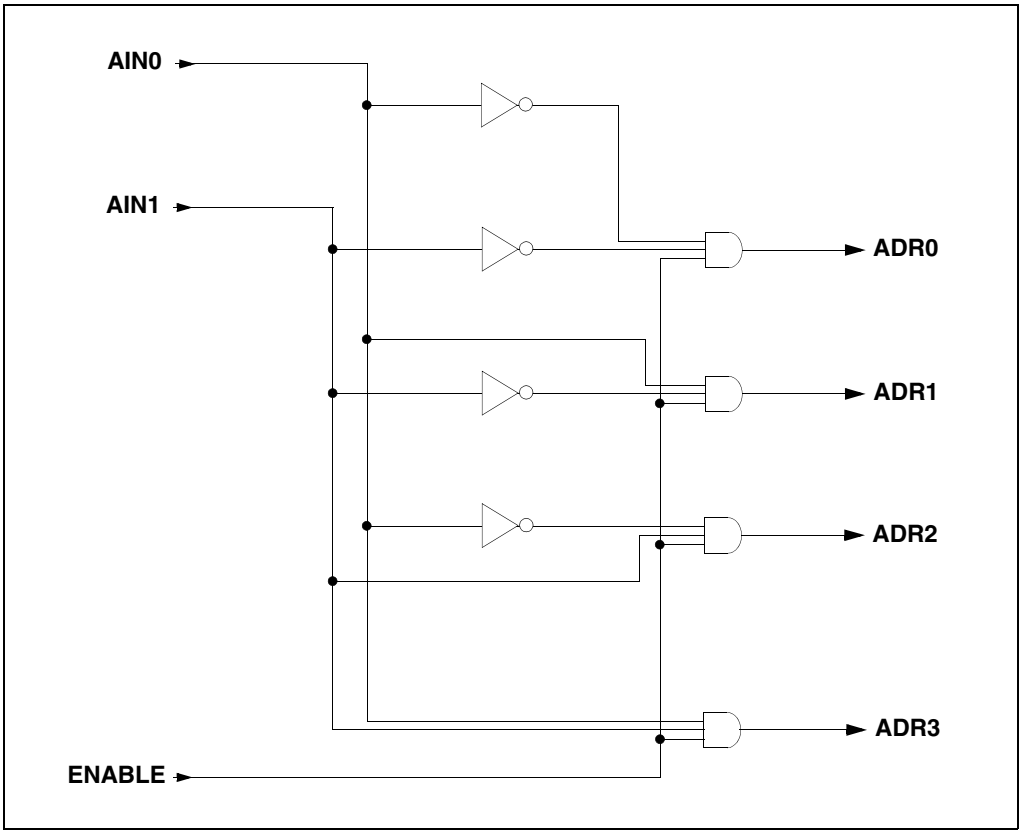
Figure 5-13 Cyclic Shift Register (SHIFTREG8)




```
SCHEMATIC_PERMUTABLE_PORTS = {(c S0 S1 S2 S3 S4 S5 S6 S7)}
```

The example in Figure 5-14 shows that groups (AIN0 ADR1) and (AIN1 ADR2) are permutable.

Figure 5-14 Decoder (ADR2DEC4)




```
SCHEMATIC_PERMUTABLE_PORTS = { (p (f AIN0 ADR1) (f AIN1 ADR2)) }
```

LVS Black Box Flow

To run Hercules LVS COMPARE with black box structures, you must have an ascii text file with a list of black box structures. If you are beginning your Hercules run with a Dracula physical verification command file that contains hcell statements, Hercules automatically creates a BLACK_BOX_FILE and specifies the path to the file. If you create a BLACK_BOX_FILE manually, you must specify the path to the file using the HEADER section option, BLACK_BOX_FILE.

The syntax for the BLACK_BOX_FILE is as follows:

```
LVS = { schematic_structure = layout_structure ...
        schematic_structure = layout_structure }
```

The BLACK_BOX_FILE contains a list of structures that are treated as if they compare, even though no comparison verification has been completed on them. Any device and connection data contained within the black box structure is ignored. The BLACK_BOX_FILE does not need to contain information about the port connections of the black box structures; Hercules LVS COMPARE determines this information automatically, if the following criteria are met:

- The file must specify the name of the both the schematic cell and the layout cell that correspond together.
- Every port in the schematic cell must have a corresponding port in the matching layout cell with the exact same port name. However, extra ports may exist in the layout only for cases of PUSH_DOWN_PIN connections, global power/ground connections, and feedthroughs.

Debug Starting Point: LVS Error File

A *block*.LVS_ERROR file is created for each Hercules LVS run. The file gives the PASS or FAIL result and lists the errors of the failed equivalence points in priority. After an LVS run, this is the first file you should investigate to debug your design.

Environment and Compare Result—The following example shows the section in the LVS error file that lists the Hercules version, the runtime environment, and the pass/fail result. It also gives you a summary of the number of successful and failed equivalence points.

```
+-------------------------------------------------------------+
|                 Hercules LVS Comparison Report              |
+-------------------------------------------------------------+

COMPARE (R) Hierarchical Layout Vs. Schematic
     2006.12.0600     2006/10/19
Copyright (C) Synopsys, Inc. All rights reserved.
---------------------------------------------------------------
LVS error file     = DAC96.LVS_ERRORS
Schematic netlist  = dac96.hrc
Layout netlist     = DAC96.net
Equivalence file   = dac96.eqv
Runset file        = test.ev
Working directory  = /remote/SEG_sys/larryke/hercules/example/
tutorial/Getting_Started_Hercules_LVS/dac96lvs1
Compare directory  = compare
Compare start time = 2006-10-19 14:44:41
```

If there are layout errors, such as text or C_THRU errors, in your ERC run, you will see a Layout Errors section before the PASS or FAIL header section. The following example shows that there are text short errors in the ERC run.

```
+-------------------------------------------------------------+
```

```
|                        Layout Errors                          |
+---------------------------------------------------------------+
ERROR: The following errors are listed in "SRAM.LAYOUT_ERRORS":

   ERR_TEXT_SHORTEST_PATH (1)
   ERR_TEXT_SHORT (1)
```

Details are in the *block*.LAYOUT_ERRORS file. You should try to fix these errors even when the LVS run shows PASS.

If the LVS compares successfully, a large block PASS is shown.

```
---------------------------------------------------------------
Top block compare result: PASS

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

                     [DAC95 == DAC96]

---------------------------------------------------------------
Comparison summary

    84 successful equivalencies
     0 failed equivalencies
```

If the LVS fails to compare, a large block FAIL is shown.

```
---------------------------------------------------------------
Top block compare result: FAIL

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

                      [DAC95 != DAC96]

---------------------------------------------------------------
Comparison summary
    83 successful equivalencies
  *  1 failed equivalencies
        1 first priority errors
        0 second priority errors
```

In the COMPARE section of the runset file, if you set any of these:

- PROPERTY_ERRORS_CONTINUE = TRUE

- MATCHED_PORTS_CONTINUE = TRUE
- MATCHED_PORTS_CONTINUE = NAME

a block might compare even though there are property or compare errors in the child blocks. In this situation, you see ~= instead of == for those child blocks, and the status of contingency is propagate up to the top. As a result, you see ~= below the block FAIL in the LVS error file.

```
------------------------------------------------------------
Top block compare result: FAIL

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

                      [DAC95 ~= DAC96]

------------------------------------------------------------
Comparison summary

    83 successful equivalencies
  *  1 failed equivalencies
       1 first priority errors
       0 second priority errors
```

There are two major sections following the PASS/FAIL result: First Priority Errors and Second Priority Errors.

- The First Priority Error section lists the failed equivalent points and their errors. Those failed equivalent points are most likely the source of mismatch. Start your debug from these points. Hercules gives more detailed diagnoses for the first priority errors than for other errors.
- The Second Priority Error section lists the failed equivalence points that are either caused by failure in its children blocks or are less important after the post analysis. Only a brief summary of unmatched devices and nets are listed for the equivalence points in this section.

First Priority Errors—In the following example, the failed equivalence point IOBUF is shown. The schematic name is always listed on the left-hand side while layout name is listed on the right-hand side. In this example, iobuf is the schematic cell name and IOBUF is the layout cell name. First a summary of the number of unmatched devices and nets is listed. Then, a detailed post-compare summary lists each type of device is listed.

```
+------------------------------------------------------------+
|                     First Priority Errors                  |
+------------------------------------------------------------+
```

```
First priority errors are equivalence points that Hercules has
determined probably contain errors that need to be fixed first.

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

> iobuf  != IOBUF (level = 2)

Summary:

           0 unmatched schematic device
           0 unmatched schematic net
           0 unmatched layout device
           0 unmatched layout net

          15 matched devices
          18 matched nets

(CMP-13) Post-compare summary (* = unmatched devices or nets):

      Matched    Schematic   Layout      Instance types
                 unmatched   unmatched   [schematic, layout]
     ---------   ---------   ---------   --------------------
             5           0           0   [INVA, inva]
             4           0           0   [N, n]
             1           0           0   [NDIO, ndio]
             4           0           0   [P, p]
             1           0           0   [RP1, rp]
     ---------   ---------   ---------   --------------------
            15           0           0   Total instances

            18           0           0   Total nets
```

To help in LVS debugging, Hercules analyzes unmatched nets and devices, and recognizes certain types of circuit errors with which it generates the diagnoses. The following types of diagnoses are reported:

- Extra or Missing Layout Devices or Nets
- Short or Open Layout Nets
- Potential Short/Open Source Groups
- Unmatched Nets with Suspicious Connection Groups
- Pins Swapped on Instances
- Potential Duplicate Equivalence Point Sets
- Filterable Instances
- Possible Falsely Filtered Devices

- Unmatched Net or Device Groups
- Suspect Wrong Connections Found by PUSH_DOWN_PINS

The following sections give an explanation for each message type and an example.

Extra or Missing Devices or Nets

There is a missing or extra device, or net, in the layout netlist. The assumption is that the schematic netlist is correct (golden).

There are four message versions:

- Missing Layout Nets
- Extra Layout Nets
- Missing Layout Instances
- Extra Layout Instances

For example:

```
DIAGNOSTIC: Missing Layout Instances

   The following schematic instances are missing in the layout netlist.

     Schematic instance (type)       Layout instance
     -------------------------       --------------------------
     m4 (n1)                         * Missing instance
```

Short or Open Layout Nets

There is a short or open net in the layout. The schematic netlist is treated as being correct for deciding whether it is a short or an open. In the report the unmatched nets are grouped for analysis. The connection instances for each unmatched net are listed, as well.

There are two message versions:

- Shorted Layout Nets
- Open Layout Nets

For example:

```
DIAGNOSTIC: Open Layout Nets

    The following unmatched nets are highly suspected to indicate
    source of opens in the layout netlist.

    Group 1 of 1:

        Schematic net : connection   Layout net : connection
        --------------------------   ------------------------
```

```
        VDD : 3                         ii : 1
                                        VDD : 2

    1 matched device connected to the open net ii

-------------------------------------------------------------------------

    Instance 1 of 1:

        Instance          Schematic                   Layout
        --------------    ------------------------    ------------------
        Instance type     NOR                         NOR
        Instance name     x105                        x105

        Class             Schematic net (port)        Layout net (port)
        --------------    ------------------------    ------------------
        1                 VDD (VDD)                   * ii (VDD)

    2 matched devices connected to the open net VDD

-------------------------------------------------------------------------

    Instance 1 of 2:

        Instance          Schematic                   Layout
        --------------    ------------------------    ------------------
        Instance type     NAND                        NAND
        Instance name     x101                        x101

        Class             Schematic net (port)        Layout net (port)
        --------------    ------------------------    ------------------
        2                 VDD (VDD)                   * VDD (VDD)

    Instance 2 of 2:

        Instance          Schematic                   Layout
        --------------    ------------------------    ------------------
        Instance type     NAND                        NAND
        Instance name     x102                        x102

        Class             Schematic net (port)        Layout net (port)
        --------------    ------------------------    ------------------
        2                 VDD (VDD)                   * VDD (VDD)
```

Potential Short/Open Net Groups

One or more groups of layout and schematic nets are potential short or open sources, except for some suspicious connections.

Net pairs are reported with schematic information on the left and layout on the right. The number after the colon (:) is the total connection count of the net. After net pairs, there is a list of matched schematic and layout devices.

For example:

```
DIAGNOSTIC: Potential Short/Open Source Group

    The following unmatched nets are potential short/open sources, except
    for some suspicious connections. Related devices are listed as two
    groups, matched and unmatched.

    Schematic net : connection    Layout net : connection
    --------------------------    ----------------------
    DATA9 : 8                     52/DATAOUT9 : 16
    DATA29 : 8
```

Note:

Matched devices connected to unmatched nets are listed immediately after the report.

Unmatched Nets with Suspicious Connection Groups

A pair of layout and schematic nets should match because most of their connections match. The analysis reports the unmatched, and possibly wrongly connected, neighbor devices between the two nets. These are the places to examine.

Net pairs are reported with the schematic on the left and layout on the right. The number after the colon (:) is the total connection count of the net. After the net pairs, there is a list of schematic and layout devices.

For example:

```
DIAGNOSTIC: Unmatched Nets With Suspicious Connections

     The following pairs of nets are apparently supposed to be the same
net
     except for some suspicious connections to devices listed below.

   Net 1/1

     Schematic net : connection             Layout net : connection
     --------------------------             -------------------------
     $1N104        : 60                     XX_281     : 65

   35 matched devices connected to unmatched net.
   ------------------------------------------
   NOTE: Pins with the same class value are logically interchangeable.

   Instance 1/35

     Instance       Schematic               Layout (1507.000, -398.000)
     -------------  ----------------------  ---------------------------
     Instance type  ADR3DEC8/ADR2DEC4/AND3  adr3dec8/and3
     Instance name  $1I31/$1I2/$1I4         adr3dec8_128/and3_50
     Properties     N/A                     N/A
```

```
     Class          Schematic net (port)    Layout net (port)
     -------------  ----------------------  -------------------------
     4              $1N104 (C)              * XX_281 (B)
```

Pins Swapped on Instances

The pins of the instances in the layout are swapped. Hercules can determine a possible way to modify the layout netlist to match the schematic, but it may not be the best solution. Normally, the unmatched nets diagnosis follows immediately. To help you debug the problem, it lists the connections for those nets.

For example:

```
DIAGNOSTIC: Pins Swapped On Instance

   The pins of the following instances in layout are swapped. One
possible
   way to modify the layout netlist is shown below.(+ = modified net)

     Instance 1 of 1:

        Name (type)
        ---------------------
        x1 (AOI)

        Class          Layout net (port)          Suggested connection:
        ------------   ------------------------  ---------------------
        4               b (b)                       + a
        4               d (d)                       + b
        4               a (a)                       + d

DIAGNOSTIC: Unmatched Nets

     Unmatched schematic and layout nets are grouped for cross probing.

     Group 1 of 2:

     Schematic net : connection                Layout net : connection
     ------------------------------            --------------------
     a : 2                                      a : 2
```

Potential Duplicate Equivalence Point Sets

There are possibly unknown duplicate equivalent points. This is reported when Hercules detects that two or more equivalence point cells have the same number of devices and nets. They are possibly swapped in layout. In that case, a warning is issued to run with `FIND_DUPLICATE_EQUIVS=TRUE`. With this option, Hercules checks for duplicate equivalences by comparing them, then uses this information for LVS comparison. With this option, swapped duplicate equivalences in the layout do not give LVS errors since the netlists are effectively equivalent.

For example:

```
DIAGNOSTIC: Potential duplicate equivalence points.

    Hercules has detected that some equivalence points with the same
    number of devices and nets are possibly swapped in the design.
    These cells are listed below in [schematic, layout] format:

  Set ID    Equivalence point set
  ------    ---------------------------------
       1    [BUF3, BUF3] [BUF2, BUF2] [BUF1, BUF1]
       2    [SLOWINV, SLOWINV] [INV2, INV2] [INV3, INV3] [INV4, INV4]
       3    [PDETX, PDETX] [PDETXB, PDETXB]
       4    [DFFS, DFFS] [DFFSF, DFFSF]
       5    [DFFR, DFFR] [DFFRP2, DFFRP2] [DFFRF, DFFRF]
```

Running with `FIND_DUPLICATE_EQUIVS=TRUE` resolves this type of LVS mismatch.

Filterable Instances

Unmatched instances exist in the schematic or layout netlist that are filterable by using FILTER_OPTIONS rules or FILTER_SCHEMATIC_OPTIONS and FILTER_LAYOUT_OPTIONS rules.

For example:

```
DIAGNOSTIC: Filterable Schematic Instances:

    The following unmatched schematic instances can be filtered out by
    using FILTER_OPTIONS or FILTER_SCHEMATIC_OPTIONS rules. Each
    instance and the corresponding rule(s) have been put together for
    your easy reference.

    You can set the FILTER_OPTIONS or FILTER_SCHEMATIC_OPTIONS of EQUATE
    commands in the runset file.

    Instance 1 of 2:

      Instance     Schematic
      -----------  -------------------------
      Name (type)  Parallel#32 (MP), n = MP, Length = 0.55, Width = 680

      Pin          VDD = GATE
                   VDD = SRC
                   PAD = DRN
                   VDD = BULK

      This instance can be filtered by one of the following rule(s)
      ------------------------------------------------------------
      PMOS-16 : Filters devices when gate and bulk pins are tied to the
      same net
      Filterable by PMOS-3 if the bulk net is specified as a power net.

    Instance 2 of 2:
```

```
Instance     Schematic
-----------  -------------------------
Name (type)  Parallel#34 (MP), n = MP, Length = 0.55, Width = 80

Pin          VDD = GATE
             VDD = SRC
             I = DRN
             VDD = BULK

This instance can be filtered by one of the following rule(s)
-------------------------------------------------------------
PMOS-16 : Filters devices when gate and bulk pins are tied to the
same net
Filterable by PMOS-3 if the bulk net is specified as a power net.
```

Possible Falsely Filtered Devices

There are unmatched devices that were filtered out by corresponding filter rules. The design may compare if these devices are not filtered. The pins causing false filtering are highlighted to facilitate debugging.

For example:

```
DIAGNOSTIC: Possible Falsely Filtered Devices in Layout:

    The following unmatched instances are filtered out by the following
    rules. Each instance and the corresponding rule(s) are put together
for
    easy reference.

    The pins causing the instance to be filtered are marked with +

    Instance 1:

      Instance          Schematic      Layout (1135.950, 368.000)
      ---------------   ------------   ------------------------
      Instance type     N              PAD_BYPASS/DRV_W_ESD/N
      Instance name     Parallel#79    286/5/M1

      Instance connections
      Class             Schematic net (port)       Layout net (port)
      --------------    ----------------------     -------------------
      4               * VSS_RING (GATE)          + GND (GATE)
      1               * VSS_RING (SRC)           + GND (SRC)
      1               * GND (DRN)                + GND (DRN)
      5               * VSS_RING (BULK)          + GND (BULK)

      This instance is being filtered by one of the following rule(s)
      ---------------------------------------------------------------
      NMOS-5  : Filters devices when source and drain pins are shorted
```

Unmatched Net or Device Groups

Hercules is not able to classify some unmatched nets and devices into any of the previous types of circuit errors. These devices and nets are partitioned into groups. Each group contains a number of unmatched devices or nets (but not both) in layout and schematic that have the same matched neighbors (devices and nets) in the net. It is highly likely that members of each group match each other between schematic and layout netlist.

The matched devices connected to an unmatched nets group are also listed, for ease in debugging. For each pair of matched devices, the first line lists the schematic instance name and device type. The second line lists the layout instance name and device type. The remaining lines are side-by-side listings of pin connections for both the schematic (left) and the layout (right). Each of the pin connection lines is prefaced with the class of the port. These class identifiers can be used to determine net connections that can be swapped without changing functional behavior of the block.

Unmatched nets are highlighted with an asterisk (*) on the layout side in the report. The instance name of a device may be a path to a hierarchical device with separators (/) if subblocks were flattened. Because the path is generated from instance names, it is always unique. By default, only the pins connected to the unmatched nets are listed for each instance. You can set `LVS_REPORT_LEVEL = {XPIN-2}` in the OPTIONS section to report all pins of the devices connected to the unmatched nets. See the *Hercules Reference Manual*, Detailed Options chapter, LVS_REPORT_LEVEL section. You can also check the corresponding summary file (sum.*block*.*block*), which has a complete pin listing.

For example:

```
DIAGNOSTIC: Unmatched Nets

   Unmatched schematic and layout nets are grouped for cross probing.

   Group 1 of 2:

   Schematic net : connection                  Layout net : connection
   ------------------------------              --------------------
   QN : 4                                      QN : 5

   4 matched devices connected to unmatched nets (* = unmatched net)
   ---------------------------------------------------------------

   Instance 1 of 4:

     Instance        Schematic                 Layout (7.500, 35.750)
     --------------  ------------------------  ----------------------
     Instance type   P                         p
     Instance name   $1I4                      M5
     Properties      Length = 1, Width = 20.5  Length = 1, Width = 20.5

     Class           Schematic net (port)      Layout net (port)
```

```
-------------  ------------------------  ----------------------
5              VDD (VBB)                 * VDD (BULK)
1              VDD (D)                   * QN (DRN)
1              QN (S)                    * VDD (SRC)
```

Suspect Wrong Connections Found by PUSH_DOWN_PINS

Some suspect connections are found by the PUSH_DOWN_PINS option. The suspect wrong connections are found among some matched instance pairs between schematic and layout. The instances are low-level equivalence points and are treated as devices after a successful matching. All ports of the cell are connected across all instances in schematic/ layout side, but are not in the other sides. This message indicates some possible shorts and opens, especially for global nets.

The following example shows that layout instance 2124/2650/3059 has open nets 2428 and VDD.

```
DIAGNOSTIC: Suspect wrong connections found by push_down_pins

    Port {VDD XOVERDIS} are connected across all instances of
AR_PREDECODE
    in schematic, but not in layout.

      Suspect Connections around Matched Device Pairs
      -----------------------------------------------

      Instance       Schematic                Layout (41.069, 1112.850)
      -------------  ---------------------    ------------------------
      Instance type  AR_BLK_COL/AR_BLK/AR_PRE ar_blk_col_red/ar_blk_red
                                              DECODE
      Instance name  XUBC/XUBS/XUPD           2124/2650/3059

      Class          Schematic net (port)     Layout net (port)
      -------------  ----------------------   ------------------------
      8              VDD (XOVERDIS)         * 2428 (SPBLKEN)
      10             VDD (VDD)              * VDD (VDD)
```

The LVS Log FIle

When you run LVS, all on-screen messages are logged to a log file for tracking purposes. The file (*block*_lvs.log) is located in the run_details/ directory. It keeps all the run time information, including execution time in each stage, pass/fail result for each equivalent point, the option settings in the runset, the runtime environment, and the starting and ending time. The following example is from a log file.

Example 5-9 block_lvs.log file

```
HLVS (R) Hierarchical Layout Versus Schematic, Release 2006.12.0045
2006/08/11
```

```
  (C) Copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004.
  Synopsys, Inc.  All rights reserved.

Call as:
/remote/code/Hercules/2006.12/build/arch/x86_64-amd-linux2.4/OPT/bin/
lsh dac96lvs4b

Hercules LVS compare start time    : 2006-08-11 18:28:51

+-------------------------------------------------------------------+
|                          Environment Status                       |
+-------------------------------------------------------------------+

Hostname            = tphxa6
Platform type       = AMD64_L24
Runset file         = dac96lvs4b.ev
Working directory   = /remote/SEG_sys/larryke/hercules/example/tutorial
/Getting_Started_Hercules_LVS/dac96lvs4
Top block           = DAC96
Layout netlist      = DAC96.net
Schematic netlist   = dac96.hrc
Equivalence file    = dac96.eqv
Compare directory   = compare

+-------------------------------------------------------------------+
|                            Options Listing                        |
|                * = different from Hercules default                |
+-------------------------------------------------------------------+

    == Compare options ==

    add_width_for_parallel_merge   = FALSE
    all_ports_texted               = FALSE
    auto_exclude_equiv             = TRUE
    combine_output_files           = FALSE
  * compare_properties             = TRUE
    delete_lay                     = [none]
    delete_sch                     = [none]
  * detect_permutable_ports        = TRUE
    equate_by_device_name          = FALSE
  * equate_by_net_name             = TRUE
    equiv_by_name                  = FALSE
. . .

+-------------------------------------------------------------------+
|                        Preprocessing Stage                        |
+-------------------------------------------------------------------+

Purging compare directory ... OK
Reading schematic netlist ...
    Generating EVaccess netlist data ...
Reading schematic time = 0:00:00 User=0.02 Sys=0.02 Mem=17.608
Reading layout netlist ...
```

```
    Generating EVaccess netlist data ...
Reading layout time = 0:00:01 User=0.02 Sys=0.03 Mem=23.920
Processing schematic netlist ...
    Propagating schematic globals ...
    Propagating schematic globals time = 0:00:00 User=0.01 Sys=0.00
Mem=25.982
Processing schematic time = 0:00:00 User=0.01 Sys=0.00 Mem=26.990
```

Understanding the Equivalence Point Summary File

To make design compares, typically you only need to use the LVS error file, in which Hercules supplies a detailed summary file for each equivalence point. The file is located in the directory run_details/compare/layout_cell_name and is named sum.schematic_cell_name.layout_cell_name, where layout_cell_name and schematic_cell_name are the names of the equivalence point on the layout and schematic side.

You can easily find the corresponding summary file by following the links in the LVS error file for each equivalence point. For example:

```
For details, please refer to compare/and2_ga/sum.AND2B.and2_ga
```

Most of the information is the same as in the LVS error file. However, there is additional information included in this file, such as a cross-reference port table, merge node information, and merge/filtering statistics.

Below are some examples of the additional tables in the summary file.

Netlist Statistics tables—Example 5-10 shows the section that contains the Netlist Statistics tables for both schematic and layout. These list the total number of devices and nets in a block both prior to and after filtering and merging. In each table:

- The first part is the device data.
  - The far right column lists the names of devices and sub-blocks referenced by the block being verified.
  - The column labeled Initial lists the beginning number of each device.
  - The PushDown, Filter, Parallel, and Path/Ser columns list the number of instances created (positive) or destroyed (negative) by Pushdown, Filtering, Parallel Merging, and Path/Series Merging.
  - Each row of the Final column is the sum of the first four columns.
  - The bottom row summarizes the results for all device types.

- The second part is the net data, similar in format to the device data. The PushDown, Dangle, 0 Connect, and Path/Ser columns list the number of nets removed by Pushdown, Remove Dangling Net, Zero Connections, and Path/Serial Merging.
- The last part is the post-merge netlist statistics section that contains a side-by-side listing of the number of devices and nets after filtering and merging. The devices are categorized by equivalence pairs. Use this table to quickly determine obvious differences in the number of nets or a particular device type.

Example 5-10 Netlist Statistics Table

```
+-------------------------------------------------------------------+
|                        Statistics Report                          |
+-------------------------------------------------------------------+

Schematic netlist statistics after filtering and merging

   Initial PushDown Filter Parallel Path/Ser  Final   Device type
   ------- -------- ------ -------- --------  ------- ---------------
         1        0      0        0        0        1 INVB
         1        0      0        0        0        1 NAND2B
   ------- -------- ------ -------- --------  ------- ---------------
         2        0      0        0        0        2 Total devices

   Initial PushDown  Dangle 0 Connect Path/Ser  Final    Net type
   ------- --------  ------ --------- --------  -------  -----------
         6         0      0         0        0        6  Total nets

Layout netlist statistics after filtering and merging

   Initial PushDown  Filter  Parallel  Path/Ser  Final    Device type
   ------- --------  ------  --------  --------  -------  -----------
         1        0       0         0         0        1  inv_gacell
         1        0       0         0         0        1  nand2_gacell
         1        0       0         0         0        1  p
   ------- --------  ------  --------  --------  -------  ------------
         3        0       0         0         0        3  Total devices

   Initial PushDown  Dangle  0 Connect Path/Ser  Final   Net type
   ------- --------  ------  --------  --------  ------  ------------
         8        0      -2         0         0       6  Total nets

Post merge netlist statistics: (* = different count)

   Schematic       Layout          Device type [schematic, layout]
   ---------       ---------       -------------------------------
           1               1       [INVB, inv_gacell]
           0               0       [N, n]
           1               1       [NAND2B, nand2_gacell]
           0               1     * [P, p]
   ---------       ---------       -------------------------------
           2               3     * Total devices
```

```
           6               6       Total nets
```

Summary of Equated Nets—This section summarizes nets which have been equated either with EQUATE_BY_NET_NAME or with EQUATE_NETS. This listing is sorted alphabetically. For example:

```
Summary of Equated Nets:
 GND == GND
 VDD == VDD
 net_1 == net_1
```

Blocks with Symmetrical Circuitry—Example 5-11shows the section that is generated for blocks which contain symmetrical circuitry. The table lists the number of devices and nets which remain after the COMPARE process has matched as much unique circuitry as possible. In order to allow further progress, a match must be assumed from similar groups of devices and/or nets. If similar groups of schematic ports exist prior to the first assumed match, they are listed at the end of this section. This listing is potentially important for understanding which groups of ports may be swappable (see previous section “Permutable Ports for LVS COMPARE” on page 5-36 for more information on symmetry).

Example 5-11 Blocks with Symmetrical Circuitry

```
These circuits contain some symmetry (42.86%)
Number of nodes not yet matched:

Schematic   Layout
---------   ------
0           1       Total Devices
3           5       Total Nets

Attempting to find a valid match for symmetrical nodes.
WARNING: A match must be assumed from similar nodes to allow
further progress.
WARNING: Symmetry may allow multiple "correct" matchings of I/O.
   The following groups of schematic I/O are not yet matched:
      I/O Group 1:
         IN1
         IN2
WARNING: Assumed net match for schematic "IN2" and layout "B"
WARNING: Deduced net match for schematic "IN1" and layout "A"
```

Post-Compare Netlist Statistics—Shows the post-Compare data table that lists the numbers of matched and unmatched devices and nets. The top of the table is divided based on equivalence pairs. Once again, note that it is necessary to add up the rows listing multiple equivalence for an accurate count.

Example 5-12 Post-Compare Netlist Statistics

```
Post-Compare Netlist Statistics:

Schematic     Layout      Schematic     Layout
Matched       Matched     Unmatched     Unmatched
```

```
---------    -------    ---------    ---------
0            0          0            0          [n, n]
0            0          0            1          [P, P]
1            1          0            0          [n--2, n--2]
1            1          0            0          [P||2, P||2]
---------    -------    ---------    ---------
2            2          0            1          Total Devices

4            4          1            3          Total Nets

WARNING: 1 matches were assumed based on symmetry.
```

Relationships Between the Schematic and Layout—Example 5-13 shows the section that details the port (I/O) relationships between the schematic and layout. All of the matched ports are listed in the far right column; the schematic port name followed by the layout port it matches. The port class of each pair is listed, with independently swappable ports, receiving the same port class. If any port net in either the schematic or layout is matched to a non-port net, then that non-port net becomes a generated port. Generated ports in both the layout and schematic are designated in the appropriate column.

Example 5-13 Relationships Between the Schematic and Layout

```
Writing cross-reference entries for I/O nets ...

Port Cross-Reference Table:

Generated     Generated     Port      Sch Net /
Sch Port      Lay Port      Class     Lay Net
---------     ---------     ------    ---------
                            1         GND / GND
                            2         VDD / VDD
              *             3         IN1 / A
              *             3         IN2 / B

4 entries written to "compare/NAND/xrf.NAND.NAND"
```

Property Differences—If property checking is enabled and there are property differences between the schematic and layout devices, the information shown in is generated. The percent difference listed is relative to the schematic device value. A positive difference indicates that the layout property is too large, while a negative difference indicates that the layout property is too small. The property will be shown as a list of multiple values if the device is a merged device where property values cannot be merged into a single value. In this case, the percent difference represents the maximum absolute difference between pairs from the schematic and layout property value lists. For example:

```
Comparing device properties ...

ERROR: Property Width mismatch (50.00%):
   Schematic: {inst SerChain#0=n--2 {prop }}
      Width = (6 6)
   Layout : {inst SerChain#1=n--2 {prop }}
```

```
      Width = (6 9)
```

Cross Reference Listing for Merged Devices—The section below is a cross reference listing for Merged devices. It provides a way to determine the components of a merged device. For each merged device previously referenced in the summary file, it lists the member instances. Note that a merged device may in fact be composed of other merged devices.

Example 5-14 Cross Reference Listing for Merged Devices

```
Schematic Merged Devices Previously Referenced:

   ParChain#0 {inst M1=P {prop n="P", x=25.000, y=32.500}}
   {inst M2=P {prop n="P", x=12.500, y=32.500}}

   SerChain#0 {inst M4=n {prop n="N", x=125.500, y=10.500}}
   {inst M3=n {prop n="N", x=25.500, y=10.500}}

Layout Merged Devices Previously Referenced:

   ParChain#1 {inst M1=P {prop n="P", x=25.000, y=32.500}}
   {inst M2=P {prop n="P", x=12.500, y=32.500}}

   SerChain#1 {inst M5=n {prop n="N", x=125.500, y=10.500}}
   {inst M4=n {prop n="N", x=25.500, y=10.500}}

Elapsed time = 0:00:00 User=0.1 System=0.0 Faults=0 Swaps=0 IO=2
```

Controllable Message Descriptions in LVS Compare

LVS compare messages are separated into MSG, ERR, FATAL, and WARN severity. You can control the printing of these messages during the compare process. At times you may want to reduce the size of output files by switching off particular types of messages. Normally, these are warnings that you understand or data that is not generally referenced. In other instances, you may want to turn on other messages to produce more verbose output.

- MESSAGE_SUPPRESS command in the OPTIONS section: Used to disable the printing of MSG and WARN messages; it does *not* work for FATAL and ERR messages.
- MESSAGE_ENABLE command in the OPTIONS section: Used to enable the printing of MSG and WARN messages; it does *not* work for FATAL and ERR messages.
- MESSAGE_IDENTIFIERS option: Set to display the message identifiers in the listing file.
- MESSAGE_ERROR option: Set to upgrade a message to a severity level of WARN messages to ERR.

The following are examples of message control:

```
OPTIONS {
   message_suppress = { CMP-39 CMP-45 }  /* Do not print
```

```
                                                these messages. */
   message_enable = { CMP-60, CMP-61 }    /* Do print these
                                                messages */
   message_identifiers = on               /* Prints message IDs
                                                in output files. */
   message_error = { CMP-75 }             /* Upgrade message(s)
                                                level to error. */
}
```

The following table describes the controllable messages that can appear in *block*_lvs.log, *block*.LVS_ERRORS file, and sum.*block*.*block* files.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-4	ON	FATAL	Cannot open netlist file "%s" for writing.	The specified equivalence netlist file could not be opened for write access.
CMP-5	ON	ERR	Cannot open database file "%s" for writing.	Cannot open cross reference database file for writing.
CMP-7	ON	WARN	Top Block "%s" is part of a multi-equiv; one member has been chosen.	The specified top block is part of a multi-equiv. One member of that multi-equiv will be arbitrarily chosen as the top level equivalence point.
CMP-8	OFF	WARN	%d matches were assumed based on symmetry.	The specified number of matches (devices and/or nets) were assumed in order to allow progress for symmetric portions of the circuitry. This message is only printed if any matches were assigned.
CMP-9	OFF	MSG	All devices and nets were matched in %d passes.	The Compare process is iterative in the mechanism used to match devices and nets. A high number of passed usually indicates a block which contains a high percentage of symmetric circuitry.
CMP-10	ON	MSG	Some devices and nets remain after matching unique elements (%.2f%%)	For circuits with symmetric circuitry, this message lists the percentage of devices and nets remaining after all unique matches were found. Subsequent matches will require one or more assumed matches to allow further matching progress.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-11	ON	MSG	%s Netlist Statistics for "%s":	This table lists the individual and total counts for devices and nets in a netlist. The number of devices and nets created or removed by merging and filtering is also shown.
CMP-12	ON	MSG	Post-Merge Netlist Statistics:	This table is a side-by-side listing of the final device and net counts after all merging and filtering operations were completed.
CMP-13	ON	MSG	Post-Compare Netlist Statistics:	This table summarizes the number of matched and unmatched devices and nets for the schematic and layout blocks.
CMP-14	ON	WARN	Ignoring equate-by-name for "%s" - %d (S) vs. %d (L) connections	The specified net name, found in both the schematic and layout, was not used as an initial match reference point because the number of connections to the net does not agree between the two blocks. Nets with different number of connections can never match.
CMP-15	ON	WARN	Invalid %snet "%s" in %s listing - ignoring entry	The specified schematic net name used in an EQUATE_NETS equivalence option could not be found in the schematic netlist. The name was probably entered incorrectly.
CMP-16	ON	WARN	Invalid %s device "%s" in %s listing - ignoring entry	The specified layout net name used in an EQUATE_NETS equivalence option could not be found in the layout netlist. The name was probably entered incorrectly.
CMP-17	ON	WARN	Ignoring equate for schematic net "%s" and layout net "%s" - %d vs. %d connections	The specified schematic and layout nets found in an EQUATE_NETS equivalence option was not used as an initial match reference point because the number of connections to the nets does not agree between the two blocks. Nets with different number of connections can never match.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-18	ON	WARN	Ignoring additional EQUATE_NETS entry for %s net "%s"	There is more than one EQUATE_BY_NET_NAME or EQUATE_NETS possibility for the specified schematic net. The first valid equate is used, and subsequent equates are ignored.
CMP-19	ON	WARN	Ignoring additional EQUATE_DEVICES entry for %s device "%s"	There is more than one EQUATE_BY_NET_NAME or EQUATE_NETS possibility for the specified layout net. The first valid equate is used, and subsequent equates are ignored.
CMP-20	ON	MSG	Summary of Equated Nets:	This table lists in alphabetical order the net names used as initial match reference points as specified by the EQUATE_BY_NET_NAME and EQUATE_NETS equivalence options.
CMP-21	ON	WARN	Device "%s" filtered from schematic netlist.	This is a warning message to indicate that the specified instance of a schematic device was filtered. While layout devices are commonly filtered, it is a more unusual occurrence for a schematic device to be filtered.
CMP-22	ON	MSG	Filtering %s unused devices ...	This message is printed prior to the filtering operations. The number of devices removed is printed after the operation is complete.
CMP-23	ON	MSG	Merging %s parallel devices ...	This message is printed prior to the parallel merging operations. The number of devices removed is printed after the operation is complete.
CMP-24	ON	MSG	Merging %s series devices ...	This message is printed prior to the series merging operations. The number of devices removed is printed after the operation is complete.
CMP-25	ON	MSG	Merging %s path devices ...	This message is printed prior to the path merging operations. The number of devices removed is printed after the operation is complete.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-26	ON	MSG	Merging %s parallel chain devices ...	
CMP-27	OFF	WARN	A match must be assumed from similar devices and/or nets to allow further progress.	Symmetry in the circuits requires that an assumed match be made between two devices or nets. This only occurs after all possible matches of unique devices and nets have been completed.
CMP-28	ON	WARN	Symmetry may allow multiple correct matchings of ports.	For circuits with symmetry, an assumed match between devices or nets must often be made. Prior to making the first assumed match, the status of the port nets of the schematic block are checked. If there are ports which are not yet matched, and appear to be similar to one another, they are listed in groups. It is possible that the ports within each group are logically interchangeable or swappable. Refer to the Symmetry / Port Swapping section for potential implications and alternatives.
CMP-29	ON	WARN	%s device "%s" has no connection to pin "%s"	The indicated device instance has no net connection specified in the netlist for the specified pin. As a result, the Compare process automatically assigns a net name beginning with the prefix No Connect#, and followed by a unique positive integer.
CMP-31	ON	WARN	%s net "%s" has 1 connection and is not a port.	The indicated net has only 1 connection, and the net is not a port of the block. This indicates that the net is dangling, since it only connects to a single device pin.
CMP-33	ON	WARN	%s net "%s" has 0 connections.	The indicated net has no connections to devices in the block. This occurs for nets which are declared as ports, but do not connect to any devices in the netlist description of the block.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-36	ON	WARN	No Check_Properties command in Equate{} for device type "%s"	This warning message is printed for any device type which has no properties specified for checking, even though property checking has been enabled for the Compare process. This may indicate an oversight by the user.
CMP-37	ON	MSG	Comparing device properties ...	This header message is printed at the beginning of device property checking during the Compare process. Properties are only checked for matched devices.
CMP-39	ON	MSG	Writing cross-reference entries for port nets ...	In order to check correct connectivity to references of a block higher in the design hierarchy, the Compare process maintains a cross reference of port nets between the schematic and layout blocks.
CMP-40	ON	WARN	Port net "%s" in schematic equates to non-port net "%s" in layout.	This warning message indicates that a port net in the schematic block was matched to a non-port net in the layout block. This could indicate a missing layout connection, since hierarchical references to the layout block will always have a dangling connection for the specified net name.
CMP-41	ON	WARN	Port net "%s" in layout equates to non-port net "%s" in schematic.	This warning message indicates that a port net in the layout block was matched to a non-port net in the schematic block. This could indicate a shorted layout connection, since hierarchical references to the schematic block will always have a dangling connection for the specified net name.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-42	ON	WARN	Schematic net "%s" does not equate to layout net with same name.	This warning message indicates that a schematic net and layout net with the same names were matched, but not to each other. Normally, it would be expected that text in the layout database would match with corresponding names in the schematic database.
CMP-43	ON	WARN	Layout net "%s" does not equate to schematic net with same name.	This warning message indicates that a schematic net and layout net with the same names were matched, but not to each other. Normally, it would be expected that text in the layout database would match with corresponding names in the schematic database.
CMP-44	ON	WARN	Matched devices connected to unmatched nets.	This heading and accompanying messages list matched devices connected to unmatched schematic and/or layout nets. The side-by-side device listings are included to facilitate the debug of unmatched blocks.
CMP-45	ON	WARN	Port "%s" on cell "%s" is unconnected for instance	The specified instance of a cell does not have a connection listed for a particular port.
CMP-47	ON	MSG	%s Merged Devices Previously Referenced:	Many messages will refer to device instances which do not exist in the original netlist description of a block. These devices are created as a result of merging operations. Any of these created devices which are referenced earlier in the equivalence summary file are listed in terms of their component members. These listings can be used to determine which physical devices have been used to formulate a merged or composite device.
CMP-54	ON	FATAL	Cannot open EQUIVALENCE file "%s" for reading.	This specified equivalence file could not be opened for read access.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-55	ON	FATAL	%s at line %d in EQUIVALENCE file "%s" near token : %s	The equivalence file contains a syntax error at the specified line. This error must be resolved before Compare can run.
CMP-56	ON	MSG	Port Cross-Reference Table:	This table lists all the ports in a matched block. Any ports generated in either the schematic or layout are marked as such.
CMP-57	OFF	WARN	Assumed %s match for schematic "%s" and layout "%s"	Because of symmetry in the design, Compare was forced to assume a particular match between nodes in the schematic and layout.
CMP-59	OFF	MSG	%s filtered device list:	This table lists all the filtered devices in both the schematic and layout netlists. By default, this table is not printed.
CMP-60	OFF	MSG	%s parallel merged device list:	This table lists all the parallel merged devices in both the schematic and layout netlists. By default, this table is not printed.
CMP-61	OFF	MSG	%s parallel chain merged device list:	This table lists all the parallel chain merged devices in both the schematic and layout netlists. By default, this table is not printed.
CMP-62	OFF	MSG	%s %s merged device list:	This table lists all the series and path merged devices in both the schematic and layout netlists. By default, this table is not printed.
CMP-63	OFF	MSG	Net Cross-Reference Table (%s ports):	This table gives a complete cross-reference listing of the matched nets, except for ports. By default, this table is not printed.
CMP-64	OFF	MSG	Device Cross-Reference Table:	This table gives a complete cross-reference listing of the matched devices. By default, this table is not printed.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-66	ON	ERR	Duplicate schematic_swap_ports entry for "%s" in EQUIV %s=%s.	This error indicates that a single schematic port is specified as being part of more than one swappable I/O group. The SCHEMATIC_SWAPPABLE_PORTS statements in the equivalence file must be changed.
CMP-67	OFF	WARN	Swappable or permutable ports entry for "%s" not used.	This warning indicates that a SCHEMATIC_SWAPPABLE_PORTS was specified for a port that does not exist.
CMP-68	ON	WARN	Global net "%s" in schematic equates to non-port net "%s" in layout.	This warning message indicates that a port net in the schematic block was matched to a non-port net in the layout block. This could indicate a missing layout connection, since hierarchical references to the layout block will always have a dangling connection for the specified net name.
CMP-69	ON	ERR	No devices or nets to match	This error message indicates that this equivalence point contains no devices or nets.
CMP-70	ON	MSG	Processing %s netlist ...	This message indicates an initial hierarchical netlist analysis is being performed.
CMP-71	ON	MSG	Schematic = %s, Layout = %s, Level = %d	This message specifies which equivalence point is currently being compared, including the level of the equivalence point.
CMP-72	ON	ERR	Instance of '%s' in cell '%s' not defined in netlist and has no EQUATE entry in runset	This netlist error indicates that a cell has been referenced in the netlist that is not defined in the netlist. If the cell being referenced is a primitive extracted device, then an EQUATE must be provided for it in the runset.
CMP-73	ON	MSG	%s EQUATE device name identical to cell in netlist: "%s"	This error indicates that the device name in an EQUATE is identical to a cell defined in the specified netlist.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-74	ON	MSG	Ignoring invalid %s: %s=%s	This message lists all invalid equivalence points and gives the reason why each one was considered invalid.
CMP-75	ON	WARN	Duplicate equiv: %s=%s	This message indicates that one equivalence point was defined multiple times.
CMP-76	ON	ERR	Illegal M:N multi-equiv created by: %s=%s	This error message indicates that the specified equivalence point resulted in a multi-equiv on both the schematic and layout side. This conflict must be resolved for Compare to run.
CMP-78	ON	MSG	Purging Compare Directory ...	If the Compare option, RETAIN_PREVIOUS_DATA, is set to FALSE, all data in the Compare directory is removed.
CMP-79	ON	FATAL	'%s' not listed as a port for device '%s' in cell %s	This fatal error indicates that in an instance definition a port is referenced which does not exist in the cell definition.
CMP-80	ON	FATAL	Cannot open %s netlist "%s" for reading.	This fatal error indicates that the specific netlist could not be opened for read access.
CMP-81	ON	FATAL	No schematic netlist specified by runset or command-line option.	A schematic netlist must be specified either by runset or command line option for Compare to run.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-82	ON	MSG	The following equivalence points have been automatically excluded because there are a different number of instances between the schematic and layout, or because they are in the 'exclude_equiv' list for the current equiv point:	This message indicates that for the current equivalence point, information about certain child equivalence points is not being used because a different number of instances of the child equivalence point are defined between the schematic and the layout.
CMP-86	ON	ERR	Illegal M:N multi-equate created by: %s=%s	This error message indicates that the specified EQUATE resulted in a multi-equate on both the schematic and layout side. This conflict must be resolved for Compare to run.
CMP-88	ON	FATAL	Undefined/Illegal primitive "%s" referenced for device "%s".	A message will be printed if the device type is undefined or illegal.
CMP-89	ON	FATAL	Undefined/Illegal pin "%s" referenced by device "%s".	A message will be printed if the device pin is undefined or illegal.
CMP-90	ON	ERR	Cannot Create directory --> '%s'	This error message indicates that the specified directory could not be created. A permissions problem often can create this error.
CMP-91	ON	WARN	Black box duplicates an equiv: Equiv is being ignored: %s=%s	This warning indicates that the equivalence file contains an EQUIV entry and a BLACK_BOX entry that are identical. The EQUIV entry will be ignored.
CMP-92	ON	MSG	Equivalent blocks:	This message lists all matched blocks at this level or in the entire run.
CMP-93	ON	MSG	Non-equivalent blocks:	This message lists all unmatched blocks at this level or in the entire run.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-94	ON	MSG	Uncompared blocks due to dependency errors:	This message lists all uncompared blocks due to prior errors at this level or in the entire run.
CMP-95	ON	MSG	**** LEVEL %d ****	This is the level banner printed when a new level is begun.
CMP-96	ON	MSG	**** SUMMARY COMPARE RESULTS ****	This is the banner indicating that the following tables summarize the entire Compare run.
CMP-97	ON	MSG	This equivalence point has symmetry ports, matching can only be resolved in its parent. (%d%%,%d%%,%d %%)	The comparison process stops in this equivs. The process will continues at the parent.
CMP-98	ON	MSG	Clean-up all unresolved suspended equivalence points.	Output the results for unresolved or suspended equivalence points.
CMP-99	ON	ERR	Device count mismatch exceeds specified tolerance of %s.	This error message indicates that the device counts in the schematic and the layout differ by more than the tolerance specified in the runset.
CMP-100	ON	ERR	Net count mismatch exceeds specified tolerance of %s.	This error message indicates that the net counts in the schematic and the layout differ by more than the tolerance specified in the runset.
CMP-101	ON	FATAL	Parse error at or near line %d in file %s.	This error message indicates that a parser error was found in the specified netlist.
CMP-102	ON	MSG	Elapsed time = %s	This message reports the elapsed time individually for each equivalence point.
CMP-103	ON	WARN	Cannot compare due to errors in dependent blocks.	This message indicates that the current equivalence point cannot be compared because of errors in its children.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-104	ON	WARN	Multi-equiv with different port counts.	For any multi-equiv situation, each member must contain an identical port count. Any violation of this is a fatal error.
CMP-109	ON	MSG	%s Merged Nets Previously Referenced:	Many messages will refer to nets which do not exist in the original netlist description of a block. These nets are created as a result of merging operations. Any of these created nets which are referenced earlier in the equivalence summary file are listed in terms of their component members. These listings can be used to determine which physical nets have been used to formulate a merged or composite net.
CMP-110	ON	MSG	Compare messages printed:	This is a listing of all equivalence points and the listing is categorized by the compare results the equivalence points.
CMP-111	ON	MSG	Matched blocks containing blocks with comparison errors:	This is a listing of all equivalence points that contained errors but were used because the MATCHED_PORTS_CONTINUE option was set. All errors were internal to the block and all port nets were matched successfully.
CMP-112	ON	WARN	This equivalence point contains blocks with comparison errors.	This message indicates that the specified equivalence point contains instances of other equivalence points that passed comparison because the MATCHED_PORTS_CONTINUE option was set. Although the top block may compare, this warning conveys that the design still contains errors.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-113	ON	WARN	This equivalence point contains unmatched internal nodes, but comparison will continue because all port nets match.	This message indicates an equivalence point that contains errors, but all the port nets were matched and the MATCHED_PORTS_CONTINUE option was set. Comparison can continue in this situation because only port connections are checked at instances of this cell.
CMP-114	ON	WARN	Invalid ignore_equiv %s = %s	This message lists all invalid IGNORE_EQUIV equivalence file entries. These entries are invalid because they do not have a corresponding EQUIV entry.
CMP-115	ON	WARN	%s block performs series or path merging but contains no power or ground nets.	This message indicates any block in which series or path merging occurs but no power or ground rails exist as specified by SCHEMATIC_POWER, SCHEMATIC_GROUND. Such situations often result in excessive traversal of certain unspecified global nets.
CMP-116	ON	WARN	Messages of ERROR and FATAL severity cannot be suppressed.	This message is caused by an attempt to suppress an ERROR or FATAL message. Message suppression is valid only for the warning and miscellaneous information reported.
CMP-118	ON	WARN	Schematic nets matching layout nets with differing text.	This message is produced only when the REQUIRE_TEXTED_NETS_ MATCH option is set. Any texted nets in the layout that do not match nets in the schematic with identical text are flagged with this message.
CMP-119	ON	WARN	Schematic ports matching layout ports with differing text.	This message is produced only when the REQUIRE_TEXTED_PORTS_ MATCH or REQUIRE_TEXTED_ NETS_MATCH options are set. Any texted port nets in the layout that do not match port nets in the schematic with identical text are flagged with this message.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-120	ON	WARN	Net "%s" traversed more than %d times during merge.	This message indicates that the specified net has violated the lower limit specified by MERGE_NET_RANGE. All such nets should be checked to ensure they are not undesignated power or ground rails.
CMP-121	ON	ERR	Exceeded traversal limit of %d for net "%s". Consider using MERGE_PATHS_DEVICE_LIMIT to reject paths that create excessive numbers of devices. A value of 20 is often appropriate.	This message indicates that the specified net has violated the upper limit specified by MERGE_NET_RANGE. All such nets should be checked to ensure they are not undesignated power or ground rails. Also, consider using the MERGE_PATHS_DEVICE_LIMIT option.
CMP-122	ON	ERR	EQUATE %s=%s creates a multi-equate with conflicting device types.	This message indicates that the specified EQUATE results in the creation of a multi-equate where the members of the multi-equate are of different device types. This results in a FATAL error.
CMP-123	ON	WARN	Multiple tolerances specified for property '%s' of device %s=%s in EQUIV %s=%s. The last value is being used	If multiple tolerances are specified for one device in the equivalence file, all earlier tolerances are ignored and the last value is retained.
CMP-124	ON	WARN	Layout port nets without text.	This is the header for a list of all ports in the layout without text. This list is produced for violations to the ALL_PORTS_ TEXTED option.
CMP-125	ON	MSG	Deleted %s cell :	This message lists all the cells deleted from the netlists as specified by the DELETE_SCH and DELETE_LAY Compare options.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-126	ON	ERR	EQUATE %s=%s creates a multi-equate with differing pin definitions.	This message indicates that the mentioned EQUATE produces a multi-equate situation with inconsistent pin definitions. Multi-equate requires uniformity of pin descriptions for the device on the common side of the multi-equate.
CMP-127	ON	ERR	%s port "%s" does not have a corresponding port in the %s in BLACK_BOX %s=%s {}	This error message indicates that the given BLACK_BOX does not reference every port on the target cell. A BLACK_BOX entry must contain information about every port in the corresponding cell.
CMP-129	ON	ERR	Duplicate cell definition in %s netlist: "%s"	This error message indicates that the specified netlist contains multiple definitions or certain cells. Duplicate definitions are not allowed in the ISS netlist format.
CMP-131	ON	ERR	The %s netlist contains WIRE constructs; unwire must be run to resolve them.	An input netlist to Compare contains WIRE constructs. Compare does not handle these, and so unwire must be run to remove them.
CMP-132	ON	MSG	Summary of Equated Devices:	This table lists in alphabetical order the device names used as initial match reference points as specified by the EQUATE_BY_DEVICE_NAME and/or EQUATE_DEVICES equivalence options.
CMP-133	ON	ERR	%s port "%s" in black box "%s" corresponds to more than one %s port while push down pin option is turned off	This error message indicates that the mentioned ports has multiple entries in a BLACK_BOX. Entries are required for every port, but only one entry for every port is allowed.
CMP-134	ON	WARN	Unresolved parameters not able to be checked. They will be checked as they are resolved.	The unresolved parameters will be treated as a string value.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-135	ON	ERR	Braces "{ }" are now required around option lists in EQUIVALENCE file "%s". Violations are found at or near lines: %s.	Lists of elements in the equivalence file that can be of indeterminate length must now be enclosed in {}. For examples of the new syntax, see EQUATE_NETS and SCHEMATIC_SWAPPABLE_PORTS, or other EQUIV constructs.
CMP-136	ON	ERR	The filtering options for EQUATE %s=%s differ with those from a previous member of this multi-equate.	Elements within a multi-equate contain inconsistent filtering option. Every member must have identical filtering options.
CMP-137	ON	ERR	Limit of 20 user defined properties has been exceeded.	Any given Hercules run can only have 20 or fewer user-defined properties.
CMP-138	ON	FATAL	Cannot open SCHEME file "%s" for reading.	The specified SCHEME file could not be opened for read access.
CMP-139	ON	WARN	Scheme functions registered in EQUATE without a specified Scheme file.	All Scheme functions registered in the EQUATE section must have an accompanying Scheme file identified in the HEADER section.
CMP-140	OFF	WARN	Port permutations were required.	It was necessary to apply dependent swappability rules to identify equivalences.
CMP-141	ON	WARN	MATCH_EQUATED_NETS conflicts with SCHEMATIC_PERMUTABLE_PORTS. Ignoring MATCH_EQUATED_NETS.	SCHEMATIC_PERMUTABLE_PORTS will be used when it conflicts with MATCH_EQUATED_NETS.
CMP-142	ON	WARN	Swappable or permutable port "%s" not found in block "%s".	A port specified in a SCHEMATIC_SWAPPABLE_PORTS or SCHEMATIC_PERMUTABLE_PORTS rule was not found in the netlist.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-143	ON	WARN	Non-existent ports in SCHEMATIC_SWAPPABLE or SCHEMATIC_PERMUTABLE_PORTS rules. Ignoring rules.	One or more swappable or permutable ports were not found. All SCHEMATIC_SWAPPABLE_PORTS and SCHEMATIC_PERMUTABLE_PORTS rules will be ignored.
CMP-144	ON	WARN	Swappable or permutable ports entry for "%s" not found.	One or more swappable or permutable ports for the current block were not found.
CMP-145	OFF	WARN	Assumed %s match based on properties for schematic "%s" and layout "%s"	Property values were used to distinguish between ports that were otherwise similar.
CMP-146	ON	WARN	Comparison of user property "%s" is not defined for merged devices.	User properties cannot be compared on merged devices (unless Scheme property comparison functions are written) because the values of those properties are not defined in certain merging scenarios.
CMP-147	ON	WARN	Comparison of "%s" is not defined for merged %s.	Length and width cannot be independently determined for merged resistors and therefore will not be compared.
CMP-148	ON	WARN	Property width_ratio violation(s) found in shorting equivalent nodes. Nodes WILL be shorted.	Parallel chains of series devices do not have consistent ratios of device widths. The logically equivalent internal nodes will still be shorted, but the nodes are not exactly electrically equivalent.
CMP-149	ON	WARN	This equivalence point contains property errors, but comparison will continue because all nodes matched.	Although this equivalent point contains property errors, those errors do not prohibit Compare to continue to other equivalence points.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-150	ON	WARN	ERROR_ON_GUESS conflicts with SCHEMATIC_PERMUTABLE_PORTS. Ignoring ERROR_ON_GUESS.	SCHEMATIC_PERMUTABLE_PORTS is used when it conflicts with ERROR_ON_GUESS.
CMP-151	ON	WARN	Layout port "%s" in BLACK_BOX %s=%s {} is not texted; The name is subject to change and may invalidate the black box.	BLACK_BOX entries require a correspondence listing of all ports between the layout and schematic. We recommend that all ports be texted because untexted net names are chosen arbitrarily and therefore may change in another layout extract.
CMP-152	ON	MSG	%s net "%s" connects to a %s net across all instances and will be treated as a %s net.	PUSH_DOWN_PINS has the ability to push power and ground information down the hierarchy. If a net in a lower level cell hooks up to power or ground across all instances of that cell, then the net can be treated as power or ground in the cell. This will help with merging and filtering in the cell.
CMP-153	ON	WARN	Design has no structure for swappability check.	The design is empty; it has no nets or devices.
CMP-154	ON	WARN	MATCH_EQUATED_NETS conflicts with DETECT_PERMUTABLE_PORTS. Ignoring MATCH_EQUATED_NETS.	DETECT_PERMUTABLE_PORTS is used when it conflicts with MATCH_EQUATED_NETS.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-155	ON	WARN	The following ports appear more than once in SCHEMATIC_PERMUTABLE_PORTS rules: %s. This may result in significantly slower run times for LVS COMPARE in the parent cell. When possible, condense rules so that no port appears more than once.	The current swappability rules in SCHEMATIC_PERMUTABLE_PORTS can cause ports to appear more than once. The rules need to be revised.
CMP-156	ON	FATAL	No matching pin for %s.	While attempting to verify a port permutation, a pin on a schematic device was found to not have a matching pin on the matching layout device.
CMP-157	ON	ERR	Device count exceeds specified maximum of %d.	Device count exceeds the limit.
CMP-158	ON	ERR	Auto_Exclude_Equiv unable to resolve some hierarchical discrepancy; Use EXCLUDE_EQUIV on the cells marked with ** in the previous table.	AUTO_EXCLUDE_EQUIV was unable to resolve a particular hierarchical discrepancy and it must be resolved by hand using EXCLUDE_EQUIV. This message appears very infrequently.

Table 5-13 CMP Messages

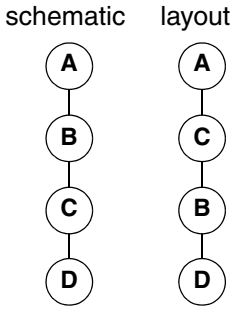
ID	Default	Severity	Message Text	Explanation
CMP-159	ON	WARN	Removing hierarchically invalid equiv: %s=%s	This message appears when equivalence points are reversed in the hierarchy. The reversing of hierarchy is invalid and the offending equivalence points are removed. In the example below, equivalence points B and C would be removed from both the schematic and layout netlists. 
CMP-160	ON	MSG	NOTE: The top block is a black box. There is nothing to be done	Black box cells are not considered during netlist comparison. When a top-level cell is a Black box, nothing can be done.
CMP-163	ON	ERR	Cannot open pushdown pin file "%s" for writing.	The specified file could not be opened for write access.
CMP-164	ON	ERR	Cannot open pushdown pin file "%s" for reading.	The specified file could not be opened for read access.
CMP-166	ON	WARN	Exceeded MERGE_PATHS_DEVICE_LIMIT for path originating at net "%s".	This message indicates that a net has been rejected as an origin for path processing. The COMPARE option MERGE_PATHS_DEVICE_LIMIT is used to restrict the number of merged devices created by MERGE_PATHS. If MERGE_PATHS is enabled and MERGE_PATHS_DEVICE_LIMIT is set, then the paths originating from each path origin in the circuit will be analyzed to determine the number of merged devices that will result.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-167	OFF	WARN	Dependent swappability found	The ports of this block exhibit a symmetry which is not pure independent swappability.
CMP-168	ON	WARN	No %s %s nets were found. Generating them automatically: %s ...	Indicates that Compare determined schematic/layout power/ground nets automatically.
CMP-169	ON	WARN	No %s %s nets were found and none could be generated.	Indicates that Compare attempted to determine schematic/layout power/ ground nets automatically, but was unsuccessful.
CMP-170	ON	WARN	No equivalence file or schematic top block specified. "%s" will be used.	The specified schematic cell name is the root of the schematic netlist and will be used as part of the top block equivalence.
CMP-171	ON	WARN	Multiple schematic root cells exist. The largest will be chosen.	Several roots exist in the schematic netlist. Since there was no obvious means to distinguish them, the largest is chosen and is used as part of the top block equivalence.
CMP-172	ON	FATAL	Cannot open automatically generated equiv file "%s" for write.	The auto-generated equivalence file cannot be produced due to a file permissions problem.
CMP-173	ON	MSG	Determining equivalence points ...	Indicates that no equivalence file was specified, and one must be produced automatically.
CMP-174	ON	WARN	Total device counts differ by more than 50%%; check scope of netlists!	The total number of devices found under the root of the schematic netlist differs from the total number under the layout root by more than 50%. This may indicate a problem in scope. For example, only a portion of the layout is being analyzed, but the entire schematic is being used to produce the equivalence file.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-175	ON	ERR	Port swappability cannot be completely determined for this block.	This cell has ports with symmetries that could not be determined by detect_permutable_ports.
CMP-176	ON	ERR	Port swappability cannot be completely determined for this block because child "%s = %s" of this block has dependently swappable ports.	This cell has complex symmetries and has a child cell with complex symmetries. detect_permutable_ports will not process this scenario.
CMP-177	ON	WARN	%s already exists, it will be renamed as %s.bak.	The runset supplement automatically produced by Compare already exists. The existing one is renamed before the new is created.
CMP-178	ON	FATAL	%s cell %s is part of a multi-equiv that is only partially black boxed.	If a multi-equiv situation is to be changed to black boxes, every member of the multi-equiv must be black boxed.
CMP-179	ON	ERR	Cannot open %s file "%s" for reading.	The specified black box file could not be opened for reading. This might indicate a file protection problem.
CMP-180	ON	WARN	Cannot find BLACK_BOX cell "%s" in %s netlist	This cell specified in the black box file does not exist in the corresponding netlist. The black box will be ignored.
CMP-181	ON	WARN	Cannot find BLACK_BOX port "%s" in %s cell "%s"	This port specified for a black box cell, does not exist on that cell in the netlist.
CMP-182	ON	ERR	%s port "%s" in black box "%s" corresponds to more than one %s port not belonging to the same pushdown pin group	When a single port in one netlist corresponds to multiple ports in the other netlist (by virtue of push_down_pins), then it is best to just do the mapping to one port, and Compare will recognize that which reconciles the rest.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-183	ON	ERR	Top block %s in %s netlist is empty. lsh exits.	An empty layout netlist has been extracted. There is nothing for Compare to do.
CMP-184	ON	WARN	Layout cell "%s" has a port count %d exceeding 50,000.	Whenever a cell is found in the netlist with more than 50,000 ports, that cell is reported with a warning. This usually means that something unexpected has occurred with this cell in the LVS Extract.
CMP-185	ON	WARN	Created new cell "%s" for %s netlist.	An empty dummy cell has been created to correspond with an undefined cell in a specified black box.
CMP-186	ON	FATAL	Parsing error at or near line %d of file "%s"	A parse error has occurred in reading the black box file.
CMP-187	ON	WARN	Merging nets %s and %s, which differ only by case. If these nets should be distinct, consider setting IGNORE_CASE to FALSE.	IGNORE_CASE has shorted two nets in a cell with the same name but different case. This may warrant further review.
CMP-188	ON	ERR	Schematic cell "%s" is instantiated without being defined and it is not a candidate for EQUATE generation	When Compare is generating EQUATEs, it looks at all blocks that are instantiated, but for which there is no cell definition. If one is found with too many ports (>4), that indicates that it must not be a candidate for EQUATEs, but instead is indication of an incomplete netlist.
CMP-190	ON	ERR	%s top block '%s' not found.	A top block, either schematic or layout, has been specified that could not be found in the corresponding netlist.
CMP-191	ON	ERR	Cannot produce equate for %s device: %s	The EQUATE generation code was not able to find a complete mapping of the primitive cells. Those primitive cells that do not match will need to be equated explicitly.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-193	ON	WARN	Layout device '%s' is unextracted	This device type has an extract command in the runset but no devices in the netlist. EQUATE generation will proceed since the device does not exist for Compare purposes, but this warning is printed.
CMP-195	ON	WARN	The following global net names are equivalent with the exception of case, '%s' and '%s'. Consider setting IGNORE_CASE to TRUE	Nets that appear to need to be schematic_global nets have been found that are the same text only in different case. Often this indicates the need to run with ignore_case=TRUE.
CMP-196	ON	ERR	"%s=%s" and "%s=%s" are multi-equiv black boxes but the former has %d ports while the latter has %d ports.	Automatically black boxing and multi-equiv type structures have stricter requirements on the port mapping. A violation has occurred and the port list on the specified cells should be reviewed carefully.
CMP-198	ON	ERR	Multi-equiv error: Primitive pin "%s=%s" found in BLACK_BOX %s=%s {} but not in BLACK_BOX %s=%s {}	Automatically black boxing and multi-equiv type structure has stricter requirements on the port mapping. A violation has occurred and the port list on the specified cells should be reviewed carefully.
CMP-199	ON	FATAL	Unexpected end of file in "%s"	An end of file was found in the black_box file before the conclusion of the last construct specified in the file.
CMP-201	ON	WARN	Can not find matched %s ports with same text for %s ports "%s".	If the matched port nets have different text between the schematic and layout, a message will be printed.
CMP-202	ON	FATAL	Cannot find schematic top block, please check your schematic netlist.	If the schematic top block is not defined, a message will be printed.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-203	ON	ERR	Swappable ports exist but option detect_permutable_ports is not turned on.	When some swappable ports are found and the detect_permutable_ports is turned off, an error message will be printed.
CMP-205	ON	WARN	Black box generation for this equiv is disabled.	The equiv was found to have unresolved permutable ports and Hercules is not going to treat this equiv as blackbox.
CMP-206	ON	MSG	NOTE: The flat device match result is base on Pwr/Gnd comparison to report, and the WARNING of unmatch flat devices will not affect the comparison: Post-Compare flat device statistics:	Post-compare statistics when FLAT_DEVICE=true.
CMP-207	ON	WARN	Members of %s %s flat devices	Shows the power/ground members by types. This is for FLAT_DEVICE=true.
CMP-208	ON	WARN	The following Pwr/Gnd/Type sets have different property between SCH and LAY	Lists the property mismatched report for flat devices. This is for FLAT_DEVICE=true.
CMP-209	OFF	MSG	Enable property comparison even when the cells have compare errors	When this message is turned on, property comparison will be executed even when the equiv has other compare errors.
CMP-210	ON	WARN	Inconsistent setting, %s, of multi-equate: EQUATE %10s = %10s EQUATE %10s = %10s	Indicates that there are multi-equates with inconsistent settings. The inconsistent settings are automatically set to TRUE.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-211	ON	WARN	Scheme function "%s" is involved in equivalence file, but scheme file is not assigned in HEADER section.	If the Scheme function is mentioned in the equivalence file, but the Scheme file is not defined in the HEADER section, a warning will be printed.
CMP-212	OFF	MSG	Pre-Merge Netlist Statistics:	By enabling this message, the pre-merge netlist statistics table will be printed to the equivalence summary file.
CMP-213	OFF	MSG	Print properties for unmatched nodes.	By enabling this message, the required properties of the unmatched nodes will be printed to the equivalence summary file.
CMP-214	OFF	MSG	Print the detail connections of the composite devices.	By enabling this message, the detailed connections of the composite nodes will be printed to the equivalence summary file.
CMP-215	ON	WARN	Flat Device "%s" filtered from %s netlist.	Reports the filtered flat devices.
CMP-216	OFF	MSG	Print detail information for composite nodes.	When the message is turned on, the inner details of the composite nodes will be printed.
CMP-217	ON	WARN	Missing required property %s for %s device "%s".	A warning about missing property for certain device.
CMP-219	ON	MSG	The following equivalence points have been automatically excluded because they have port swappability problem.	This message indicates that for the current equivalence point, information about certain child equivalence points is not being used because the child has an unresolved swappable problem.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-220	ON	MSG	The following equivalence points have been automatically excluded because they only contain one device.	This message indicates that for the current equivalence point, information about certain child equivalence points is not being used because the child contains only one device. When you set CMP-220 to OFF, the one-device child equivalence points are still considered in the option AUTO_EXECLUDE_EQUIV=TRUE when comparing the current equivalence point.
CMP-224	ON	WARN	The following child cells have exactly the same netlist and they will be treated as identical devices during compare. Any schematic cell in the left side could be matched to any layout cell in the right side.	When the duplicated equivalence points are found, a message will be printed that lists the matched pairs of the cells between layout and schematic.
CMP-225	OFF	MSG	Print the Merged Devices Referenced in the netlist	Members of the merged devices will be printed in the netlist files, sch.block and lay.block, as the comment of the netlists.
CMP-226	ON	WARN	Schematic empty cell %s is defined in EQUATE command, treat it as a primitive device	If a name exists for both a cell definition in a netlist and a device definition in an EQUATE command, the message will be printed and the name will be treated as a device name.
CMP-227	ON	MSG	Generating dynamic EQUATE %s=%s based on EQUATE %s=%s	The message lists all equates that are dynamically generated.
CMP-228	ON	ERR	Illegal delete list:	User can use delete_lay and delete_sch to remove specified cells from the design during comparison. If the device is removed only in one side, the message will be printed and user needs to check these two delete lists and EQUATE.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-229	ON	MSG	Post-Processing for Unmatched Devices/ Nets Analysis:	Start the analysis for the failed equivalence point.
CMP-241	ON	ERR	Unmatched schematic and layout %s are grouped for cross probing.	This message partitions into groups unmatched devices and nets resulting from uncommon circuit errors.
CMP-242	OFF	MSG	"%s" is refined as follows:	By enabling this message, the refined nets of push_down_pins will be printed to summary files.
CMP-243	ON	WARN	Property %s violation, zero value property.	If the value of the property is zero, this warning will be printed to summary files.
CMP-244	ON	FATAL	EQUATE device name "%s" identical to equivalence point name: "%s=%s"	The message will be printed if any duplicated cells are found during comparison.
CMP-245	OFF	WARN	The %s port "%s" has no corresponding %s port with the same name in top block	The message will check the port names on top blocks of schematic and layout netlists. If any port name doesn’t exist on both sides, the error will be printed to the summary files
CMP-246	OFF	WARN	Top block port net "%s" in %s equates to top block non-port net "%s" in %s	If the port net is matched to a non-port net after comparison of the top blocks, the error message will be printed to summary files.
CMP-247	ON	WARN	%s was merged by %d devices, to avoid tremendous amount of messages, only first 50 will be printed.	If there are too many devices to be merged, the message will be printed and only the first 50 member devices will be printed.
CMP-248	ON	WARN	The property "%s" of %s device "%s"is a string property. Thus, the tolerance will be ignored	We don't support the tolerance for string properties.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-249	OFF	WARN	The gate connections between %s and %s are not consistent with the inner properties "%s".	This message indicates that the external gate connections and the inner device properties are not consistent between two schematic and layout composite devices.
CMP-250	ON	MSG	Propagating %s globals ...	This message indicates lsh is propagating global nets.
CMP-253	ON	MSG	Comparing ...	This message indicates lsh is comparing schematic and layout netlists in certain equivalent point.
CMP-263	ON	MSG	Restart equivalence point.	This message indicates lsh is restarting the equivalence point for better results according to previous results.
CMP-264	ON	MSG	Removed bad pushdown pin groups.	This message indicates lsh has identified and removed some bad pushdown pin groups.
CMP-265	ON	MSG	Detected potential duplicate equivalence point.	This message indicates lsh has detected some potential duplicate equivalence point.
CMP-266	ON	MSG	The following equivalence points have been automatically excluded because they are potential duplicate equivalence points.	A list of excluded equivalence point according to hierarchy checking.
CMP-267	ON	MSG	NOTE: Since no equivalence file is specified, COMPARE option FIND_ADDITIONAL_EQUIVS is automatically turned on to find the equivalence points.	This message indicates lsh will automatically generate equivalence points when the equivalence file is not specified.

Table 5-13 CMP Messages

ID	Default	Severity	Message Text	Explanation
CMP-269	ON	ERR	Swapped Pins Devices Group.	This heading and accompanying messages list of potential swapped-pin matched devices connected to unmatched schematic and/or layout nets.
CMP-270	ON	ERR	%s instance type %s is not defined in netlist and has no EQUATE entry in runset.	No valid type definition is available for this instance in the netlist.
CMP-271	ON	WARN	WRITE_NETLISTS is set to FALSE, VUE will not be able to display netlist information or allow cross-probing to/from the netlists	A warning for VUE user that there will be any netlist information for future usage.
CMP-272	ON	WARN	CREATE_VUE_OUTPUT is set to TRUE, therefore WRITE_NETLISTS is set to TRUE.	WRITE_NETLISTS is set to TRUE automatically to provide the netlist information for VUE users.
CMP-274	OFF	ERR	Found empty cell %s not defined as device.	Reports empty cells as errors if not defined in the EQUATE command.
CMP-276	OFF	MSG	Always check top cell port.	This message indicates that CMP-40 is enabled for layout cells not having ports.

6

Analyzing and Debugging for LVS Extraction and COMPARE

This chapter begins with a quick checklist for LVS debug, then describes diagnostic information and the HTML interface for debugging.

Quick Checklist for LVS Debug

Step 1: Account for texting, device extraction errors, and C_THRU errors.

Step 2: If you have major texting errors, review your TEXT_OPTIONS, ASSIGN, and TEXT sections to make sure you have the correct text layers attached to the correct polygon layers.

Step 3: Debug all device extraction errors.

Step 4: Rerun your Hercules LVS job if changes to the design or runset were made.

Step 5: Open the Compare_Results.html file in your browser and review the equivalence points listed in the main LVSDEBUG window.

An example of the LVS debug process follows the expanded discussion of the debugging steps below, but it does not have examples of all of the potential problems listed below.

Device Extraction Debugging

Steps 1 and 2:

Account for all text errors, make sure that any errors reported would not negatively affect the layout netlist. If there are changes to the design or runset, rerun Hercules before proceeding to debugging device extraction errors.

Debug all device extraction errors using the GUI debugging tool, or open the input layout database in a layout editor of your choice, and view the errors using the coordinates given in the *block*.LAYOUT_ERRORS file.

Below are some of the more common text-related problems with examples you might encounter during this debugging step.

Note:

Investigate C_THRU errors if the SCON_MODE option is used. When SCON_MODE is used, the layer1 polygons associated with the net with the highest polygon count are written to the output. However, if two or more nets have the same layer1 polygon count, the selected net might vary depending on the hierarchy, version, and platform. This might result in inconsistent LVS reports when using C_THRU output to construct circuit connectivity which eventually becomes netlisted.

Step 1 and 2 Examples - Unused Text

If you define a text layer in the ASSIGN section and use it in the TEXT section, Hercules generates an Unused Text ERROR for each piece of that text that is not *attached* to a polygon.

Note:

Attached is defined by the value of the ATTACH_TEXT variable in the TEXT_OPTIONS section, and by whether or not the text string overlaps the layer with which it was associated in the TEXT section.

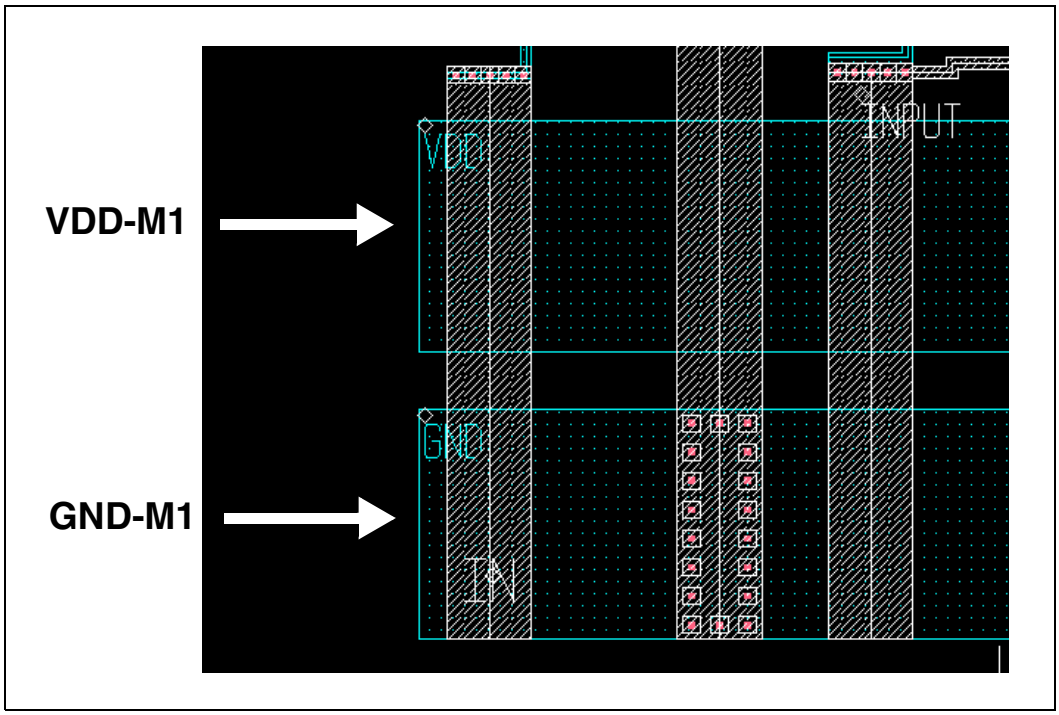
Here is an example of an Unused Text ERROR in *block*.LAYOUT_ERRORS file:

```
Library name:  dac96.db
Structure name:  IOBUF

#####--- ERR_TEXT_UNUSED ----------------------------------

Reports unused text of M1.txt file after all text assignments.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure    Unused Text    l;dt     ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IOBUF        GND            30;0     (-89.500, -303.500)
```

```
IOBUF          VDD              30;0      (-89.500, -253.300)
```

Figure 6-1 illustrates the text strings overlapping M2, not M1. The TEXT on layer 30 should be attached to M2 instead of M1 to avoid these errors.

Figure 6-1 M1 and M2 Text in IOBUF



Text Opens

Whenever you have two unconnected nets in a cell with the same text string, Hercules will generate a Text Open error in the *block*.LAYOUT_ERRORS file. Example 6-1 shows a Text Open error message in the *block*.LAYOUT_ERRORS file.

There are special TEXT_OPTIONS, such as USE_COLON_TEXT and USE_SEMI_COLON_TEXT, to suppress certain Text Open errors in library (or standard) cells. See the *Hercules Reference Manual*, Detailed Options chapter, for details on these options.

Example 6-1 Text Open ERROR in block.LAYOUT_ERRORS File

```
Library name:  dac96
Structure name:  buf4x

#####--- ERR_TEXT_OPEN ----------------------------------
```

```
Nets in named structures are connected by text.

Connect and Text commands used for opens processing:

CONNECT {
   ngate pgate BY [ TOUCH OVERLAP ] field_poly
   M2 BY [ TOUCH OVERLAP ] PADTOP
   M1 M2 BY [ TOUCH OVERLAP ] V1
   M1 ndiffdio res_term field_poly nsd psd welltie subtie BY [ TOUCH
      OVERLAP ] CONT
   NWELL BY [ TOUCH OVERLAP ] welltie
   SUBSTRATE BY [ TOUCH OVERLAP ] subtie
}

TEXT {
   M1 BY M1.text
   M2 BY M2.text
   field_poly BY POLY.text
   PADTOP BY PAD.text
   NWELL BY "VDD"
   SUBSTRATE BY "GND"
}


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Parent Struct  Base  l;dt  Parent  Inst origin  Base   Text From  Path
Text   Text    (x, y)      (x, y)    Net ID            or Info
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
buf4x   IN           30;0  (3.000, 56.000) XX_5  LAYER  Signal
                     30;0 (57.000, 54.500)       LAYER  Signal
```

Text Shorts

Whenever there are two different text strings attached to the same electrically connected net, Hercules will generate a Text Short error in the *block*.LAYOUT_ERRORS file. Below is an example of how that error appears in the file. You can use the TEXT_OPTION FIND_SHORTEST_PATH_BETWEEN_TEXT_SHORTS and the short finding utility in the GUI debugging tool to trace these text shorts. For more information on FIND_SHORTEST_PATH_BETWEEN_TEXT_SHORTS, see the *Hercules Reference Manual*, Detailed Options chapter.

Example 6-2 Text Short Error in block.LAYOUT_ERRORS File

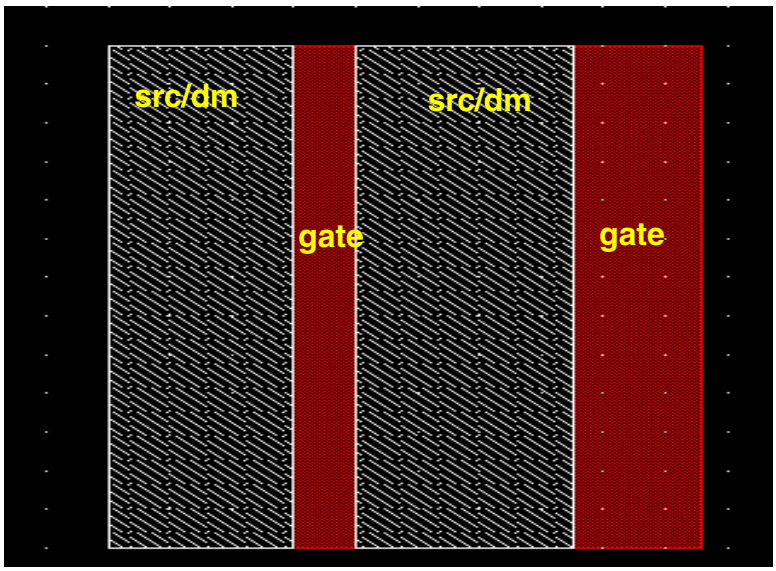
```
Library name:  dac96.db
Structure name:  IOBUF

#####--- ERR_TEXT_SHORT ----------------------------------

Text for net in named structures are shorted.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure   Net  ID   Used   Text    l;dt   (position x, y) Text From
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inva        XX   1    *      A       30;0   (3.000, 2.000)   LAYER
                             Z       30;0   (9.000, 2.000)   LAYER
```

Step 3: Missing Terminal(s) - Devices

Hercules uses a device recognition layer to find devices. Once one is found in the design, Hercules checks to see if the devices are formed correctly. If they are not, an error is reported in the *block*.LAYOUT_ERRORS file, and the device is not extracted. Below are some of the more common device extraction problems with examples you might encounter during this debugging step.

Figure 6-2 Layout of NMOS Defined with Missing Terminals



Example 6-3 NMOS Defined with Missing Terminals in block.LAYOUT_ERRORS File

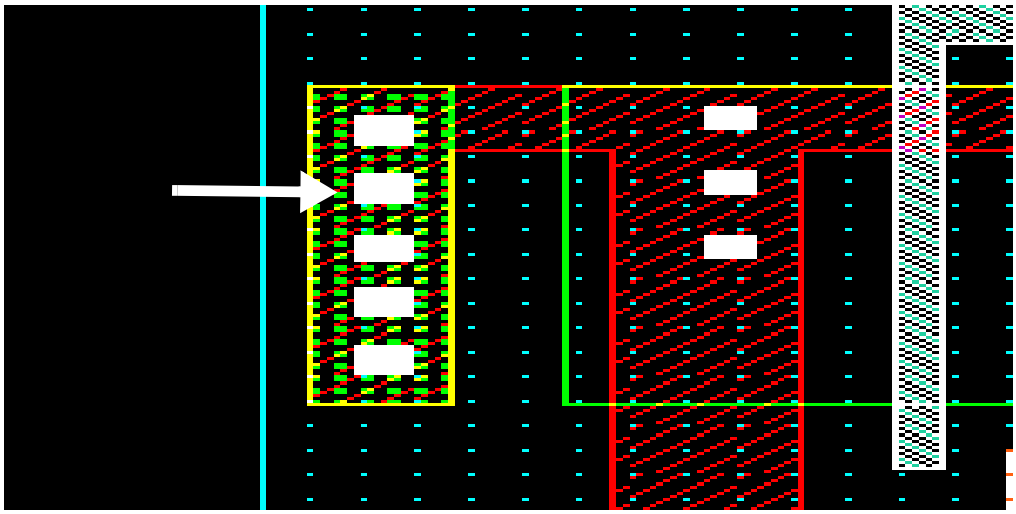
```
Library name:  dac96
Structure name:  inva

#####--- ERR_DEVICE ----------------------------------

NMOS n ngate nsd nsd SUBSTRATE {
   MOS_PRINT_XY_POSITION=FALSE;
   MOS_PRINT_STATS=FALSE;
   MOS_SAVE_ALL_PROPS=FALSE;
   MOS_REFERENCE_LAYER="POLY";
   MOS_HIERARCHICAL=TRUE;
   MOS_CALC_NRS_NRD=TRUE;
   MOS_MULTITERM_EXTRACT=FALSE;
   MOS_NODE_BASE_EXTRACT=TRUE;
   MOS_LEVEL_SD=FALSE;
```

```
   MOS_COMBINE_SOURCE=TRUE;
   MOS_SINGLE_SD=NORMAL;} TEMP=ndevice

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure        Error Type       Layer   Value   ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inva        MISSING_TERMINALS     nsd     1       (11.000, 10.500)
```

Step 3 Examples - Device Has Too Many Terminals

This example shows a diode where, instead of using the diffusion layer as both of the terminals, the diffusion derived layer is used as one and the diocont layer as the other. Figure 6-3 shows a picture of the device and Example 6-4 shows the ERROR in the *block*.LAYOUT_ERRORS file. Notice that the VALUE in the *block*.LAYOUT_ERRORS file is 2. This value will always be one number greater than the number of terminals you are allowed to have for a specific layer. In the case of this example, you should have 1 polygon for the ndiffdio layer, and 1 polygon for the diocont layer.

Figure 6-3 Layout of Diode Defined with Too Many Terminals



Example 6-4 Diode Defined with Too Many Terminals in block.LAYOUT_ERRORS File

```
Library name:  dac96.db
Structure name:  IOBUF


#####--- ERR_DEVICE ---------------------------------

DIODE ndio ndiffdio diocont substrate {
   DIODE_PRINT_XY_POSITION=FALSE;
   DIODE_PRINT_STATS=FALSE;
   DDIODE_SAVE_ALL_PROPS=FALSE;
   DIODE_TYPE=NP;
```

```
   DIODE_HIERARCHICAL=TRUE;
   DIODE_RECOGNITION_LAYER_USED=TRUE;} TEMP=ndiode

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure      Error Type      Layer      Value   ( position x, y )
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IOBUF      TOO_MANY_TERMINALS  diffcont   2       (-88.000, 112.500)
```

To fix this error, use the ndiffdio layer for both terminals. The DIODE is formed from the N material in the terminals and the P material in the substrate.

Step 4:

Once all of the Device Extraction and texting ERRORS have been fixed, rerun the Hercules LVS job.

LVS Comparison Debug

Step 5:

When Steps 1 through 4 are completed, debugging LVS errors can be started. Open the COMPARE.html file in your browser or open the *block*.LVS.ERRORS in a text viewer to review the LVS diagnostics output from Hercules. Below is a list of common LVS.ERRORS and what they imply. A complete list of Hercules diagnostic messages can be found in Table 6-1.

Possibility of Shorts and Opens

In many cases, the possibility of shorts or opens will be discovered in LVS even if there were no text shorts/opens reported in the *block*.LVS.ERRORS file. This could be the result if one or more of the nets involved are untexted. Hercules could also find two differently texted nets that would match one net in the schematic. A table is written to the sum.*sch_block.lay_block* file (referred to as the sum.block.block file) showing the suspected nets. Example 6-5 shows an example of this table.

Example 6-5 Diagnostic OPEN/SHORT Table in sum.block.block File

```
Diagnostic analysis recognizes the following correspondence between
unmatched nets in the schematic and layout. These may indicate the source
of shorts or opens:

Schematic     Layout        Net Name
Connections   Connections
-----------   -----------   ----------------
              9             GND
4                           GND
5                           $1N7
```

Merging Options Missing

If COMPARE determines that there are extra devices in the layout or schematic that can be merged in series or parallel to help your design match, it generates a message. This message gives the merging option that applies to the extra devices and recommends that you set that option to TRUE to help your design compare.

Filtering Options Missing

If COMPARE determines that there are extra devices in the layout or schematic that match one of the available Hercules filter options, it generates a message. This message gives the filtering option that applies to the extra devices and recommends that you set it to help your design compare.

LAYOUT POWER or LAYOUT GROUND Definitions Missing

When the LAYOUT_POWER or LAYOUT_GROUND definitions are missing, Hercules generates a warning advising that these nets are not defined, but Hercules will try to automatically generate these net names based on string matching. If the list of LAYOUT_POWER or LAYOUT_GROUND nets is incorrect, filtering and merging might not occur correctly. Example 6-6 shows an example of this warning.

Example 6-6 Warning: LAYOUT_POWER or LAYOUT_GROUND Definitions Missing

```
HLVS (R) Hierarchical Layout Versus Schematic, Release DATA OMITTED
  Copyright. Synopsys.  All rights reserved.


   ** Environment Status **

runset              = test.ev
root                = DAC96

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

stop_on_no_explode_error    = FALSE  static_equated_nets      = TRUE
text_resolves_port_swap     = TRUE   use_total_width          = FALSE
write_netlists              = TRUE   zero_connection_warning  = FALSE

Purging Compare Directory ... OK

Reading schematic netlist ... OK
Reading layout netlist ... OK

WARNING: No layout power/ground nets were found. Generating them
automatically:  VDDIO VDD GND VSSIO
... Processing schematic netlist ... OK
Propagating schematic globals ... OK
Processing layout netlist ... OK

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

Schematic Globals Missing

Hercules automatically tries to generate schematic globals when they are not defined in the runset. As described in Chapter 5, “Layout Versus Schematic Comparison,” a search of the schematic netlist is done to generate this list, but it is possible for Hercules to miss less commonly defined schematic globals. If this section is missing from your runset, you might want to double check that Hercules found all of the necessary schematic globals, or else false opens could appear in the comparison output.

Step 6

Once you have reviewed and fixed all of these major problems, rerun Hercules LVS on your design and follow the detailed instructions below for debugging Hercules LVS error output.

Text files

When Hercules LVS is run, there are two text files in the working directory you need to look at in the beginning of the debugging process: the *block*.RESULTS file and the *block*.LVS_ERRORS file. These two files, in conjunction with the detailed summary file (sum.*block.block*) and the GUI debugging tool, form a complete LVS debugging tool.

Run Results

The *block*.RESULTS file is the first step in the LVS debugging process. This is a very simple file that states whether each stage of the Hercules run finished successfully and whether the design passed or failed.

If the run finishes successfully, the *block*.RESULTS file will tell you if there are any failed equivalence points and whether the top block passed or failed. In the following example, the *block*.RESULTS file directs you to the *block*.LVS_ERRORS file for more information.

Example 6-7

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

  DATE OMITTED
  Copyright Synopsys.  All rights reserved.

Called as: hercules -C lvs.ev

- Parsing runset "lvs.ev" ... DONE

- Checking input conditions ... DONE

- Performing layout vs schematic comparison ... DONE
    ### LVS errors, refer to DAC95.LVS_ERRORS
```

```
- Creating EVaccess data ... DONE

Hercules Run: Time=  0:02:43 User=99.81 System=4.78
Hercules is done.
```

The ### indicates that there are LVS errors. Hercules will then direct you to the *block*.LVS_ERRORS file for diagnostic analysis of the problem.

The following is an example of a *block*.RESULTS file from a Hercules run that did not finish because of a runtime error:

Example 6-8

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

  DATE OMITTED
  Copyright Synopsys.  All rights reserved.

Called as: hercules -C lvs.ev

- Parsing runset "lvs.ev" ... DONE

- Checking input conditions ... DONE

- Performing layout vs schematic comparison ... ERROR
Hercules Run: Time=  0:01:10 User=65.21 System=4.79
Hercules did not complete.
```

The ERROR tells you that there was a runtime error. This could be the result of a lack of system resources, network issues, segmentation violation, or bus error. If Hercules LVS stops because of a runtime error, check the *block*_lvs.log file to see what occurred.

Diagnostic Information

At the end of an LVS comparison run by Hercules, a text file called *block*.LVS_ERRORS is created. After the extraction errors are clean, look here to find diagnostic information on the design.

The file is divided into 3 sections.

- Header
- General Messages
- Equivalence Point Messages.

Header

At the top of the *block*.LVS_ERRORS file is the header section. As shown in Example 6-9, this tells you whether or not the top block passed or failed, and the total number of blocks that passed and failed.

Example 6-9 Header Section for PASS

```
Top block compare result: PASS

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

                         [DAC95 == DAC95]

=================================================
Comparison completed with 85 successful equivalencies.
Comparison completed with 0 equivalence errors.
=================================================
```

In this design, the top block passed and there were no lower level equivalence errors.

If the design does not compare at the top block, the header section will look as shown in Example 6-10.

Example 6-10 Header Section for FAIL

```
Top block compare result: FAIL

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


                         [DAC95 != DAC95]


============================================================
Comparison completed with 28 successful equivalencies.
ERROR: Comparison completed with 57 equivalence errors.
============================================================
```

If the top block compares, the design is clean. You can, however, have lower level equivalence errors. In this case, the header section will say PASS across the top but give you a warning on the lower level equivalence errors, as shown in Example 6-11.

Example 6-11 Header Section for PASS with WARNING

```
Top block compare result: PASS

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

                         [DAC95 == DAC95]

================================================
Comparison completed with 79 successful equivalencies.
WARNING: Comparison completed with 6 lower level
         equivalence errors
================================================
```

Notice that the ERROR message has been downgraded to a warning.

General Messages

The General Message section gives you diagnostic information for the whole chip. The information mainly applies to power and ground issues, COMPARE options that might be incorrectly set, strange device count mismatches, and so forth. Most of this information is not easily obtained from the detailed equivalence summary files. Not all failed designs will have general messages. There are a lot of situations where Hercules determines that the problem with the chip lies within a sub-block and will only print equivalence point messages. Example 6-12 shows the header and general message sections of a *block*.LVS_ERRORS file.

Example 6-12 Header and General Message

```
Top block compare result: FAIL

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

                         [DAC95 != DAC95]

============================================================
Comparison completed with 28 successful equivalencies. ERROR: Comparison
completed with 57 equivalence errors.
 ============================================================

(DBG-4) It may be possible to eliminate some unconnected nets by
declaring the following nets as schematic_global:
                    - GND
```

This general message tells you that the net GND is not a schematic global. This is very important information. If this is the case, all GND nets will not get propagated throughout the schematic.

Equivalence Point Messages

After the general message section comes the equivalence point messages. Each equivalence point that fails is output into the text file along with detailed diagnostics that tell you what the possible problems with the cell are.

The first equivalence points written are the starting point cells. They are denoted by the asterisks (**) before the cell name. There is also a note before listing the equivalence file that tells you that the following equivalence points are the ones to start debugging. After all the starting point equivalence points have been output, the remaining equivalence points are output.

Example 6-13 shown the header and equivalence messages of a *block*.LVS_ERRORS file. For this run, there were no general messages output.

Example 6-13 Header and Equivalence Message

```
Top block compare result: FAIL

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

                       [DAC95 != DAC95]

================================================================
Comparison completed with 79 successful equivalencies.
ERROR: Comparison completed with 6 equivalence errors.
================================================================

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

        Hercules has determined that the following
        equivalence points probably contain errors
        that need to be fixed first.

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

** NOR2B != NOR2C (level = 10) **

  (DBG-14) Filterable layout devices.

  Please check run_details/compare/nor2c/sum.NOR2B.nor2c
```

```
============================================================
============================================================
        Remaining equivalence point diagnostics
============================================================
============================================================

OR2B != OR2C (level = 9)

  (DBG-12) Suspicious bulk connections.

      Device            Reason
      ------            ------
       P                Untexted Bulk

  (DBG-19) Extra layout devices.

  Please check run_details/compare/or2c/sum.OR2B.or2c
```

Each equivalence point has the following general format:

```
FOO != FOO
   diagnostic message 1
   diagnostic message 2
Please see: path_to_detailed_summary_file.
```

There are several messages that can be printed in the *block*.LVS_ERRORS file to provide diagnostic information about the COMPARE errors in their design. A number can be printed along with the message if the option MESSAGE_IDENTIFIERS = TRUE. Table 6-1 has a detailed explanation of each message.

Table 6-1 Diagnostic Messages

Message Number	Message
DBG-0	Schematic and layout agree at all equivalence points. No further debugging necessary.
DBG-1	Schematic and layout agree at the top block. No single failed equiv point required excessive processing time. Removal of the failed equiv points listed below will reduce processing time in subsequent runs. An ignore_equiv file, *block*.ignore_equiv, has been generated. EXPLANATION: The design is clean because the top block passed but there were lower-level equivalence points that failed. By removing these failed equivalence points, you will save time and have cleaner results.

Table 6-1 Diagnostic Messages(Continued)

Message Number	Message
DBG-2	Top block does not compare but the number of devices and nets match between the schematic and layout. Consider setting DETECT_PERMUTABLE_PORTS to TRUE. Also, consider setting MERGE_PATHS to TRUE. EXPLANATION: This is a general message saying that the number of nets and devices at the top match. This usually means that the problem preventing the design from comparing is simply a logic problem. This kind of problem is much easier to detect.
DBG-3	Processing of the entire hierarchy might result in a more complete diagnostic report. Consider setting STOP_ON_ERROR to FALSE. Also, consider setting EXPLODE_ON_ERROR to TRUE. EXPLANATION: If STOP_ON_ERROR is set to true, Hercules does not continue to compare other equivalence points if there is an error. Hercules has several helpful diagnostics that are not run if the top block is not processed.
DBG-4	a) It is possible to eliminate some unconnected nets by declaring the following nets as schematic_global: b) The power/ground net, FOO, is apparently disconnected in the layout as shown by these unmatched nets in the top block: EXPLANATION: a) Hercules has determined that many of the mismatched nets in the schematic could be fixed if the net was labeled a global. b) Hercules has determined that there is an open in a PWR/GND net and another net. This message is a subset of DBG-21
DBG-5	TEXT_OPEN_RENAME has been performed on some unmatched nets. This means that two or more nets in the same cell have the same text but NEVER connect anywhere in the hierarchy. The file *block*.LAYOUT_ERRORS contains more details, but a partial listing of unmatched, renamed nets is listed below: EXPLANATION: If an unmatched net is found in an equivalence point, that net is checked to see if it has been renamed by CONNECT_BY_NAME. If so, you might have open nets that need to be investigated.
DBG-6	There are too many 1-connection nets in the schematic; the schematic netlist might have some problems.
DBG-7	This equivalence point failed to compare, but all of its parents did. It should be removed from consideration. EXPLANATION: The logic in the cell has been exploded into its parents and has been verified clean there. There is no reason to continue debugging this cell.

Table 6-1 Diagnostic Messages(Continued)

Message Number	Message
DBG-8	Missing layout devices which also appear in the *block*.LAYOUT_ERRORS file. These are likely the result of extraction errors. EXPLANATION: Hercules looks in the *block*.LAYOUT_ERRORS file to see if there are any device extraction errors. If they are, and they match the type that is missing in the layout, this message is output.
DBG-9	TEXT SHORT DISCARD problems. EXPLANATION: If a text short is found during Extraction, and REMOVE_TEXT_FROM_SHORT is turned on, all text is removed from the nets and the net receives a Hercules net name. This is a secondary check informing you that there was a text short in an equivalence point.
DBG-10	Untexted layout power problems. EXPLANATION: Connection analysis indicates that the following pairs are apparently supposed to be the same net. This indicates that the layout nets might have untexted power problems.
DBG-11	Untexted layout ground problems. EXPLANATION: Connection analysis indicates that the following pairs are apparently supposed to be the same net. This indicates that the layout nets might have untexted ground problems.
DBG-12	Suspicious bulk connections. EXPLANATION: If the bulk connection to a device is not a PWR/GND net, this message is output.
DBG-13	Filterable schematic devices. EXPLANATION: If a device is unmatched in an equivalence point, Hercules checks to see if it is filterable. If so, this message is written out.
DBG-14	Filterable layout devices. EXPLANATION: If a device is unmatched in an equivalence point, Hercules checks to see if it is filterable. If so, this message is written out.
DBG-15	There are unmatched primitive devices due to a different number of placements of child cells in schematic and layout. EXPLANATION: This message will be output is AUTO_EXCLUDE_EQUIV is turned off and there is a hierarchy problem.

Table 6-1 Diagnostic Messages(Continued)

Message Number	Message
DBG-16	Inadvertently filtered devices. EXPLANATION: Falsely filtered devices can cause a real problem. They show evidence of an untexted short or open, PWR/GND problems or other connectivity problem that might otherwise fall through the cracks. These devices should be closely examined.
DBG-17	AUTO_EXCLUDE_EQUIV was not able to resolve a hierarchical discrepancy. Consider adding an EXCLUDE_EQUIV entry for each of the following child equivs. EXPLANATION: AUTO_EXCLUDE_EQUIV is designed to fix hierarchy problems where the number of instances of an equivalence point differ between the layout and schematic. However, there are certain situations where Hercules cannot determine this during the run. For subsequent runs, the output will be cleaner if you manually omit the cells in question from the COMPARE process.
DBG-18	Extra schematic devices. EXPLANATION: The following equivalence points matched in the layout, but have extra devices in the schematic, and warrant further examination.
DBG-19	Extra layout devices. EXPLANATION: The following equivalence points matched in the schematic, but have extra devices in the layout, and warrant further examination.
DBG-20	Suspicious layout ports.
DBG-21	Possibility of shorts or opens. EXPLANATION: This message is output if the SHORT_ANALYSIS_TABLE is written to the sum.block.block file. This is an indication of untexted shorts or opens in the block.
DBG-22	Layout nets swapped with respect to the schematic. EXPLANATION: This message assumes that the schematic is correct. In this case, the layout nets referenced in the sum.block.block file are swapped. If the layout is changed, the design would compare.
DBG-23	Unmatched schematic devices adding connections that are not in the layout.
DBG-24	Unmatched layout devices adding connections that are not in the schematic.
DBG-25	Property mismatches. EXPLANATION: If property mismatches are the only reason an equivalence point fails, this message is output.

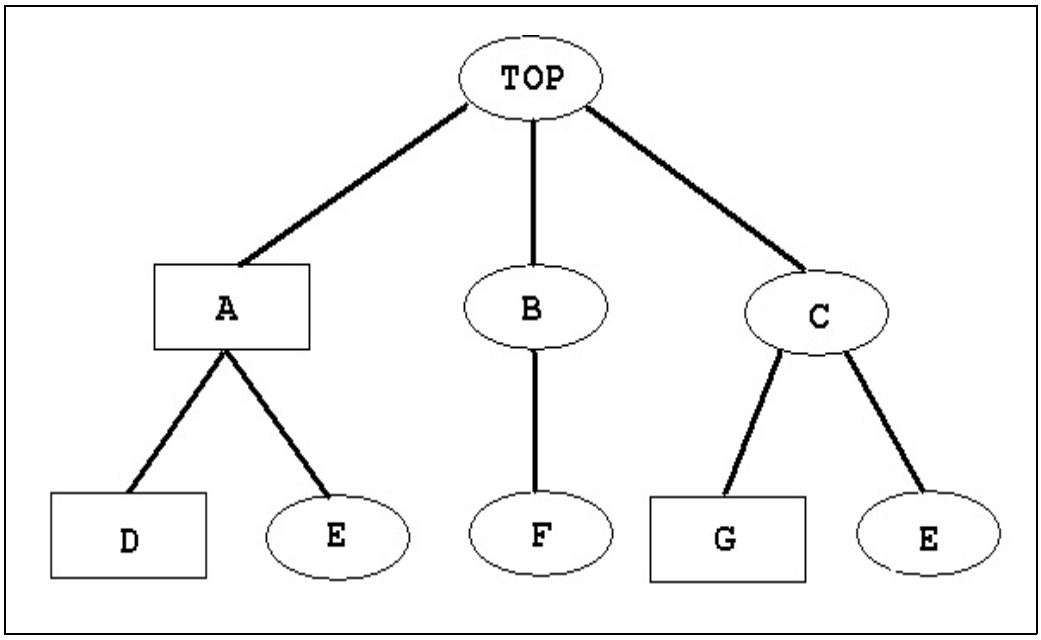
Table 6-1 Diagnostic Messages(Continued)

Message Number	Message
DBG-26	Port swappability problems. EXAPLANATION: This message should not appear if DETECT_PERMUTABLE_PORTS is set to true. This message appears if there are swappable ports in the design that can't be resolved. Because the logic of the cell is compared before swappability is checked, removing this equivalence point will make subsequent runs will be faster.
DBG-28	Top block does not compare but the number of devices and nets match between schematic and layout. Consider removing the following equiv points, which appear to be redundant and might result in unresolvable hierarchy: EXPLANATION: There are several cases where Hercules has determined that, even though an equivalence point passes, it will make the design fail at the top. If these equivalence points are removed in subsequent runs, the design will have a better chance of passing.
DBG-30	Errors occurred in %s, please see run_details/compare/sum.%s.%s for further details. EXPLANATION: Hercules could not determine or make a suggestion on the cause of the problem in this equivalence point.

Finding a Starting Point

To help you find a starting point, Hercules has a cell priority approach to LVS debugging. Consider the following example: (CIRCLE means the equivalence point failed; SQUARE means that it passed. Assume the same name between the schematic and layout.)

Figure 6-4 Hierarchy Example 1



The failed equivalence points are represented by cells B, C, E, F and TOP. To debug this design, you would start by examining the hierarchy. You need to find the equivalence points that are the most likely to have the problem that is causing the design to fail. F and C become the logical choices after looking at the results. Below is a brief explanation of the analysis that you should go through.

B would not be a good starting point because it has a child that fails (F). The child will have to be fixed before B can pass. If there is a physical problem in B in addition to F, it will be harder to debug because the two problems can make it hard to pinpoint the problem.

You would not start with E even though it is a leaf cell that failed. The reason is that, in one instance, it had a parent that compared. Even though another instance had a parent that failed, it would be a lower priority.

F is an obvious choice to start debugging. It is a leaf cell and none of its parents passed.

C is also a good place to start. Even though it has a child that failed, the problem probably lies in C because of the explanation about E above.

In this simple example, you can do this kind of reasoning without much trouble. But if the chip has any complexity to the hierarchy, it becomes almost impossible. It is not impossible for Hercules to do this analysis though. This is what is meant by Cell Priority debugging—having Hercules give you starting points that will allow the quickest and easiest debugging.

What is a Starting Point?

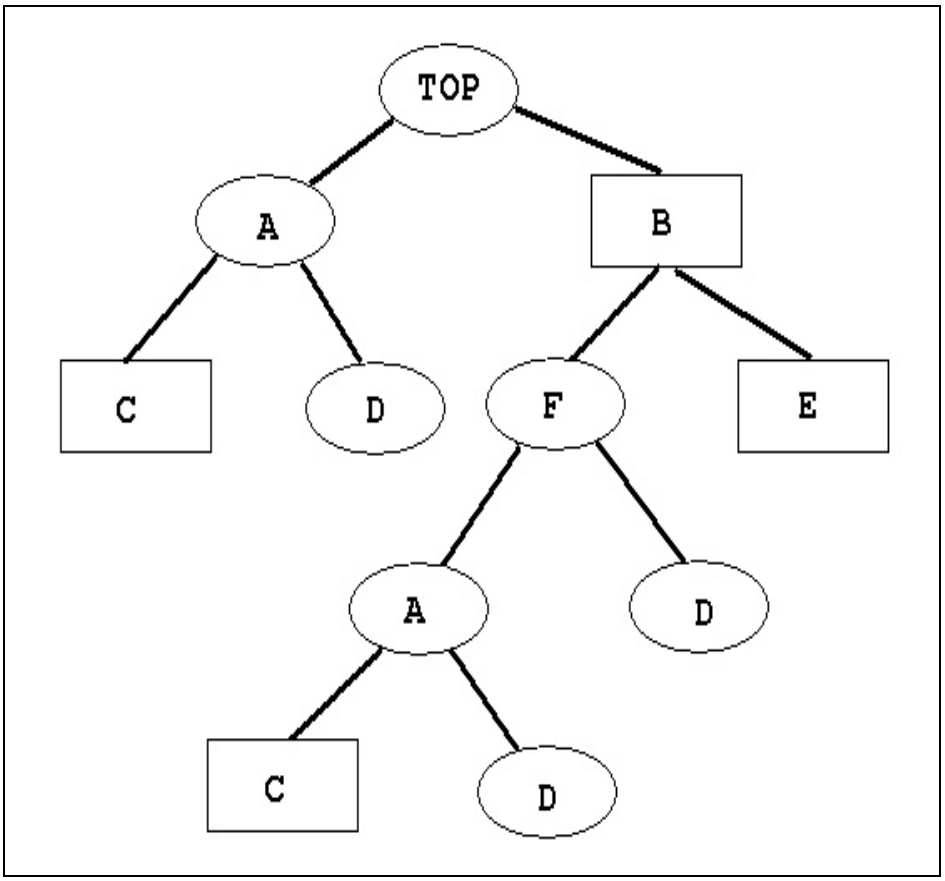
If the top block fails, what are the cells in each branch of the hierarchy that are most likely to contain the real problem?

In other words, which cells, if fixed, would give the most significant progress towards a clean design?

The answer to these questions are the Starting Point Cells or High Priority Cells.

Let us consider another example:

Figure 6-5 Hierarchy Example 2



What are the high-priority cells? The possible answers are A, D, F, and TOP.

F is not because it has a parent (B) that compares.

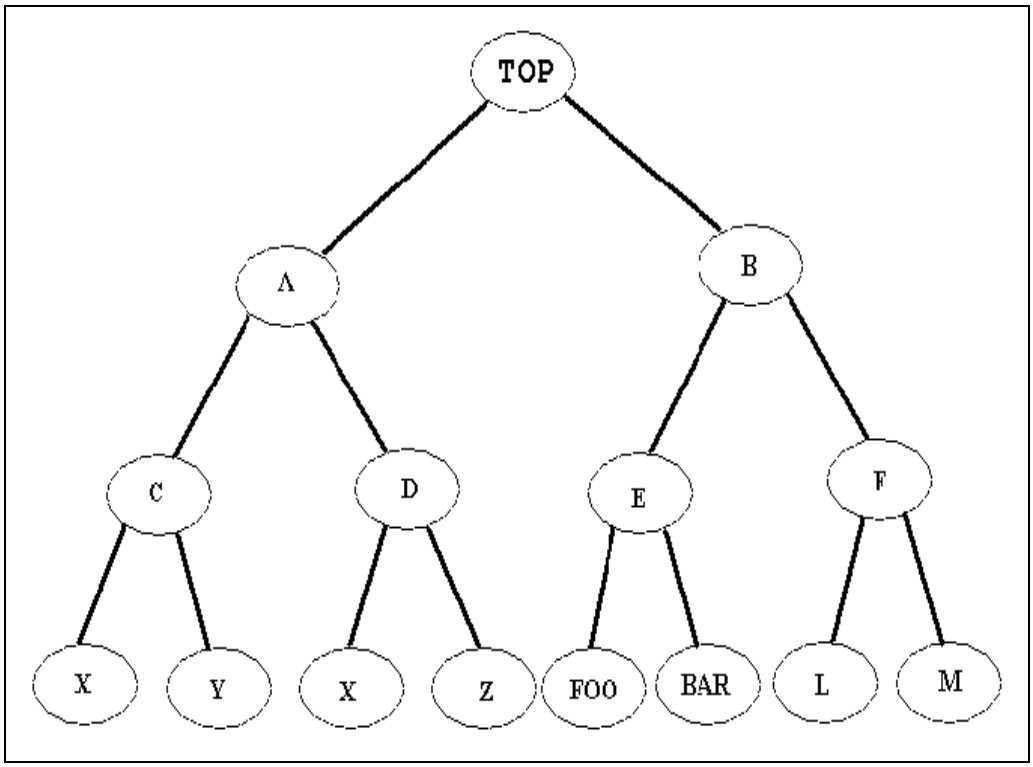
A is not because it has a grandparent (B) that compares.

D is not because it has a great-grandparent (B) that compares.

This leaves only cell TOP as a high priority. This makes sense because after the hierarchy is examined, the probable problem resides in TOP. If you make lower-level equivalence points pass, the runtime might possibly decrease. However, there is also the possibility that fixing a lower-level equivalence point could break a parent. This would result in more problems.

In another example:

Figure 6-6 Hierarchy Example 3



All of the leaf cells (X,Y,Z, FOO, BAR, L, M) are picked out by Hercules as high priority because they are the most likely place where the real problem exists.

There are situations where a problem in the parent block can cause problems in child cells. In these cases however, it is usually easier to debug from the child. A common example of this is with tub ties. Some standard cell libraries do not have them included in the design and the end user must place them later. Often, these are not put in the individual standard cells

but along the PWR/GND rail at a higher level of hierarchy. If one of these is either misplaced or missing, there will be an opening in the child cell. Instead of trying to find the missing tap in the context of the parent, debugging the child leads to the problem more quickly.

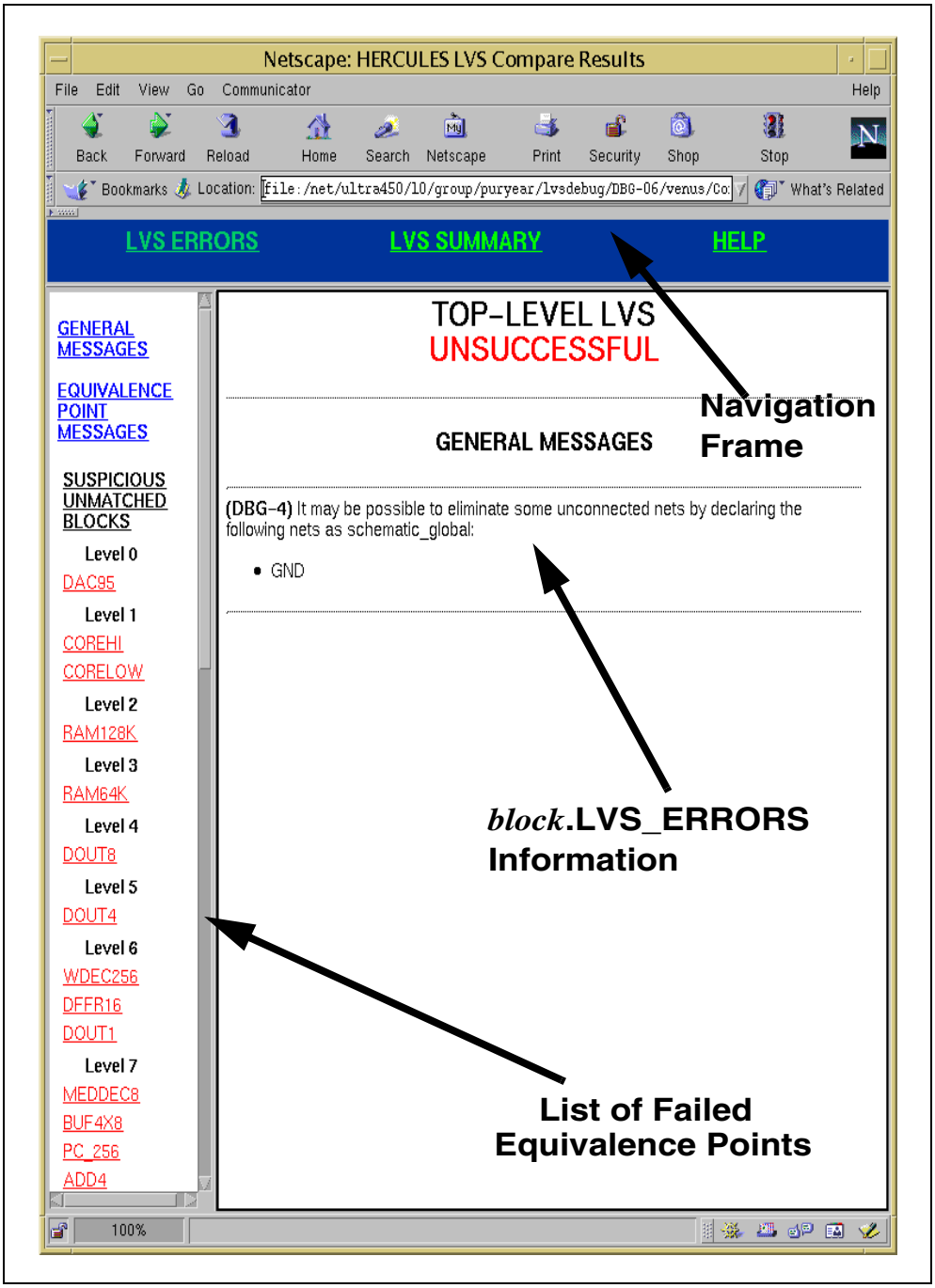
HTML Interface for Debugging

The HTML interface has the same information as the text files. The only difference is that you can move between files with a click of the mouse, making the diagnostic information completely interactive. To have Hercules generate the HTML output, set the command-line argument -html or -html-nobrowse. The former brings up Netscape with the Hercules HTML pages loaded automatically after the run is completed. The latter generates the HTML pages, but does not load them automatically. The COMPARE option HTML_OUTPUT=TRUE will also generate the HTML output and it behaves like the -html command-line option.

After the Hercules run completes, there will be a file called Compare_Output.html in the working directory. This is the main page of the HTML interface. You can either bring up the HTML page manually or you can use the browse script that is also generated.

When the HTML page comes up, there are three frames. The small frame that spans the top is the navigation frame. It has links to the *block*.LAYOUT_ERRORS file, *block*.LAYOUT_ERRORS file, the compare summary file (*block*_lvs.log), and a HELP link. The left frame has links to all unmatched equivalence points as well as links to the general and the equivalence point diagnostic messages. The largest frame is the *block*.LVS_ERRORS file; this file will have a link in the HTML pages only if netlist extraction was run immediately before the LVS run.

Figure 6-7 HTML Output

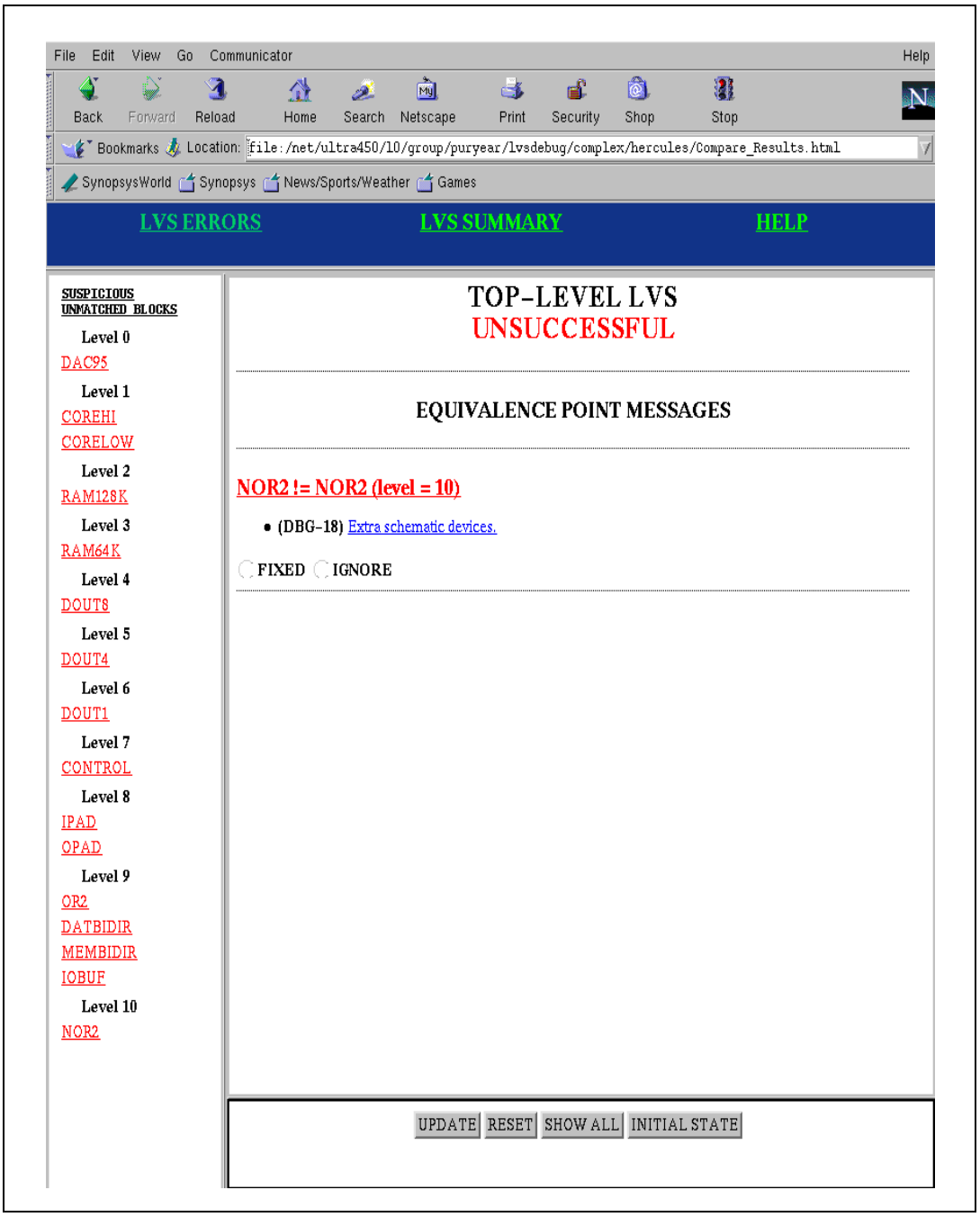


If Hercules outputs general messages, you will not see any information about individual equivalence points when the HTML page is first opened. This is designed to give you a sense of the overall problems before going into the details of the equivalence points. If you want to now see the diagnostics for the equivalence points, you need to click on the Equivalence Point Messages near the top of the left frame.

Figure 6-8 General and Equivalence Point Links

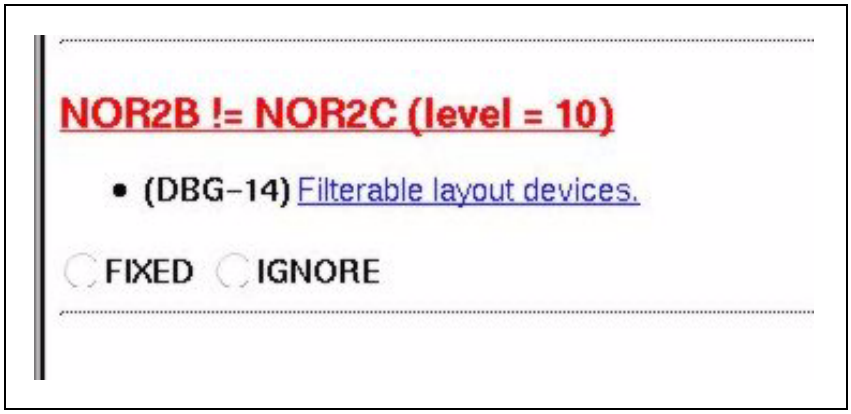
The equivalence point messages look very similar in format to the text form in the *block*.LVS_ERRORS. Only Starting Point equivalence points are viewable. This is so you can see the equivalence points that you should start debugging first and just work your way through them. The name of the equivalence points are linked to the detailed comparison summary file. If clicked on, the page is updated to have the diagnostic information no longer showing. To get back to the diagnostics, either click the BACK button in your browser, or click on the *block*.LVS_ERRORS link at the top of the page. Some of the diagnostic messages are also linked. If this is the case, they will take you to a certain point in the detailed summary file that related to that message.

Figure 6-9 Equivalence Point Messages



The equivalence point messages are completely interactive. There are several buttons underneath that aid in the debugging process.

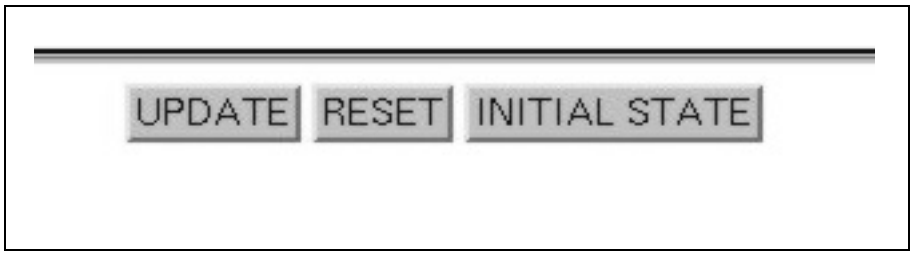
Figure 6-10 FIXED and IGNORE Buttons



When FIXED is pushed, you are telling Hercules that this cell has been fixed in the layout and doesn't need to be considered any more. When the update button is pressed, this cell will no longer show up in the *block*.LVS_ERRORS file. This will help remove some of the unnecessary information from the page. But, marking something fixed will not remove it from the text version of the *block*.LVS_ERRORS file.

Select IGNORE if you determine that a certain equivalence point Hercules identifies as a starting point is not the source of a problem in the design. In this situation you can ignore this cell. By selecting IGNORE, you are telling Hercules you do not want to see diagnostic information on this cell. When the update button is pressed, the cell is removed from the *block*.LVS_ERRORS file. However, the branches of hierarchy that were affected by this equivalence point will still not pass because nothing has been fixed. Hercules will then provide a list of secondary high priority cells that, if fixed, would potentially correct the problem in that branch.

Figure 6-11 UPDATE, RESET, and INITIAL STATE Buttons



When you press UPDATE, the HTML output changes according to the settings that you chose. See the section on the FIXED and IGNORE buttons to see the exact functionality of this button for each of those cases.

RESET sets all buttons back to their default settings. This will not change the list of equivalence points; it will simply unmark equivalence points that have been marked fixed or ignored.

When you mark some equivalence points fixed or ignored they are removed from the HTML page. Press INITIAL STATE to get all of the original equivalence point messages back.

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U