Hercules™ DRC and ERC 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 DRC for Checking

Preface

This preface includes the following sections:

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

What’s New in This Release

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

To see the *Hercules Release Notes*,

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

   https://solvnet.synopsys.com/DownloadCenter

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

About This Guide

The *Hercules DRC and ERC User Guide*, assists the advanced designer in running design rule checking (DRC) and electronic rule checking (ERC) 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 DRC and ERC 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 DRC and ERC flows, further enhancing their knowledge of the Hercules tool and its functions. For the more advanced users, this guide has information bout analyzing and debugging DRC and ERC results. This user guide includes information on

- Executing DRC and ERC verifications
- Using the DRC for checking including layer definition, projection and errors, corner and nodal DRC checking, and texting in a DRC runset
- Using DRC for complex data generation
- Procedures that can be used for analyzing or confirming the electrical connectivity of an IC design

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 DRC and ERC

Using Hercules, a hierarchical physical verification tool, you can perform design rule checking (DRC) and electronic rule checking (ERC) on your integrated circuit design. This user guide includes how to run DRC and ERC using Hercules, check layers, corners, and nodals using DRC, use DRC for complex data generation, and use ERC, analyze and debug DRC and ERC.

What are DRC and ERC?

Hercules DRC is a design rule checker that analyzes the data in the layout and calculates its interactions (spacings and overlaps) to other layers and itself. While doing this, it checks to see if the dimensions are within the design rules.

Hercules ERC is an electronic rule checker that can check for antenna violations and to make sure devices have a path through to a specific net.

DRC and ERC Features and Benefits

DRC

- Internal layer checks
- Wide metal checks
- Layer-to-layer checks
- All design checks are performed hierarchically
- Incremental DRC by layer or cell

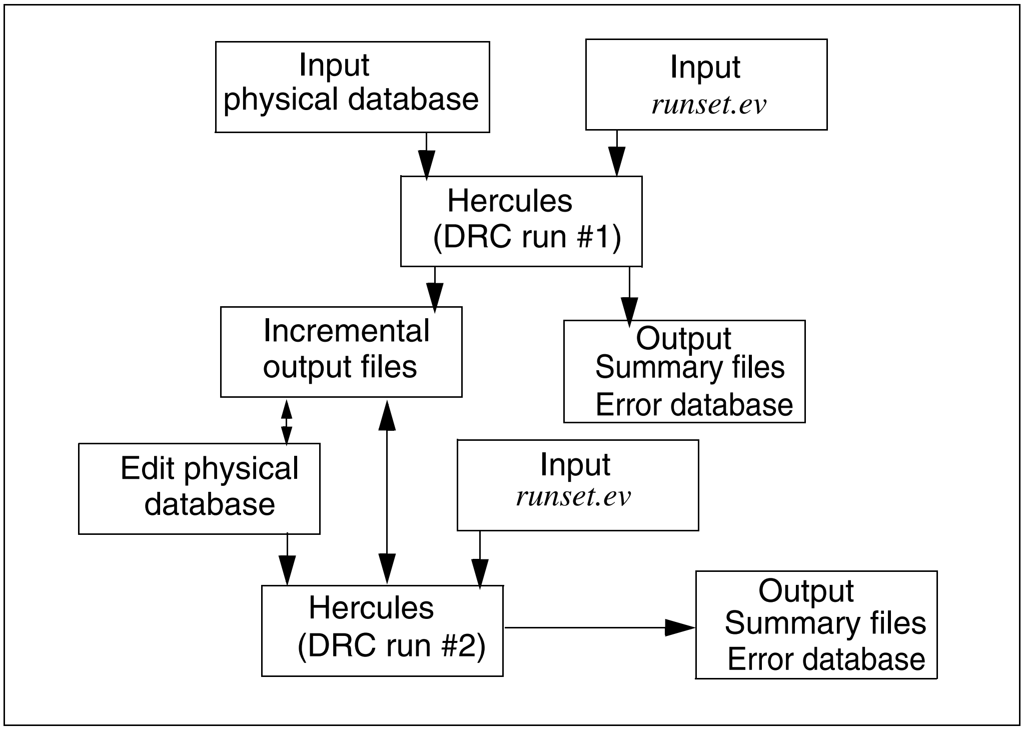
ERC

- Antenna checking with back annotation into Astro
- Through device path checking

Using the DRC and ERC Design Flow

You can run a standard or incremental DRC on your physical database and runset file to generate a summary file and error database. The flow diagram in Figure 1-1 shows standard and incremental DRC flows.

Figure 1-1 DRC Flow



For a standard DRC flow, follow these steps:

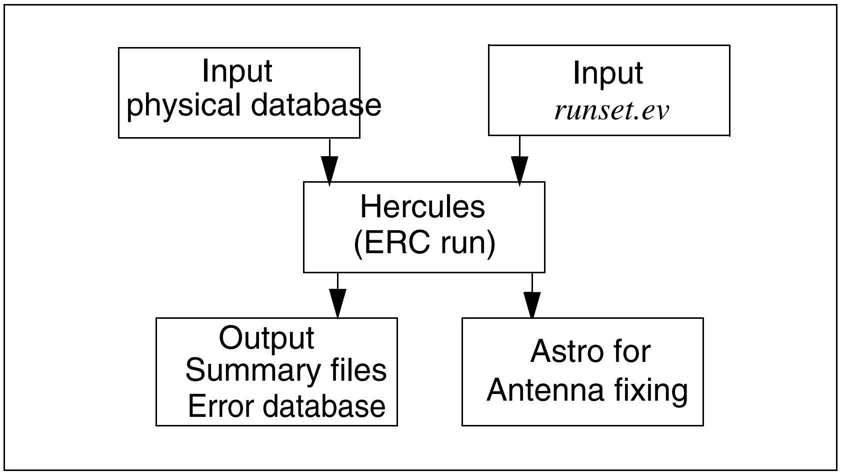
1. Read in your input files (physical database and *runset.ev*.)
2. Run Hercules DRC.
3. Review summary files and the error database.
4. Debug any errors.

For an incremental DRC flow, follow these steps:

1. Read in your input files (physical database and *runset.ev*).
2. Run Hercules DRC.
3. Repeat steps 3 and 4 for as many incremental runs as necessary.
4. Review the output summary files and the error database.
5. Debug any errors.

ERC is used to analyze or confirm the electrical connectivity of an IC design. The flow diagram in Figure 2 shows a standard ERC flow.

Figure 1-2 ERC Flow



For a standard ERC flow, follow these steps:

1. Read in your input files (physical database and runset).
2. Run Hercules ERC.
3. Review the summary files.
4. Debug any errors; fix errors in Astro (antenna only).

2

Executing Hercules DRC and ERC

This chapter describes the execution and analysis of a Hercules runset for both design rule checking (DRC) and electrical rule checking (ERC), as well as information on advanced Hercules DRC usage models.

See *Hercules General Usage Information*, General Hercules Execution chapter, for basic usage models.

The Hercules Command Line

For a complete list of Hercules command line syntax, see *Hercules General Usage Information*, General Hercules Execution chapter. For details on executing Hercules from VUE, see the *Hercules VUE User Guide*.

Hercules High Performance

Hercules high performance capabilities 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.

When distributing processing is used, different commands in a runset are processed by different CPUs at the same time (in parallel). Hercules can distribute on multiple hosts across a network as well as on a single host with multiple CPUs. Long DRC jobs should be run with distributed processing, especially for using idle CPUs on different hosts.

Unlike distributed processing, multithreading is parallelism within the command rather than at the runset level. The work within each command is separated among multiple CPUs (on a single host only). Multithreading should be used for large chips because it is more memory- and disk-efficient than distributed processing. Multithreading is also useful for cutting time on long ERC jobs, because most commands are dependent on the results of previous commands that make distributed processing less efficient.

Since distributed processing and multithreading are complementary, you can use distributed processing for servers across a network and also use multithreading to take advantage of multiple CPUs on each server. This combination is found to be the most effective in general.

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

Selective Command Runs

On many occasions during the incremental layout development process, it is not necessary to run all DRC checks on the whole chip. Hercules allows you to perform incremental DRC runs by selecting types of checks, parts of runsets, layout layers, or blocks to be run on.

For more information on selective command runs, see *Hercules General Usage Information*, General Hercules Execution chapter.

Incremental DRC by Cell

Hercules has the ability to perform incremental DRC on specific cells. A combination of options in the Hercules OPTIONS section allows you to specify which cells Hercules will rerun DRC checks based on the cell's time stamp (last modification time), or the cell's name. To avoid the creation of too many false errors, Hercules checks all commands in the runset and finds a proper value to grow the bounding box of modified cells. If the number of new cell placements is more than 100, or if the top cell has a newer time stamp, incremental checking is disabled and all data in the design are checked.

Incremental DRC checking is a quick way to recheck small areas of the design that might have changed during an Engineering Change Order (ECO) process. Not all incremental runs are followed by a full chip DRC run. A full chip DRC run should be done prior to tapeout.

Detailed Hercules Incremental DRC by Cell Execution

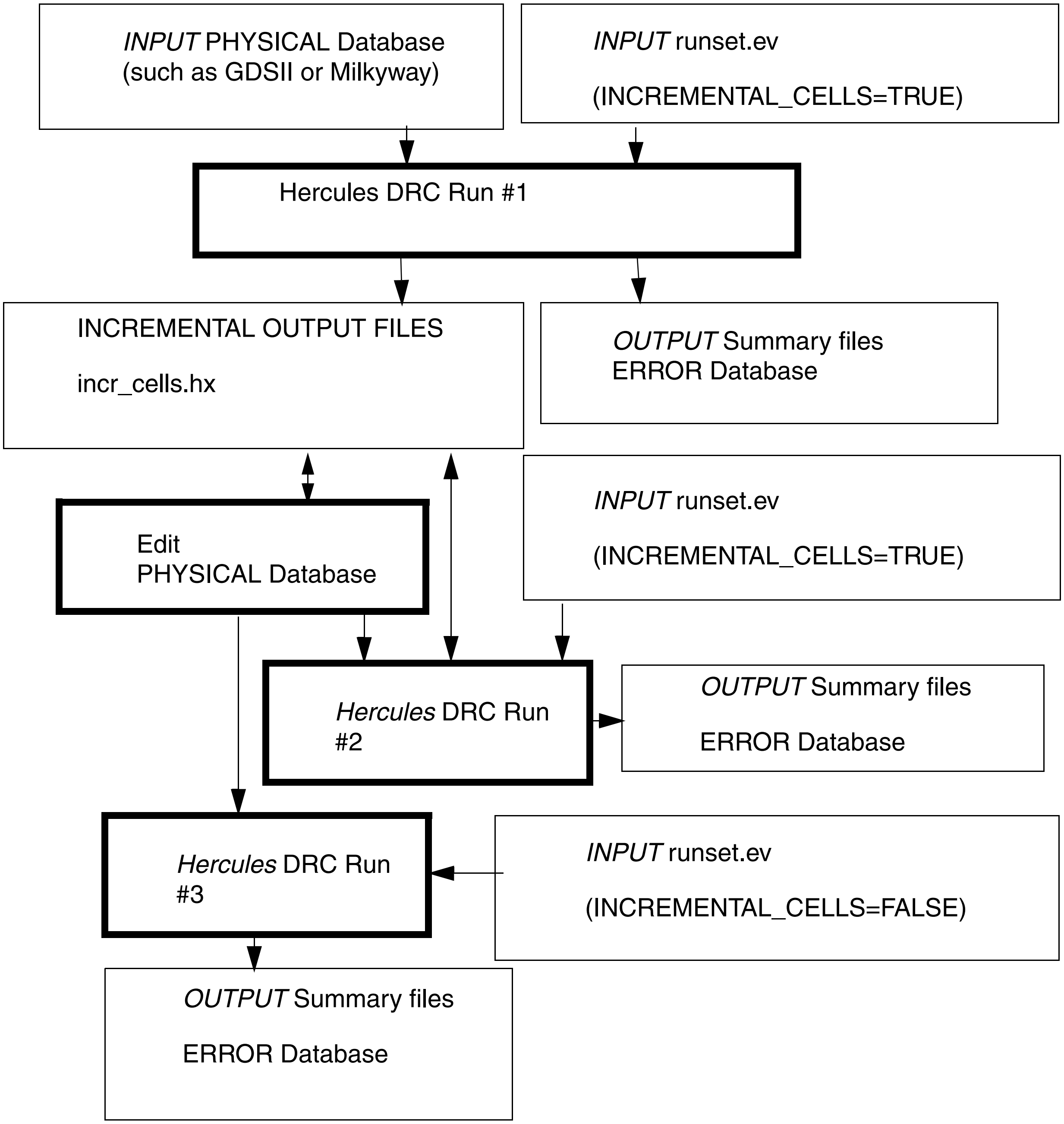
Figure 2-1 illustrates the detailed flow in which Hercules uses the cell's library time stamp information to determine which cells in your design should be re-checked. Following this section is a short section on the two options that allow you to manually specify the cell names you want Hercules to recheck.

Notice that in this time stamp flow, you must either run directly on the Milkyway database or GDSII. (OASIS is not supported because this format does not include time stamps.) Milkyway and GDSII formats guarantee that the time stamps are updated on only the cells that are edited. You can use other environments at your own risk, if you can verify that the GDSII data streamed out of these environments accurately reflects the time stamps of updated cells.

Note:

Overwrite your old GDSII library with the new one because the time stamp file is library specific. If you manually specify your own cell list based on what you know was edited, then the layout editor environment is not an issue.

Figure 2-1 Flow Diagram Using Incremental DRC by Cell Time Stamps



The first step of this flow is to set the INCREMENTAL_CELLS = TRUE option in the Hercules OPTIONS section of your *runset*.ev file. This tells Hercules to read the time stamps for all the cells in your database, and create a file, *incr_cells*.hx. It also creates a database of time stamps under the *./group* directory called *.library_name.hx_time*. In this first Hercules run, the *incr_cells*.hx file is generated with a list of all the cells in your database, and Hercules executes the DRC commands on all of these cells.

The second step of this flow is to edit your design where necessary. After you complete your edits in a layout editor, you can rerun Hercules on your edited Milkyway database. Hercules will automatically use the library database of time stamps created in the previous run and compare it to the new library database of time stamps. For each cell that has been updated, an entry is made in the *incr_cells*.hx file. After all of the time stamps are checked, the *incr_cells*.hx file should contain only the cells that you edited since your last Hercules DRC run.

A new library database of time stamps is created and you can continue editing your design and running Hercules incrementally, until you have completed your edits and they are DRC-correct.

After you have completed your incremental Hercules runs, and are ready for a final DRC check of your design, you need to set INCREMENTAL_CELLS = FALSE and rerun Hercules on your entire design. Always do a complete DRC run on your design before tapeout.

Incremental DRC By Cell Using Explicit Cell Names

In Figure 2-1, Hercules automatically determines which cells in your design have been updated. There are two Hercules OPTIONS that allow you to specify the cells that have been updated and run incrementally.

- The INCREMENTAL_CELLS_FILE = *file_name* option allows you to specify a file that lists the updated cells.
- The INCREMENTAL_CELLS_LIST = {*cell*, ..., *cell*} option allows you to specify a list of updated cells in your Hercules runset.

The overall flow is similar to the cell time stamp flow, except that with this flow, you have to update the file or list of cells with the second option each time you rerun Hercules. This flow is useful if the environment does not store your time stamp when converting to GDS.

Incremental DRC by Layer

Hercules has the ability to perform incremental DRC based on layer names. This is a very simple flow and therefore does not require a flow diagram. The INCREMENTAL_LAYERS option in the OPTIONS section of the Hercules runset allows you to list all layers that have been modified since the previous run. Only the commands that either directly or indirectly depend on one or more of the listed layers will be run.

INCREMENTAL_LAYERS can be used with DRC or ERC runs. Device extraction will not function as expected in an incremental-by-layer run, and the extracted netlist will not be correct. Always do a complete DRC run on your design before tapeout.

An example of a situation where this command should be used is when you have made changes to a metal mask layer and you want to verify quickly that your changes are correct. First run DRC with the changed layer listed in the INCREMENTAL_LAYERS option. Next, correct any violations you find and rerun with INCREMENTAL_LAYERS until no violations are detected. Finally, rerun your DRC, LVS, and ERC runsets on all layers and verify that your design is clean.

Incremental DRC with GUI Debugging Tool

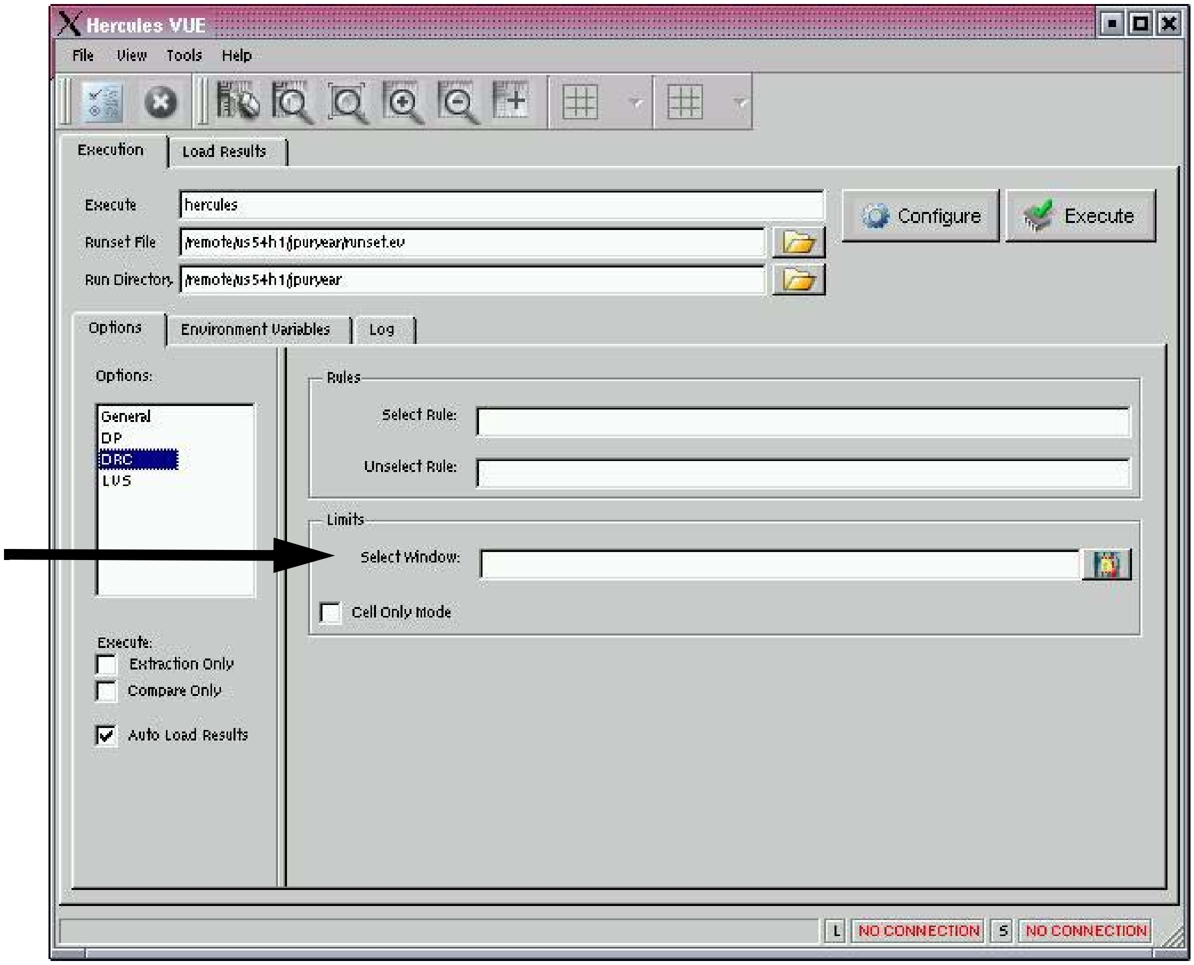
Hercules in combination with Hercules-VUE GUI debugging tool has the ability to interactively select a region or window of the design, and execute a Hercules DRC runset on that area. This flow is executed using the Hercules SELECT_WINDOW command and the GUI debugging tool.

Note:
For details on executing Hercules from VUE, see *Hercules VUE User Guide*, Running Hercules Within VUE chapter.

1. Start Hercules-VUE and a layout tool such as IC Workbench EV Plus, or Cadence® Virtuoso® Layout Editor.
2. Establish that the GUI debugging tool is connected to your layout tool.
3. To rerun Hercules only on selected areas of the design, click the Options tab in the Execution pane. The Select Window text area accepts input in two modes (see the arrow in Figure 2-2):
   - *Automatic Mode:* Click the User button to the right of the Select Window text box. Then, select the region to be checked in layout editor with the mouse. You can repeat this process to select multiple areas, but you must click the User button for each area.
   - *Manual Mode:* In the Select Window text area, enter four space-separated coordinates to define each desired area. The coordinates for multiple areas are not delimited. For example, to specify two areas, enter eight space-separated coordinates.

     Note:
     The coordinate list in VUE is different than as specified in the SELECT_WINDOW runset option. In VUE list the coordinates without brackets or delimiters. For example, in the SELECT_WINDOW runset option, the syntax is `[x1,y1,x2,y2]`, whereas in the VUE Select Window text area it is `x1 y1 x2 y2`.

Figure 2-2 Execute Hercules-VUE Window (with Options)



Execute: hercules

Runset File: /remote/us54h1/jpuryear/runset.ev

Run Directory: /remote/us54h1/jpuryear

Options: General, DP, DRC, LVS

Select Rule:

Unselect Rule:

Select Window:

□ Cell Only Mode

Execute:

□ Extraction Only

□ Compare Only

☑ Auto Load Results

Hercules Run Analysis for DRC and ERC

Analysis During the Run

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

When a Hercules DRC run is submitted, the first objective is to parse the runset for any typographical errors or undeclared variables. If an error is encountered, the run stops and the errors are logged to both the screen and the file *parse*.out or the *block*.sum. The *block*.RESULTS file will reference the appropriate file. Any parse errors must be fixed for the run to be successfully submitted.

The second step in a Hercules run is to preprocess, create, and sort the group files. Group files are written into the directory specified in the HEADER section of the runset as GROUP_DIR. The initial group files are written in the order they exist in the ASSIGN statements. A group file is not written until all processing is done. All other PERM or TEMP output group files are created as they are encountered in the runset.

Group files, by default, are deleted when no longer needed during the Hercules run, with any remaining files deleted after the run is complete. If group file creation takes more than a few minutes, the CHECK_POINT and RESTART commands should be used while modifying check algorithms to optimize performance. If you modify the explode list, delete list, or any runset options, then new group files must be created.

Analysis After the Run

Hercules outputs two primary files and multiple secondary files for an HDRC run. The two primary files are *block*.RESULTS and *block*.LAYOUT_ERRORS. The suffixes of the primary files are capitalized for easy reference.

The *block*.RESULTS file is the place to start the analysis of the run. It indicates whether or not the run completed, and indicates DRC errors, if any are present. The *block*.RESULTS file references the *block*.LAYOUT_ERRORS file if there are errors.

The *block*.LAYOUT_ERRORS file identifies which check has failed, the level of the hierarchy in which the check failed (identified by structure name), and a relative coordinate for the error vector. The *block*.LAYOUT_ERRORS file also includes any structures and coordinates which failed the CHECK_45, CHECK_90, and GRID_CHECK commands. Duplicate placements are also reported in the *block*.LAYOUT_ERRORS file, and must be corrected. Duplicate placements are identical structures placed directly on top of one another, and, while not fatal to HDRC, can present problems when HLVS is run at a later time. Particular error types, such as duplicate information, can be suppressed so that they are not reported. The information in the *block*.LAYOUT_ERRORS file can be reviewed using HDRC, which greatly simplifies the error finding and fixing methodology.

The *block*.sum file is a secondary file which contains the information displayed on the screen as the run was proceeding. This file contains real and CPU time, and required memory for preprocessing, creating group files, sorting group files, and performing the checks. At the end of the file, the total run time and peak memory for all checks to complete is listed after the Overall ev_engine time file. Wall (real) run time might not be the best way to judge performance due to system load from other processes running simultaneously with Hercules. A better measure of job runtime is to look at the number in seconds after the User field. This CPU time provides an approximate job runtime if no other processes are running concurrently. The *block*.sum file also includes information which helps determine the limits of existing system hardware. An entry after each check documents the maximum amount of memory used to complete the check after the Mem field. Large numbers indicate that you might want to run Hercules on a machine with more memory, especially if runtimes increase drastically as data volume increases.

The *block*.tree file provides a list of the design's structure hierarchies, as well as a count of each structure at each level of the design. The file also shows a count of levels of hierarchy in the design, unique cells, hierarchical placements, total design placements, number of placements exploded or deleted, and the number of polygons in the top level. Also documented is a cell-by-cell definition of the number of primitive elements used to create the cell, times placed in the design, and other pertinent data, such as explosions and deletions count by cell name.

Description of Output Files

Analysis and debugging of DRC run can be also done using text output files. Although it is much more efficient to use the GUI debugging tool, text files give all the information necessary to locate and fix DRC errors.

The *block*.sum file informs you that a certain check is done by listing the command and options used to do the check, the number of violations found, and the time and memory used for the check.

For example:

```
INTERNAL met2 {
   COMMENT = "Metal2 width check"
        SPACING<4.000
        CONVEX_TO_CONVEX<4.500 } (105)
WARNING - 2 width violations found.
   Check time = 0:00:01  User=0.00 Sys=0.00 Mem=7.230
```

In the preceding example, a metal2 width check is done with the INTERNAL command and two errors are found. Graphical error layers are stored on layer 105 in the output library. Check time is the total (wall) time; User is the CPU time; and Sys is the I/O time (all times are in seconds). Mem is used memory in KB.

The *block*.LAYOUT_ERRORS file first lists the summary for each error (if PRINT_ERRSUM_FILE = TRUE in the runset) at the top of the file. Then it gives the location of each violation, including precise coordinates, the name of the cell and the measured distance that is in violation of DRC rule. See Example 2-1.

Example 2-1 Layout Errors

```
 LAYOUT ERRORS RESULTS: ERRORS

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

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

Library name:  users_drc_labs.lib
Structure name:  TOPCELL2

Metal2 width check
 INTERNAL met2 { } (105) ............................. 2 violations
found.

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

INTERNAL met2 {
        COMMENT = "Metal2 width check"
        SPACING<4.000
        CONVEX_TO_CONVEX<4.500
        WIDTH=0.500 } (105)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Structure         ( lower left x, y ) ( upper right x, y )   Distance
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ADFULAH            (204.000, 53.000)   (207.000, 60.000)      3.000
TOPCELL2           (368.000, 268.000)  (371.000, 269.000)     3.162
```

3

Using DRC for Checking

This chapter includes details on many topics relating to DRC-licensed commands including polygon processing theory for these commands, hierarchy issues, and advanced dimensional applications.

Data Creation Operations

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

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

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

BOOLEAN Command

BOOLEAN command operators are used for merging, intersecting, subtracting, and finding unique data from the specified layers of data to create new layers of data. Each of the BOOLEAN command operators is defined in Table 3-1 and described in the *Hercules Reference Manual*, Detailed Commands chapter.

Note:

The BOOLEAN command is enhanced to allow more than two inputs.

Table 3-1 BOOLEAN Operators

Operator	Description
AND	The intersection of data from layer1 and layer2.
ANDOVERLAP	The overlapping of data from layer1 and layer2 from different hierarchical levels or contained in different cells
NOT	The subtraction of data of layer2 from layer1
OR	The merge of data from layer1 and layer2
XOR	The unique data of layer1 and layer2
Multiple Layers	Enables several Boolean operations to be combined and executed at one time

BOOLEAN: Multiple Layers

Several Boolean operations can be combined and executed at one time. Benefits of combining these operations include the following:

- Reduction of clutter caused by named temporary layers.
- Intermediate results stored on local disk space during distributed runs.
- Possible runtime improvement.

If P-transistor source drain region (psd) is defined as all P-diffusion in an NWELL minus the polysilicon, then

```
BOOLEAN ((PDIFF AND NWELL) NOT POLY) {} TEMP=psd
```

Operator Precedence Rules

As with any equation, operator precedence is very important in evaluating Boolean equations. It is recommended that parentheses be used liberally for clarity and to avoid mistakes.

For the same operator, the layers are grouped from left to right.

```
BOOLEAN A NOT B NOT C {} TEMP=t5
```

. . . is evaluated as if it is written:

```
BOOLEAN (A NOT B) NOT C {} TEMP=t5
```

Operator precedence (from highest to lowest) is OR, XOR, NOT, ANDOVERLAP, AND

```
BOOLEAN a ANDOVERLAP b OR c XOR d NOT e AND f {} TEMP=t6
```

. . . is evaluated as if it is written as:

```
BOOLEAN (a ANDOVERLAP (((b OR c) XOR d) NOT e)) AND f {} TEMP=t6
```

Note:

If subexpressions are repeated in different commands, calculate this only once and store the result in a temporary layer. Use the following:

```
BOOLEAN NDIFF AND POLY {} TEMP=ngate
BOOLEAN PDIFF AND POLY {} TEMP=pgate
BOOLEAN ngate OR pgate {} TEMP=gate
BOOLEAN (NDIFF NOT NWELL) NOT ngate {} TEMP=nsd
BOOLEAN (PDIFF NOT NWELL) NOT pgate {} TEMP=psd
```

instead of:

```
BOOLEAN (NDIFF AND POLY) OR (PDIFF AND POLY) {} TEMP=gate
BOOLEAN (NDIFF NOT NWELL) NOT (NDIFF AND POLY) {} TEMP=nsd
BOOLEAN (PDIFF AND NWELL) NOT (PDIFF AND POLY) {} TEMP=psd
```

Layer Generation Grammar

```
Term_1         : layer
               | '(' Layer_Equation ')'

Term_2         : Term_1
               | Term_2 OR Term_1

Term_3         : Term_2
               | Term_3 XOR Term_2

Term_4         : Term _3
```

```
                  | Term_4 NOT Term_3

Term_5            : Term_4
                  | Term_5 ANDOVERLAP TERM_4

Layer_Equation : Term_5
                  | Layer_Equation AND Term_5
```

Example 1

```
BOOLEAN A NOT B NOT C {} TEMP=t5
```

Example 2

```
BOOLEAN (A NOT B) NOT C {} TEMP=t5
```

Example 3

```
BOOLEAN a ANDOVERLAP b OR c XOR d NOT e AND f {} TEMP=t6
```

Example 4

```
BOOLEAN (a ANDOVERLAP (((b OR c) XOR d) NOT e)) AND f {} TEMP=t6
```

SELECT Command

The SELECT commands are used to create new layers resulting from operations on a single input layer or two distinct input layers. For edge selection, see the SELECT_EDGE command. Each of the SELECT operators is defined in Table 3-2 below and described in the *Hercules Reference Manual*, Detailed Commands chapter.

Table 3-2 SELECT Operators

Operator	Description
BY_PROPERTY	Selects polygons if they have the specified property associated with them
CUTTING	Selects data from one layer which cuts (intersects) data of another layer
EDGE_TOUCH	Selects data from one layer that has coincident edges with another layer
ENCLOSING	Selects data from one layer that fully encloses data of another layer
INSIDE	Selects data from one layer fully inside data of another layer

Table 3-2 SELECT Operators

Operator	Description
INSIDE_HOLE	Selects data from one layer fully surrounded by and fully touching data of another layer (This works for either 1 or 2 layers)
INTERACT	Selects data from one layer which interacts in any way with data from another layer
OUTSIDE	Selects data from one layer fully outside data of another layer
TEXTED_WITH	Selects data from specified layers that are texted with specified text string or strings
NOT TEXTED_WITH	Selects data from specified layers that are NOT texted with specified text string or strings
TOUCHING	Selects data from one layer which is fully outside but touches data of another layer (May be used with the POINT_TOUCH option)
VERTEX	Selects data elements from one layer that have the number of vertices within the specified range

SIZE Command

There are two kinds of sizing operations: basic and bounded. Basic sizing enlarges or shrinks the polygon data of a layer. Bounded sizing is a repetitive procedure involving basic sizing and Boolean operations which provides electrical path length and distance checking capability. The syntax and operation of the basic and bounded SIZE operations are very different and are described separately in the following sections.

The SIZE operator, both basic and bounded, adds or subtracts data by the appropriate value. The SIZE algorithms are assumed to be correct only for data which is on-grid and for a SIZE value which is a multiple of the grid resolution. Data which is off-grid must be placed on-grid before the SIZE. The SIZE value, if it is not a multiple of the grid resolution, is rounded to be a multiple of the grid resolution.

Basic SIZE Command

The basic SIZE operation changes the size of polygons on the specified layer by adding or subtracting a specified value to or from each edge of each polygon. Each of the basic SIZE operators is defined in Table 3-3 below and described in the *Hercules Reference Manual*, Detailed Commands chapter.

Table 3-3 Basic SIZE Operators

Operators	Description
EDGESIZE	Adds a specified value to the edges of each polygon without extending the corners
GROW	Increases the size of one or more sides of a polygon by the user-specified value
OVERLAP	Outputs only the overlap of sized polygons
OVERSIZE	Adds a specified value to the edges of each polygon and extends the corners
OVER_UNDER	Performs an oversize and then an undersize on the polygon data and gives one result
SHRINK	Reduces the size of one or more sides of a polygon by the user-specified value
UNDER_OVER	Performs an undersize and then an oversize on the polygon data and gives one result
UNDERSIZE	Subtracts a specified value from each edge of each polygon
VSIZE	Sizes directed vector paths (This is a cell-level operation)

Bounded SIZE Command

Bounded SIZE operators oversize polygons on layer1 subject to the edge boundaries of polygons on layer2. These operators (shown in Table 3-4) allow measuring electrical path lengths of polygons or electrical distances between polygons. They are typically used for checking maximum electrical distance process rules. The inside operator oversizes polygons of one layer enclosed by polygons on a second layer. The OUTSIDE operator oversizes polygons of layer1 to polygons on layer3 without overlapping the polygons on layer2. Both operators are repetitive procedures that combine incremental oversizing steps and BOOLEAN command steps in order to measure an electrical path length (INSIDE operator) or to measure the electrical distance between polygons (OUTSIDE operator). Each of the bounded SIZE operators is defined in Table 3-4 below and described in the *Hercules Reference Manual*, Detailed Commands chapter.

Table 3-4 Bounded SIZE Operators

Operator	Description
INSIDE	Repetitive sizing operation that oversizes polygons on one layer inside (enclosed by) polygons on another layer.
OUTSIDE	Repetitive sizing operation that oversizes polygons on one layer subject to the edge boundaries of another layer.

Layer Definitions

This section describes the Hercules tool vector functionality, which is primarily within the Hercules commands. Some of the information in this chapter is duplicated from other chapters in the*Hercules Reference Manual*. However, this chapter compiles all the vector information in one place for easy use and learning.

A vector is a line segment connecting two points. In most cases, the purpose of using vectors is to select an edge of a polygon and still maintain that edge's original relationship to the polygon. The original relationship of a polygon to a vector is maintained by a vector's direction, which indicates where the inside and outside of a vector is located. (The direction of a vector is determined by following the edges around a polygon in a clockwise direction.) When a vector has direction, the inside of a vector is towards the right of the vector, and the outside is towards the left of the vector, relative to the direction the vector is pointing.

Creating Vectors

The EXTERNAL, INTERNAL, ENCLOSE, INSIDE_EDGE, CUT, NOTCH, and VECTORIZE commands provide output in the form of vectors. Each command is described with the necessary syntax to create vectors.

By specifying the OUTPUT_EDGES option with the EXTERNAL, INTERNAL, ENCLOSE, INSIDE,_EDGE, or NOTCH commands, output is in the form of violating edges, rather than violating regions. The portion of the edge from the input data that caused the violation will be output in directed vector form. For two-layer checks, the layer1 edges are designated as output1, and the layer2 edges are designated separately as output2. If only one output is specified, then only layer1 edges are flagged.

When TOUCH and OUTPUT_EDGES are set to TRUE, Hercules reports layer1 TOUCH violations to the Output_Definition. When OUTPUT_TOUCH_BOTH (default is FALSE) is set to TRUE, Hercules reports layer1 and layer2 TOUCH violations to the Output_Definition.

Note:

> The OUTPUT_TOUCH_BOTH option must be used in conjunction with the OUTPUT_EDGES and TOUCH options, and is valid only for the EXTERNAL, ENCLOSE, INSIDE_EDGE, and MULTI_RULE_ENCLOSE commands.

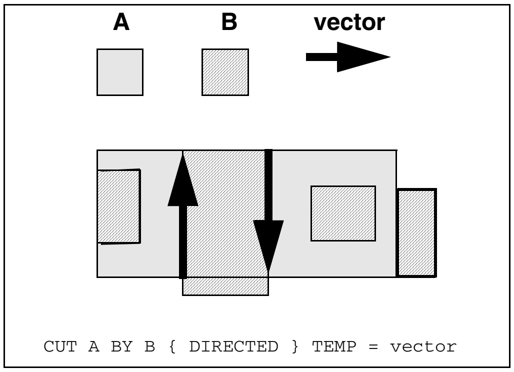
The CUT and VECTORIZE commands produce vectors by default. Use the DIRECTED option with CUT and VECTORIZE to specify a direction for the output vectors.

CUT Command

The CUT command is used to generate edges. For more information on the CUT command, see the *Hercules Reference Manual*, Detailed Commands chapter.

For example, Figure 3-1 shows the CUT command operation with the DIRECTED option specified. The DIRECTED option provides the edge direction information from the original polygon.

Figure 3-1 CUT Command Operation with the DIRECTED Option Specified



```
CUT A BY B { DIRECTED } TEMP = vector
```

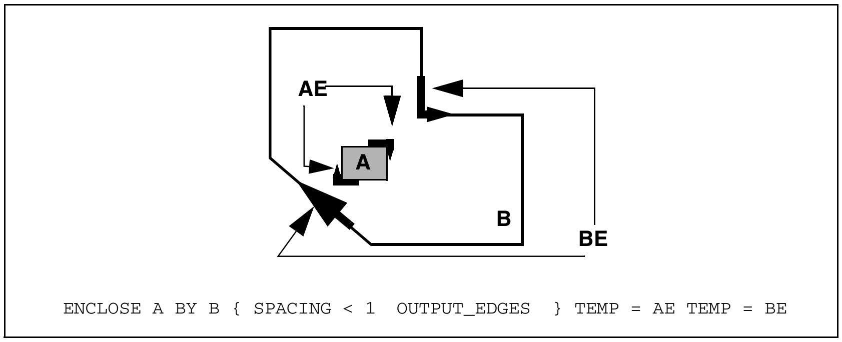
The intersecting edges of layer B are flagged and placed in a temporary file named vector. By using the DIRECTED option, the CUT vectors are a subset of layer B and are given a direction so that the vector/polygon relationship is maintained.

ENCLOSE Command

The ENCLOSE command measures the depth of one polygon enclosed by another. For more information on the ENCLOSE command, see the *Hercules Reference Manual*, Detailed Commands chapter, as well as the example provided in “Vector Examples” on page 3-45.

For example, Figure 3-2 shows the ENCLOSE command operation with the OUTPUT_EDGES option specified. When OUTPUT_EDGES is specified, vectors are produced.

Figure 3-2 ENCLOSE Command Operation with the OUTPUT_EDGES Option Specified



```
ENCLOSE A BY B { SPACING < 1  OUTPUT_EDGES  } TEMP = AE TEMP = BE
```

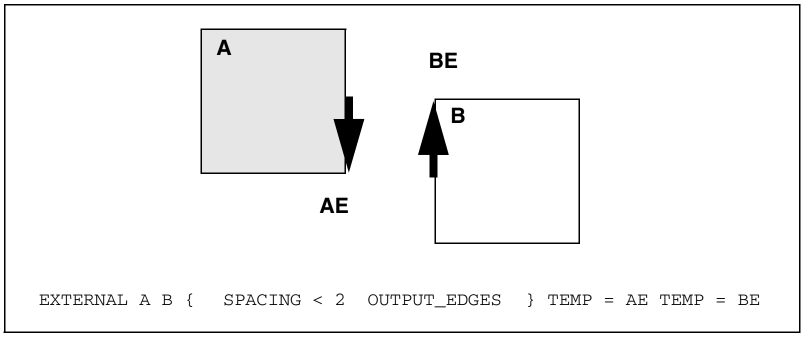
For corner spacing violations, a portion of the edges forming the corner is output. The output edge length is equal to the spacing value that was flagged as a violation. For corner-to-edge violations, the output edge segment is equal to twice the spacing value.

EXTERNAL Command

The EXTERNAL command measures outside edge to outside edge of polygons on one or two layers. Separate check values must be specified for 45-degree edges, corner-to-corner spacing, corner-to-edge spacing, and data within a length range. For detailed information on the EXTERNAL command, see the *Hercules Reference Manual*, Detailed Commands chapter.

For example, Figure 3-3 shows the EXTERNAL command operation with the OUTPUT_EDGES option specified. When OUTPUT_EDGES is specified, vectors are produced.

Figure 3-3 EXTERNAL Command Operation with the OUTPUT_EDGES Option Specified



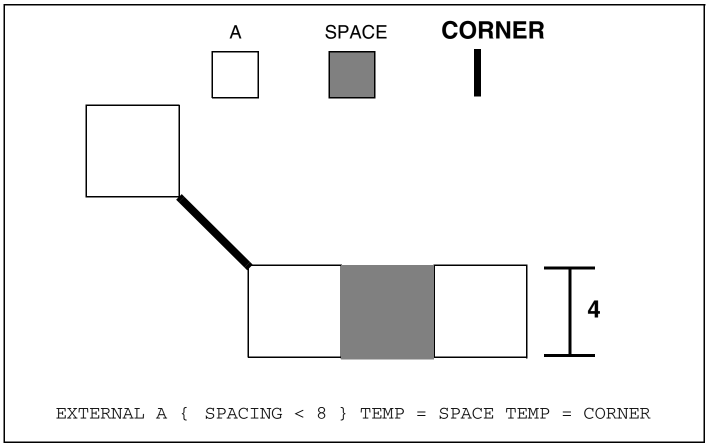
```
EXTERNAL A B {  SPACING < 2  OUTPUT_EDGES  } TEMP = AE TEMP = BE
```

Note:

For edge-to-edge spacing violations, OUTPUT_EDGES outputs only the portion of the edge in violation; that is, AE and BE.

When OUTPUT_EDGES is not used, corner violations must be stored as vectors by specifying a second output file, as shown in Figure 3-4.

Figure 3-4 EXTERNAL Command Operation with the OUTPUT_EDGES Option Not Specified

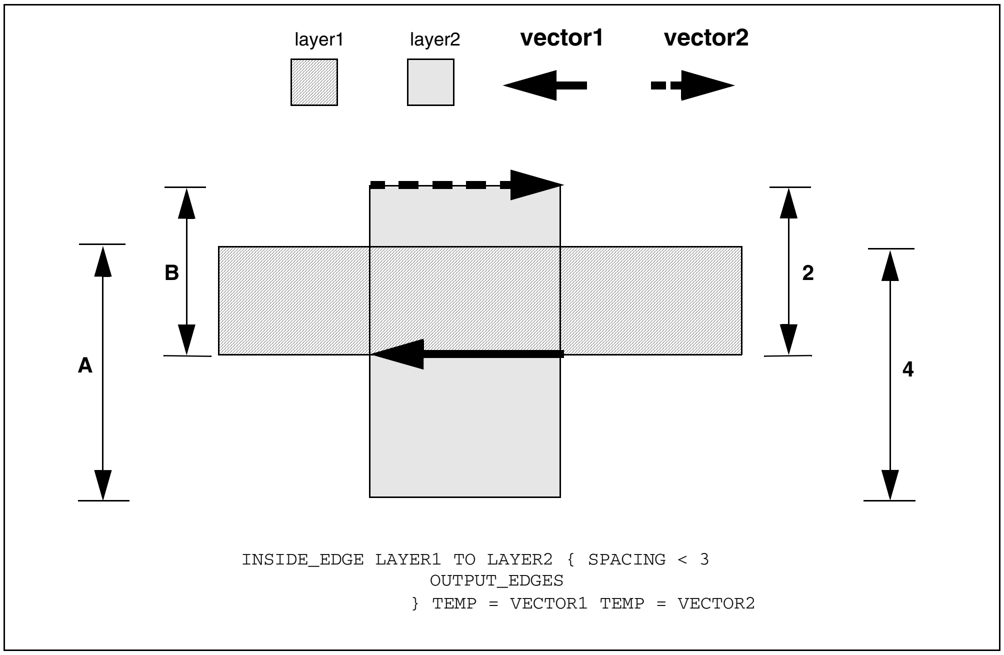


INSIDE_EDGE Command

The INSIDE_EDGE command is a two-layer internal check that measures edge-to-edge distances of polygons that intersect or touch. For detailed information on the INSIDE_EDGE command, see the *Hercules Reference Manual*, Detailed Commands chapter.

For example, Figure 3-5 shows the INSIDE_EDGE command operation with the OUTPUT_EDGES option specified. When OUTPUT_EDGES is specified, vectors are produced.

Figure 3-5 INSIDE_EDGE Command Operation with the OUTPUT_EDGES Option Specified



```
INSIDE_EDGE LAYER1 TO LAYER2 { SPACING < 3
                     OUTPUT_EDGES
                   } TEMP = VECTOR1 TEMP = VECTOR2
```

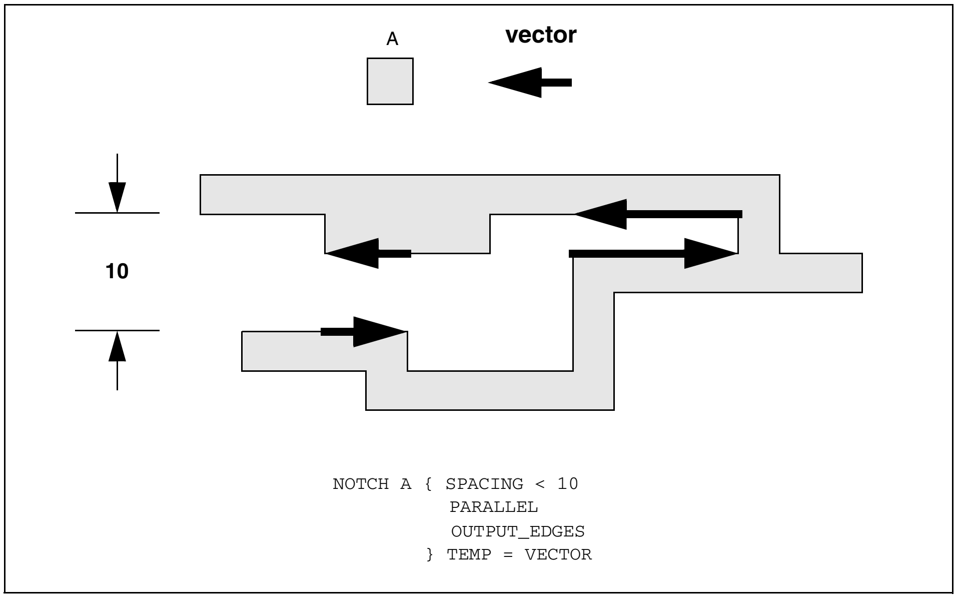
The region from the top of layer1 through the bottom of layer1 to the bottom of layer2 (see A) does not violate the spacing and is not flagged as an error. The distance measured from the bottom of layer1 through the top of layer1 to the top of layer2 (see B) is less than three microns. The edges are flagged as a violation and stored as vectors.

NOTCH Command

The NOTCH command measures outside edge to outside edge distances within a single polygon on one layer. It then flags any distances less than the user-defined check value. Separate check values must be specified for 45-degree edges, corner-to-corner spacing, corner-to-edge spacing, and data within a length range. For detailed information on the NOTCH command, see the *Hercules Reference Manual*, Detailed Commands chapter.

For example, Figure 3-6 shows the NOTCH command operation with the OUTPUT_EDGES option specified. When OUTPUT_EDGES is specified, vectors are produced.

Figure 3-6 NOTCH Command Operation with the OUTPUT_EDGES Option Specified



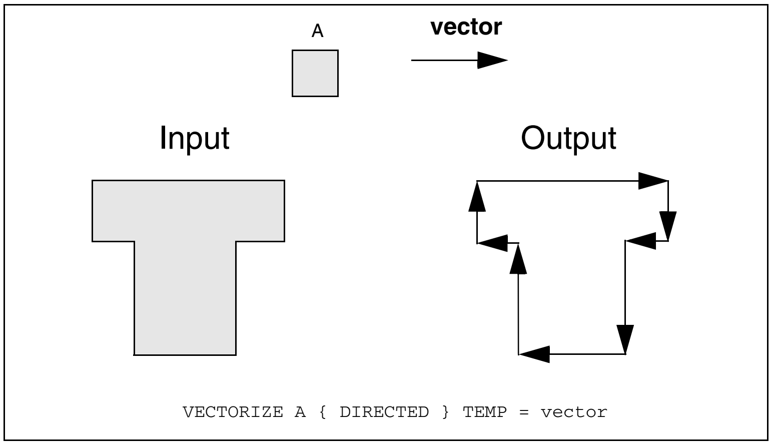
All areas where a layer A edge is parallel to and within 10 microns of another layer A edge are spacing violations. The PARALLEL option is used in this example to disable corner checking.

VECTORIZE Command

The VECTORIZE command converts polygons to vector data. The command generates a closed vector chain for each polygon, where each vector corresponds to an edge of the polygon. See the following example.

The DIRECTED option provides the edge direction information from the original polygon.

Figure 3-7 VECTORIZE Command



Note:

> The vector direction is oriented so that the polygon from which the vector was created is always on a known side (the right).

Processing Vectors

After vector data has been created, the following commands process vectors.

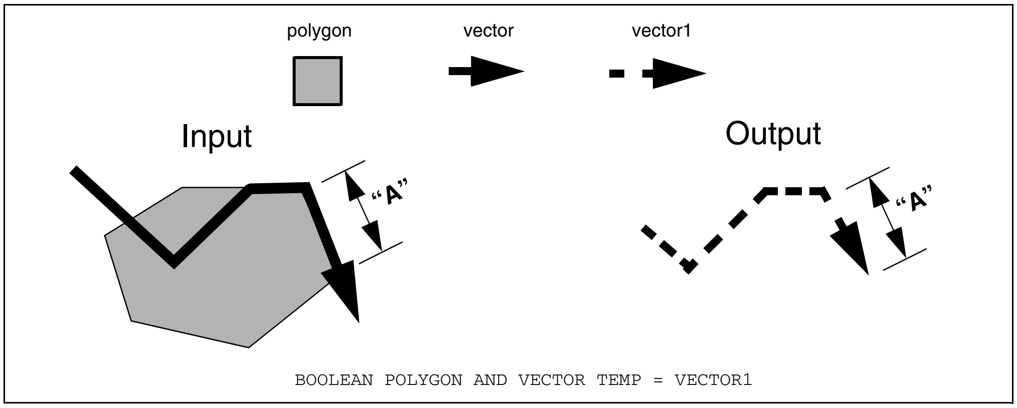
BOOLEAN AND

The BOOLEAN AND command processes vectors according to the combinations described in Table 3-5. The examples that follow the table explain each combination. For more information, see “BOOLEANS Used with Directed Vectors” on page 3-17.

Table 3-5 BOOLEAN AND Vector Processing Combinations

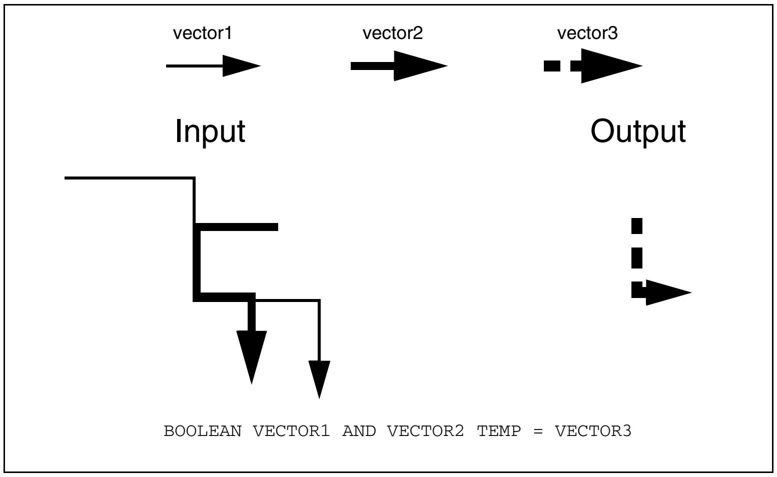
Input One	Input Two	Output
polygon	vector	vector
vector	vector	vector

Figure 3-8 BOOLEAN AND



When one input group file contains polygons and the other contains vectors, the output is a vector.

Figure 3-9 BOOLEAN AND Example 2



When both input group files contain vectors, the output is a vector.

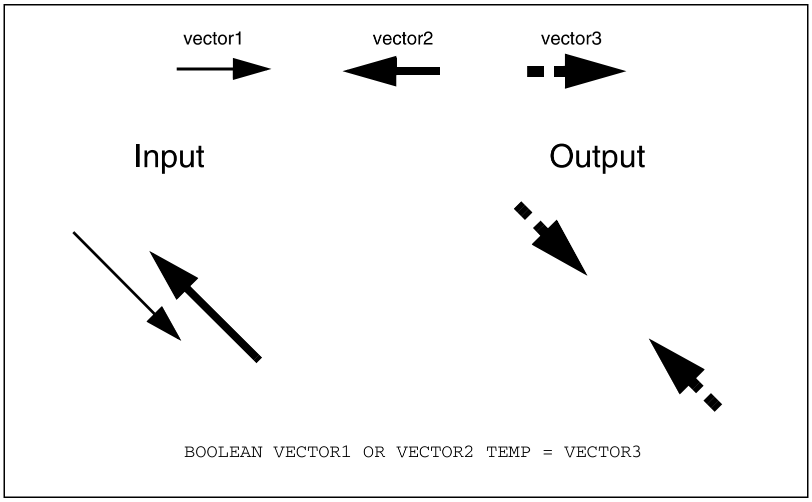
BOOLEAN OR

The BOOLEAN OR command processes vectors according to the combination described in Table 3-6. The example that follows the table explains the combination. For more information, see “BOOLEANS Used with Directed Vectors” on page 3-17

Table 3-6 BOOLEAN OR Vector Processing Combination

Input One	Input Two	Output
vector	vector	vector

Figure 3-10 BOOLEAN OR



When both input group files contain vectors, the output is a vector.

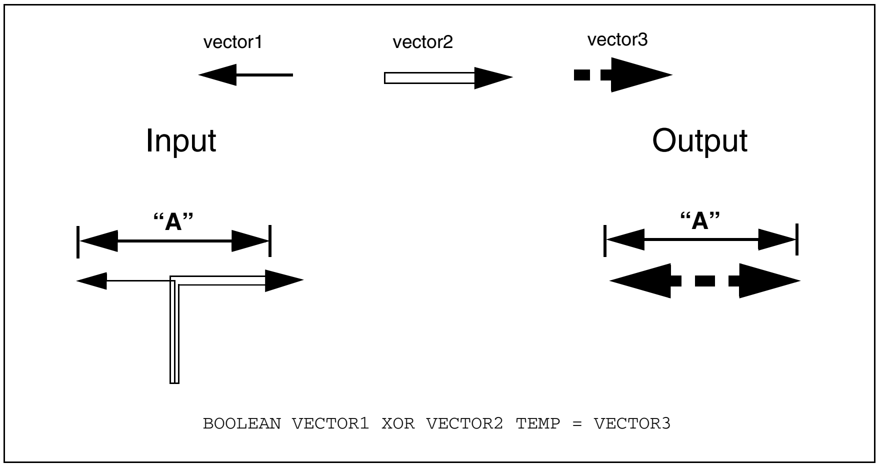
BOOLEAN XOR

The BOOLEAN XOR command processes vectors according to the combination described in Table 3-7. The example that follows the table explains the combination.

Table 3-7 BOOLEAN XOR Vector Processing Combination

Input One	Input Two	Output
vector	vector	vector

Figure 3-11 BOOLEAN XOR



When both input group files contain vectors, the output is a vector.

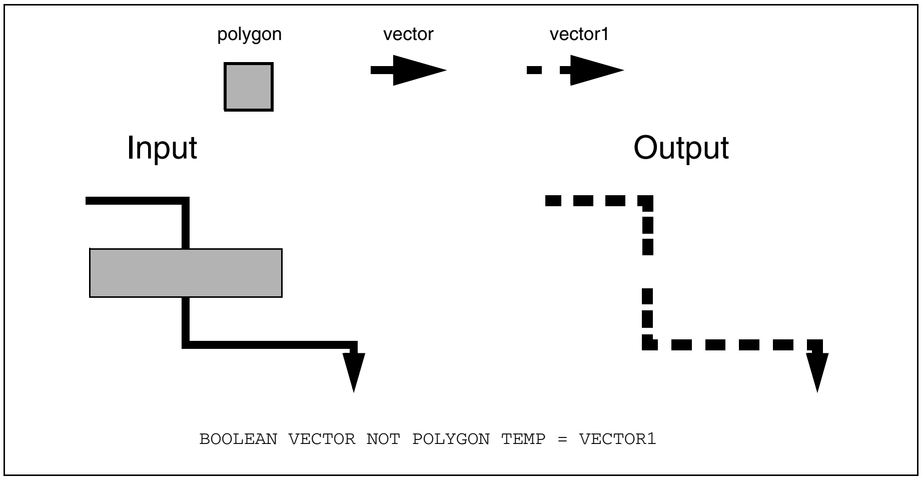
BOOLEAN NOT

The BOOLEAN NOT command processes vectors according to the combinations described in Table 3-8. The examples that follow the table explain the combinations. For more information, see “BOOLEANS Used with Directed Vectors” on page 3-17

Table 3-8 BOOLEAN NOT Vector Processing Combination

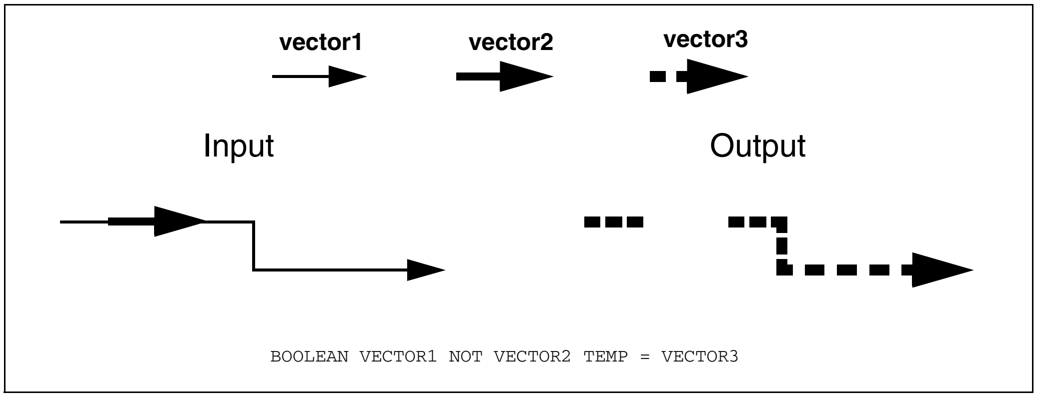
Input One	Input Two	Output
vector	polygon	vector
vector	vector	vector

Figure 3-12 BOOLEAN NOT



When one input group file contains vectors and the other contains polygons, the output is a vector.

Figure 3-13 BOOLEAN NOT Example 2



When both input group files contain vectors, the output is a vector.

BOOLEANS Used with Directed Vectors

Boolean operations used with directed vectors are executed differently than ordinary vectors. When vectors are co-linear, their directions are monitored to determine the output of the specific Boolean operation.

The following table illustrates two overlapping vectors. The top row is designated for undirected vectors. In the second row, the vectors point in opposite directions and their materials are on opposite sides. In the third row, the vectors point in the same direction and their materials are on the same side. The columns represent the outcome from each Boolean operation.

Figure 3-14 BOOLEANS Used with Directed Vectors

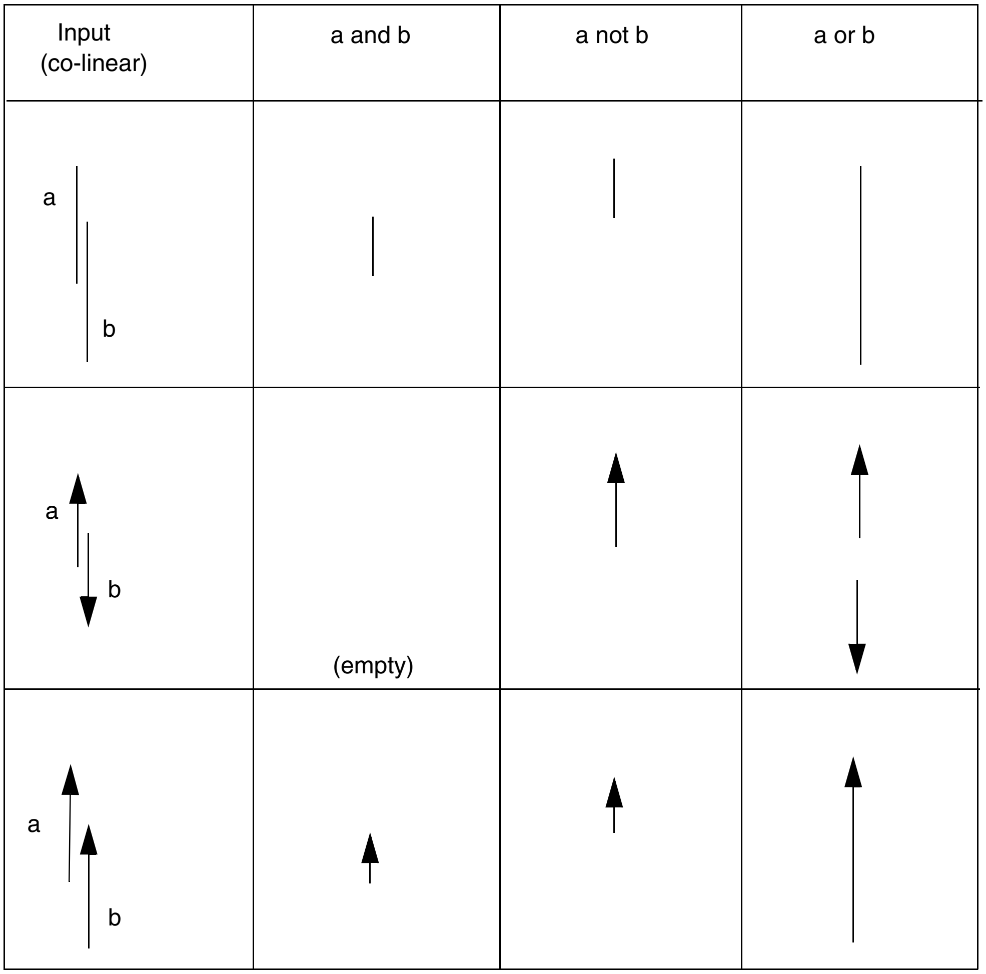
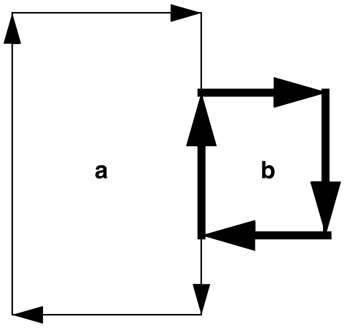


Figure 3-15 BOOLEANS Used with Directed Vectors Example 2



```
BOOLEAN AVECTOR AND BVECTOR {  } TEMP = CVECTOR  OUTPUT = NO OUTPUT
```

Figure 3-16 BOOLEANS Used with Directed Vectors Example 3

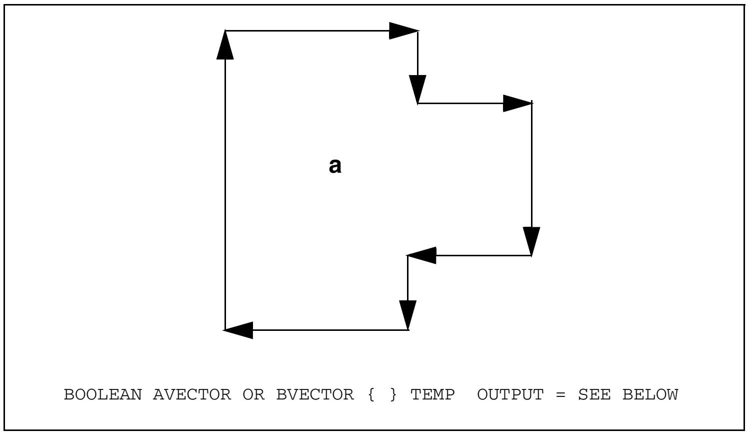
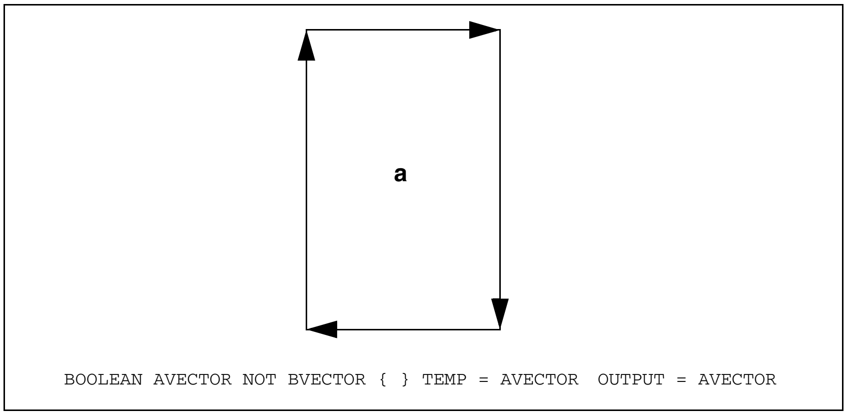


Figure 3-17 BOOLEANS Used with Directed Vectors Example 4


a
BOOLEAN AVECTOR NOT BVECTOR { } TEMP = AVECTOR OUTPUT = AVECTOR

ENCLOSE Command

Vector output can be used in a subsequent ENCLOSE command as a data input layer. However, the ENCLOSE command cannot enclose a vector group by a vector group or a polygon group by a vector group.

Figure 3-18 Example Input Data for ENCLOSE Command

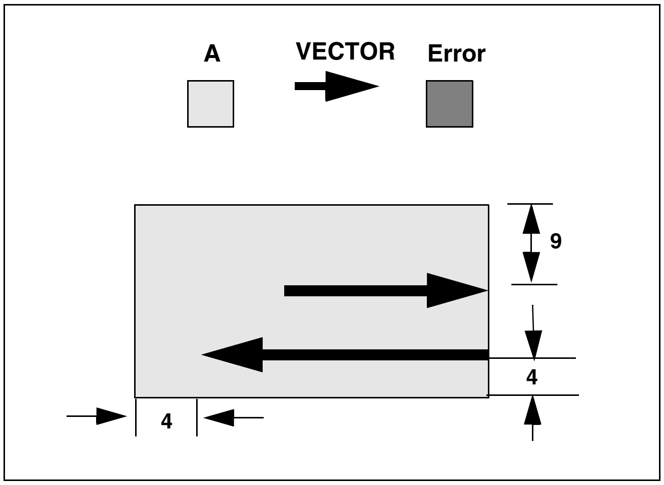
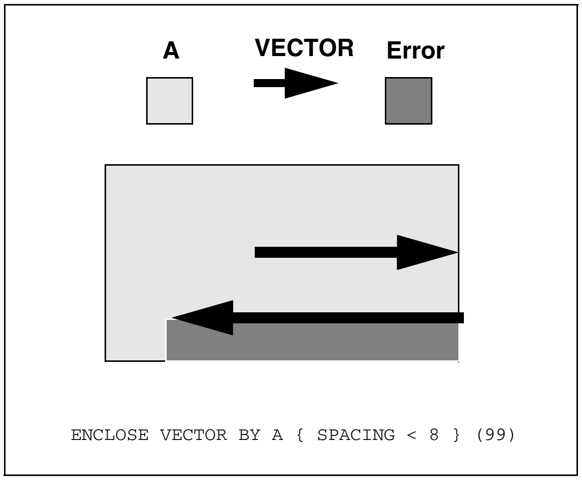


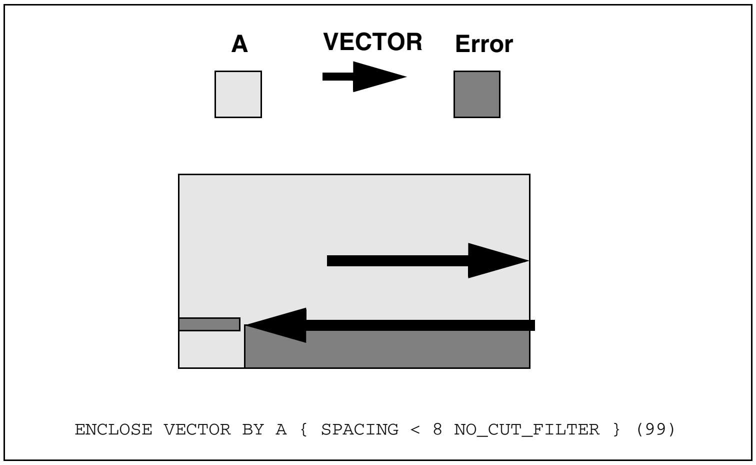
Figure 3-19 Output Data 1 for ENCLOSE Command



```
ENCLOSE VECTOR BY A { SPACING < 8 } (99)
```

The sides of the vector are checked for enclosure violations less than 8 microns with layer A. The errors are flagged and placed in the error hierarchy on layer 99.

Figure 3-20 Output Data 2 for ENCLOSE Command



With the NO_CUT_FILTER option set, the vector is checked for enclosure violations with respect to the endpoints and sides of the vector. The errors are flagged and placed in the error hierarchy on layer 99.

Note:

The ENCLOSE command does not allow the following options to be used for vector processing:

- NODAL

EXTERNAL Command

Vector output can be used in a subsequent EXTERNAL command as a data input layer.

Figure 3-21 Example Input Data for EXTERNAL Command

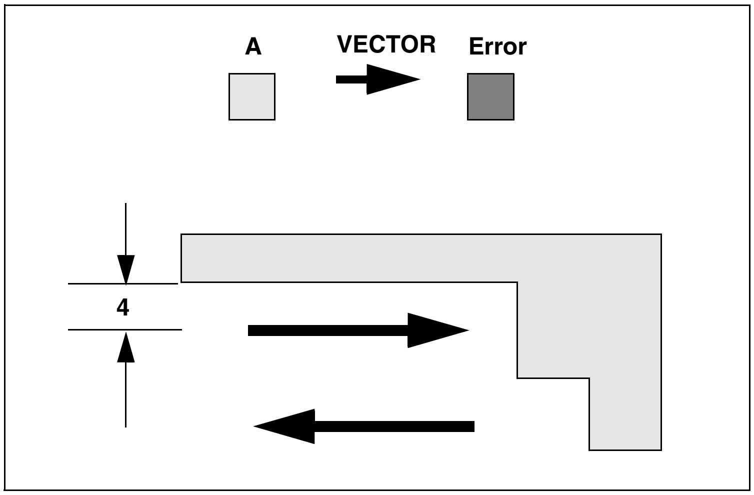
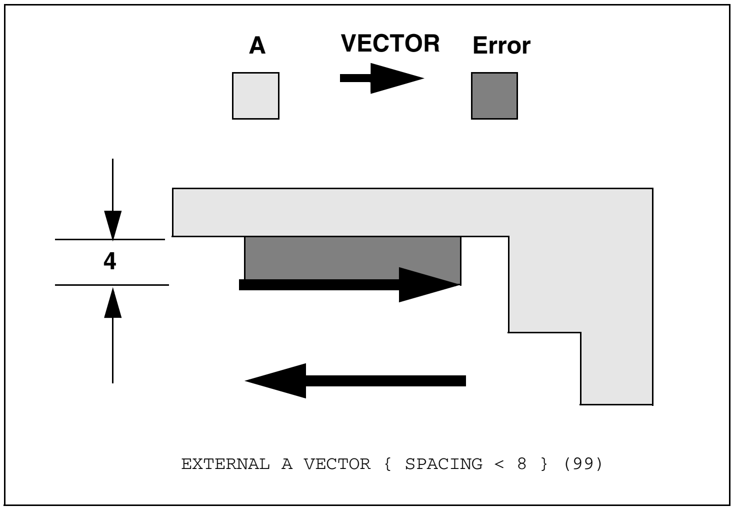
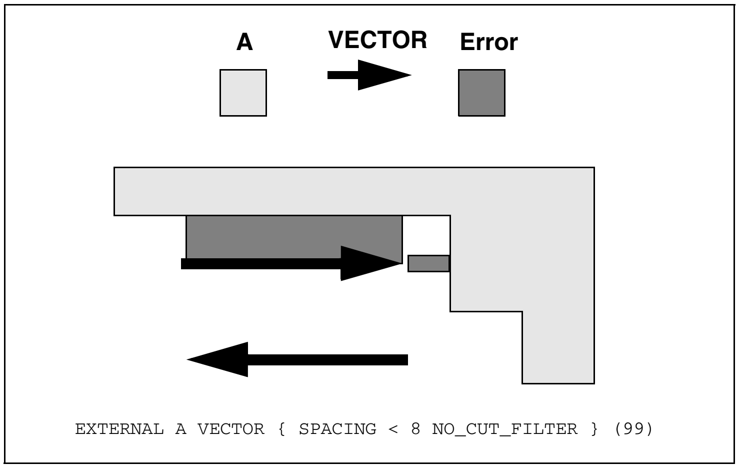


Figure 3-22 Output Data 1 For EXTERNAL Command



The sides of the vector are checked for external violations less than 8 microns with layer A. The errors are flagged and placed in the error hierarchy on layer 99.

Figure 3-23 Output Data 2 for EXTERNAL Command



With the NO_CUT_FILTER option set, the vector is checked for external violations with respect to the endpoints and sides of the vector. The errors are flagged and placed in the error hierarchy on layer 99.

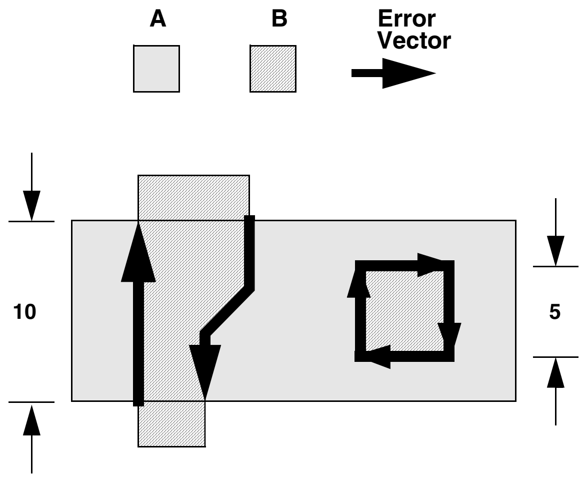
Note:

The EXTERNAL command does not allow the NODAL option to be used for vector processing.

LENGTH Command

The LENGTH command is a hierarchical check that operates on vectors. It flags all contiguous edge intersections whose length is between the minimum and maximum range values inclusively.

Figure 3-24 Example for LENGTH Command



```
CUT A BY B { ENCLOSED = TRUE DIRECTED } TEMP=CUT_RESULT
LENGTH CUT_RESULT { RANGE=[0,25] } (99)
```

All vectors resulting from the CUT command that are between 0 and 25 microns long are flagged and placed in the error hierarchy on layer 99. The length of the vector for enclosed data is the perimeter of that data.

LEVEL Command

The LEVEL command moves data that overlaps hierarchically to a common level. When processing vectors, the LEVEL command moves interacting vectors and polygons to the same level. Table 3-9 describes the combinations of input and output that the LEVEL command processes.

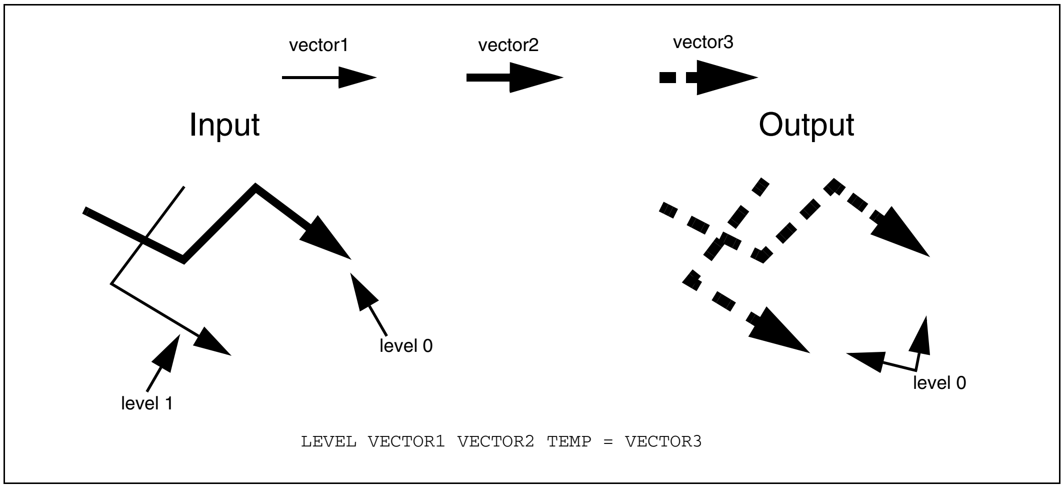
Table 3-9 LEVEL Vector Processing Combination

Input One	Input Two	Output One	Output Two
vector		vector	
vector	polygon	vector	polygon
polygon	vector	polygon	vector

Table 3-9 LEVEL Vector Processing Combination

Input One	Input Two	Output One	Output Two
vector	vector	vector	vector

Figure 3-25 LEVEL Command



NOTCH Command

Vector output can be used in a subsequent NOTCH command as a data input layer.

Figure 3-26 Example Input Data for NOTCH Command

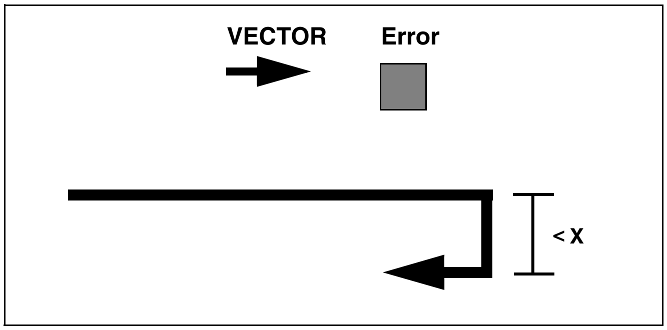
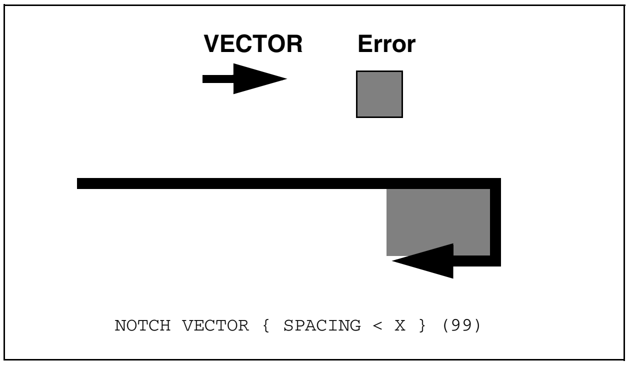


Figure 3-27 Output Data for NOTCH Command



The individual vectors are checked for spacing less than x. Errors are placed in the error hierarchy on layer 99.

REVERSE Command

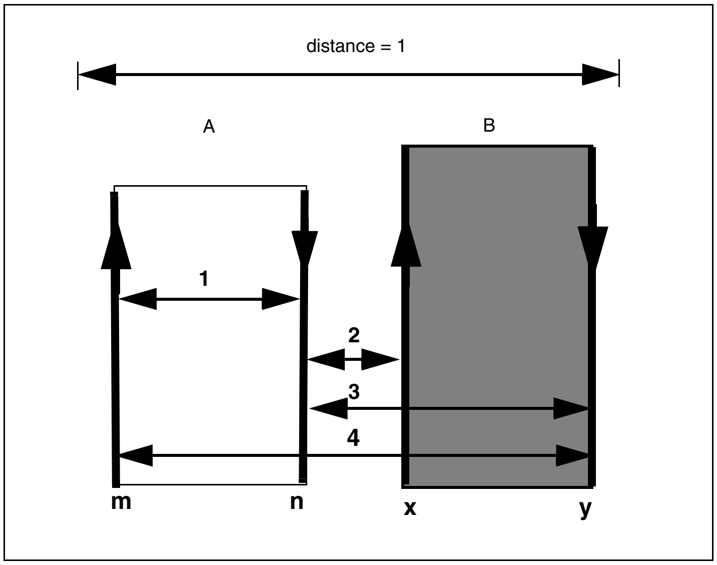
The REVERSE command changes the direction, or which side has material, for all vectors in a layer. This command is used in conjunction with the vector spacing capability of the EXTERNAL, ENCLOSE, and NOTCH commands to provide full capability of dimensional checking for edges (outside-to-outside, inside-to-outside, inside-to-inside). For more information on spacing and edges, see the OUTPUT_EDGES option, *Hercules Reference Manual*, Detailed Commands chapter.

The basic syntax is

```
REVERSE layer1 {} Output_Definition
```

where *layer1* is the vector layer.

Figure 3-28 REVERSE Command



In the preceding example, edges in m and n are derived from polygon A and stored in *aedges*. Edges x and y are derived from polygon B and stored in *bedges*. Distances 1, 2, and 3 can be measured as follows:

- *distance 1*

```
reverse aedges temp = revaedges

external revaedges {spacing < 1 } (100)
```

- *distance 2*

```
external aedges bedges {spacing < 1} (100)
```

- *distance 3*

```
reverse bedges temp = rebbedges

external aedges revbedges {spacing < 1} (100)
```

- *distance 4*

```
reverse aedges temp = revaedges

reverse bedges temp = revbedges

external revaedges revbedges {spacing < 1} (100)
```

SELECT Command

The SELECT command is used to create new layers. When processing vectors, the SELECT command uses the following operators:

- INTERACT
- CUTTING
- ENCLOSING
- INSIDE
- OUTSIDE
- TOUCHING

Table 3-10 describes the combinations of input and output that the SELECT command processes.

Table 3-10 SELECT Vector Processing Combination

Input One	Input Two	Output One
vector	vector	vector
polygon	vector	polygon
vector	polygon	vector

Note:
You cannot SELECT polygon INSIDE vector or SELECT vector ENCLOSING polygon.

Figure 3-29 and Figure 3-30 display all of the input vectors and polygons that are used in the examples for the SELECT command on the following pages. Refer to these figures to see the input that the SELECT command does not choose.

Figure 3-29 SELECT Command Input Vectors

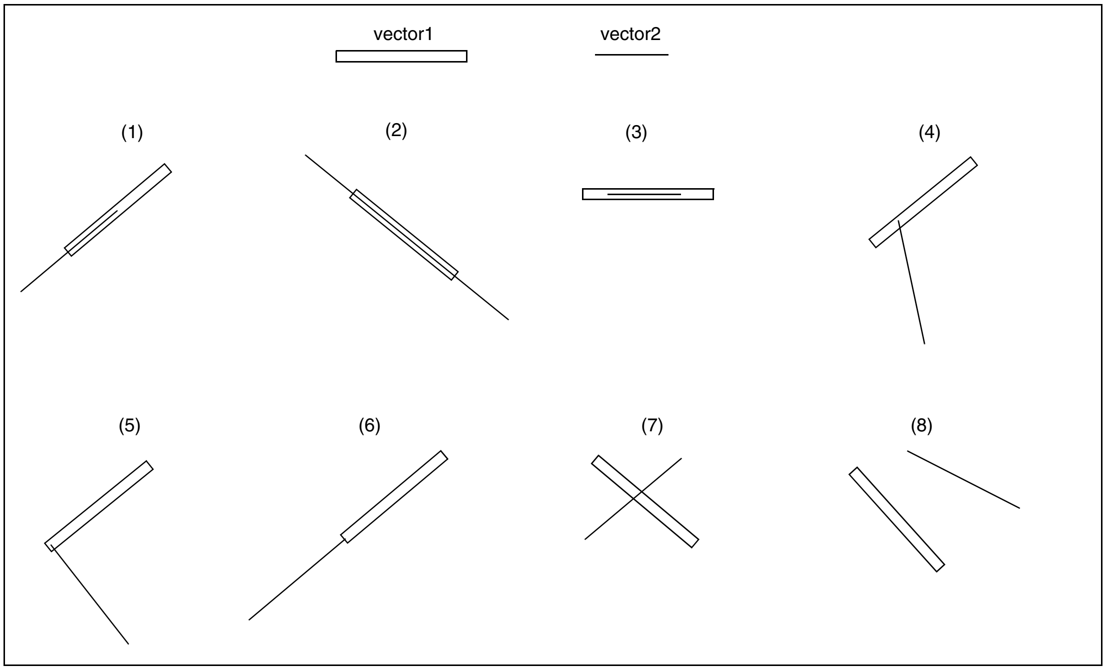
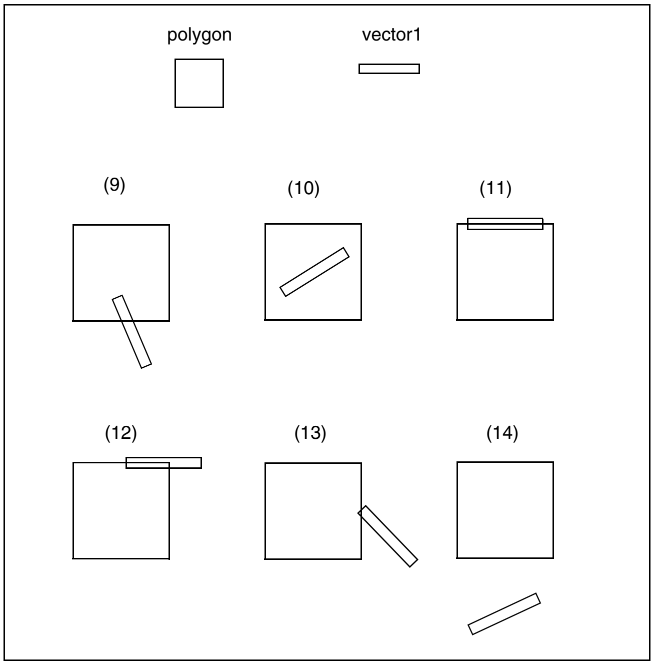


Figure 3-30 SELECT Command Input Polygons and Vectors

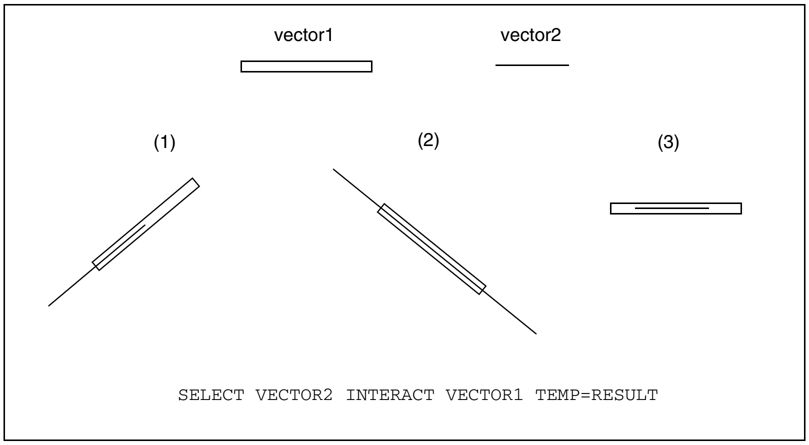


SELECT INTERACT (without POINT_TOUCH Option)

When both input group files contain vectors, SELECT INTERACT (without the POINT_TOUCH option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (1) cut
- (2) enclosing
- (3) inside

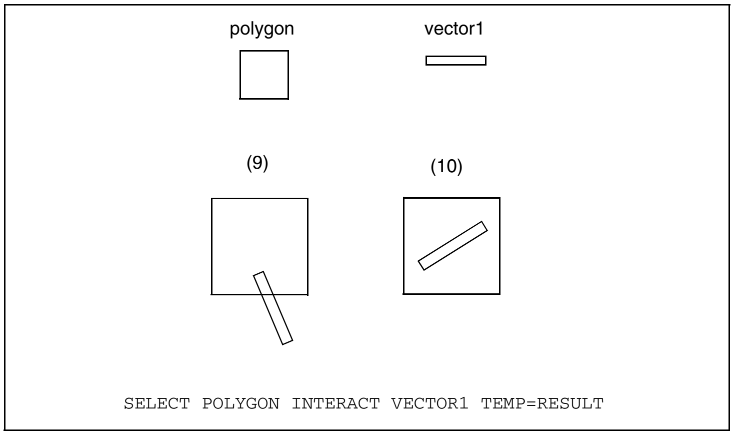
Figure 3-31 SELECT INTERACT (without POINT_TOUCH Option) Example 1



When one input group file contains polygons and the other input group file contains vectors, SELECT INTERACT (without the POINT_TOUCH option) elects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (9) cut
- (10) enclosing

Figure 3-32 SELECT INTERACT (without POINT_TOUCH Option) Example 2

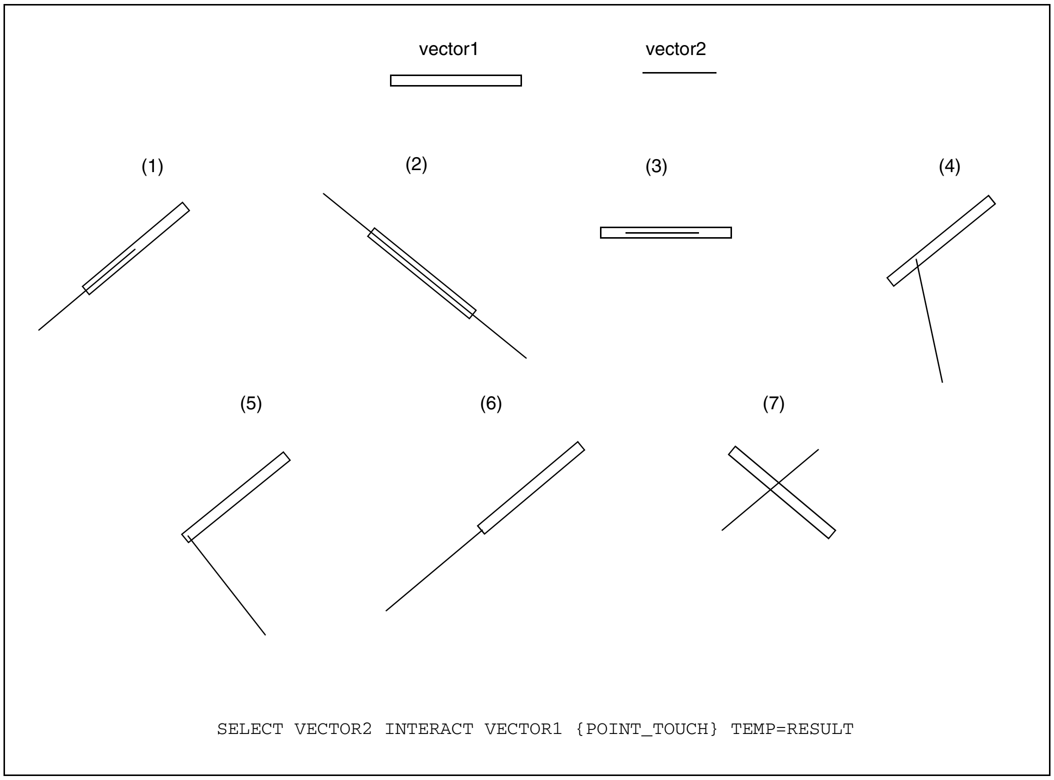


SELECT INTERACT (with POINT_TOUCH Option)

When both input group files contain vectors, SELECT INTERACT (with the POINT_TOUCH option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (1) cut
- (2) enclosing
- (3) inside
- (4) point_touch
- (5) end_point_touch
- (6) point_touch_collinear
- (7) cross

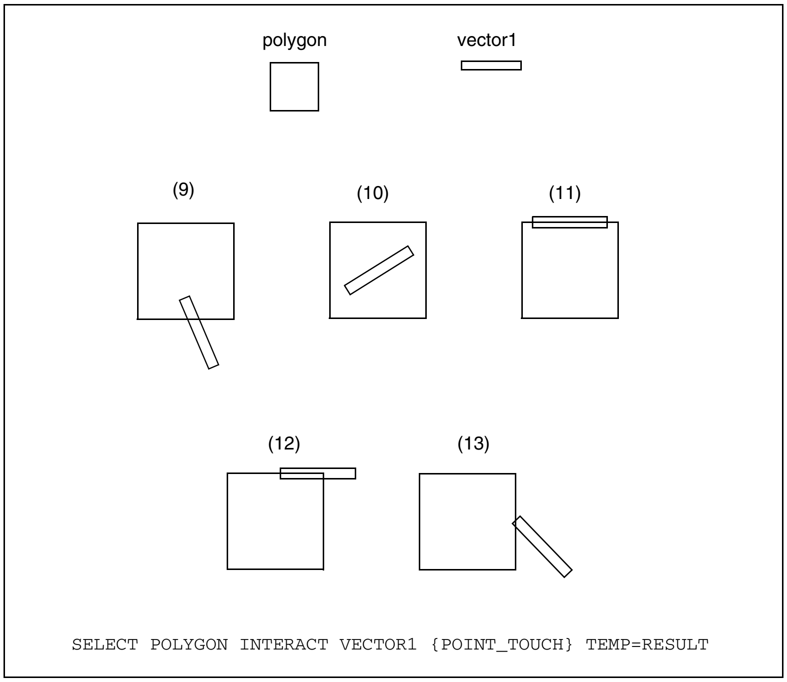
Figure 3-33 SELECT INTERACT (with POINT_TOUCH Option) Example 1



When one input group file contains polygons and the other input group file contains vectors, SELECT INTERACT (with the POINT_TOUCH option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (9) cut
- (10) enclosing
- (11) line_touch
- (12) line_touch
- (13) point_touch

Figure 3-34 SELECT INTERACT (with POINT_TOUCH Option) Example 2



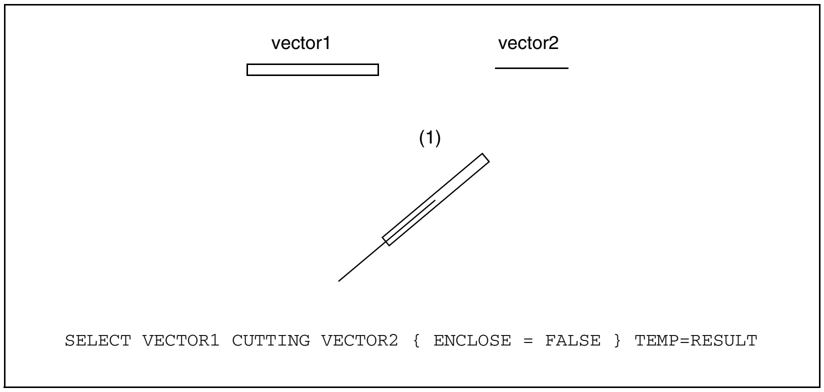
SELECT CUTTING (without ENCLOSE Option)

When both input group files contain vectors, SELECT CUTTING (without the ENCLOSE option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (1) cut

Note:
The ENCLOSE option defaults to TRUE.

Figure 3-35 SELECT CUTTING (without ENCLOSE Option) Example 1

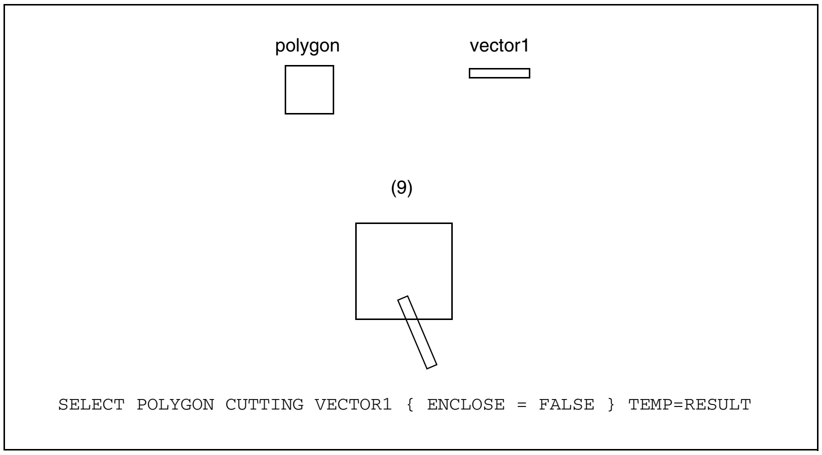

vector1
vector2
(1)
SELECT VECTOR1 CUTTING VECTOR2 { ENCLOSE = FALSE } TEMP=RESULT

When one input group file contains polygons and the other input group file contains vectors, SELECT CUTTING (without the ENCLOSE option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (9) cut

Note:
The ENCLOSE option defaults to TRUE.

Figure 3-36 SELECT CUTTING (without ENCLOSE Option) Example 2


polygon
vector1
(9)
SELECT POLYGON CUTTING VECTOR1 { ENCLOSE = FALSE } TEMP=RESULT

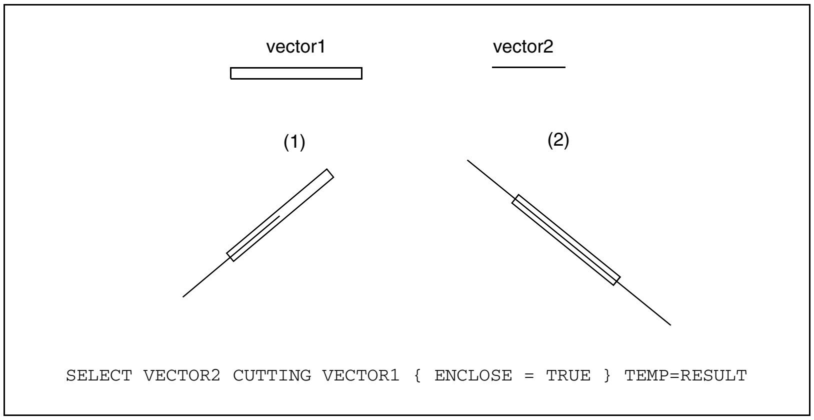
SELECT CUTTING (with ENCLOSE Option)

When both input group files contain vectors, SELECT CUTTING (with the ENCLOSE option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (1) cut
- (2) enclosing

Note:
The ENCLOSE option defaults to TRUE.

Figure 3-37 SELECT CUTTING (with ENCLOSE Option) Example 1

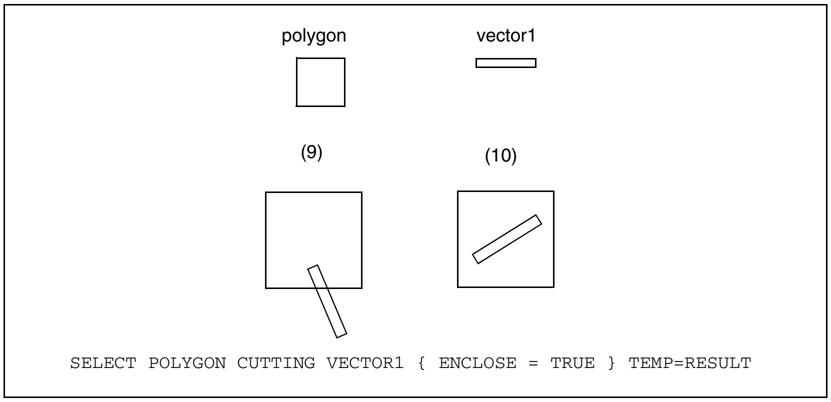

vector1
vector2
(1)
(2)
SELECT VECTOR2 CUTTING VECTOR1 { ENCLOSE = TRUE } TEMP=RESULT

When one input group file contains polygons and the other input group file contains vectors, SELECT CUTTING (with the ENCLOSE option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (9) cut
- (10) enclosing

Note:
The ENCLOSE option defaults to TRUE.

Figure 3-38 SELECT CUTTING (with ENCLOSE Option) Example 2

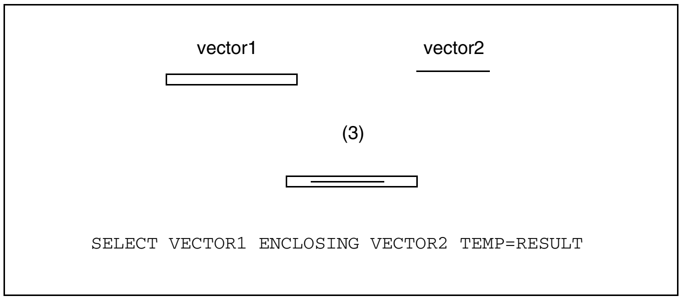

polygon
vector1
(9)
(10)
SELECT POLYGON CUTTING VECTOR1 { ENCLOSE = TRUE } TEMP=RESULT

SELECT ENCLOSING

When both input group files contain vectors, SELECT ENCLOSING selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (3) enclosing

Figure 3-39 SELECT ENCLOSING Example 1

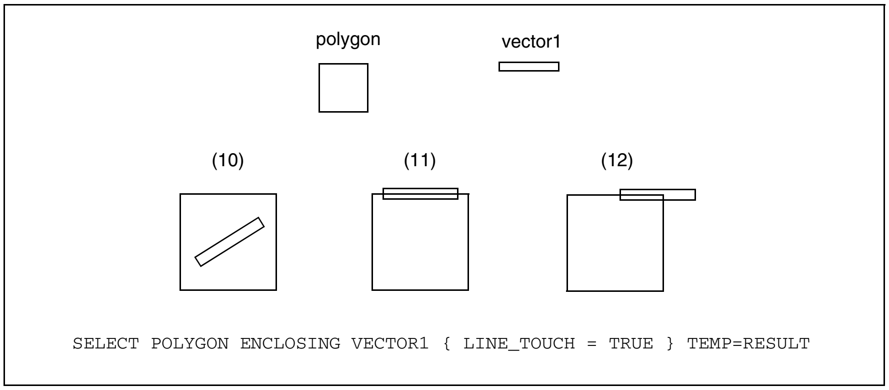


When one input group file contains polygons and the other input group file contains vectors, SELECT ENCLOSING selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (10) enclosing
- (11) line_touch
- (12) line_touch

Note:
When selecting polygons, the LINE_TOUCH option defaults to TRUE.

Figure 3-40 SELECT ENCLOSING Example 2

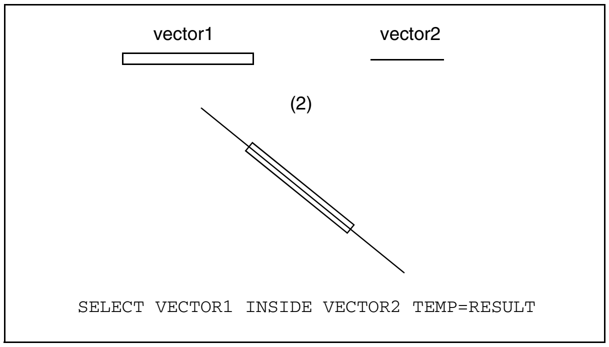

polygon
vector1
(10)
(11)
(12)
SELECT POLYGON ENCLOSING VECTOR1 { LINE_TOUCH = TRUE } TEMP=RESULT

SELECT INSIDE

When both input group files contain vectors, SELECT INSIDE selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (2) inside

Figure 3-41 SELECT INSIDE Example 1

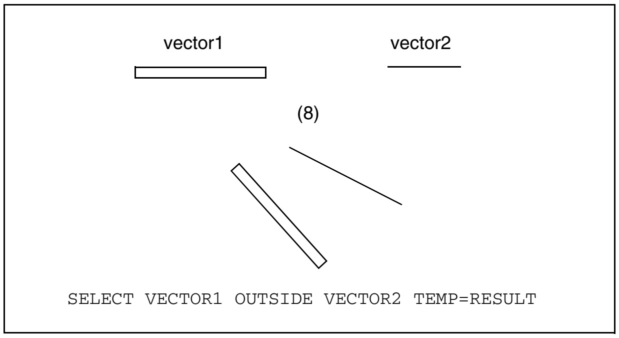


SELECT OUTSIDE (without POINT_TOUCH Option)

When both input group files contain vectors, SELECT OUTSIDE (without the POINT_TOUCH option) elects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (8) no interaction

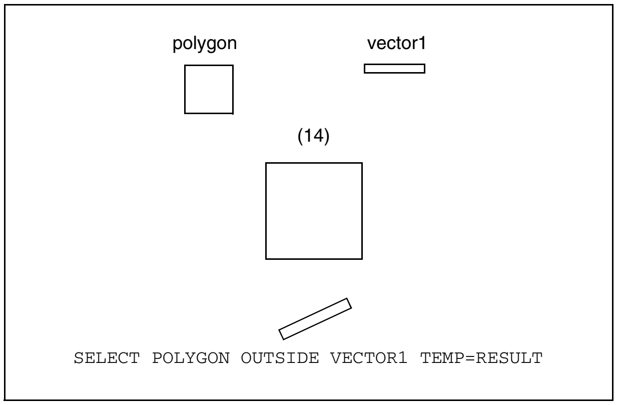
Figure 3-42 SELECT OUTSIDE (without POINT_TOUCH Option) Example 1



When one input group file contains polygons and the other input group file contains vectors, SELECT OUTSIDE (without the POINT_TOUCH option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (14) no interaction

Figure 3-43 SELECT OUTSIDE (without POINT_TOUCH Option) Example 2

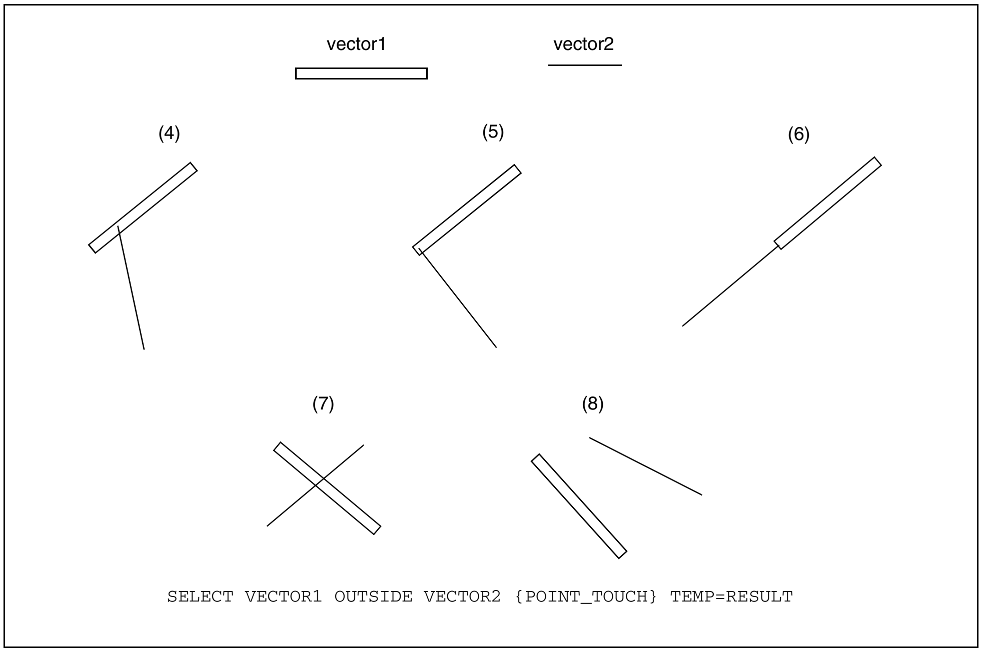


SELECT OUTSIDE (with POINT_TOUCH Option)

When both input group files contain vectors, SELECT OUTSIDE (with the POINT_TOUCH option) selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (4) point_touch
- (5) end_point_touch
- (6) point_touch_collinear
- (7) cross
- (8) no interaction

Figure 3-44 SELECT OUTSIDE (with POINT_TOUCH Option) Example 1

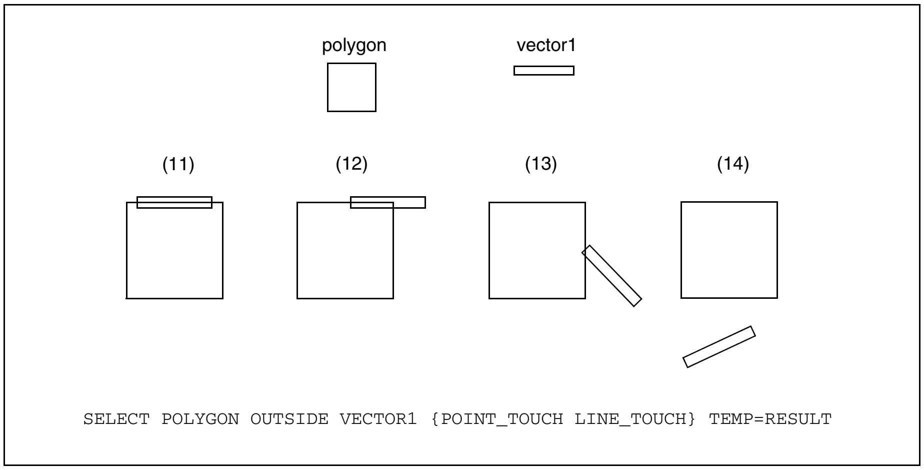


When one input group file contains polygons and the other input group file contains vectors, SELECT OUTSIDE (with the POINT_TOUCH and LINE_TOUCH options)

- (11) line_touch
- (12) line_touch
- (13) point_touch
- (14) no interaction

Note:
When selecting polygons, the LINE_TOUCH option defaults to TRUE.

Figure 3-45 SELECT OUTSIDE (with POINT_TOUCH Option) Example 2

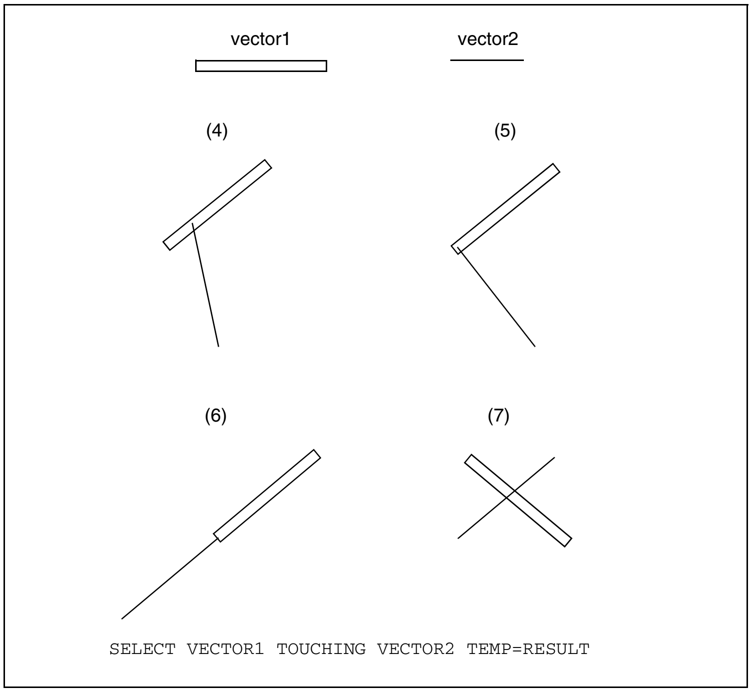

polygon
vector1
(11)
(12)
(13)
(14)
SELECT POLYGON OUTSIDE VECTOR1 {POINT_TOUCH LINE_TOUCH} TEMP=RESULT

SELECT TOUCHING

When both input group files contain vectors, SELECT TOUCHING selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (4) point_touch
- (5) end_point_touch
- (6) point_touch_collinear
- (7) cross

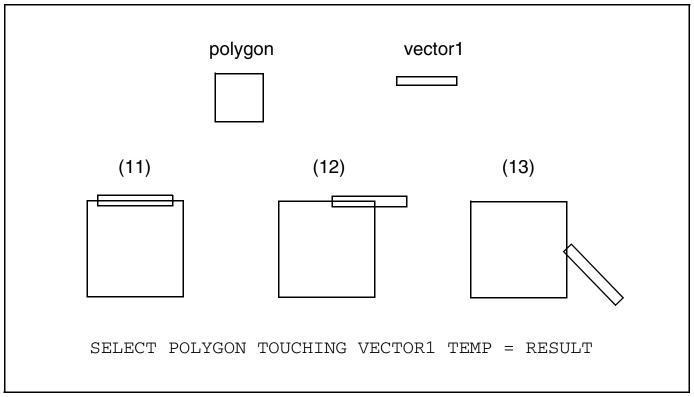
Figure 3-46 SELECT TOUCHING Example 1



When one input group file contains polygons and the other input group file contains vectors, SELECT TOUCHING selects the following vector interactions from the input in Figure 3-29 and Figure 3-30:

- (11) line_touch
- (12) line_touch
- (13) point_touch

Figure 3-47 SELECT TOUCHING Example 2



SIZE Command

The SIZE command creates polygons from vectors. Directed vectors can be undersized or oversized. By default, this is a hierarchical operation, but you can set it to a cell-level operation by setting CELL_LEVEL = TRUE.

When oversizing a directed vector in which the corner is formed at the cell-level, a corner will be formed correctly. (See Figure 3-48.) When oversizing a directed vector in which a corner is formed hierarchically, path endpoints are not expanded and the oversized vector will have a missing corner. (See Figure 3-49.)

Figure 3-48 OVERSIZING a Directed Vector with a Corner Formed at Cell-Level

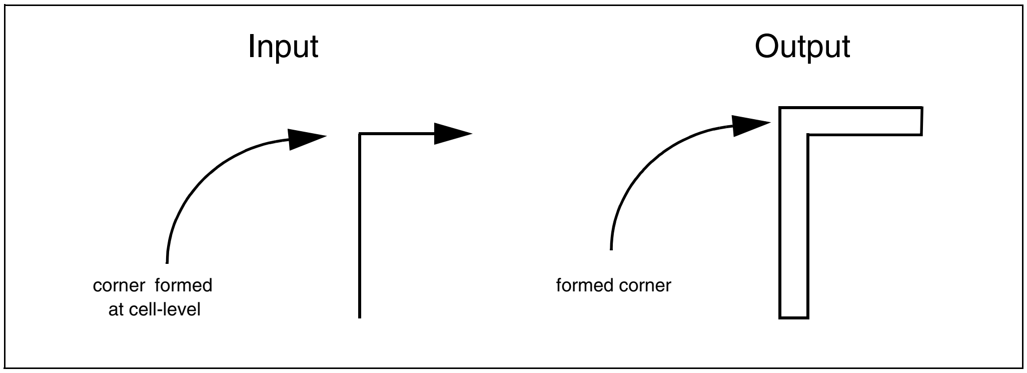
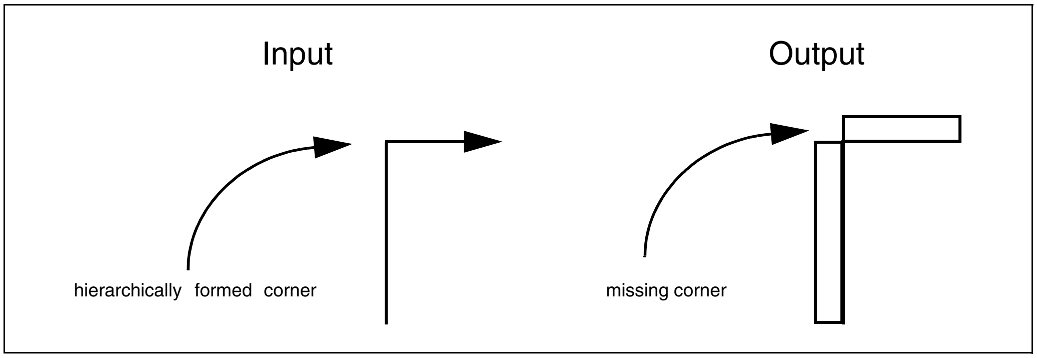


Figure 3-49 OVERSIZING a Directed Vector with a Hierarchically Formed Corner



A directed vector maintains its original relationship to a polygon. Therefore, a directed vector has an inside and an outside. When a vector is undersized, the vector is sized toward the direction of the polygon from which the vector was created (the “inside” of the vector).

Figure 3-50 Undersized Vector

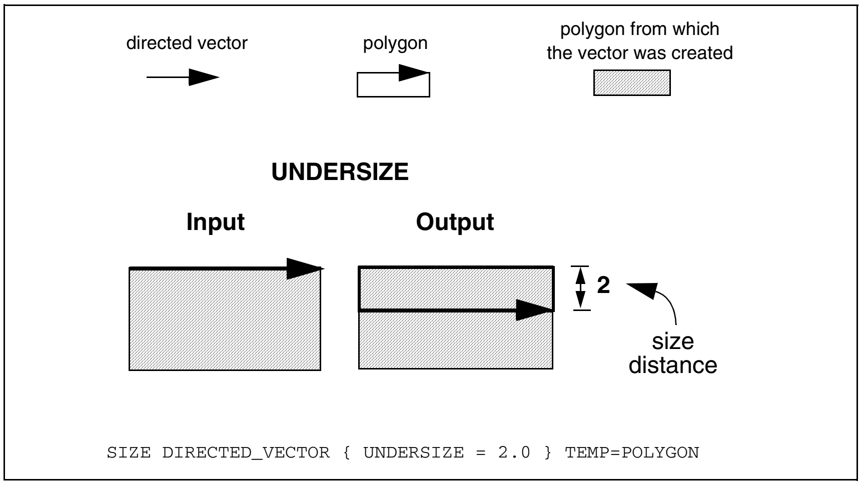
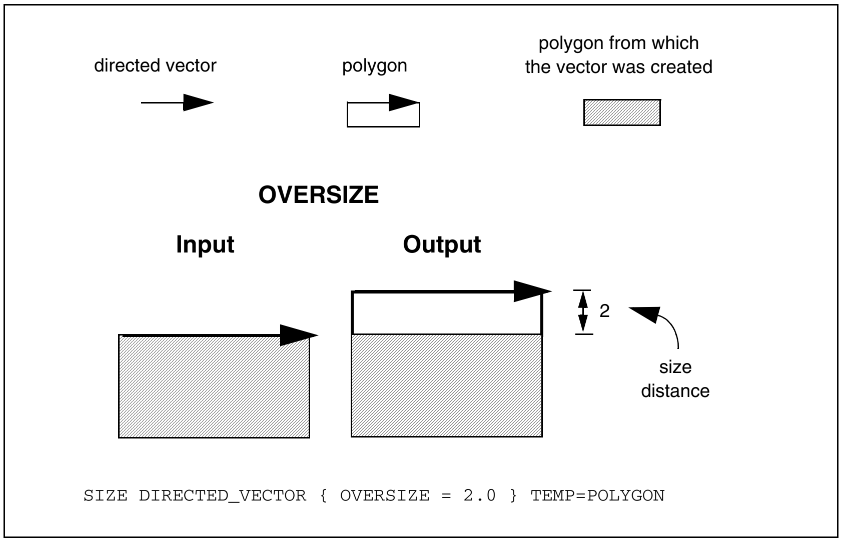
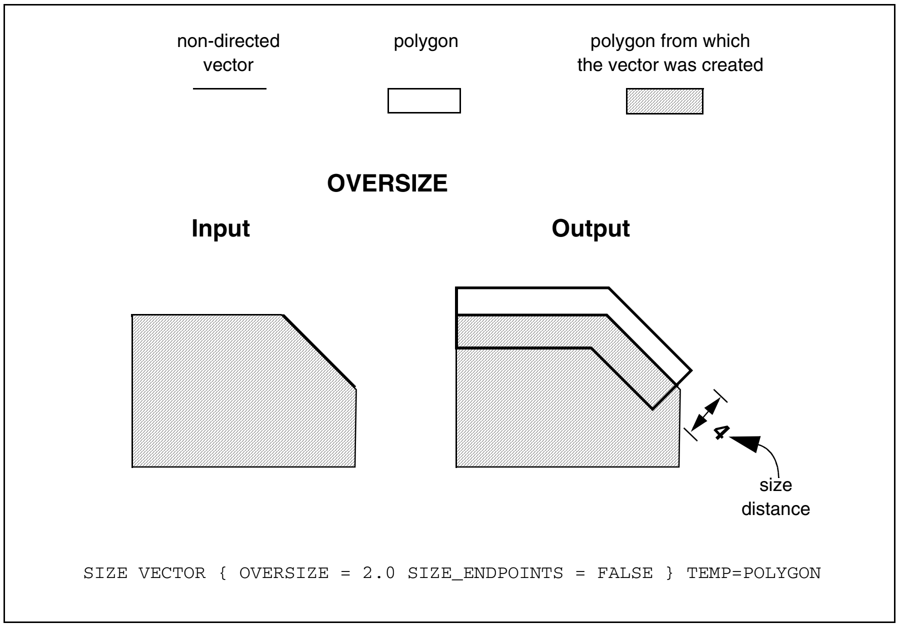

directed vector
polygon
polygon from which the vector was created
UNDERSIZE
Input
Output
2
size distance
SIZE DIRECTED_VECTOR { UNDERSIZE = 2.0 } TEMP=POLYGON

Figure 3-51 Oversized Vector


directed vector
polygon
polygon from which the vector was created
OVERSIZE
Input
Output
2
size distance
SIZE DIRECTED_VECTOR { OVERSIZE = 2.0 } TEMP=POLYGON

When a vector is oversized, the vector is sized away from the direction of the polygon from which the vector was created (the “outside” of the vector).

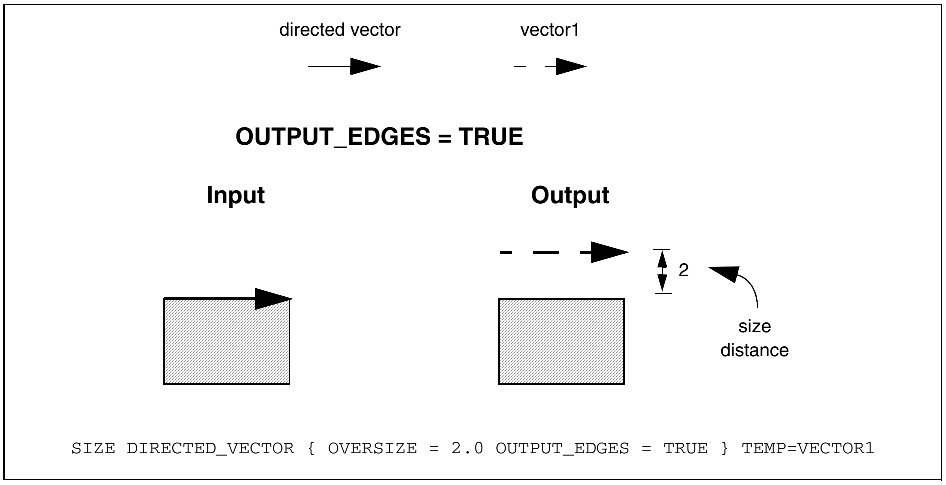
Figure 3-52 Nondirected Oversized Vector



```
SIZE VECTOR { OVERSIZE = 2.0 SIZE_ENDPOINTS = FALSE } TEMP=POLYGON
```

Nondirected (normal) vectors can only be oversized.

Figure 3-53 Vector with OUTPUT_EDGES Option



```
SIZE DIRECTED_VECTOR { OVERSIZE = 2.0 OUTPUT_EDGES = TRUE } TEMP=VECTOR1
```

The OUTPUT_EDGES option can be used on directed vectors to produce vector output.

VECTOR Processing Summary

Table 3-11 and Table 3-12 display the valid combinations of input (vectors and polygons) for the vector processing commands.

Table 3-11 Processing Vectors with Data Creation Commands

COMMAND	LAYER1	LAYER2	OUTPUT1	OUTPUT2
CUT	polygon	polygon	vector	
VECTORIZE	polygon	N/A	vector	
SELECT	polygon	polygon	polygon	
SELECT	vector	vector	vector	
SELECT	polygon	vector	polygon	
SELECT	vector	polygon	vector	
AND	polygon	polygon	polygon	
AND	polygon	vector	vector	
AND	vector	vector	vector	
OR	polygon	polygon	polygon	
OR	vector	vector	vector	
XOR	polygon	polygon	polygon	
XOR	vector	vector	vector	
NOT	polygon	polygon	polygon	
NOT	vector	polygon	vector	
NOT	vector	vector	vector	
SIZE	polygon	N/A	polygon	
SIZE	vector	N/A	polygon	
LENGTH	vector	N/A	vector	
LEVEL	polygon		polygon	

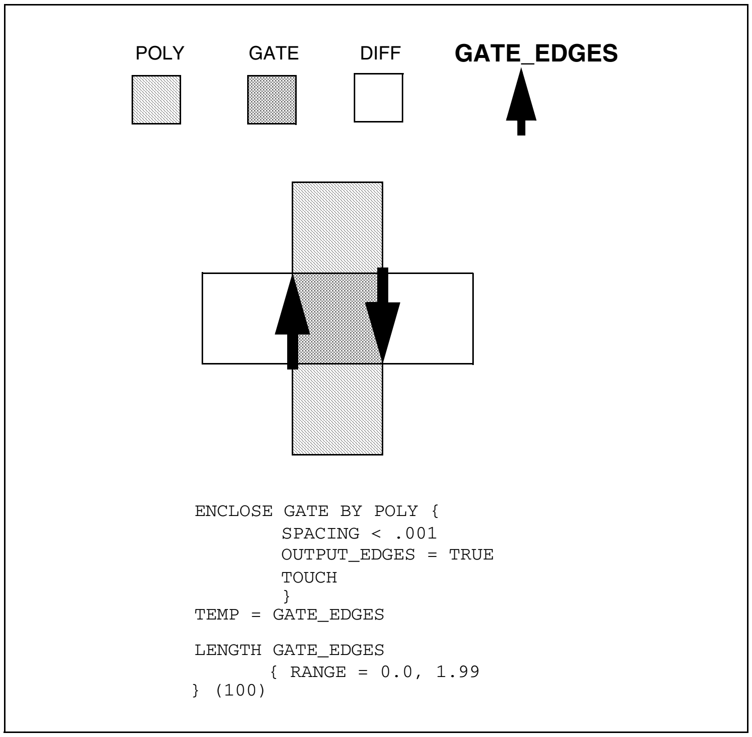
Table 3-11 Processing Vectors with Data Creation Commands

COMMAND	LAYER1	LAYER2	OUTPUT1	OUTPUT2
LEVEL	vector		vector	
LEVEL	polygon	polygon	polygon	polygon
LEVEL	vector	polygon	vector	polygon
LEVEL	polygon	vector	polygon	vector
LEVEL	vector	vector	vector	vector

Vector Examples

The following examples display two common uses of the vector options. For more information on the ENCLOSE command, see the *Hercules Reference Manual*, Detailed Commands chapter.

Figure 3-54 Gate Width Checking



```
ENCLOSE GATE BY POLY {
        SPACING < .001
        OUTPUT_EDGES = TRUE
        TOUCH
        }
TEMP = GATE_EDGES

LENGTH GATE_EDGES
       { RANGE = 0.0, 1.99
} (100)
```

Gate widths less than 2 are flagged and placed on layer 100 in the error hierarchy.

Projection and Errors

This section shows how to control error output using the EDGE_METRIC option. The EDGE_METRIC option can be used to attain different types of error projection capabilities with Hercules dimensional commands

EDGE_METRIC

The EDGE_METRIC option provides control over the violation edges when using the OUTPUT_EDGES specification. There are three settings for the EDGE_METRIC option: OPPOSITE, RADIAL, and SQUARE.

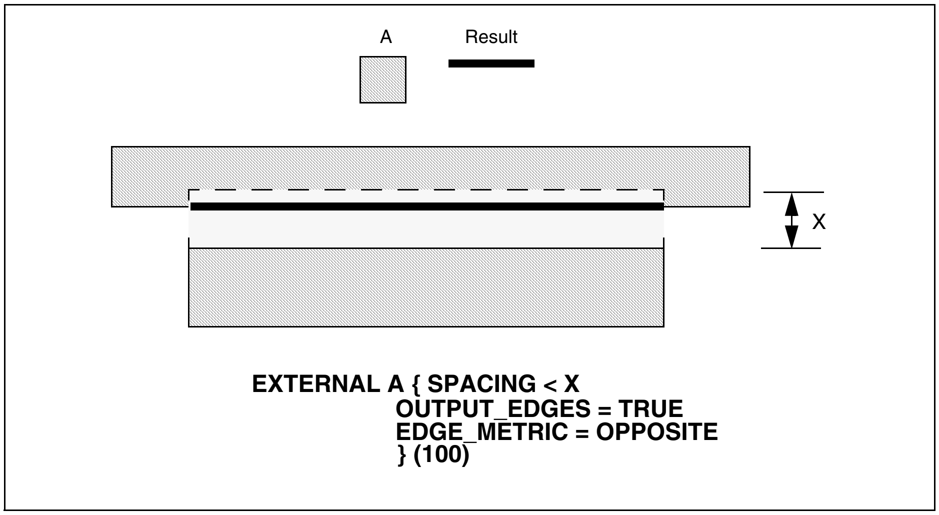
The following diagrams show the check regions and edge output for various examples created when using the three settings. The dotted region consists of all points in the half-plane, outside of the edges that are within the check value. The dark lines show the violating edge, which is the output. The portion of the edge that is actually in error is flagged. The following examples are for the EXTERNAL check. When applying this concept to an inside edge check, the term “outside” means the inside of the edge; it is basically the check side of the edge.

`Syntax:  EDGE_METRIC =` *`OPPOSITE| RADIAL|SQUARE`*

`Default:` *`OPPOSITE`*

Note:
To provide readable diagrams, only one or two projections are shown in each figure.

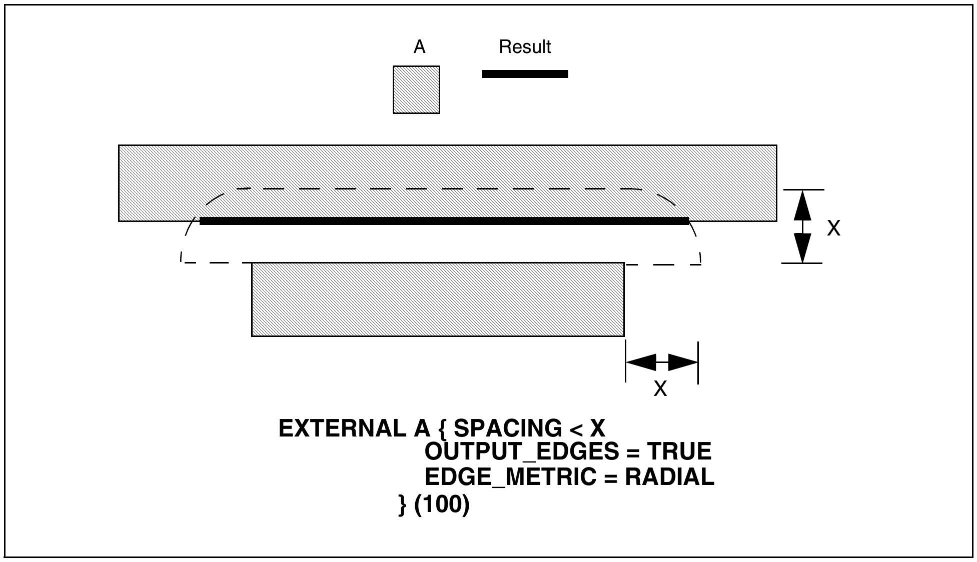
Figure 3-55 Example 1: OPPOSITE



```
EXTERNAL A { SPACING < X
             OUTPUT_EDGES = TRUE
             EDGE_METRIC = OPPOSITE
             } (100)
```

This is the default setting. Only the portion opposite the check region is output. The perpendicular projection between the edges defines the check region.

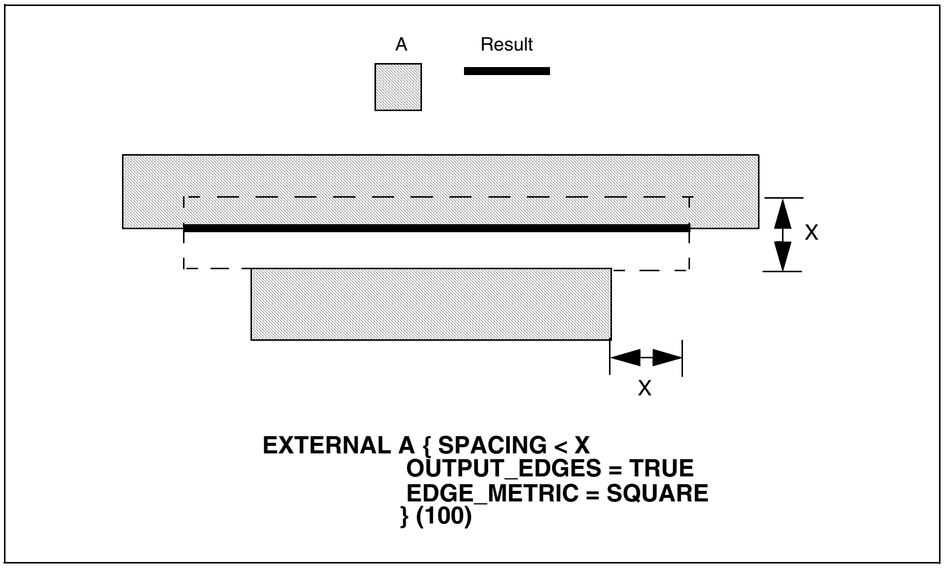
Figure 3-56 Example 2: RADIAL



```
EXTERNAL A { SPACING < X
                  OUTPUT_EDGES = TRUE
                  EDGE_METRIC = RADIAL
              } (100)
```

This is the portion of the edge that is within the specified distance of the check edge. The endpoints of the edge are extended by the check value and the region is formed with a radius of the check value.

Figure 3-57 Example 3: SQUARE



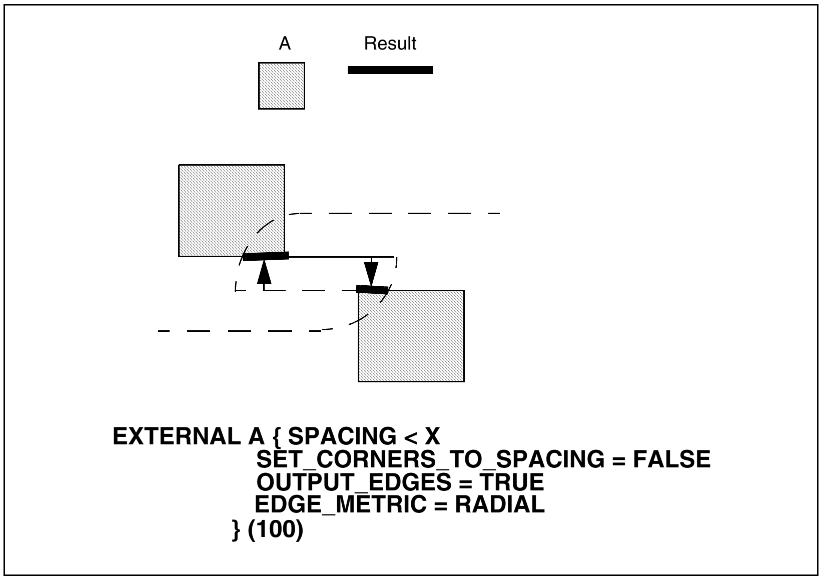
```
EXTERNAL A { SPACING < X
                  OUTPUT_EDGES = TRUE
                  EDGE_METRIC = SQUARE
               } (100)
```

This setting uses the same edge extension but forms a square on the extension thus providing a rectangular check area.

Non-Projecting Edges

When EDGE_METRIC = RADIAL or SQUARE, non-projecting edges are also checked for violations, with the same check regions as projecting edges.

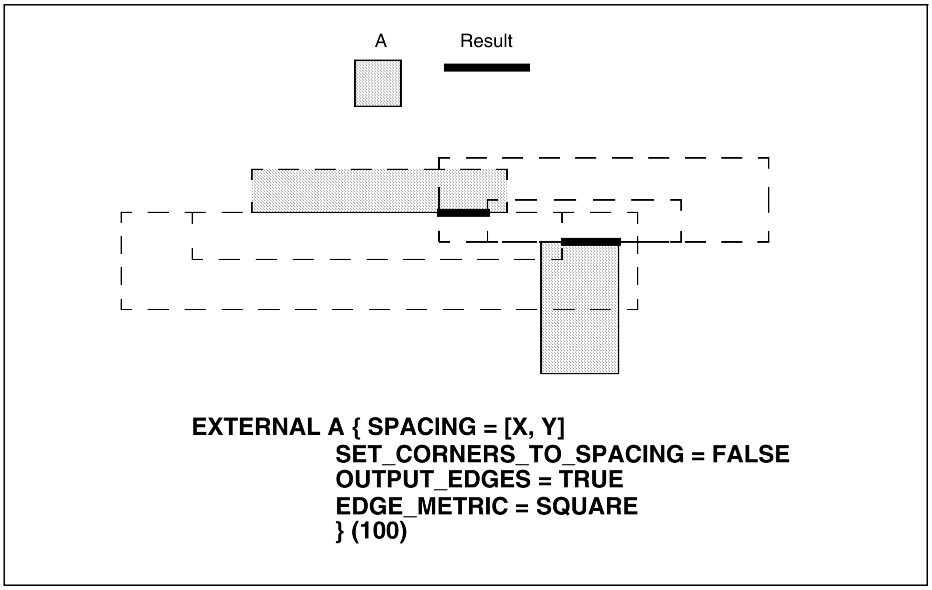
Figure 3-58 Example 4: EDGE_METRIC=RADIAL



```
EXTERNAL A { SPACING < X
             SET_CORNERS_TO_SPACING = FALSE
             OUTPUT_EDGES = TRUE
             EDGE_METRIC = RADIAL
           } (100)
```

The preceding diagram is an example of using an EDGE_METRIC = RADIAL check. The dotted regions show the check region and the dark edge is the violation output.

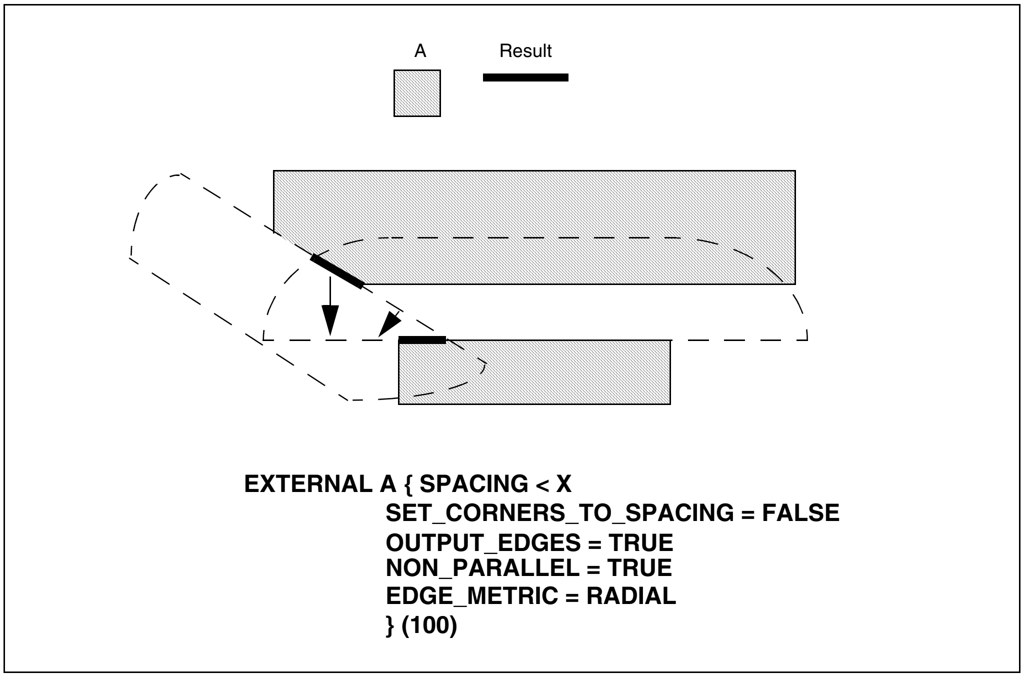
Figure 3-59 Example 5: EDGE_METRIC=SQUARE



```
EXTERNAL A { SPACING = [X, Y]
             SET_CORNERS_TO_SPACING = FALSE
             OUTPUT_EDGES = TRUE
             EDGE_METRIC = SQUARE
             } (100)
```

The preceding diagram is an example of a range check on non-projecting edges, using EDGE_METRIC = SQUARE.

Figure 3-60 Example 6: Non-parallel, non-projecting



```
EXTERNAL A { SPACING < X
             SET_CORNERS_TO_SPACING = FALSE
             OUTPUT_EDGES = TRUE
             NON_PARALLEL = TRUE
             EDGE_METRIC = RADIAL
             } (100)
```

The preceding diagram illustrates a non-parallel, non-projecting example.

Rules for Non-Projecting Edges

For two edges to be checked:

Each edge must intersect the half-plane which consists of all points on the check side of the line, and is determined by the other edge.

The angle between the check sides of the two edges must be less than 90 degrees.

Note:

> It is highly recommended that the corner checks be turned OFF when using the EDGE_METRIC = RADIAL or SQUARE (use SET_CORNERS_TO_SPACING = FALSE). In addition, the corner checks disregard edge orientation.

Corner Checking

Several options can be used with all dimensional check commands that do corner checking: ENCLOSE, Multiple Rules ENCLOSE, EXTERNAL, INSIDE_EDGE, Multiple Rules INSIDE_EDGE, INTERNAL, and NOTCH. These options are specified by adding them to the bracket-enclosed option list of each desired command.

Table 3-12 Dimensional Corner Checking Command Options

Option	Description	Default
BOX_CORNER	Uses box-type corner checking	FALSE
CONVEX_TO_CONVEX_FILTER	Maximum length filter for convex_to_convex error vectors	filter_vector
CONVEX_TO_EDGE_FILTER	Maximum length filter for convex_to_edge error vectors	filter_vector
FILTER_VECTOR	Enables maximum length filter on error vectors	NONE
SQUARE_CORNER	Checks only 90-degree corners	FALSE
TOGGLE_BOX_CORNER	Toggles setting of BOX_CORNER	box_corner
TOGGLE_SQR_CORNER	Toggles setting of SQUARE_CORNER	square_corner

Note:

If specifying either CONVEX_TO_CONVEX_FILTER or CONVEX_TO_EDGE_ FILTER, the corresponding command-specific option *must* also be specified.

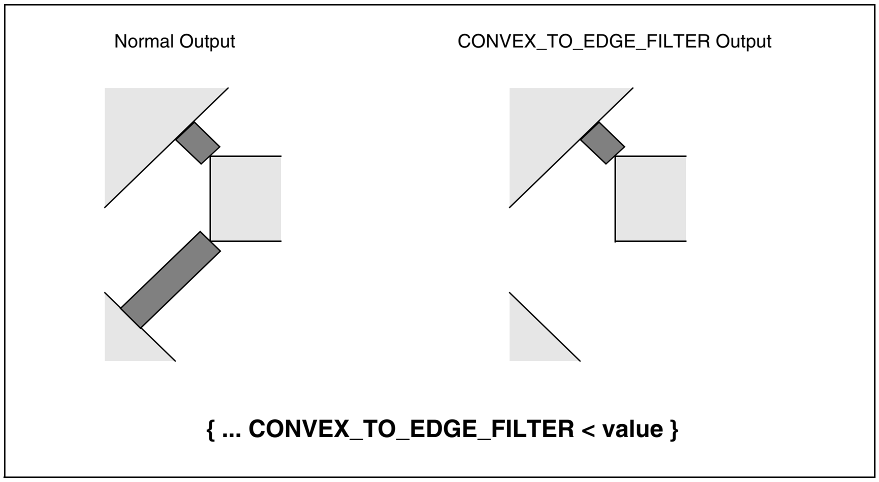
- BOX_CORNER. Corner-to-edge and corner-to-corner spacing measurements can be made using two methods: radial arc or box corner. The radial arc method is the default checking method used by Hercules. Box corner checking is an alternative corner checking method that is useful when process specification rules are stated in terms of box corner spacings. This option is not available with the CUT, MOSCHECK, and RESCHECK commands.

Note:

This option is also available as a dimensional option. see the *Hercules Reference Manual*, Detailed Commands chapter. Setting this option locally overrides the global specification.

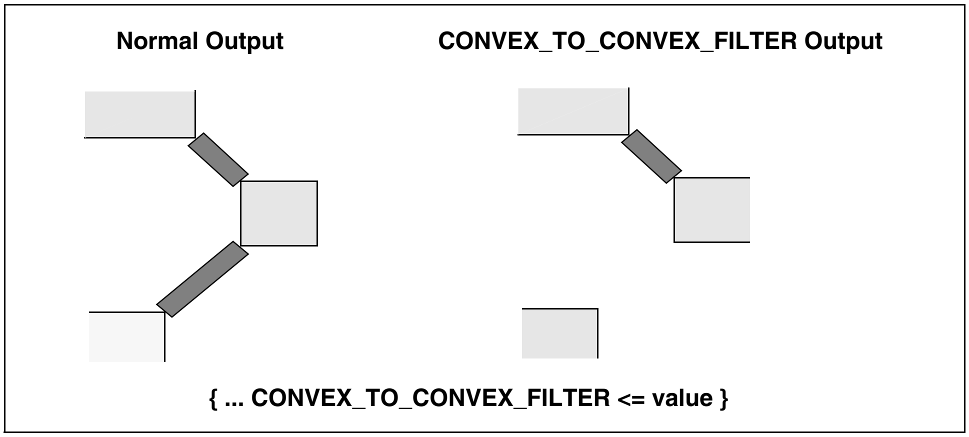
- CONVEX_TO_EDGE_FILTER. Eliminates *all* error vectors longer than the specified length. If you want to filter only the error vectors from a CONVEX_TO_EDGE check, or if a different filter value is required for CONVEX_TO_EDGE error vectors, you can use the CONVEX_TO_EDGE_FILTER option. For this option, the relational operators < and <= yield different results. This option is not available with the CUT, MOSCHECK, and RESCHECK commands.

Figure 3-61 CONVEX_TO_EDGE_FILTER


Normal Output
CONVEX_TO_EDGE_FILTER Output
{ ... CONVEX_TO_EDGE_FILTER < value }

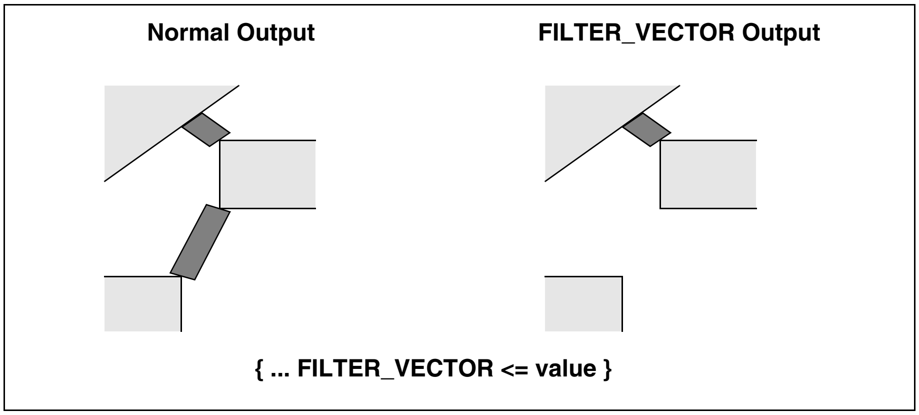
- CONVEX_TO_CONVEX_FILTER. Eliminates *all* error vectors longer than the specified length. If you want to filter only the error vectors from a CONVEX_TO_CONVEX check, or if a different filter value is required for CONVEX_TO_ CONVEX error vectors, you can use the CONVEX_TO_CONVEX_ FILTER option. For this option, the relational operators < and <= yield different results. This option is not available with the CUT, MOSCHECK, and RESCHECK commands.

Figure 3-62 CONVEX_TO_CONVEX_FILTER


Normal Output
CONVEX_TO_CONVEX_FILTER Output
{ ... CONVEX_TO_CONVEX_FILTER <= value }

- FILTER_VECTOR. All errors that are reported as error vectors must be filtered against a maximum length using the FILTER_VECTOR option. If specified, the FILTER_VECTOR option will discard errors that are longer than the given value. For this option, the relational operators < and <= yield different results. This option is not available with the MOSCHECK and RESCHECK commands.

Figure 3-63 FILTER_VECTOR



- SQUARE_CORNER. By default, Hercules uses a broad definition of corners during checking operations. All 45-, 90-, and 135-degree angles in the data for a layer constitute corners, and are checked for spacing violations against other corners and against edges. Setting the SQUARE_CORNER option to TRUE restricts corner checking to 90-degree (square) corners. This option is not available with the CUT, MOSCHECK, and RESCHECK commands.

  Note:

  > This command is also available as a dimensional option. See the *Hercules Reference Manual*, Detailed Commands chapter. Setting this option locally overrides the global specification.

- TOGGLE_BOX_CORNER. The corner checking method used by each command must be toggled from the default corner option by specifying the TOGGLE_BOX_CORNER option. The radial arc method is the default checking method used by Hercules to check corner-to-edge and corner-to-corner spacings. Box corner checking is an alternative corner checking method that is useful when process specification rules are stated in terms of box corner spacings. See the BOX_CORNER option in the *Hercules Reference Manual*, Detailed Commands chapter, for an illustration of both corner checking methods.
- TOGGLE_SQR_CORNER. By default, Hercules uses a broad definition of corners during checking operations. All 45-, 90-, and 135-degree angles in the data for a layer constitute corners and are checked for spacing violations against other corners and edges. The SQUARE_CORNER option, assigned in the runset OPTIONS section, restricts checking to only 90-degree (square) corners. For individual commands, the TOGGLE_SQR_CORNER universal option is used to toggle the assigned corner check setting. If the SQUARE_CORNER option is set to FALSE, then specifying the TOGGLE_SQR_CORNER universal option limits the corner checking for that one check to square corners only. If the SQUARE_CORNER option is set to TRUE, then specifying the TOGGLE_SQR_CORNER universal option expands the corner checking for that one command to include all corners.

CONNECTIVITY-Based DRC Checking

Connectivity-based Hercules DRC checking is a powerful tool that checks for DRC errors based on connectivity. The EXTERNAL, CENTER_TO_CENTER, INSIDE_EDGE, ENCLOSE, and SELECT DRC commands have connectivity-based checking capabilities.

The following options and operators are used in connectivity DRC checks.

- NODAL. When set to TRUE, data that is part of the same electrical node is not checked for EXTERNAL, INSIDE_EDGE, ENCLOSE, and CENTER_TO_CENTER command spacing violations. However, virtual connections are still checked. When set to FALSE (default), data that is part of the same electrical node, including virtual connections, is checked for EXTERNAL, INSIDE_EDGE, ENCLOSE, and CENTER_TO_CENTER command spacing violations.
- FLAG_DISCONNECTED. Setting this option to FALSE prevents data electrical disconnected data from being checked for EXTERNAL, INSIDE_EDGE, ENCLOSE spacing violations. The default is TRUE.
- TEXTED_WITH/NOT TEXTED_WITH. The operators are used with the SELECT command to select data that is or is not texted with specified text strings.

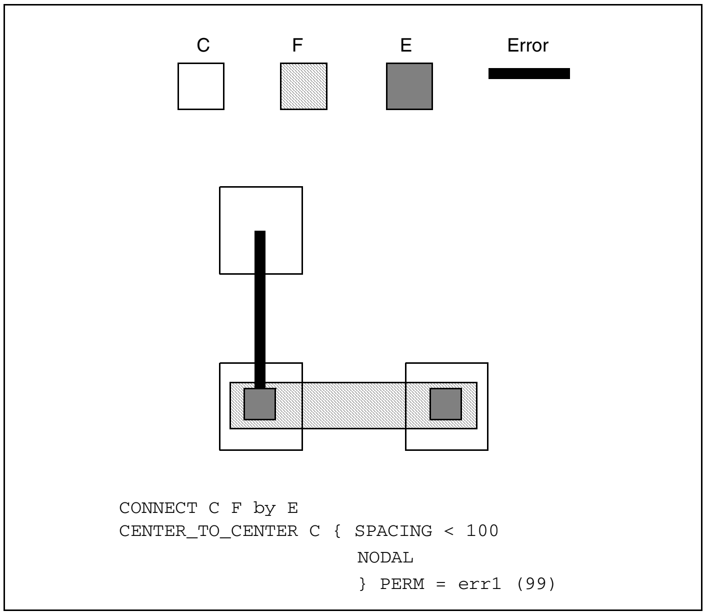
The following parser rules are for connectivitybased DRC checking:

- The layers of EXTERNAL, INSIDE_EDGE, ENCLOSE, and CENTER_TO_CENTER commands must be contained in preceding CONNECTs.
- SELECT_TEXTED_WITH requires a preceding TEXT statement.
- NODAL checks require a preceding CONNECT statement.
- SELECT_NODAL_TEXTED_WITH requires a preceding TEXT statement following the latest CONNECT statement.
- All DRC nodal checks input layers must appear in a preceeding CONNECT.

NODAL Option

CENTER_TO_CENTER Command

Figure 3-64 CENTER_TO_CENTER Command

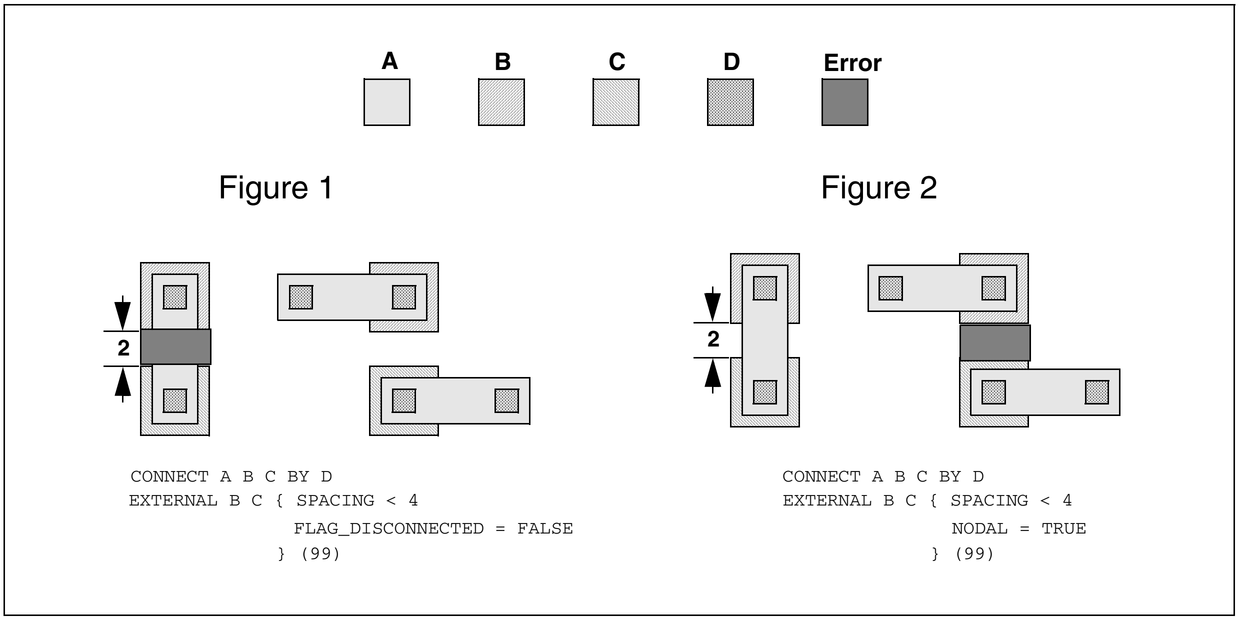


```
CONNECT C F by E
CENTER_TO_CENTER C { SPACING < 100
                     NODAL
                     } PERM = err1 (99)
```

In Figure 3-64, a CENTER_TO_CENTER check is performed on all rectangles that are not connected. Any spacing found from the center of one rectangle to the center of another rectangle that falls within the specified spacing value is flagged and placed in err1 on layer 99. Notice that the CONNECT command must first be run to define data that is identified as connected, much like NODAL for the EXTERNAL command.

EXTERNAL Command

Figure 3-65 EXTERNAL Command



```
CONNECT A B C BY D
EXTERNAL B C { SPACING < 4
               FLAG_DISCONNECTED = FALSE
             } (99)
```

```
CONNECT A B C BY D
EXTERNAL B C { SPACING < 4
               NODAL = TRUE
             } (99)
```

Figures 1 and 2 show how the EXTERNAL command performs connectivity based DRC checks for layer1 (B) and layer2 (C). In Figure 1, only electrically connected nodes with spacing less than 4 microns are flagged. In Figure 2, electrically unequal or disconnected nodes with spacing less than 4 microns are flagged. All violations are flagged and placed in the error hierarchy on layer 99.

SELECT Command

- The SELECT command selects data that is or is not texted with specified text strings in a DRC runset; for example, when text is the mechanism for filtering data prior to processing by subsequent Hercules commands. The following example shows how signal IO pads are filtered from Power IO pads via text selection.

  All signal IO PADS must be connected to a diffusion layer. Power pads texted with “VSS” or “VDD” are exempt from checking.

Example 3-1

```
ASSIGN {
   PAD (22) TEXT (22;1)
TEXT{
   PAD BY PAD.TEXT
}
SELECT PAD NOT TEXTED_WITH "VDD" "VSS" {GLOBAL_NET_ANY}  TEMP =
signal_pads
/*select PADS that do not contain “VDD” or “VSS” at any level of the
```

```
heirarchy.*/
/* connect the diffusion, interconnect, and PAD layers.*/
CONNECT { METAL1 diff   BY contact
          METAL2 METAL1 BY VIA1
          METAL2 METAL3 BY VIA2
          METAL3 PAD    BY VIAPAD
          signal_pads BY PAD
}
/* Using NET_FILTER to select all "signal_pads" not connected to
diffusion.
NET_FILTER {
   LAYER_SET[1] = {signal_pads}
   FILTER_RULE = {EV_AREA[1] > 0}
   OUTPUT_SET = {LAYER_SET[1]}
   CONNECTED = {!diff}
} (120;3)
```

Advanced Dimensional Applications

This section provides several examples on how to implement various Hercules dimensional checks. They illustrate some of the more sophisticated data processing that Hercules is capable of.

In addition to looking at the dimensional checks listed in this section, take a look at other chapters in this manual for additional examples of dimensional checks. The discussion on generating metal fill patterns in "Generating Metal Fill for Planarization" on page 4-5, for example, contains information on performing density checks.

- Complex Enclosure Checking
- Differentiate Via-to-Via Spacing: Same Node Versus Different Node
- Longedge checks
- Wide Metal Checking
- Exact Dimensional Checks
- Waivers

Complex Enclosure Checking

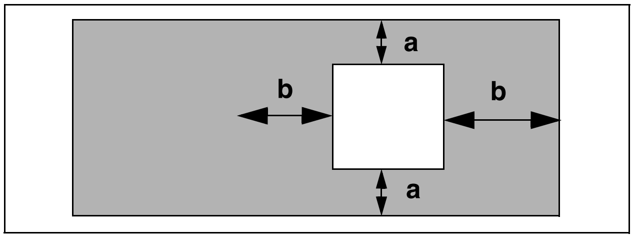
Via Checks

For a Via -End -Of -Line design rule:

1. Via must be enclosed by at least 0.05 µm of metal.
2. End-of-line metal must extend at least 0.12 µm beyond the last via.

In this set of rules, any enclosed spacing less than 0.05 µm is an error. Spacings between 0.05 µm and 0.12 µm are only errors, though, if the spacings on adjacent sides are also less than 0.12 µm. This is illustrated in Figure 3-66, where the a dimensions must be less than 0.12 µm (but greater than 0.05 µm) only if the b dimensions are at least 0.12 µm. Also, any via not enclosed by the metals must be flagged as an error.

Figure 3-66 “End-of-line” Via Check



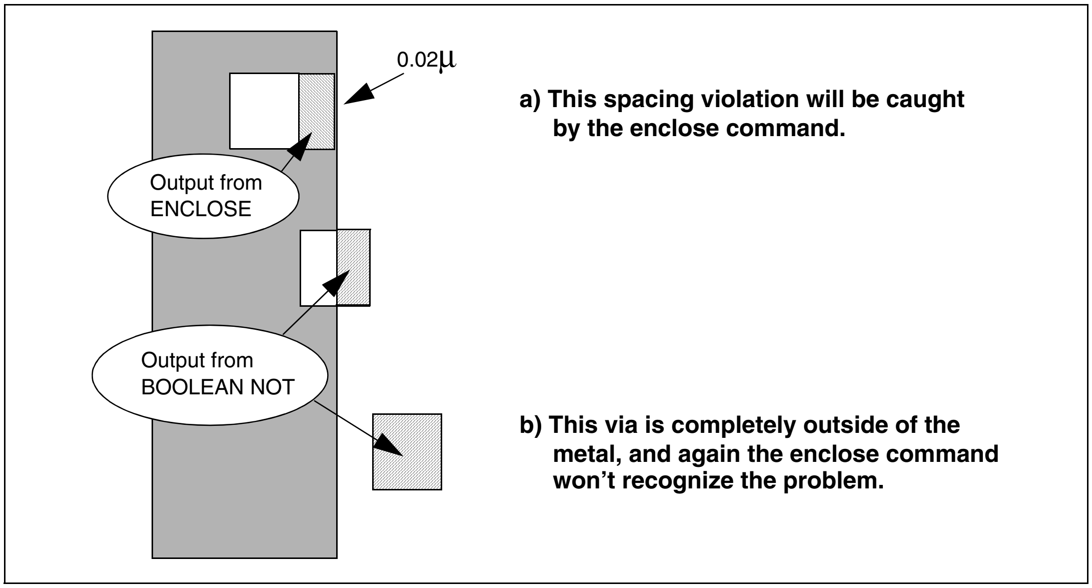
The following example shows one way these checks might be written. This example uses the Multiple Rules ENCLOSE command. The command first determines if any side of the via is enclosed by less than 0.05 µm of metal, since this is always an error. Vias that pass this test are then checked to determine if they have any sides enclosed by less than 0.12 µm of metal. If they do, a secondary check ensures that the sides adjacent to the potential violation meet the 0.12 µm requirement. If they do, the via satisfies rule b and is not reported as a violation. If they do not, the via is reported as an error.

Example 3-2

```
BOOLEAN VIA1 NOT MET1 {
   COMMENT=“Via not contained within metal”
} (100;2)
ENCLOSE VIA1 BY MET1 {
   COMMENT = “End-of-line via”
   RULE = [SPACING < 0.05
           CONTINUE = FALSE]
   RULE = [SPACING < 0.12
           ADJ_SPACING < 0.12]
} (100;4)
```

Notice that the first check requires a BOOLEAN NOT command to flag vias that are not completely contained within the metal layer.

Figure 3-67 Via Outside Metal Layer


0.02µ
Output from ENCLOSE
Output from BOOLEAN NOT
a) This spacing violation will be caught by the enclose command.
b) This via is completely outside of the metal, and again the enclose command won't recognize the problem.

The next example of via-end-of-line rules requires TOUCH capabilities for adjacent errors.

1. Vias must be enclosed by at least 0.0 µm of metal.
2. "End-of-line" metal must extend at least 0.04 µm beyond the last via.

The rule states that vias must be enclosed by 0.0 µm of metal on two opposite sides. However, the remaining two opposite sides must be enclosed by 0.04 µm of metal. This rule requires TOUCH capabilities for adjacent errors. Since the Multiple Rules Enclose command does not have this capability, the MULTI_RULE_ENCLOSE command is required to adequately meet design rule specifications. For more details on this command, see the *Hercules Reference Manual*, Detailed Commands chapter.

Example 3-3 MULTI_RULE_ENCLOSE VIA BY METAL

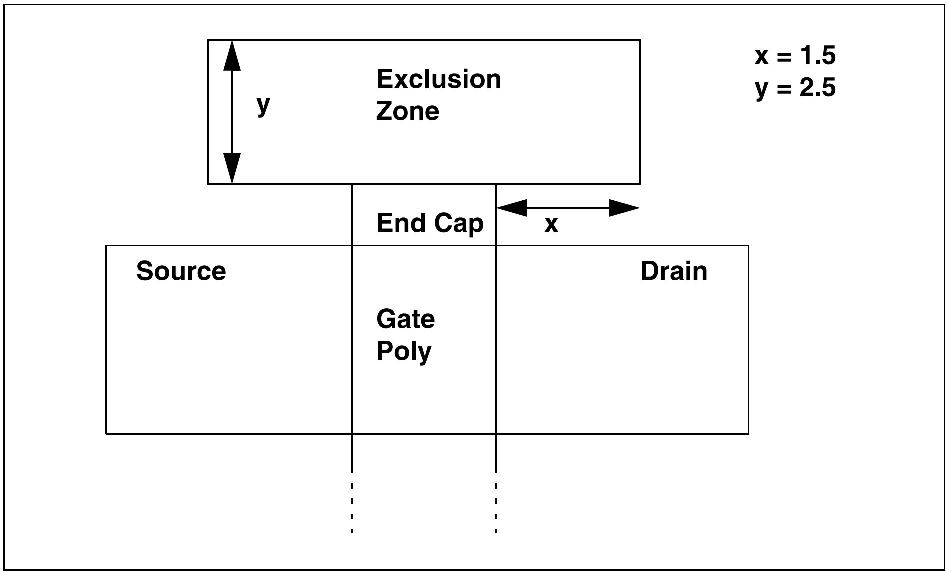
```
MULTI_RULE_ENCLOSE VIA BY METAL{
   OUTPUT_OPTIONS = {
   COMMENT = "Via-end-of-line checks"
   WIDTH = 0.1
   }
   RULE = {
      PRIMARY_ERRORS = {
         SPACING < 0.04
         SET_CORNERS_TO_SPACING = FALSE
         PARALLEL_POINT_PROJECTION=FALSE
         POINT_TOUCH = FALSE
         TOUCH=TRUE
         }/* point touch, corner spacing, and parallel point
             projections are not desired for primary errors*/
      ADJACENT_ERRORS ={
      SPACING < 0.04
      SET_CORNERS_TO_SPACING = FALSE
      PARALLEL_POINT_PROJECTION=FALSE
      POINT_TOUCH = FALSE
      TOUCH=TRUE
      }
   }
}
```

The PRIMARY_ERRORS will check all sides of the via for < 0.04 enclosed violations excluding corner spacing, parallel projections, and point touch violations. The via is then checked by the secondary section for adjacent errors since these are the errors of interest. The ADJACENT_ERRORS section will catch touch violations that will be missed by the MULTI_RULE_ENCLOSE command.

Poly “Exclusion” Checks

This rule defines an exclusion zone at the end of poly end caps where no other poly can be placed. This zone is intended to prevent DRC errors that might result from modifications of the existing poly after OPC processing.

Figure 3-68 Poly “Exclusion” Zone



How do I perform this check in Hercules?

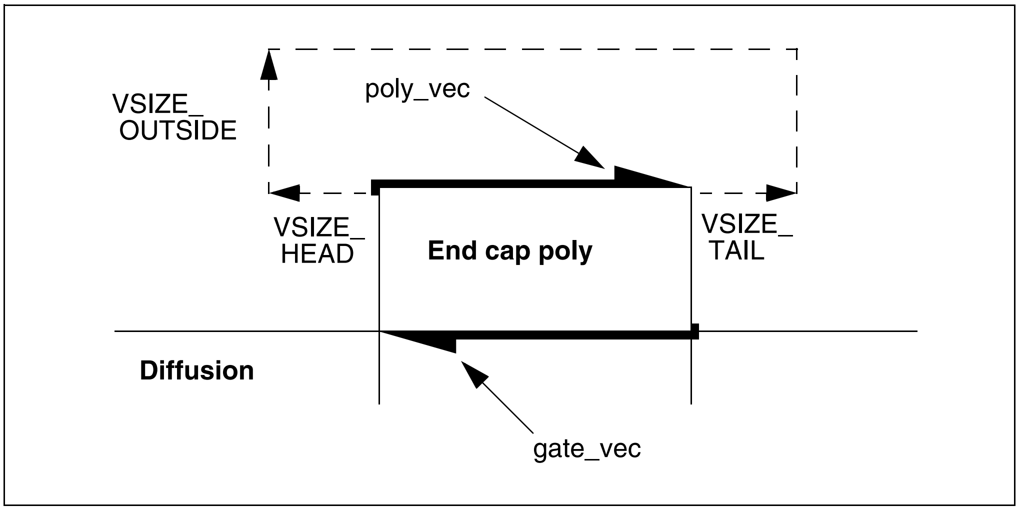
During OPC, the poly end cap polygons are modified to compensate for changes that occur as part of the manufacturing process. These modifications add poly to the end cap inside the exclusion zone, and if other poly already exists in this region it could merge with this new poly during manufacture and produce a short in the design. Since this short is created during mask fabrication and doesn’t actually exist in the design, a preemptive approach is required to prevent the problem. This approach uses Hercules to create a marker layer defining the poly exclusion zone, and any poly existing within this region is flagged as an error.

Example 3-4 ENCLOSE DIFF BY POLY

```
ENCLOSE DIFF BY POLY {
    OUTPUT_EDGES = TRUE
    SPACING < ENDCAP_MAX  /* replace ENDCAP_MAX with
                  the appropriate dimensional value */
    TOUCH = TRUE
    POINT_TOUCH = TRUE
    SET_CORNERS_TO_SPACING = FALSE
    PARALLEL_POINT_PROJECTION = FALSE
} TEMP = gate_vec   TEMP = poly_vec
SIZE poly_vec {
    VSIZE_INSIDE = 0
    VSIZE_OUTSIDE = 2.5
    VSIZE_TAIL = 1.5
    VSIZE_HEAD = 1.5
    } TEMP = poly_exclude
BOOLEAN poly_exclude AND POLY {
    COMMENT = “Poly in exclusion zone”
} (100;5)
```

The key to this example is the use of the vector form of the ENCLOSE command, invoked by the OUTPUT_EDGES option. As shown in Figure 3-69, this command outputs two vectors; one (gate_vec) where the end cap touches the gate, and the other (poly_vec) on the opposite edge. (The command’s syntax requires output definitions for both vectors.) ENDCAP_MAX is set to a value that will distinguish end cap poly, which is usually much shorter than the typically much longer interconnect poly.

Figure 3-69 Vectors Produced by the ENCLOSE Command



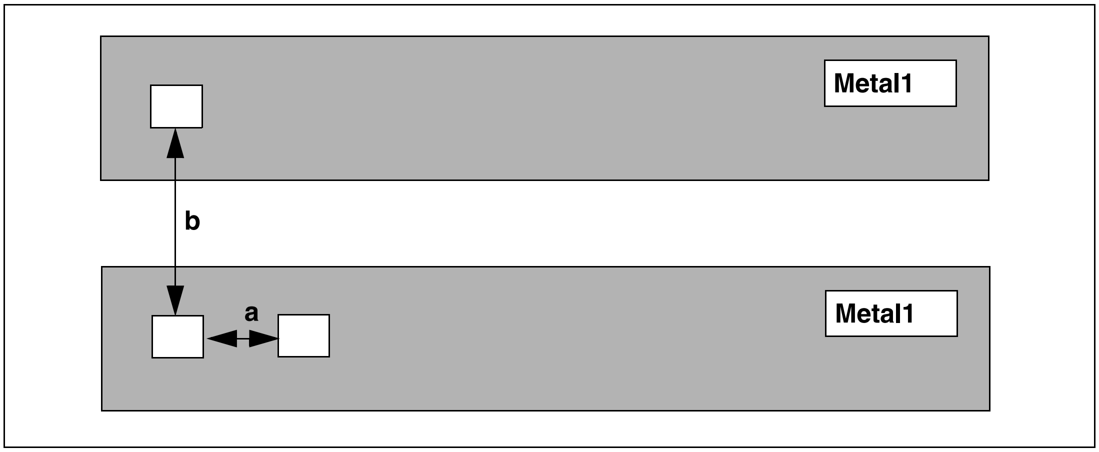
The poly exclusion zone is generated by converting the poly_vec vector into a polygon using the vector SIZE command. Since the “inside” of the vector will always be the side facing toward the gate, setting a non-zero value for VSIZE_OUTSIDE will grow the new polygon away from gate and away from the endcap regardless of the MOS device’s orientation. VSIZE_HEAD and VSIZE_TAIL are used to extend the new exclusion zone beyond the sides of the endcap. The final Boolean identifies any poly that has been placed within the exclusion zone.

Differentiate Via-to-Via Spacing: Same Node Versus Different Node

This design rule requires different via-to-via spacing. The minimum via1-to-via1 spacing is:

- same node: 0.3 µm
- different node: 0.35 µm

Figure 3-70 Via-to-via Spacing on Different Node



How do I perform this check in Hercules?

If minimum spacing requirements for vias on the same nodes are identical to vias on the different nodes, then use an EXTERNAL command to perform the via-to-via spacing check. In Figure 3-70, via1s on the same nodes should be at least 0.3 µm away while via1s on the different nodes should be at least 0.35 µm away. To differentiate vias on different nodes, use the CONNECT command followed by the EXTERNAL command and FLAG_DISCONNECTED and NODAL options.

Example 3-5 CONNECT

```
CONNECT {
        METAL1 METAL2 BY [ OVERLAP ] VIA1
        METAL2 METAL3 BY [ OVERLAP ] VIA2
        METAL3 METAL4 BY [ OVERLAP ] VIA3
        METAL4 METAL5 BY [ OVERLAP ] VIA4
        METAL5 METAL6 BY [ OVERLAP ] VIA5
        }
EXTERNAL VIA1 {
        COMMENT = "equal potential VIA1 spacing"
        SPACING < 0.3
        FLAG_DISCONNECTED = FALSE
        TOUCH = TRUE } (1;1)
EXTERNAL VIA1 {
        COMMENT = "non-equal potential VIA1 spacing"
        SPACING < 0.35
        NODAL = TRUE
        TOUCH = TRUE } (1;2)
DISCONNECT
```

Longedge checks

Spacing Rules Based on the Run Length of a Common Edge

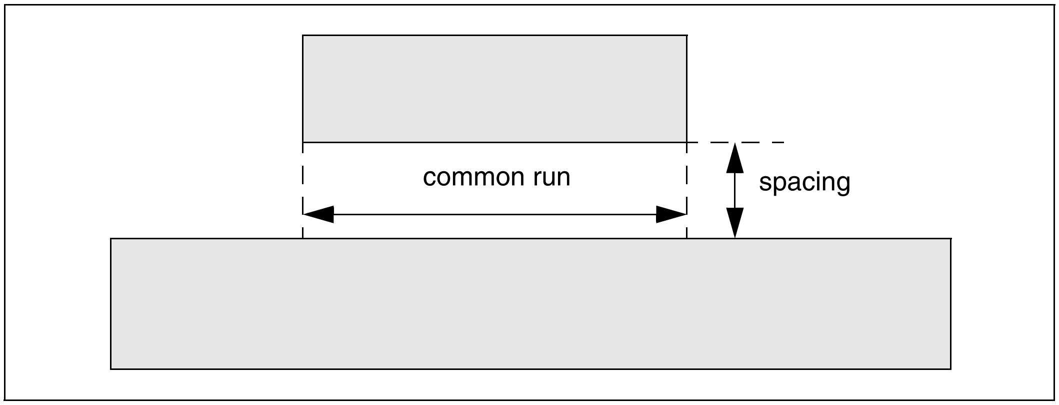
Design rule

This rule sets a spacing value that must be satisfied by polygons whose common edge meets a particular length specification, and a second spacing that must be satisfied by all other polygons.

For example:

1. Diffusions must be at least 1.0 µm apart except as noted in (2).
2. Diffusions must be spaced at least 1.5 µm apart when they share a common run of at least 10.0 µm.

Figure 3-71 Spacing Rules Based on a Common Run Length



How do I perform this check in Hercules?

This rule requires a simple EXTERNAL command with various LONGEDGE options.

```
EXTERNAL layer1 layer2 {
    SPACING < 1.0
    LONGEDGE_TO_EDGE < 1.5
    LONGEDGE >= 10.0
} (100;14)
```

The LONGEDGE option defines the trigger that determines which of the two spacing values is to be used in the check. When the length of the common run between the polygons satisfies the value specified by this option, the spacing is determined by the LONGEDGE_TO_EDGE option. When the common run doesn't meet the LONGEDGE specification, the regular SPACING value is used in the check.

Multiple Range Spacing Checks

Sometimes design rule specifications require DRC checking of multiple spacing values on the same runset layers. Runset algorithms written with Hercules multiple range spacing capabilities is an efficient method to handle such requirements. The EXTERNAL, INTERNAL, NOTCH, ENCLOSE, and INSIDE_EDGE Dimensional commands have Multiple Range Spacing capabilities. For more information see the *Hercules Reference Manual*, Detailed Commands chapter.

Example 1 shows two ENCLOSE commands with different spacing criteria. The more efficient method, Example 2, uses one ENCLOSE command with Hercules multiple range spacing capabilities. The *filter_errors1* output layer contains all enclose violations < 0.16. The *filter_errors2* output layer contains all enclose violations < 0.12. From this point, you can choose the necessary Hercules commands to process the output from *filter_errors1* or *filter_errors2*.

Example 1

```
ENCLOSE layer1 BY layer2 {
   SPACING < 0.16
} TEMP = filter_errors1
ENCLOSE layer1 BY layer2 {
   SPACING < 0.12
} TEMP = filter_error2
```

Example 2

```
ENCLOSE layer1 BY layer2 {
   SPACING < [0.16, 0.12]
} TEMP = filter_errors1, filter_errors2
```

Wide Metal Checking

The design rule for metals indicates that there should be minimum distance between two metal wires. Wide metal checking takes place while checking for different spacings of metals (that is, checking to see whether the metal width is greater than or equal to specific value).

Example rules can be any of the following:

- metal1 spacing of metal1 with a width greater than or equal to 10.0 μm is to have a spacing of at least 1.5 μm.
- Minimum metal1 to metal1 spacing:
  - metal1 width < 10 μm 1.0
  - metal1 width >= 10 μm 1.5

metal1 space greater than or equal to 1.5 (at least one metal line > 10 μm wide)

How do I perform this check in Hercules?

```
SIZE METAL1 { UNDER_OVER = 4.999 } TEMP=bigmetal
EXTERNAL bigmetal METAL1 {
        SPACING<1.500
        FLAG_ACUTE_ANGLE=TRUE
        NON_PARALLEL=TRUE
        SET_CORNERS_TO_SPACING=FALSE
        PARALLEL_POINT_PROJECTION=FALSE
        OUTPUT_EDGES=TRUE } TEMP = error_edges
```

45-degree angle

When dealing with 90-degree angle data, use the SIZE command with the undersize/oversize option to eliminate the skinny metal. When dealing with 45-degree angle data, additional steps are required to get rid of unwanted pieces of data.

Figure 3-72 Wide Metal Checking with 45-degree Angle

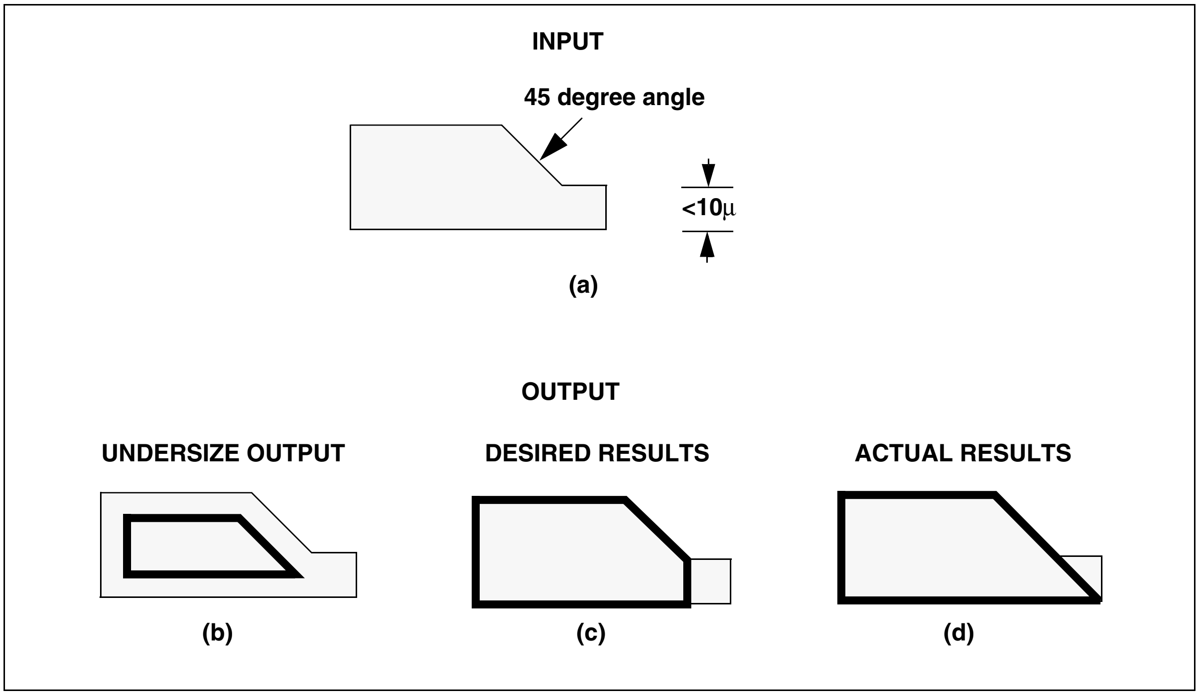


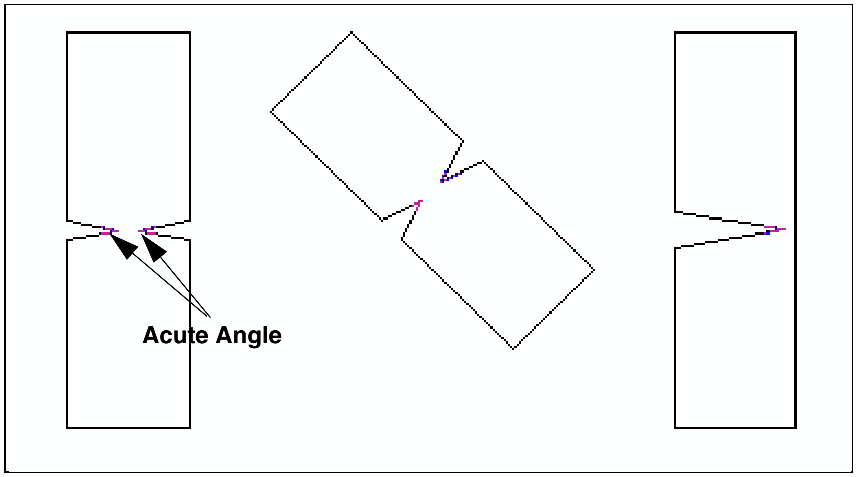
Figure 3-72 shows an example of how using the SIZE command does not remove all the skinny metal pieces when performing undersize/oversize on 45-degree angle data. Using SIZE undersize/oversize results in the output shown in Figure 3-72(d). In order to achieve the desired results, as shown in Figure 3-72(c), additional checks are needed to filter out the excess data.

Example 3-6

```
/* Unless acute angle clipping is an issue, these can
        be combined into a single under_over */
SIZE METAL1 {
        LEVEL_NON_ORTHOGONAL = TRUE
        UNDERSIZE = 5.50} TEMP = under_metal1

SIZE under_metal1 INSIDE METAL1 {
        DELETE_NOT = TRUE
        INCREMENT = 0.1
        OVERSIZE = 5.50} TEMP = uo_metal1
/* Operate clipping filter only on fat pieces */
SELECT METAL1 INTERACT under_metal1 {} TEMP = contains_fat
/* Find skinny regions to not away */
INTERNAL contains_fat {
        SPACING <= 10.00
        FLAG_ACUTE_ANGLE = FALSE
                       /* Gets rid of large square blobs */
        FLAG_ADJACENT_EDGES = TRUE
                       /* Flags acute regions internal to shape */
        NON_PARALLEL = TRUE
        } TEMP = skinny
/* Clip off angles and corners of uo_metal1 to get fat */
BOOLEAN uo_metal1 NOT skinny {} TEMP = fatPC
/* End of fat metal selection algorithm */
/* Once the wide_metal has been identified, perform the necessary spacing
check */
/* For the example spacing rule used in this section */
EXTERNAL bigmetal METAL1 {
        SPACING<1.500
        FLAG_ACUTE_ANGLE=TRUE
        NON_PARALLEL=TRUE
        SET_CORNERS_TO_SPACING=FALSE
        PARALLEL_POINT_PROJECTION=FALSE

        OUTPUT_EDGES=TRUE } TEMP = junk2_edges
/* END. */
```

Figure 3-73 provides examples of shapes with 45-degree edges. If the SIZE undersize/oversize combination is performed, the EXTERNAL check results in tiny vector output at the acute angles. To resolve this, re-code to remove the vector output.

Figure 3-73 Wide Metal Checking



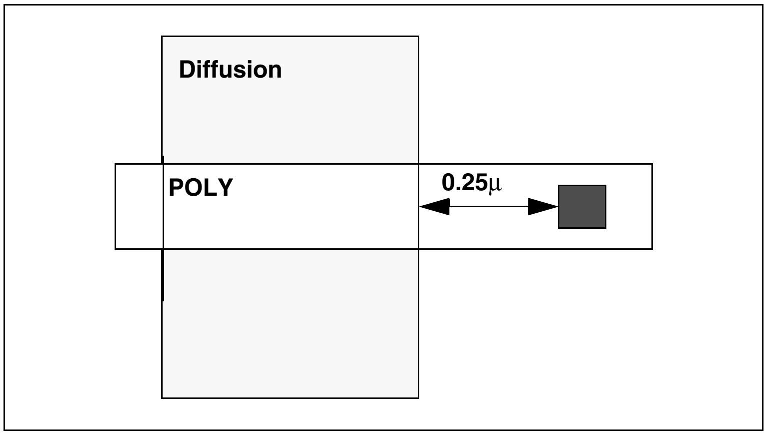
Exact Dimensional Checks

Design rule

This rule states that contacts must be an exact distance away from a given layer.

For example, poly contacts must be exactly 0.25 µm away from gate diffusions.

Figure 3-74 Poly Contact Spacing Rule



How do I perform this check in Hercules?

For this check, first identify all of the contacts that satisfy the rule. The contacts are then NOTed away from the input data to identify the remaining contacts that violate the rule.

For example:

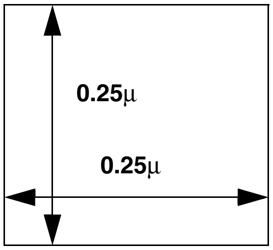
```
SIZE gate {
    OVERSIZE = 0.25} TEMP = big_gate
SELECT poly_cont EDGE_TOUCH big_gate {
    OUTSIDE
    RANGE = [1,1] } TEMP = good_cont
BOOLEAN poly_cont NOT good_cont { } (100;52)
```

Design rule

Vias must be squares or rectangles that have a fixed size.

For example, vias must be squares with sides 0.25 µm long.

Figure 3-75 Via DRC Rule



How do I perform this check in Hercules?

This is one of the few cases in Hercules where the command's output is the set of polygons that do not satisfy the conditions of the check.

For example:

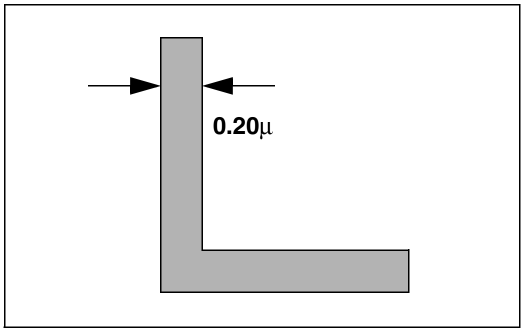
```
INTERNAL VIA1 {
    DIMENSION = [0.25, 0.25] } (100;28)
```

Design rule

This rule specifies an exact width for polygons on a given layer.

For example, image registration markers must have a width of 0.20 µm.

Figure 3-76 Image Registration Marker



How do I perform this check in Hercules?

This check has to be performed in two steps. In the first step, the dimensional check flags all polygons on the layer whose widths are less than the specified amount as errors. In the second step, the dimensional check shrinks the polygons so all that are the target size or smaller disappear, and only those that are larger than the target size remain. This approach is important because the INTERNAL command does not support a "greater than" operator.

In the first example, the two commands involved in the check each write their results out to the error hierarchy separately. This has the advantage of allowing you to tell at a glance whether a problem polygon is too narrow or too wide. In the second example, the two commands each write out to a TEMP layer, and these are ORed together for the error output. This approach would be used if you are more interested in flagging all of the errors at one time.

Example 1

```
INTERNAL reg_mark {
    SPACING < 0.20 } (100;34)
SIZE reg_mark {
    UNDER_OVER = 0.100 } (100;35)
```

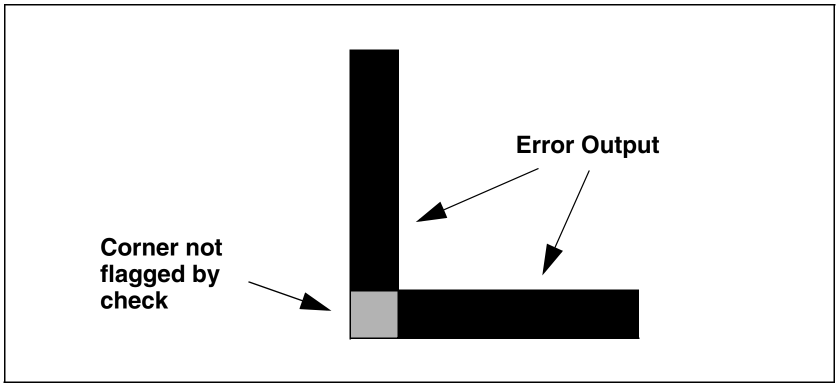
Example 2

```
INTERNAL reg_mark {
   SPACING < 0.20 } TEMP = too_skinny
SIZE reg_mark {
   UNDER_OVER = 0.100 } TEMP = too_fat
BOOLEAN too_skinny OR too_fat {
   COMMENT = “Bad width” } (100;36)
```

Note:

As shown in Figure 3-77, when both legs of the registration mark are narrower than the desired width the error polygons don’t cover the corner of the mark. This should not be a problem, however, since it is the widths of the legs that determine whether or not the shape meets the design rule.

Figure 3-77 Error Output When Both Legs of the Registration Mark Are Too Narrow

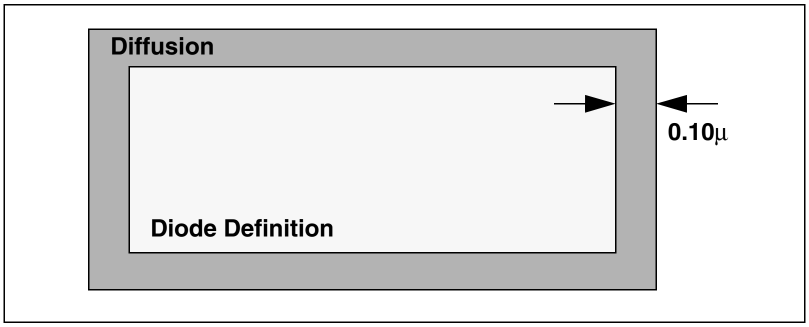


Design rule

This rule specifies an exact amount by which polygons on one layer must enclose polygons on another layer.

For example, a diode definition layer must be enclosed by exactly 0.10 µm of diffusion on all sides.

Figure 3-78 Diode Definition Enclosure Rule



How do I perform this check in Hercules?

This is another check that must be performed in two steps. In the first step, the polygons that are supposed to be enclosed are enlarged by the enclosure amount. In the second step, these enlarged polygons are XORed with the enclosing polygons to identify violations. The XOR is required since the “A NOT B” and “B NOT A” conditions need to be checked to ensure that the enclosed polygons are neither too large nor too small. Additional processing is also required to select only those diffusion polygons that contain the marker layer; other diffusion polygons must be excluded to prevent false errors from being reported.

For example:

```
/* Insert code here to select diffusion that interacts
   with the marker layer to prevent false errors */
SIZE diode_def {
    OVERSIZE = 0.10 } TEMP = big_diode_def
BOOLEAN big_diode_def XOR diode_diff { } (100;47)
```

Waivers

The WAIVE option allows the user to waive or disregard unwanted dimensional check violations. It is available with the EXTERNAL, INTERNAL, ENCLOSE, INSIDE_EDGE, and NOTCH commands. Waived violations do not appear in the output structure or the *block*.LAYOUT_ERRORS file. Waived violations are reported in the *block*.waive file, which has the same format as the *block*.LAYOUT_ERRORS file. The WAIVE option is a pattern recognition function based on specified waiver patterns. When enabled, all violations that are caused by patterns in the layout that match a waiver pattern are waived.

In order to waive an error, two steps are required. In the first step, cells that contain waiver patterns and reside in the input library should be generated. In the input library, cells are specified in the WAIVER command of the runset as follows:

```
Waiver {
   prefix = waiver
   CELLS = {<waiver_cell_name>, <waiver> ...}
   ASSIGN {
      layer_name ( Layer_Definition ) ( Layer_Definition )...
      .
      .
      layer_name ( Layer_Definition ) ( Layer_Definition ) ...
      layer_name VECTOR ( Layer_Definition ) (Layer_Definition)
   }
}
```

For more information on the WAIVE option, see the WAIVER Command in the *Hercules Reference Manual*, Detailed Commands chapter.

In the second step, violations that are unwanted during a particular command should be specified. This is demonstrated below within each of the following commands: ENCLOSE, EXTERNAL, INTERNAL and so on.

```
Command {
        WAIVE = { cell_name, cell_name ... }} Output_Definition
```

The cell_name represents waiver cell_names listed in the WAIVER command. The cell_names only pertain to waiver cells in a particular check. For example:

```
HEADER {
        INLIB = waiverdemo
        OUTLIB = EV_OUT
        BLOCK = chip
        LAYOUT_PATH = .
}
WAIVER {
        PREFIX = waiver
        CELLS = {wav1}
        ASSIGN {
           one  (1)
           two  (2)
        }
}
ASSIGN {
        one     (1)
        two     (2)
}

INTERNAL one {spacing < .3
              width = .05
              waive = {wav1}
             } (111)
```

Note:

Layer numbers only have to correspond to layer numbers in the waiver cell. A design can feasibly have a metal on layer18. If a waiver cell has a metal on layer1, then ASSIGN must use layer1.

Critical boundary

For a violation to be waived, it must exactly match a violation from the waiver cell. Also, the data surrounding the violation must exactly match the waiver pattern within the context of the critical boundary. The critical boundary is the extents of the waiver cell. The extents of the waiver cell are the total extents of all layers in the cell that appear in ASSIGN in the WAIVER section. When testing a candidate pattern, anything outside the critical boundary is considered a “don’t-care” condition.

A waiver pattern consists of critical and non-critical edges. A critical edge is an edge that does not touch (point_touch or line_touch) the cell extents. A non-critical edge may intersect the cell extents. Non-critical edges are not required for a match with a candidate pattern. A boundary layer can be added to a waiver cell to determine the cell extents and provide complete control over edge criticalness. This boundary layer must be in ASSIGN in the WAIVER command.

Limitations

Relocation and orthogonal rotations of the waiver patterns are recognized. The pattern-matching algorithm of the WAIVE option is based on violations found in the waiver cell. Only CORNER_TO_CORNER, CORNER_TO_EDGE and EDGE_TO_EDGE spacing violations are supported. To be specific, all types of violations can be waived, but there must be at least one spacing violation in the waiver cell for the pattern to be processed. Only violations that are completely enclosed in the critical boundary will be waived.

Since non-critical edges of a waiver may be false edges, they cannot be used for pattern matching. Waiver cells must have at least one spacing violation that is completely enclosed in the cell extents for the pattern to be processed.

A runtime message indicates if a waiver cell cannot be processed.

Note:

> The waiver cell cannot contain any hierarchy. Hercules will only consider the layers in the waiver cell and ignore cell placements.

Hierarchical considerations

The WAIVE option does not look up the hierarchy for matching patterns. For a pattern to be recognized, it must be formed in the cell where the suspect violation resides.

The WAIVE option detects if a matching pattern is disrupted anywhere in the hierarchy, and that the associated violations are reported as usual. If violations are waived in a cell, and the matching pattern is disrupted above in the hierarchy, the violation in the cell will remain waived, and a violation is reported in the parent where the pattern was disrupted.

Whenever possible, select waiver patterns that do not exceed cell boundaries.

Making a Good Waiver Cell

- Make sure edges that form errors do not touch cell extents.
- Use as little data as possible in the waiver cell.
- Use false edges to make the waiver cell smaller.
- For errors on created data, the waiver cell must contain the created data. For example, if the error is on a gate, the waiver cell cannot have just poly and diffusion.

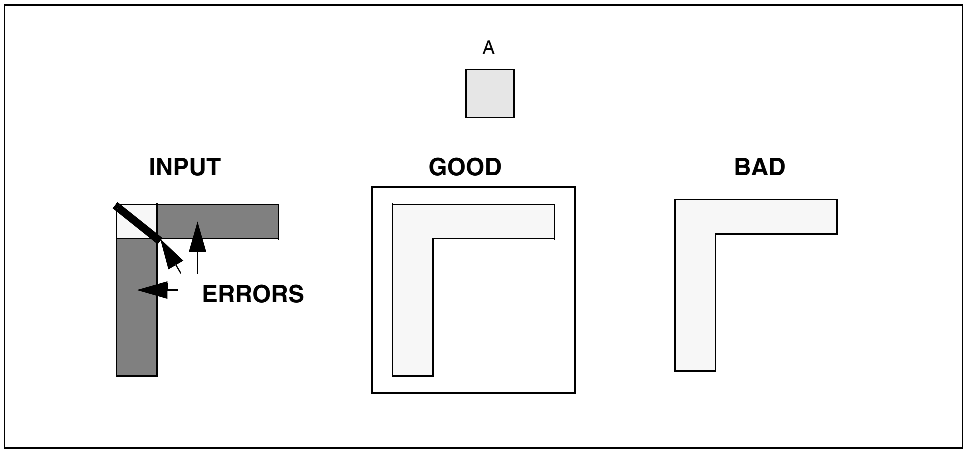
Waiver cell examples

Here are some examples of good versus bad waiver cells:

```
INTERNAL A { SPACING < 0.25 } (100)
```

When designing a waiver cell for this example, use a boundary layer because it more accurately defines the cell extents.

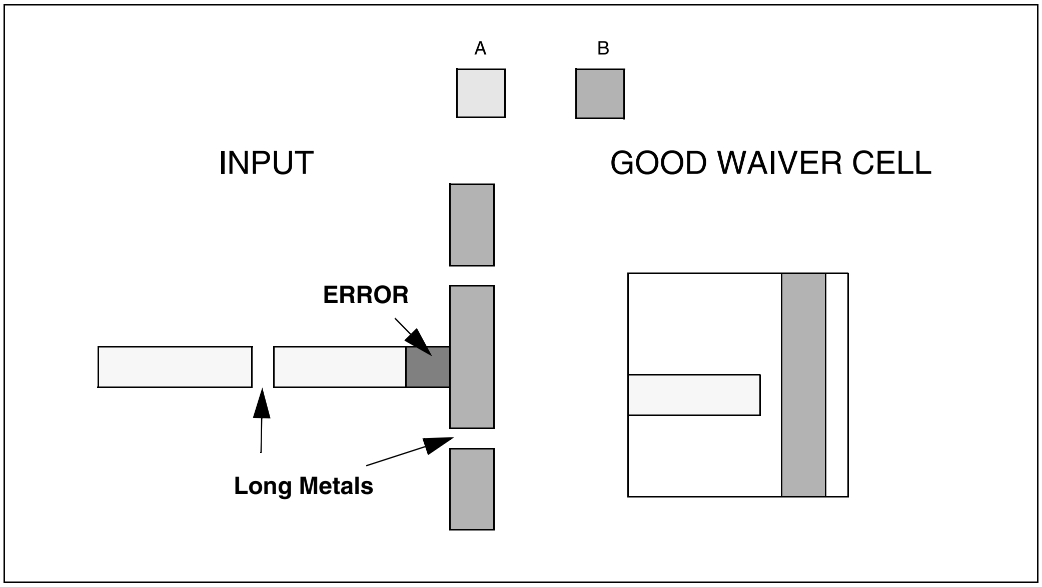
Figure 3-79 Example of a Good Waiver Cell Design



```
EXTERNAL A B { SPACING < .25 } (101)
```

For this example, it is not advantageous to design a waiver cell with long metals. For optimal performance, design the waiver cell to be as small as possible (as demonstrated in Figure 3-80).

Figure 3-80 Example of a Good Waiver Cell Design



4

Using DRC for Complex Data Generation

Hercules is designed and intended to be a verification tool, but its data creation capabilities can also be used to modify a design and influence the patterns on the different layer masks. This chapter describes at a high level Hercules commands that can be used to do Complex Data Generation. Refer to SolvNet for other detailed examples of applications.

One command used for performing Complex Data Generation is POLYGON_FEATURES. This is a general purpose data creation command intended for use when other commands do not offer the functionality required by a particular application.

Another powerful command discussed in this chapter is FILL_PATTERN. It is used for the creation of metal fill patterns for planarization.

Generating Data from Polygon Coordinates

This section begins by discussing the concept of generating data from polygon coordinates and then illustrates the concept with an example.

Generating Data

One of the most common applications of POLYGON_FEATURES is the generation of a regular array of features where they do not already exist.

Examples of these applications include the generation of:

- via arrays where different levels of metal overlap on power and ground buses
- metal fill in areas where there is not much low-level metal already present
- slots in wide-metal layers

When you consider the geometry involved in each of these applications, they are all three variations on a theme: the placement of a regular array of shapes within the boundaries of a given layer.

Example 4-1 an excellent illustration of the use of POLYGON_FEATURES for this type of situation. The example is written to generate an array of slots in selected metal polygons, but it is obvious that these slots are just polygons, and that they could just as easily be metal fill, vias, contacts, or any other regularly repeating figure. It may be necessary to change some of the parameters in the code for different applications, but this would be easy to do.

Generating Slots Example

Example 4-1 includes one example of a runset using the EQUATIONS option of the POLYGON_FEATURES command and works *only* on 90-degree angle input data. This example uses the EQUATIONS option to generate slots for metal that is 3.4 µm wide. The output is named SLOTS and stored on layer 200.

Example 4-1 Runset Using EQUATIONS Option

```
/* This internal check eliminates all data larger than the fixed width.*
 * It also removes dicontinuities and flattens data. */

INTERNAL met {spacing <= 3.4} TEMP=fixed

/* Generates the slots layer */
POLYGON_FEATURES fixed { EQUATIONS
{

   /* These are user defined parameters. */
```

```
n = 2;         /* number of rows of slots */
w = .5;        /* width of slots */
l = 2;         /* length of slots */
sl = 2;        /* spacing between slots along length */
sw = .45;      /* spacing between slots along width */
cs = 1;        /* spacing from beginning and end of */
               /* input polygon or any discontinuity */
rs = 1;        /* amount of staggering between each row of slots */
fixed_width = 3.4;   /* This should match the above value */
                     /* in the internal check */
res = 0.5;           /* desired grid unit */
/* end of user defined parameters */
/* set center offset */
center_offset = (fixed_width / 2) - (((w * n) + (sw * (n - 1))) / 2);
   center_offset = center_offset + (res / 2);     /* grid it */
mod = (center_offset%res);
center_offset = center_offset - mod;
eps = .000001;
min_fixed = fixed_width - eps;
max_fixed = fixed_width + eps;
vnum = EV_VNUM_IN; /* This is the number of input points. */
/* It is assumed the input data consists only of rectangles. */
ev_num_out = 4
if (vnum == 4)
{

   /* Get the extents of the rectangle */
   minx = EV_VXCOORD_IN[0];  /* Input points x array */
   maxx = EV_VXCOORD_IN[0];
   miny = EV_VYCOORD_IN[0];  /* Input points y array */
   maxy = EV_VYCOORD_IN[0];
   for (i=0; i < vnum; i++)
   {
      if (EV_VXCOORD_IN[i] < minx) minx = EV_VXCOORD_IN[i];
      if (EV_VXCOORD_IN[i] > maxx) maxx = EV_VXCOORD_IN[i];
      if (EV_VYCOORD_IN[i] < miny) miny = EV_VYCOORD_IN[i];
      if (EV_VYCOORD_IN[i] > maxy) maxy = EV_VYCOORD_IN[i];
   }
   xdim = maxx - minx;
   ydim = maxy - miny;

   /* Here's the horizontal case */
   if ((ydim > min_fixed) && (ydim < max_fixed))
   {
      starty = miny + center_offset;  /* bottom y of bottom row */
      startx = minx + cs;             /* left x of first slot */
      stopx = maxx - l - cs;          /* No slots beyond stopx. */
      stagger = 0;
      boty = starty;
      row = 0;
      for (row=0; row < n; row++)
      {
      if (stagger == 0)
      {
         leftx = startx;
         stagger = 1;
      }
      else
      {
         leftx = startx + rs;
         stagger = 0;
      }
      while (leftx <= stopx)
      {
         EV_VXCOORD_OUT[0] = leftx;        /* This is the output */
         EV_VYCOORD_OUT[0] = boty;         /* points array. */
         EV_VXCOORD_OUT[1] = leftx;
         EV_VYCOORD_OUT[1] = boty + w;
         EV_VXCOORD_OUT[2] = leftx + l;
         EV_VYCOORD_OUT[2] = boty + w;
         EV_VXCOORD_OUT[3] = leftx + l;
         EV_VYCOORD_OUT[3] = boty;
         /* This function saves the new polygon. */
         EV_VNUM_OUT = 4;
         ev_save_polygon();
         leftx = leftx + l + sl;
      }
      boty = boty + w + sw;
   } }

   /* Here's the vertical case. */
   if ((xdim > min_fixed) && (xdim < max_fixed))
   {
      startx = minx + center_offset;  /* left x of 1st slot */
      starty = miny + cs;             /* bottom y of bottom slot */
      stopy = maxy - l - cs;          /* No slots beyond stopy. */
      stagger = 0;
      leftx = startx;
      row = 0;
      for (row=0; row < n; row++)
      {
         if (stagger == 0)
         {
            boty = starty;
         stagger = 1;
         }
         else
         {
            boty = starty + rs;
            stagger = 0;
         }
            while (boty <= stopy)
         {
            EV_VXCOORD_OUT[0] = leftx;
            EV_VYCOORD_OUT[0] = boty;
            EV_VXCOORD_OUT[1] = leftx;
```

```
               EV_VYCOORD_OUT[1] = boty + l;
               EV_VXCOORD_OUT[2] = leftx + w;
               EV_VYCOORD_OUT[2] = boty + l;
               EV_VXCOORD_OUT[3] = leftx + w;
               EV_VYCOORD_OUT[3] = boty;
               EV_VNUM_OUT = 4;
               ev_save_polygon();
               boty = boty + l + sl;
            }
            leftx = leftx + w + sw;
   } } } }
} PERM = slots (200)
```

When examining Example 4-1, notice that there is a comment in the code stating that “it is assumed the input data consists only of rectangles.” This assumption is easy to ensure. The code shown below could be used to generate rectangles that identify locations for via arrays, where different metal levels in power and ground buses intersect for use in generating a via array. Preliminary data processing will have selected the metal bus layers.

```
BOOLEAN M1_pwr AND M2_pwr { } TEMP=potential_via_target

POLYGON_FEATURES potential_via_target {MAX_RECT = TRUE}
   TEMP = all_via_target
```

Generating Metal Fill for Planarization

Complete IC designs sometimes contain regions that have very little low-level metal in them. When this is the case, higher-level metals that are deposited tend to dip or sag in these areas since there is insufficient material underneath to support them. This can distort the surface of the chip and cause a variety of manufacturing problems. The problem can be prevented by adding filler to the design to support the higher-level metals. The following information describes how Hercules can be used to identify parts of the chip where this problem is likely to occur, and fill them with a regular pattern of metal to support the higher-level metals.

Adding metal fill to a design using Hercules is a three-part process:

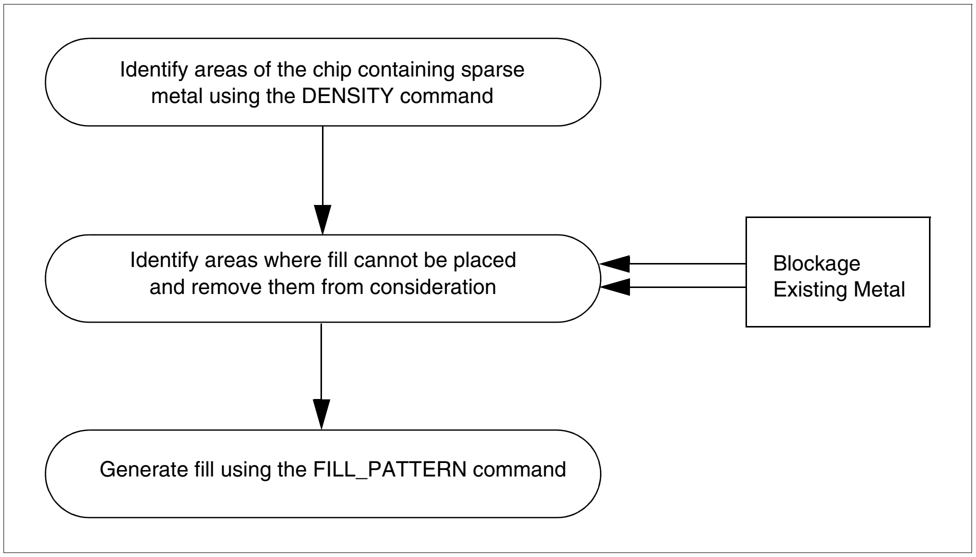
1. Identifying where the fill is needed.
2. Identifying where the fill is not needed.
3. Generating the fill polygons.

The flow is summarized in Figure 4-1. Each of these steps is described in detail in the following process steps, including examples of each command.

Important:

Before beginning this discussion, however, step 2 requires an additional comment. There are two reasons why you might not want to fill an area that has low metal density. First, the area might contain memory or sensitive analog circuits that you might not want to be covered with fill. Second, it might contain existing metal that you want to avoid to keep a design from shorting out. Both of these reasons are discussed.

Figure 4-1 Recommended Flow for Generating Metal Fill Patterns

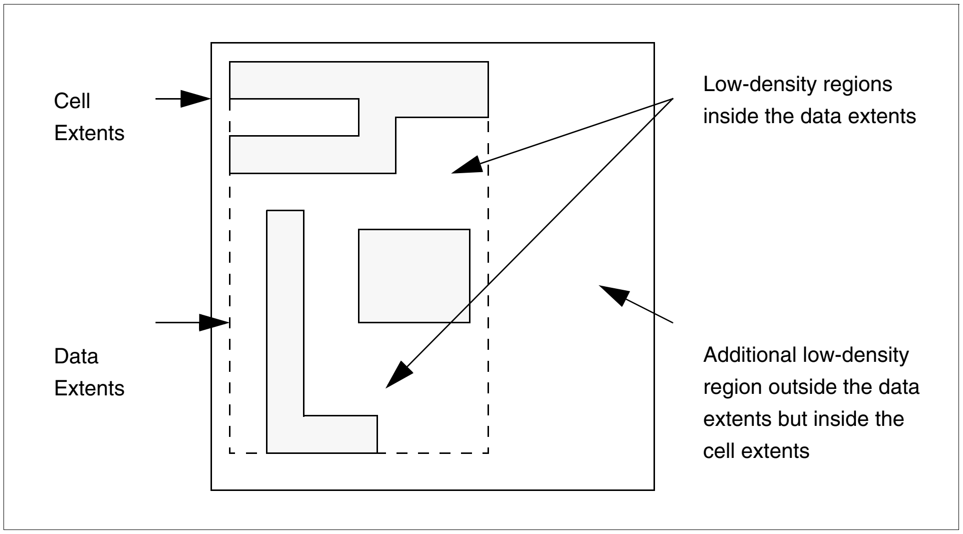

Identify areas of the chip containing sparse metal using the DENSITY command
Identify areas where fill cannot be placed and remove them from consideration
Blockage
Existing Metal
Generate fill using the FILL_PATTERN command

Step 1: Identifying Sparse Metal

Sparse metal regions of a chip are identified in Hercules using the DENSITY command. The DENSITY command operates at the top level of the design, and works by dividing the design into one or more windows. The area of the target layer inside these windows is compared to the area of the windows themselves. The results are compared to the sparse-metal criteria. The boundary of any window determined to contain sparse metal is then output for later use with the FILL_PATTERN command.

When writing the DENSITY command for a sparse-metal check, the command should be explicitly directed to check the entire design. (Considerations on how to deal with protected areas will be discussed later.) The easiest way to do this is to create a boundary layer polygon by applying the CELL_EXTENTS command to the top cell. This layer is then passed to the DENSITY command using the BOUNDARY_LAYER option. Otherwise, the command checks only within the data extents of the layer being checked, and sparse metal regions that lie outside the data extents are potentially missed. This situation is illustrated in Figure 4-2.

Figure 4-2 Cell Extents Versus Data Extents in DENSITY Command

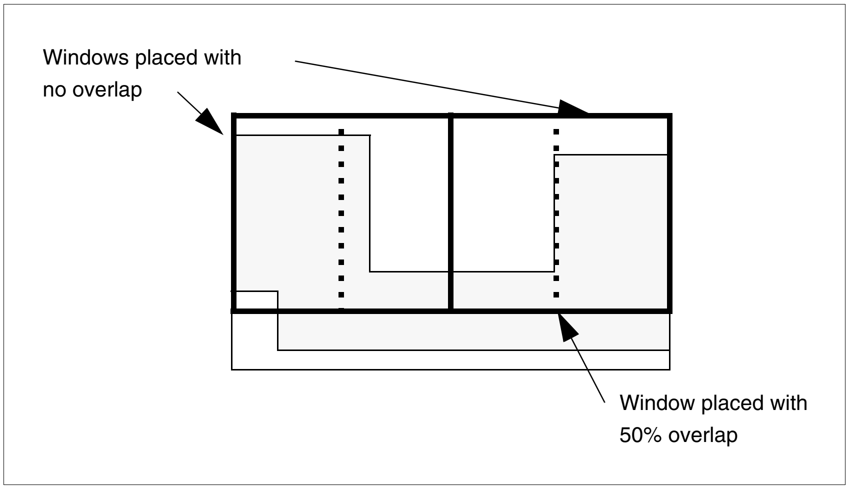

Cell
Extents
Data
Extents
Low-density regions inside the data extents
Additional low-density region outside the data extents but inside the cell extents

After you have identified the region to be checked for sparse metal, you need to divide that region into several smaller windows to perform the actual check. However, setting the size of these windows (using the DELTA_WINDOW option) is going to involve a trade-off between runtime and accuracy. If you set a single window over the entire chip, for example, you'll get faster runs but all you will find is the average metal density over the entire chip. You will have to experiment to find the optimum window size, but it should be the largest size that still detects all regions that must be filled to meet your requirements. For instance, many users have found that 50 µm x 50 µm windows work well with their designs.

After you have set your preferred window size, you should have each window overlap the preceding window. The reason for this is shown in Figure 4-3. The metal layer in this figure contains a notch that is assumed to be large enough to require some fill. The two windows shown with a solid edge, though, are placed so that they don't recognize this. As far as they are concerned, they contain more than the minimum required amount of metal, and so no sparse-metal data is generated for this check. A third window, though, placed so it overlaps the previous two, easily identifies this notch.

The amount of overlap between succeeding windows is specified using the DELTA_X and DELTA_Y options. A high degree of overlap will identify sparse metal regions more accurately but will require longer run times. A small degree of overlap will allow the command to complete faster but is more likely to miss some sparse metal regions. Fifty percent overlap in both directions should give a reasonable combination of accuracy and speed. If runtimes are an issue, however, you should reduce the overlap in the x-direction. When executing the DENSITY command, Hercules places the first window in the lower-left corner of the design with subsequent windows being placed in the positive Y direction. Hercules therefore does not have to redetermine how much metal is in the overlapped area. When Hercules finishes with the first column, it starts the window-placement process over again at the bottom of the next column. Since information about the metal density in the overlapped region was not stored during the first pass through the region, Hercules has to redetermine the metal area in the entire overlap region.

Figure 4-3 Effect of Window Overlap on DENSITY Command


Windows placed with no overlap
Window placed with 50% overlap

One final recommendation for the DENSITY command is to turn on the RESIZE_DELTA_XY option. When this option is set to FALSE, the last window in each row and column might extend beyond the cell's extents and produce unwanted violations. When the option is set to TRUE, the placement of these windows is adjusted so that they coincide with the edges of the BOUNDARY_LAYER.

Example 4-2 shows Hercules code that implements these recommendations.

Note:

> This example uses macro preprocessing and @DEFINE statements to set the parameters for the DENSITY command for convenience, but these could be set just as easily within the command itself. This example sets a window for the DENSITY checks that is 50 x 50 µm. Each window will also be offset 25 µm in the Y-direction and 10 µm in the X-direction from the previous window. Metal must cover at least 50 percent of the window to avoid the need for metal fill.

Example 4-2 Runset Fragment Illustrating Density Checks for Metal Fill

```
/* Set parameters for density check */

@DEFINE window_width 50
@DEFINE window_height 50
@DEFINE step_vert 25
@DEFINE step_horz 10
@DEFINE min_den 0.50
/* Set boundary for density check */
```

```
CELL_EXTENT {CELL_LIST={ top_block } } TEMP=full_chip_boundary

/* Identify regions requiring metal fill */

DENSITY metal2 {
     BOUNDARY_LAYER = full_chip_boundary
     DELTA_WINDOW = [0,0,window_width,window_height]
     DELTA_X = step_horz
     DELTA_Y = step_vert
     RATIO <= min_den
} TEMP = fill_this
```

Step 2: Identifying Restricted Fill Areas

As noted earlier, there will be parts of the chip where you don’t want fill to be generated regardless of the metal density in that area. There are two ways these areas can be dealt with. First, you can NOT them away from the output of the DENSITY command so that fill is put only where it can legally be placed. Second, you can fill all of the low-density regions identified by the DENSITY command and then remove the fill from those locations where it should not be placed. The first approach is recommended since it allows certain options in the FILL_PATTERN command (designed to limit the amount of disk space required by the command’s output) to function more effectively.

Possible places where you might want to avoid placing fill is in areas overlaying memory cells or sensitive analog circuits. These circuits are most likely contained in separate cells, so the easiest way to deal with them is to generate a no_fill_zone layer using the CELL_EXTENT command. You can either list the cells explicitly within the command or refer to a file containing a list of protected cells (for details refer to the CELL_EXTENT command discussion in the *Hercules Reference Manual*, Detailed Commands chapter). The layer generated by this second CELL_EXTENT command can then be NOTed away from the layer output by the DENSITY command to show where fill can legally be placed.

The other place where you are going to want to avoid placing fill polygons is on top of existing metal. Remember that the DENSITY command identifies windows that contain less than a specified minimum amount of metal, not just those that contain no metal. This fill must also be placed so that it is DRC-clean, which means that you will need to place the fill so that it satisfies minimum metal-to-metal spacing requirements. This applies at the edges of the low-density regions as well as at their cores, so you need to consider all of the existing metal that interacts with the to-be-filled layer written by the DENSITY command, and not just the metal wholly contained within this layer.

Example 4-3 shows an example of code that identifies places from which metal fill should be restricted, and removes them from the layer to be filled.

Example 4-3 Identifying Restricted Metal-fill Zones

```
/* Set metal-to-metal drc spacing requirements */
```

```
@DEFINE drc_spacing 0.25

/* Identify 'restricted' cells */

CELL_EXTENT {CELL_FILE=RESTRICTED_CELLS} TEMP=fill_restricted

/* Identify 'blockage' due to existing metal */

SIZE fill_this {OVERSIZE=drc_spacing} TEMP = within_drc_spacing
SELECT metal2 CUTTING within_drc_spacing { } TEMP = existing_metal2_cut
SELECT metal2 INSIDE within_drc_spacing { } TEMP = existing_metal2_inside
BOOLEAN existing_metal2_inside OR existing_metal2_cut { }
   TEMP = existing_metal2
SIZE existing_metal2 {OVERSIZE=drc_spacing} TEMP=fill_blockage

/* Remove blockages and restrictions from low density zone */

BOOLEAN fill_restricted OR fill_blockage { } TEMP = no_fill_zone
BOOLEAN fill_this NOT no_fill_zone { } TEMP = fill_this_zone
```

Step 3: Generating the Fill

The last step in the process of generating metal fill is to use the FILL_PATTERN command to actually generate the fill. This command generates a regular repeating array of rectangular polygons on the specified layer. Options are available that allow you to control the size of these polygons and the spacing between them (the stagger between different rows and columns, and the offset between the edge of the fill region and the first polygons in the pattern). Values for these parameters will typically be set by your foundry, and you are advised to contact your foundry representative for the necessary information.

Two concerns you might have about the FILL_PATTERN command are how much data it is going to generate and how large the resulting output files are going to be. To limit the size of these output files, it is strongly recommended that you set the option OUTPUT_AREF=TRUE. When this option is turned OFF, the FILL_PATTERN creates a set of individual polygons. When the option is turned ON, the output is a set of AREFs that give the equivalent pattern but can require several hundred times less disk space, depending on how much data is generated.

If you do use the OUTPUT_AREF option in the FILL_PATTERN command, the output generated by this command should not be further manipulated later in the runset. If you do, you will likely modify specific instances of the AREFs and thereby create hierarchy problems with the data. This is why you were advised to remove blockage regions from the areas needing fill before adding the fill to the design.

When you write out the metal fill data to the output database, you will want to do so in a manner that makes it easy to remove the fill if the design changes. A convenient way of doing this is to write the FILL_PATTERN output to a TEMP or PERM layer. Later, when you create the output database using the GRAPHICS or WRITE_MILKYWAY commands, you can write these layers to a separate output database that can be referenced by the input database. This way, if subsequent design changes require you to recreate the fill pattern, the old fill is easy to delete and replace. Before tape-out, the fill layer would then be merged with the original metal layer for mask creation. Your GRAPHICS or WRITE_MILKYWAY commands should also have the COMPRESS option turned on when you write out the fill data. If the OUTPUT_AREF option in the FILL_PATTERN command has done a good job the COMPRESS option may have no effect, but it won't hurt your disk space requirements and it might help them, so it's worth having it on.

Example 4-4 shows an example the FILL_PATTERN command. This example creates a regular pattern of 0.8 x 0.4 µm rectangles spaced 0.4 µm apart in both the X and Y directions. There is a 0.4 µm stagger in the X-direction, but no Y-stagger and no offset.

Note:

The fill generation flow described here is suitable for incorporation into a macro that would be applied to different metal layers as needed.

Example 4-4 Example FILL_PATTERN Command

```
/* Generate metal fill */

FILL_PATTERN fill_this_zone {
        WIDTH = 0.8
        HEIGHT = 0.4
        SPACE_X = 0.4
        SPACE_Y = 0.4
        STAGGER_X = 0.4
        OUTPUT_AREF=TRUE
        } PERM = metal2_final_fill (101)
```

5

Using ERC

This chapter describes various Hercules procedures that can be used either to analyze or confirm the electrical connectivity of an IC design. The chapter begins with a discussion of how Hercules can be used for substrate processing. The chapter goes on to outline procedures for connecting and texting your netlist, and then explains several commands that can be used to check the electrical connectivity of the design and confirm that there are no inappropriate net connections present. This is followed by a discussion on how to confirm the appropriate contacts necessary to avoid latch-up problems. The chapter concludes with a description of how Hercules can be used to identify potential charge-accumulation antenna violations and repair them.

Substrate Processing

This section describes how the substrate should be processed in Hercules, including the use of the keyword SUBSTRATE versus generating a substrate layer when multiple, isolated substrate regions are required.

Building the Substrate

Single-Potential Substrates

For extraction purposes, when the substrate is a single potential region, you are encouraged to use the keyword SUBSTRATE for making substrate connections in the CONNECT command, rather than generate a substrate as a digitized layer. It is also recommended that you do not rename the SUBSTRATE layer since it might affect the performance of some DRC commands such as the BOOLEAN NOT operation.

Processing the substrate as a layer can be bypassed in all-digital designs where everything is electrically tied to a common substrate. Substrate processing is not needed in such a design and the flattening issue is avoided completely. This does not mean that Hercules is ignoring the substrate. For example, the MOS extraction commands extract four-terminal devices, as shown by the syntax here:

```
NMOS device_name gate source drain [bulk]
```

```
NMOS device_name gate source drain SUBSTRATE
```

The fourth terminal (the backgate terminal, indicated by *bulk*) is optional and, if it is not specified, Hercules assumes that the device is being extracted as a three-terminal device. However, when the keyword SUBSTRATE is as the bulk connection, the backgate of the device will be connected to the substrate.

Note:

> For devices which have a bulk connection to a digitized well layer, such as a PMOS device in an NWELL, the NWELL layer would be specified as the bulk connection in the PMOS device command.

Multi-Potential or Isolated Substrates

Substrate isolation is often used when noise-sensitive circuit elements need to be isolated, or when different circuit elements require different ground potentials. Substrate processing offers unique challenges to Hercules, in that the substrate typically encompasses the entire chip's extents. If you are not careful, attempts to process the substrate can result in flattening of the design and a significant degradation in Hercules tool performance. This is especially true for runsets translated from ruledecks written for flat tools (such as Cadence Dracula® physical verification), where the design is going to be flattened as part of the verification process. The Dracula physical verification runsets frequently contain BOOLEAN commands with NOT operators that involve the substrate (for example, `SUBSTRATE NOT NWELL`). The Drac2He translator attempts to replace these commands with command sequences that are designed to maintain a design's hierarchy.

Substrate processing is more common in designs containing mixed analog and digital signal environments. You might have to isolate the analog side of a design from noise generated by the chip's digital side. If there are only one or two well-defined instances where a different substrate exists (for example, a PWELL inside an NWELL), it may be easier to deal directly with the second substrate. In such a case, it is important to make sure that any devices extracted reflect the presence of the second substrate in the connectivity and device extraction commands.

The outcome of the second substrate is going to depend on the particulars of the design flow. Typically, devices are extracted from different parts of the design by using different device recognition layers, as shown below:

```
NMOS digital_nmos d_gate d_nsd d_nsd [d_bulk]
```

```
NMOS analog_nmos a_gate a_nsd a_nsd [a_bulk]
```

The decision to list the fourth terminal in these commands (the bulk connection) depends on the flow, and particularly on whether or not the fourth terminal is included in the schematic netlist that will eventually be compared against the existing design. Example 5-1 shows one way to list the fourth terminal in the device extractions. This example assumes that an NWELL isolation ring in the design separates an isolated PWELL tub from the rest of the substrate. Since this PWELL material is surrounded by a ring structure, a SELECT INSIDE_HOLE operator is used to identify the PTUB material. The PTUB material is then used in the subsequent device creation sequence to define the device's bulk terminal material. Commands used to identify the isolation ring material and derive the other device layers are not shown here.

Example 5-1 Substrate Extraction for a Small, Well-Defined Substrate

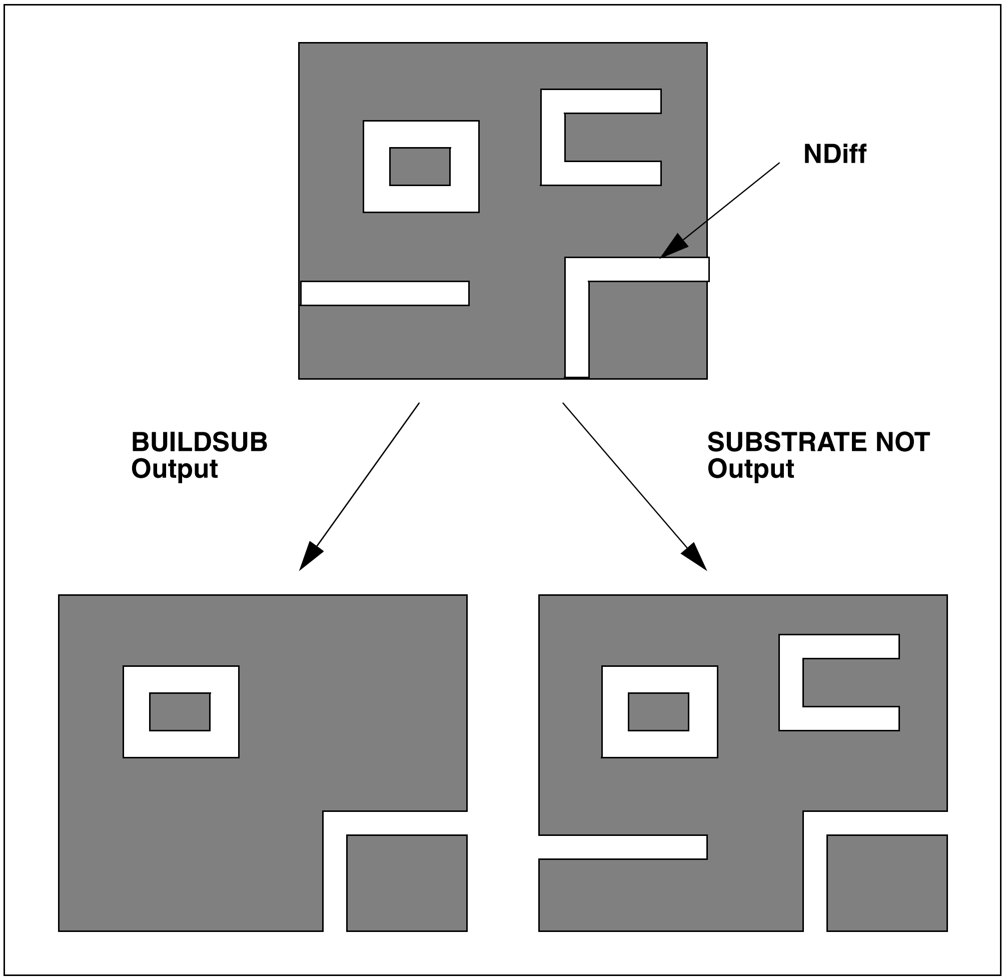
```
/* Commands identifying the isolation ring material here '*/
. . .
SELECT PDIFF INSIDE_HOLE isolation_ring { } TEMP = ptub
. . .
/* Commands creating the MOS device definition and */
/* device terminal layers would go here. */
. . .
NMOS N10_analog na_gate na_nsd na_nsd ptub {
     /* Insert appropriate NMOS options here */
} TEMP = n10_a
```

When a given chip contains several different bulk regions, the substrate should be partitioned into these regions using the BUILDSUB command. In order to use this command, the substrate regions you are trying to isolate with the command must be isolated from each other in the design. This will usually be accomplished through the use of isolation rings. Isolation trenches that divide the chip into two separate regions can also be used, as long as the isolation trench crosses the entire chip's extents. Hercules will only create different substrate regions if no direct electrical connection between them is possible.

The same objective *cannot* be accomplished using a BOOLEAN command with a NOT operator to isolate the material from the substrate. As noted earlier, this flattens the design and can have a negative impact on Hercules tool performance. A BOOLEAN NOT produces a completely different output, as shown in Figure 5-1. Since the C-shaped diffusion in the upper right and the trench in the lower left do not isolate different parts of the chip, they are ignored by the BUILDSUB command. They are not ignored, however, by the BOOLEAN command.

The NEGATE command is also *not* useful in this situation. NEGATE is not intended to generate substrate or electrical connectivity data; it is used strictly for mask preparation and DRC purposes. Like the substrate NOT, the output of the NEGATE and BUILDSUB commands are not the same, and the NEGATE output is not geared toward maintaining the design's hierarchy.

Figure 5-1 Comparison of BUILDSUB versus SUBSTRATE NOT Output



Once different substrate regions have been created, device extraction may proceed as normal. There is no need to extract different devices from different substrate regions because the different regions will form the bulk terminals for different instances of the device. For example, the following device extraction command with bulk generated by BUILDSUB could produce the extracted layout netlist shown below.

```
NMOS MN_dev ngate nsd nsd buildsub_layer TEMP=ndevice
```

A netlist segment after extraction of a device from different substrates:

```
{INST M1=MN_dev {PROP .....}
    {PIN N_3=GATE  N_6=SRC  N_5=DRN  “sub1”=BULK}}
{INST M2=MN_dev {PROP .....}
    {PIN N_7=GATE  N_14=SRC  N_12=DRN  “sub2”=BULK}}
```

Using CONNECT and TEXT

This section describes the use of the Connect with respect to making the substrate connections, as well as several advanced applications for processing text in Hercules. It explains why text is so important, how to make sure that text is applied properly, and how various TEXT_OPTIONS control text processing. In addition, text open suppression and debugging of text shorts is discussed (shorts being one net with two or more names and opens being two or more nets with the same name in the same cell).

Connecting the Bulk Layers

Once all of the device layers are formed and the substrate is generated with BUILDSUB (in the event that isolated substrate regions are required), the layers need to be connected prior to device extraction or other ERC-related processing. In the single-potential substrate case, where the substrate will be connected by a keyword, the bulk layers may be connected as shown in Example 5-2 and Example 5-3:

Example 5-2 Single-Potential Substrate Connect

```
CONNECT{
NWELL by ntap
SUBSTRATE by ptap
ngate by poly
pgate by poly
...
}
```

Example 5-3 Multi-Potential Substrate Connect

```
CONNECT{
NWELL by ntap
buildsub_layer by ptap
ngate by poly
pgate by poly
...
}
```

In the single-potential example, ptap is connected to the substrate by using the SUBSTRATE keyword, while in the multi-potential example, ptap is connected to a substrate layer generated by BUILDSUB.

Text Processing in Hercules

Hercules views text hierarchically. Proper texting methodology is extremely important to optimal LVS performance. Device filtering is performed during the COMPARE process based on power and ground text. Symmetry is broken by using text. In addition, significant runtime improvement may be achieved with text that allows the use of EQUATE_BY_NET_NAME. Text provides better diagnostics when debugging wiring shorts and opens. For more information on hierarchical texting and how it affects the COMPARE process, refer to the *Hercules Getting Started Tutorial*, "Introduction to Hercules HLVS" chapter.

Hercules provides COMPARE options that are designed for enforcing optimal texting methodology.

- ALL_PORTS_TEXTED option forces all ports for equivalence points to be texted.
- REQUIRE_TEXTED_PORTS_MATCH option flags mislabeled ports.

It is not necessary to have all nets texted to achieve optimal results. At a minimum, top level I/Os should be texted, including power supplies. Every layout port of every equivalence point should also be texted with names that match schematic port names. Using a layout editor, place text at hierarchical levels. Text should not be in a cell that is lower in the hierarchy than the polygon it is texting. To avoid text being attached to the wrong layer, keep text for certain layers on different layers.

Once you have the proper text, it is important to understand how it is read from the database or Edtext file, how it is attached using the TEXT command, and how applied text is controlled by TEXT_OPTIONS in Hercules.

Reading Text

Text layers must be referenced in the ASSIGN block. For example:

```
ASSIGN { METAL1 (8) TEXT (108) }
/*Text is read from layer 108 in layout or edtext*/
```

The text on layer 108 is assigned the name METAL1.TEXT due to its association with the layer name METAL1 in this example. However, reading in text in the ASSIGN block does not attach the text to the layer. Text attachment is achieved with a separate TEXT command in the runset. Therefore, using the TEXT command, the text now labeled METAL1.TEXT can actually be applied to any layer.

Attaching Text

The TEXT command attaches specific text layers to polygons. Attachments are made in the order of the statements in the TEXT command.

If a text's origin is not on a shape on the layer to which it is attached, Hercules reports an UNUSED_TEXT message. If different text is applied to a net which is already texted, a TEXT_SHORT message is reported. If different physical nets have the same text, a TEXT_OPEN message is reported.

The effect of texting by a string depends on the order of statements in the TEXT command. If text from a named *layer*.text is already attached to a polygon or net, text applied from a subsequent string assignment does not attach and no shorts are generated. If a string is applied to a polygon or net, and subsequently additional text is applied from a named *layer*.text, a short is generated. For example:

```
/****************************************************/
/* NOTE: "VSS" only applied to SUBSTRATE when NOT on the */
/* same net as a POLY or MET1 shape with attached text.  */
/****************************************************/
TEXT {
   POLY   BY POLY.TEXT  /* Attach POLY.text to POLY    */
   MET1   BY MET1.TEXT  /* Attach MET1.text to MET1    */
   SUBSTRATE  BY "VSS"  /* Attach VSS to SUBSTRATE if  */
                        /* not texted.                 */
}
/****************************************************/
/* NOTE: "VSS" is always applied to SUBSTRATE.      */
/* Any net connected to SUBSTRATE with other text   */
/* creates text short error messages.               */
/****************************************************/
TEXT {
   SUBSTRATE BY "VSS"  /* Attach VSS to SUBSTRATE  */
   POLY BY POLY.TEXT   /* Attach POLY.text to POLY */
   MET1 BY MET1.TEXT   /* Attach MET1.text to MET1 */
}
/****************************************************/
```

There are several options available in the TEXT command that suppress messages associated with text opens or shorts. For example, the USED_EDTEXT_FILE option allows you to create an ASCII file containing text that was applied to polygons for previous text statements.

TEXT_OPTIONS

TEXT_OPTIONS allow you to define how text is attached to polygon data, and replace text strings or text characters, if necessary. You can also set options to help debug text shorts. For more details on all the TEXT_OPTIONS, refer to the *Hercules Reference Manual*, Detailed Options chapter.

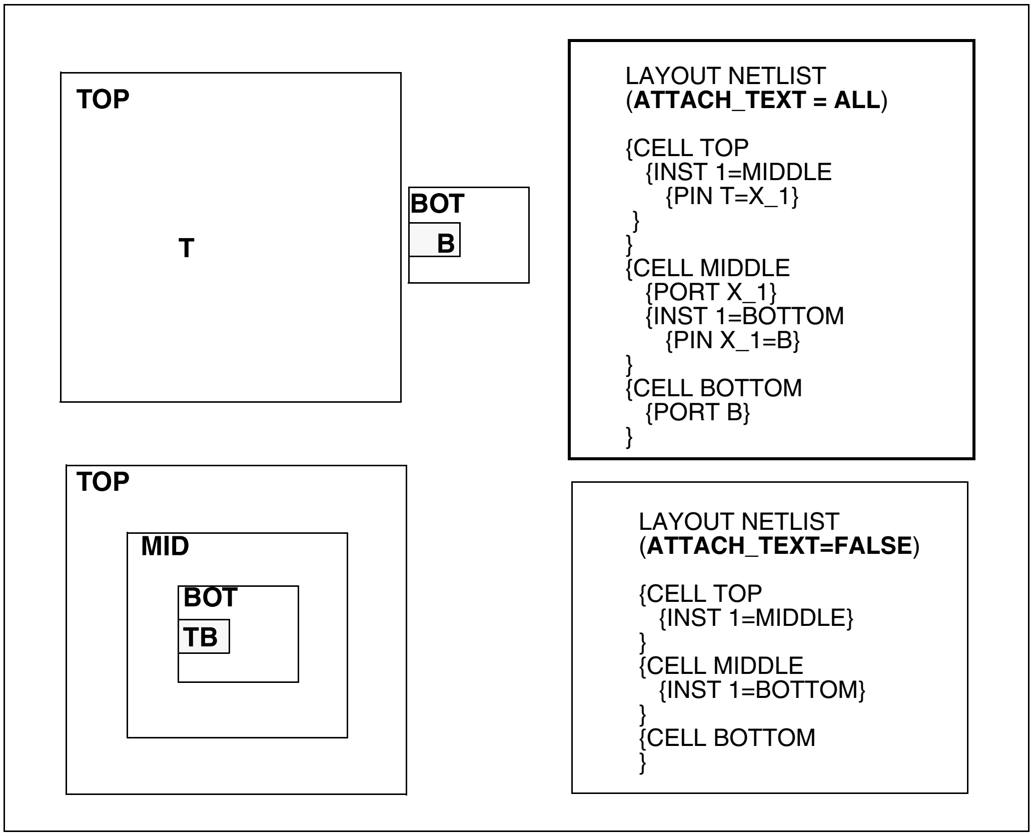
ATTACH_TEXT

The ATTACH_TEXT option determines how text creates ports. The default is FALSE, but the recommended setting is ALL. Set ATTACH_TEXT to TOP if you encounter problems with lower-level text.

- ATTACH_TEXT=ALL or ATTACH_TEXT=TRUE
    - Allows text to attach to polygons that are lower in the hierarchy by creating ports.
    - May attach text through *n* levels of hierarchy.
- ATTACH_TEXT=TOP
    - Only allows level-0 text to attach to polygons lower in the hierarchy by creating ports.
    - May attach text through *n* levels of hierarchy.
- ATTACH_TEXT=FALSE
    - If ports exist due to shape interaction, text will be attached.
    - If ports do not exist, text will not be attached.

Figure 5-2 shows how the ATTACH_TEXT option influences text processing and changes the extracted netlist. The layout consists of three levels of hierarchy. There are two text polygons in the design, B in cell BOT and T in cell TOP. Notice that cell MID contains only a reference to cell BOT. Setting ATTACH_TEXT to ALL creates a port for cells but setting it to FALSE does not.

Figure 5-2 ATTACH_TEXT



```
LAYOUT NETLIST
(ATTACH_TEXT = ALL)

{CELL TOP
   {INST 1=MIDDLE
      {PIN T=X_1}
 }
}
{CELL MIDDLE
   {PORT X_1}
   {INST 1=BOTTOM
      {PIN X_1=B}
}
{CELL BOTTOM
   {PORT B}
}
```

```
LAYOUT NETLIST
(ATTACH_TEXT=FALSE)

{CELL TOP
   {INST 1=MIDDLE}
}
{CELL MIDDLE
   {INST 1=BOTTOM}
}
{CELL BOTTOM
}
```

REPLACE_TEXT / REPLACE_TEXT_CHARS

- REPLACE_TEXT replaces a complete text string with another complete text string.
- REPLACE_TEXT_CHARS replaces a portion of the text string with another text string.
- Any REPLACE_TEXT_CHARS section must follow all REPLACE_TEXT sections.
- Both REPLACE_TEXT and REPLACE_TEXT_CHARS accept wildcards (a question mark [?] or an asterisk [*]) in the first string.
- In addition, text strings must contain alphabetic characters. For example, 123AB is a legal text string but 123 would be reported as an “All Digits” error.
- If the first string contains a question mark (?) or a square bracket ([or]), then a preceding backlash (\?, \[, or \]) is used to specify those characters.
- Do not place a single quotation mark (’) in the replacement text, because it will end up in the output string.

- In addition to space and tab characters, the following characters are illegal in text strings and should be replaced: = { } , * " /
- Text strings with reserved characters are discarded and reported in the *block*.LAYOUT_ERRORS file. For example, the text string VD*S5 would be discarded. To prevent this, use REPLACE_TEXT_CHARS to replace the asterisk (*).

See Example 5-4 for example text replacements.

Example 5-4 Text Replacements

```
TEXT_OPTIONS {
   REPLACE_TEXT {
      "GROUND" WITH "VSS"
      "GND"    WITH "VSS"
   }
   REPLACE_TEXT_CHARS {
      "AB" WITH "CD"
      "\[" WITH "<"
      "\]" WITH ">"
      "="  WITH "_"
      "{"  WITH "]"
      "}"  WITH "["
      "\*" WITH "_asterisk_"
      ","  WITH "_comma_"
      "'" WITH "_quote_"
      "/"  WITH "_slash_"
   }
}
```

USE_EXPLODED_TEXT

The USE_EXPLODED_TEXT option provides more control over text by using only the text layers specified in this option from exploded or flattened cells.

The USE_EXPLODED_TEXT option may be set to TRUE, FALSE (default), or VERBOSE.

- When set to TRUE, all text from exploded and flattened cells is retained and used.
- When set to FALSE, all text in a cell that is exploded or flattened is ignored.
- When set to VERBOSE, instance names are prefixed to text strings as they are flattened.

Beware of cells with text that are automatically exploded using TECHNOLOGY_OPTIONS.

USE_EXPLODED_TEXT_LAYER

The USE_EXPLODED_TEXT_LAYER option is another mechanism for providing control over text contained in exploded or flattened cells. This option takes a list of *layer*.text text layers. Text contained in the layers named in USE_EXPLODED_TEXT_LAYER is never ignored, even if the cell containing the text is exploded or flattened. This is equivalent to setting USE_EXPLODED_TEXT = TRUE for specific text layers.

NONCAT_TEXT

The NONCAT_TEXT option is another mechanism which provides control over text in exploded cells. This option takes a list of cells. Text that is in the cells named in the cell list is never ignored, even if the cell is exploded or flattened. This is equivalent to setting USE_EXPLODED_TEXT=TRUE for specific cells.

REMOVE_TEXT_ FROM_SHORT

The REMOVE_TEXT_FROM_SHORT option controls the disposition of shorted text.

- Text shorts are always reported.
- When setting REMOVE_TEXT_FROM_SHORT=TRUE, shorted text is removed and the nets are treated as untexted. Text shorts are reported in the *block*.LAYOUT_ERRORS file.
- When setting REMOVE_TEXT_FROM_SHORT=FALSE (default), text shorts are reported in the *block*.LAYOUT_ERRORS file. Hercules arbitrarily chooses which text to use.
- Results may be confusing if the wrong text is chosen (for example, when input is shorted to VDD).

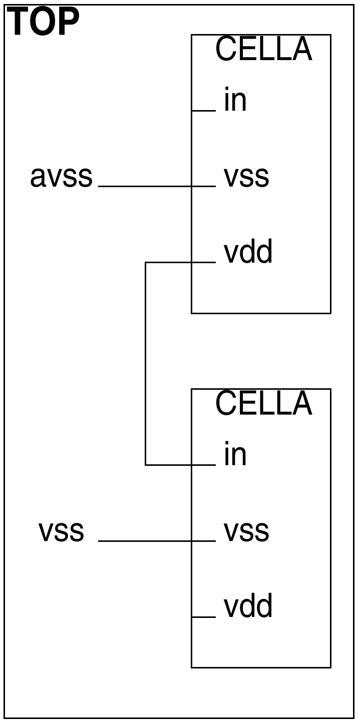
LAYOUT_GLOBAL

The LAYOUT_GLOBAL option in the OPTIONS section is used to specify a set of strings as globals.

- Global strings propagate up the layout hierarchy.
- Non-global strings only specify a net's value for a particular hierarchy level.
- Certain commands propagate non-globals down the layout hierarchy in some applications.
- Convenient method of allowing lower-level text for power supply names.
- Non-globals connected to globals lower in the hierarchy will be reported as a text short.

Figure 5-3 below shows how the LAYOUT_GLOBAL option works. For this case, setting LAYOUT_GLOBAL results in a text short and open. Therefore, using this option should be a known methodology decision.

Figure 5-3 LAYOUT_GLOBAL



```
LAYOUT NETLIST {LAYOUT_GLOBAL={vdd vss}}
-------------------------------------------------------------------
{CELL TOP
   {INST 1=CELLA
     {PIN X_1=in vss=vss  vdd=vdd}
   }
   {INST 2=CELLA
      {PIN vdd=in  vss=vss  2/vdd=vdd}
   }
}
/*****************************************/
/* Text short for vss and avss                        */
/* Text open for vss                                  */
/* Text open (global unconnected) for vdd             */
/*****************************************/

LAYOUT NETLIST (LAYOUT_GLOBAL=NONE)
----------------------------------------------------------------
{CELL TOP
   {INST 1=CELLA
      {PIN X_1=in  avss=vss  X_2=vdd}
   }
   {INST 2=CELLA
      {PIN X_2=in  vss=vss  X_3=vdd}
   }
}
/*****************************************/
/* No text open or shorts!                            */
/*****************************************/
```

Including Top-Level Port Names in Layout Netlist1

By default, no ports are netlisted in the top block of a Hercules layout netlist. You must identify the marker polygons that represent the pins, or top-block ports for Hercules. The CREATE_PORTS command is used to designate these ports, especially for the top block, so that the ports are netlisted. The COMPARE process then verifies that ports in the layout netlist match the schematic netlist ports.

For example:

1. Add the TEXT_POLYGON command to create marker polygons around the text designated as ports.

```
TEXT_POLYGON METAL1.TEXT {
   size = 0.001
   cell_list = {*}
   text_list = {*}
} TEMP = metal1_port
```

2. Add the port to the CONNECT command.

```
CONNECT {
   ...
   metal1_port BY [OVERLAP TOUCH] METAL1
   ...
}
```

3. Add the CREATE_PORTS command.

```
CREATE_PORTS {
   TOP_CELL_ONLY=FALSE
   metal2_port BY metal2_port
   metal3_port BY metal3_port
   metal1_port BY metal1_port
   VERBOSE=TRUE
}
```

You should now be able to run Hercules and generate a *block*.net file with top-level ports in the port list. The VERBOSE=TRUE option prints the ports that were created in the *block*.LAYOUT_ERRORS file.

Suppressing TEXT OPEN Messages

This section gives an overview of general text open suppression and text open suppression based on pin groups.

By default, Hercules generates text open messages in the *block*.LAYOUT_ERRORS file for all identically texted nets. In some instances, it is convenient to be able to suppress these text open messages. For library verification in particular, open nets may be intentionally left open at the cell or circuit level, but are physically connected once the cell is placed in a chip.

Suppressing Top-Level TEXT OPENS

By texting open nets with a text string followed by a colon (:), Hercules can suppress text open reporting for the top-level block.

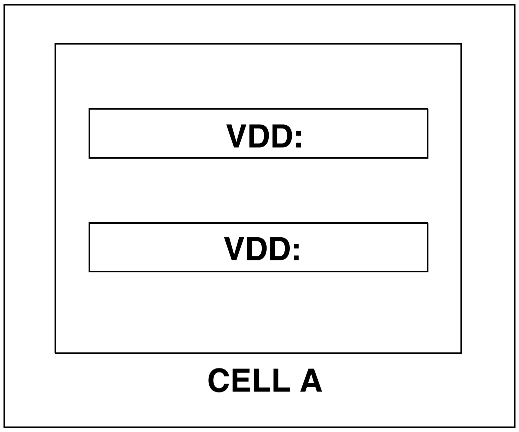
The USE_COLON_TEXT option suppresses error messages. Although this suppression is useful in some circumstances, this option is not recommended for use during final verification or when all nets are being tested for physical connections. The syntax is:

```
use_colon_text = TRUE|FALSE  (where the default is FALSE)

truncate_flag = TRUE|FALSE   (where the default is TRUE)
```

Note:
> The TRUNCATE_FLAG option must be set to FALSE when the USE_COLON_TEXT option is set to TRUE.

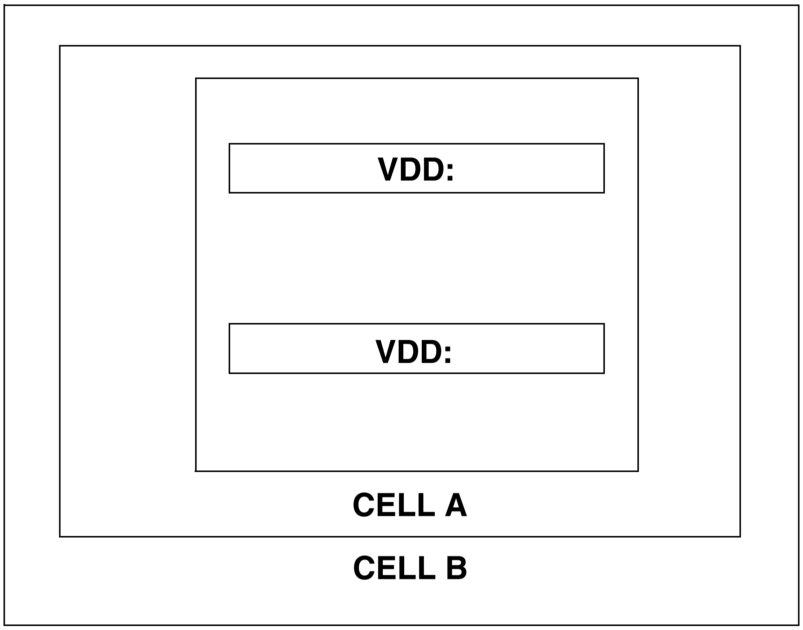
In Figure 5-4, there is no text open reported for VDD. Nets and polygons must be texted with the same text string followed by a colon(:), and BLOCK=CELL A must be specified in the HEADER section in order for net opens in CELL A to be suppressed. In nested cells, only the top-level block will have text for net opens that are suppressed.

Figure 5-4 Suppressing Top-Level TEXT OPENS



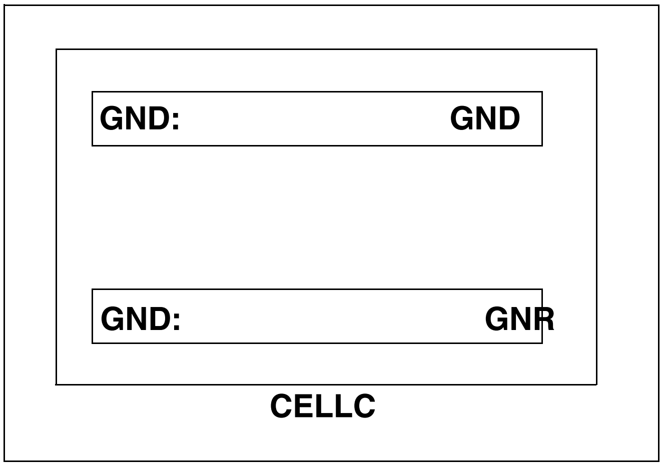
In Figure 5-5, CELL A is placed inside CELL B. If the HEADER section specifies BLOCK=CELL B, a text open in CELL A is reported. If BLOCK=CELL A is in the HEADER section, no text open is reported.

Figure 5-5 Suppressing Top-Level TEXT OPENS



In Figure 5-6, as long as BLOCK=CELL C, a text open is not reported. A text short is reported for the polygon texted with both GND and GNR. It is important to note that the polygon texted with both GND: and GND does not have a text short. For text short reporting, the colon is ignored. As long as the text strings (not including the colon [:]) are identical, there is no text short.

Figure 5-6 Suppressing Top-Level TEXT OPENS



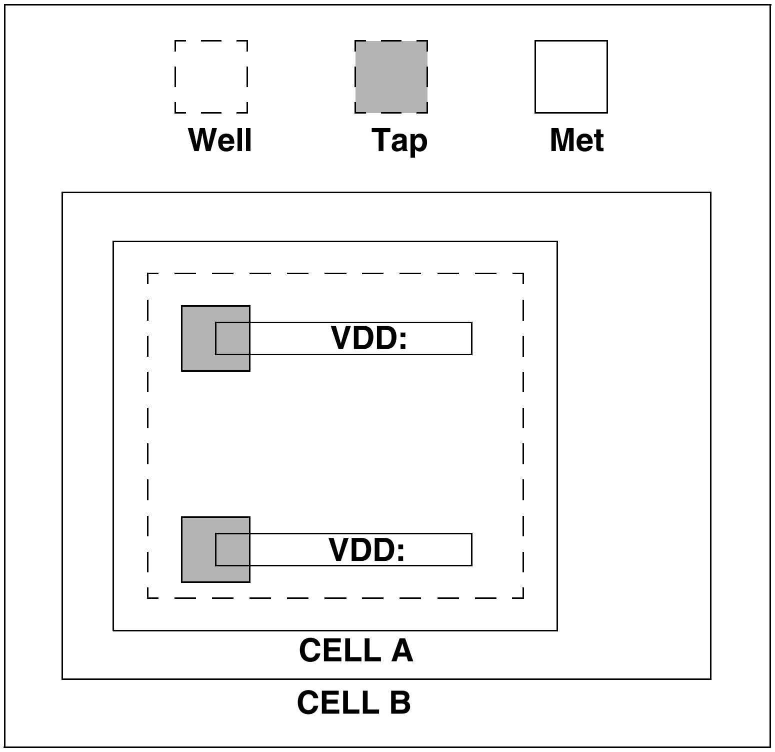
You can suppress the C_THRU error messages for nets texted with colons(:) by using the USE_COLON_TEXT option, and adding the TEXT and PROCESS_TEXT_OPENS commands as follows:

```
Connect {
   met by tap
}
Text {
   met by met.text
}
process_text_opens {}
C_THRU tap INSIDE well ...
```

Use the TEXT command after the CONNECT command to attach text to the polygons. You must also use the PROCESS_TEXT_OPENS command after the TEXT command, but preceding the C_THRU command. Implementing these commands suppresses C_THRU errors for any nets texted with colons, regardless of whether or not they are top-block.

In Figure 5-7, no text open or C_THRU error is reported if BLOCK=CELL A. If BLOCK=CELL B, a text open is reported, but no C_THRU error is reported.

Figure 5-7 Suppressing C_THRU Errors



When USE_COLON_TEXT=TRUE, the colon (:) is considered to be a special suffix that instructs Hercules to suppress error messages under the conditions described. The netname is the portion preceding the colon(:). Therefore, only the netname appears in the extracted netlist generated from the NETLIST command.

When USE_COLON_TEXT=TRUE, the colon (:) is ignored for any layout globals having this suffix. Only the netname is processed as global. For example, GND or GND: have the same meaning as LAYOUT_GLOBAL when using this option.

In some cases, text may be placed in a higher-level cell while the polygons are placed at a lower-level cell. The ATTACH_TEXT option can be used to build ports and nets from the lower-level polygons to the higher-level text. In these cases, when USE_COLON_TEXT = TRUE there is no text open reported at the higher-level cell (which contains text only).

Semicolon Text Processing

Semicolon text opens processing is a method of suppressing opens messages for pin groups within a black_box cell. Black_box cell data is an abstraction of the real layout data in the cell, and usually contains polygons representing the pins that need to be connected from outside the cell.

Semicolon text is used to provide the missing connectivity information needed to determine whether or not there is really an open. It also identifies pin groups within the black_box cell.

Definitions

A *pin group* is a group of pins that are connected to each other in the black_box cell. Only one connection from outside the cell to a pin in the group is needed for the whole group to be considered connected. A net can have multiple pin groups.

Pins are identified as being part of the same group if they all have the same text ending with a semicolon (;) and number. For example, all polygons in the black_box cell texted with vdd;1 would be part of the same pin group, and all polygons in the black_box cell texted with vdd;2 would be part of a different pin group for the same net.

Semicolon texting provides connectivity information in two ways:

- Any nets with the same text and suffix at the top level of a black_box are internally connected to each other.
- Each pin group for a net is externally connected to the same net in the parent cell.

Semicolon processing differs from colon processing in the following ways:

- Colon processing is for the top block only, and semicolon processing is for black_box cells. However, semicolon text in the top block will be treated as a special case, as an extension of colon processing for opens suppression in the top block.
- Colons on texted nets indicate that the net is connected above. Semicolons with suffixes in black_box cells indicate that the nets with the same suffix are connected to each other within the cell. Semicolon text in the top block indicates that the net is connected above.

Design Rules

The following are the design rules for semicolon processing:

- Only semicolon texted nets at the top level of a black_box cell are considered for black_box opens suppression. Semicolon text is treated as regular text in all non-black-boxed cells, except for the top block. In the top block, semicolon text is treated similarly to colon text for suppressing opens in the top cell between different pin groups of the same net.
- Semicolon texted nets at the top level of a black_box cell are assumed to provide correct connectivity information. If nets at the top level of the black_box have the same extension after the semicolon, they are considered to be physically connected within the black_box.
- Verification that semicolon text is correctly applied is done by running the cell as a non-black-boxed cell, with the complete physical layout data in the cell.

- For text short reporting, semicolons and colons are ignored. As long as the text strings (not including the colon [:] or semicolon [;]) are identical, there is no text short. For instance, there is not a short between:

  VDD and VDD:
  VDD and VDD;1
  VDD;1 and VDD;2

- In a black_box cell, it is an open if there is more than one pin group for a net and the different pin groups do not all connect up to the same parent net.
- In a black_box cell, if there is only one pin group for a net and none of the pins connect up, it is not an open.
- In a black_box cell, if a non-semicolon-texted or colon-texted net have the same netname as a semicolon-texted net, it is an open. For example, there is an open between:

  VDD and VDD;1
  VDD: and VDD;1

- Both semicolon and colon processing can be on at the same time.
- The semicolon (;) is considered to be a special suffix that is used by Hercules to suppress error messages under the conditions described. The netname is the portion preceding the semicolon. Therefore only the netname will appear in the netlist generated by the NETLIST command.
- When semicolon processing is on, the semicolon will be ignored for any layout globals having the specified suffix. Only the netname will be processed as global. For example, if either GND or GND;1 is specified as a layout global, it will be propagated as GND when the net is dispersed through connections up the hierarchy.
- If input data is from the Drac2He translator, the Dracula physical verification hedtext file, which defines pin group connectivity, will be converted to an Edtext file, and the list of black_box cells will be specified in the black_box_file.
- If input data is not from Milkyway or Drac2He, you will need to supply the semicolon text in the usual ways (that is, manually, either by applying text directly in the layout, or through the use of an Edtext file) and the list of black-box cells in the black_box_file.
- The top block cannot be a black_box cell. However, semicolon text in the top block will be used to suppress opens messages between different pin groups of the same net. They will be treated as if they were colon-texted.

USE_ SEMICOLON_ TEXT Option Usage

By default, Hercules generates text open messages in the *block*.LAYOUT_ERRORS file for all identically texted nets. In some instances, it is convenient to be able to suppress these text open messages. In particular, for library verification, open nets may be left open intentionally at the cell or circuit level, but are physically connected once the cell is placed in a chip.

By texting pin groups in black_box cells with (;) and a suffix, Hercules can suppress text open reporting for the black_box cells.

The USE_SEMICOLON_TEXT option suppresses error messages. Although this suppression is useful under some circumstances, this option is not recommended for use during final verification or when all nets are being tested for physical connection. The syntax is:

```
use_semicolon_text = TRUE|FALSE
```

The default is FALSE. When set to TRUE, use_truncate_flag automatically sets to FALSE.

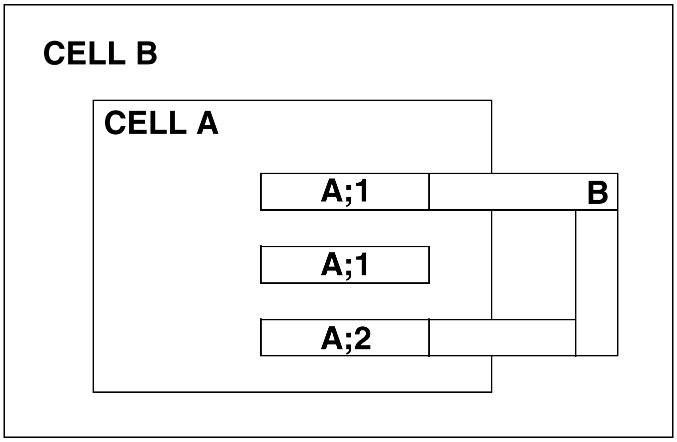
The list of black_box cells to be considered for opens suppression is obtained from the black_box_file in the HEADER section. The syntax is:

```
lvs = { sch_cell_1=lay_cell_1 ... sch_cell_n=lay_cell_n }
```

The following are examples of semicolon text usage. In each figure, CELL A is a black_box cell.

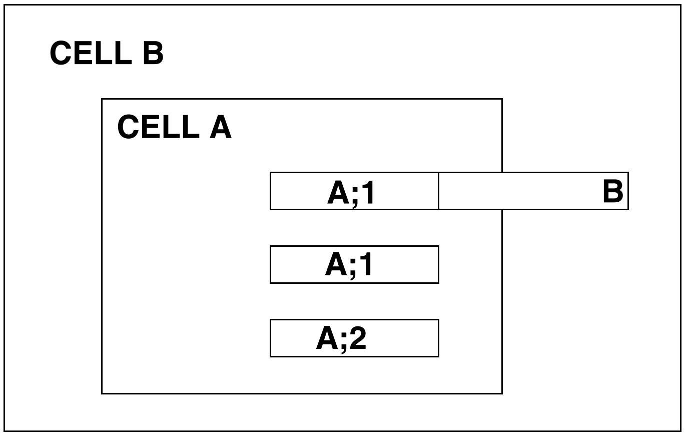
In Figure 5-8, there is no open because both pin groups for net A in the black-boxed CELL A are connected to the same net in CELL B.

Figure 5-8 Example of SEMI_COLON Text Processing



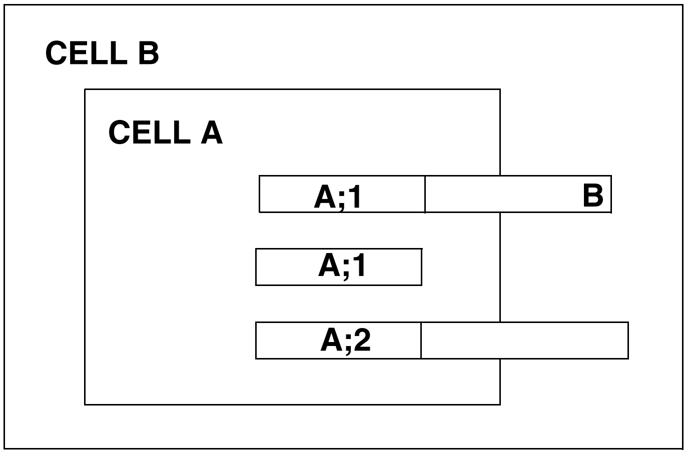
In Figure 5-9, there is an open. A;2 in the black-boxed CELL A is reported as an open on CELL B because it does not connect up.

Figure 5-9 Example of SEMI_COLON Text Processing



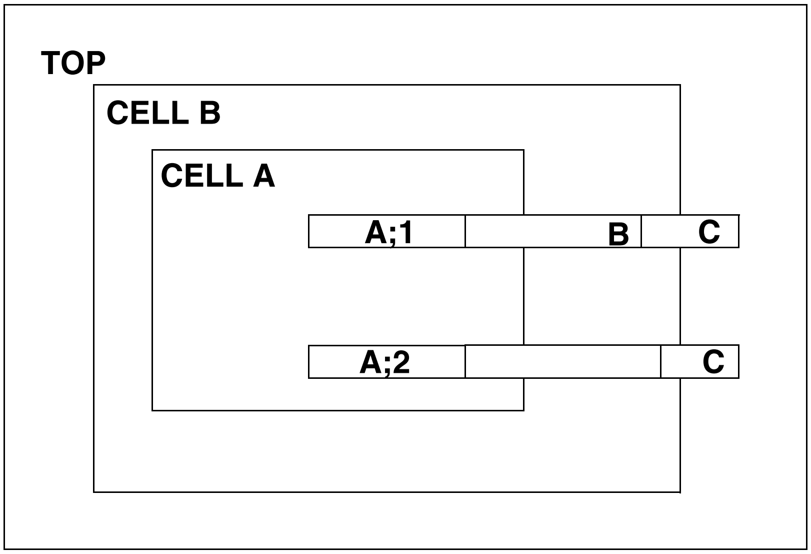
In Figure 5-10, there is an open reported in CELL B because the two pin groups in the black-boxed CELL A do not connect to the same net.

Figure 5-10 Example of SEMI_COLON Text Processing



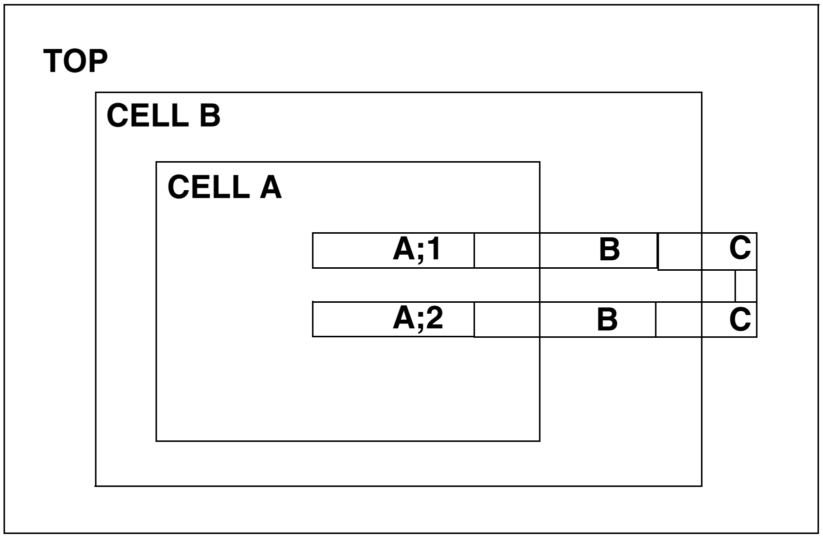
In Figure 5-11, there is an open in TOP because the two pin groups in the black-boxed CELL A are not connected to the same net at the top.

Figure 5-11 Example of SEMI_COLON Text Processing



In Figure 5-12, there is no open because the two pin groups in the black-boxed CELL A are connected to the same net at the top.

Figure 5-12 Example of SEMI_COLON Text Processing



In Figure 5-13, there is also no open because the connection for net C is assumed to be made inside the black-boxed CELL A.

Figure 5-13 Example of SEMI_COLON Text Processing

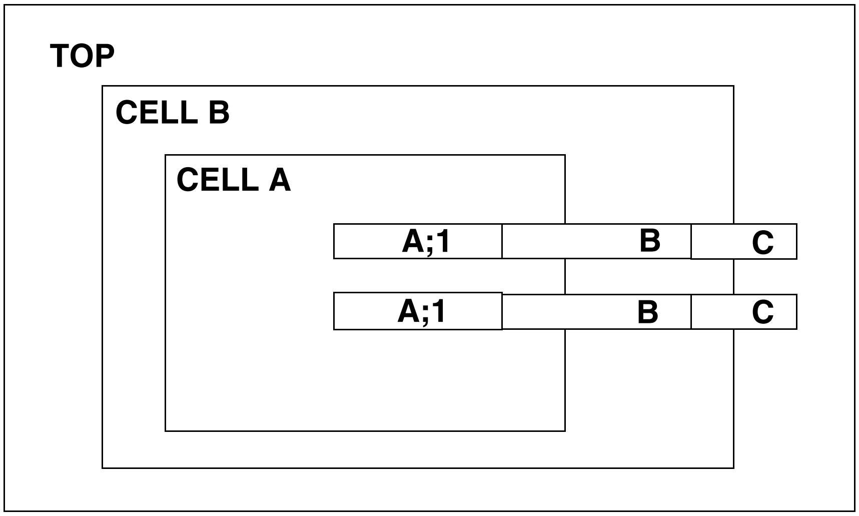
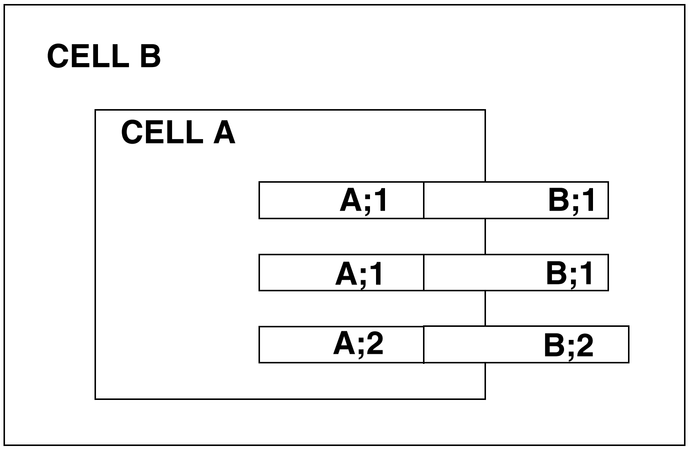


Figure 5-14 is an example of semicolon text processing for the top block. CELL B is the top block of the design and CELL A is black-boxed. The Bs with semicolons in the top block are treated as if they all ended in a colon (:). There are no opens reported because the pin groups in the black-boxed CELL A all point up to the same net.

Figure 5-14 Example of SEMI_COLON Text Processing



Connecting Nets by Name

CONNECT_BY_NAME controls how text opens are reported and/or renamed.

When CONNECT_BY_NAME=NONE:

- Reports TEXT_OPEN_RENAME messages in the *block*.LAYOUT_ERRORS file.
- Renames open nets with a unique prefix (for example, ev_0_GND) regardless of whether or not the open is resolved in parent hierarchy.
- Causes blocks containing opens to fail during COMPARE.
- Treats each piece of a net in a text open as a different net in ERC and NODAL checks.

When CONNECT_BY_NAME=CONNECTED:

- Reports TEXT_OPEN_RENAME messages in the *block*.LAYOUT_ERRORS file.
- Renames open nets with a unique prefix (for example, ev_0_gnd) *only* if they are *not* connected somewhere in parent hierarchy.
- Causes blocks that contain unresolved opens to fail during COMPARE.
- Treats each piece of a net in a text open as a different net in ERC and NODAL checks.

When CONNECT_BY_NAME=ALL:

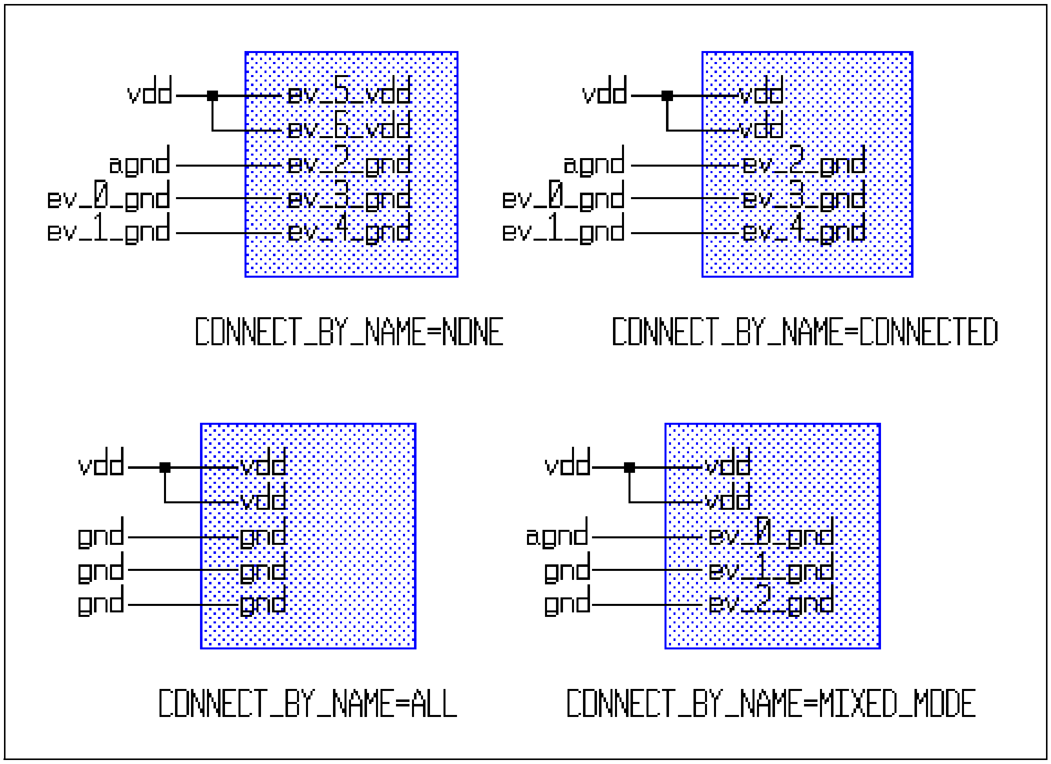
- Reports TEXT_OPEN messages in *block*.LAYOUT_ERRORS file.
- Merges opens nets and treats the nets as one net regardless of whether or not the open is resolved in the parent hierarchy.
- Allows blocks to compare even though text opens may exist.
- All pieces of a net in the text open must be treated as one net for ERC and NODAL checks.
- May result in shorts due to opens in child cells. (See Figure 5-15 below.)

When CONNECT_BY_NAME=MIXED_MODE:

- Reports top-level open nets with TEXT_OPEN messages in *block*.LAYOUT_ERRORS file.
- Merges top-level open nets and treats them as one net.
- Each physical net that is open in a sub-block results in a TEXT_OPEN_RENAME message, and is given a unique prefix (for example, ev_0_GND) *only* if they are *not* connected somewhere in parent hierarchy.
- Allows blocks to compare, even though text opens may exist in top block.
- All pieces of a net in the text open must be treated as one net for ERC and NODAL checks.

Figure 5-15 shows how CONNECT_BY_NAME controls the renaming of nets associated with text opens.

Figure 5-15 CONNECT_BY_NAME



Suppressing TEXT_OPENS When Connecting by Name

CONNECT_BY_NAME and colon texting are independent functions. There is *no* setting for CONNECT_BY_NAME which suppresses opens.

Setting CONNECT_BY_NAME to ALL is *not* equivalent function to colon texting; they are very different functions. There are two major questions related to how Hercules treats multiple physical nets which are identically texted (text open).

- Do you want to merge these nets into one logical net for NODAL or ERC commands, such as C_THRU?
- Do you want to suppress the opens messages?

CONNECT_BY_NAME=ALL allows merging of identically texted nets *regardless* of the use of colon texting. However, setting CONNECT_BY_NAME=ALL will *not* suppress any opens messages.

TEXT Short Finding

Text Short messages are reported when more than one text string is assigned to a given net. An example of a short error message is located in the *block*.LAYOUT_ERRORS file, and the tables of descriptions can be found within the TEXT Command in the *Hercules Reference Manual*, Detailed Commands chapter.

The cause for a text short may not always be obvious. Hercules has three methods for locating shorts in your layout. The first method is to use a feature within the GUI debugging tool. It is an interactive short finder, run from the GUI debugging tool interface. The SHORTEST_PATH feature generates graphical output to identify a path between the selected start and stop coordinates. Use the interactive short finder for small chips, macro blocks, or untexted net shorts.

The second way to locate shorts in your layout is to use the FIND_SHORTEST_PATH_BETWEEN_TEXT_SHORTS (FSPBTS) option in the TEXT_OPTIONS section of the Hercules runset. Use the FSPBTS option for chip-level power/ground or large texted net shorts. FSPBTS locates the path with the minimum distance between two shorted text points. When using FSPBTS, the REMOVE_TEXT_FROM_SHORT option must be set to FALSE.

When using the FSPBTS option, the shortest path is reported in two phases, polygon extraction (polygon_path) and edge refinement (edge_path). This information is printed in the summary file.

- Polygon_path is equivalent to the path found by the GUI debugging tool. It highlights the path between start and stop points. Since this path can consist of many layers, Hercules writes out each layer on different datatypes to make debugging easier.
- Edge_path is the actual minimum path outputted to a user-specified output layer, and an error is written to the *block*.LAYOUT_ERRORS file.

Output can be viewed in the GUI debugging tool and/or written to a graphics output file (GDSII). Where polygon_path output highlights the entire polygon, edge_path output is a line drawn along the center of the polygon.

There are options associated with FSPBTS that allow you to filter out the analysis of some shorts. For instance, if you are only interested in detecting shorts associated with VDD text, you can use:

```
SHORTEST_PATH_NETS = {*!VDD }
```

For more details on how to use these methods, refer to the *Hercules LVS User Guide*, Analyzing and Debugging for LVS Extraction and COMPARE chapter.

PROCESS_TEXT_OPENS Command

The PROCESS_TEXT_OPENS command locates shorts and opens caused by text. In addition to locating opens, PROCESS_TEXT_OPENS also merges the nets with identical text in the same cell. This function is automatically run with the following commands:

GRAPHICS
GRAPHICS_PROPERTIES
NETLIST
SAVE_NETLIST_DATABASE
SPICE

To merge identically texted nets, PROCESS_TEXT_OPENS should be run prior to the following nodal commands:

C_THRU
DEV_CONNECT_CHECK
DEV_NET_COUNT
DEVICE_COUNT
NET_FILTER
NET_PATH_CHECK
NODAL DRC Checks
RATIO

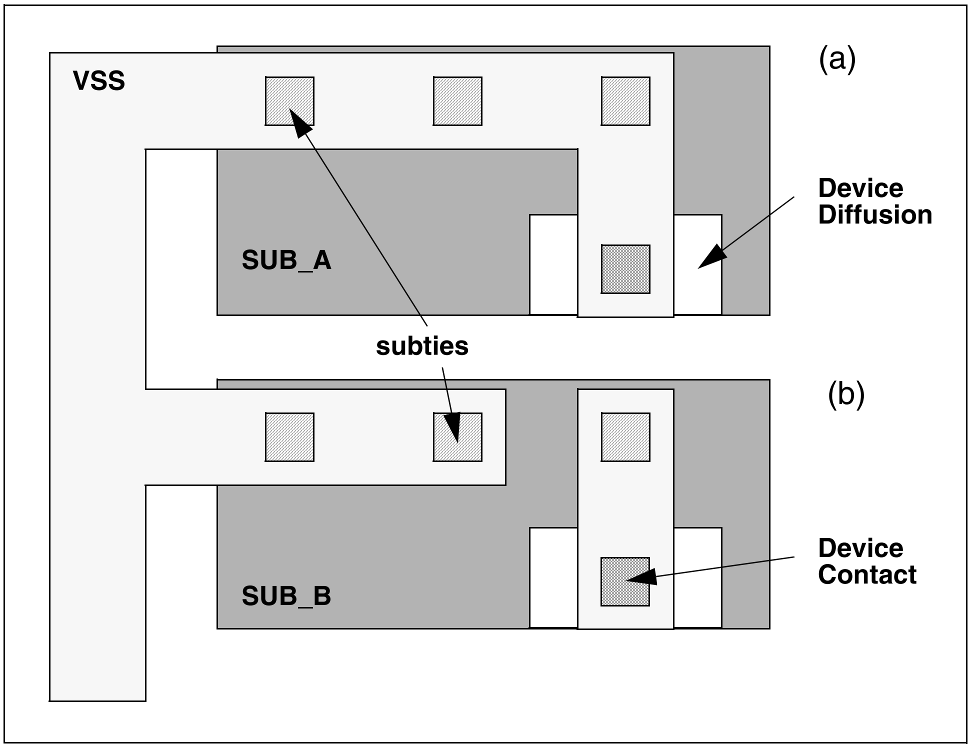
The command syntax is:

```
PROCESS_TEXT_OPENS  { }
```

Identifying Power-through-the-Well Situations

In order to maintain different isolated substrate regions at different potentials (as in a mixed digital-analog chip), each region needs to be tied to a different power rail. Any MOS devices in those regions that are tied to power should also be tied to the same power rail. A common problem arises when both the rail and the devices are tied to the substrate but not to each other. This can cause the voltage available for driving the device to fall below specifications and interfere with the chip's operation. Figure 5-16 shows (a) how the layout should be drawn to prevent this problem, and (b) how this problem can occur when the rail does not extend to the device.

Figure 5-16 Example of the Power-through-the-Well Problem



Power-through-the-well problems cannot be identified by Hercules LVS operations because, as far as LVS is concerned, there is no error. In Figure 5-16 there is a path along the VSS net to each subtie and, ultimately, to the device diffusion in both cases a and b. LVS makes no distinction whether this path is formed by metal or by NWELL. From an LVS standpoint, the two circuits are identical. However, from an ERC viewpoint, the path in b formed only by the subties and SUB_B layer is significant.

Power-through-the-well problems are identified in Hercules using the C_THRU (connected through) ERC command. This command checks to see that the layerA polygons inside the same layerB polygon all belong to the same net when layerB is not included in the CONNECT sequence. The three subties in Figure 5-16 (a) are all connected to the power rail and therefore pass a C_THRU check. Two of the subties in Figure 5-16 (b) are also connected to this net, but the third is connected to a different net. The C_THRU command will flag this as an error. To use the C_THRU command, first establish the electrical connectivity of all of the layers that interact with the subties, except that of the substrate. Then use the C_THRU command to see if the different subties can only be connected to each other by the substrate. The following example shows how the check might be coded using the absolute minimum required connectivity information.

```
CONNECT { METAL1 BY subtie }
C_THRU subtie INSIDE sub_well {
} (100;20)
```

Debugging C_THRU Errors

The C_THRU error text report in the *block*.LAYOUT_ERRORS is formatted to help debug C_THRU violations. The following points describe certain sections of a C_THRU text report:

- The violation count in the ERROR SUMMARY section matches the number of violating well polygons.
- C_THRU nets in the ERROR DETAILS section are sorted based on the number of connected layer1 polygons. Nets with the least amount of layer1 polygons are printed first since these nets are usually the problematic nets. Nets with the highest number of connected layer polygons are sorted last and are usually the good nets.
- The C_THRU error report shows the top and bottom hierarchical points of the violating net. The bottom hierarchical point generally contains the layer1 polygons. The top hierarchical point is the location of the disconnected net. All reported nets are reported relative to the parent cell.
- The layer coordinates are reported relative to the parent cell.

Example 5-5   C_THRU Error Report

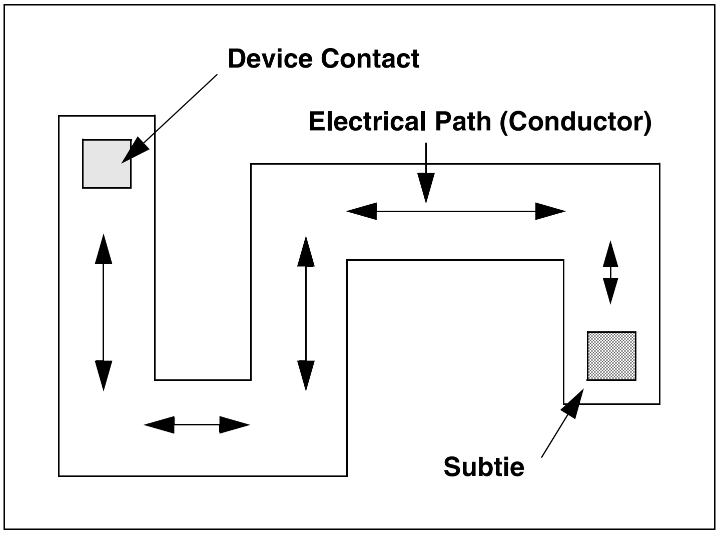
```
 ERROR SUMMARY

No Comment
 C_THRU subtie INSIDE subwell { } (223) ....... 6 violations found.

 ERROR DETAILS
#####--- ERR_C_THRU -----------------------------------

C_THRU subtie INSIDE subwell { ERROR_OUTPUT=FULL } (223)

- - - - - - - - - - - - - - - - - - - - - - - - - - - -

Parent Cell Layer Pgon Bounding Box Net Hierarchical End Points
(x1, y1)(x2, y2)(top of net <-> bottom of net)
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
CHILD_CELL SUBWELL (195.33, -196.75) (207.2,-181.30)
--- C_THRU Summary: 2 Nets ---

subtie    (196.58, -193.00) (196.710, -192.57)
./N_20 <-> CHILD_CELL_2/N_2
--- Net Summary: 1 pgon ---

subtie     196.04, -185.86) (196.36, -185.22)
./N_18 <-> CHILD_CELL_5/N_5
subtie    (196.04, -183.12) (196.36, -182.69)
./N_18 <-> CHILD_CELL_5/N_2
subtie    (196.58, -195.64) (196.710, -195.11)
./N_18 <-> CHILD_CELL_2/N_5
--- Net Summary: 3 pgons ---
```

Preventing Latchup

Latchup occurs when an effective short circuit between power and ground is created as a result of parasitic interactions in a chip. Some technologies (for example, Silicon-on-Insulator, or SOI) are inherently resistant to latchup phenomena, while others are open to modifications that increase their resistance to the problem. Layout design techniques are also available to help prevent latchup problems. A common approach to this problem is to ensure that the electrical path between a MOS-device contact subject to latchup and the nearest subtie does not exceed a maximum critical value specific to a given technology.

Figure 5-17 illustrates a common situation. The conductor in this example is the substrate to which the device is being tied through the subtie indicated. Its path is drawn to emphasize the fact that it is the electrical path length that matters when determining whether latchup conditions exist, and not just the straight-line distance between the contact and the subtie. Because of this, an EXTERNAL check will not provide the desired check.

Figure 5-17 MOS Device Connected to Subtie for Latchup Protection



The bounded SIZE INSIDE command is used to perform this check. This command repeatedly oversizes the initial target layer (for example, the device contact) and ANDs the result with the conductor layer (see the *Hercules Reference Manual*, Detailed Commands chapter, for details). Example 5-6 shows how the check can be coded. In this example, a subtie must be less than 25 µm (measured along the electrical path) from a device contact to provide latch-up protection. Preliminary operations (not shown) have identified the device contacts that are being protected, the layers defining the electrical path being checked, and the subties that are supposed to be providing the protection.

Example 5-6 Basic Latchup Violation Check

```
SIZE dev_cont INSIDE conductor {
     INCREMENT = 2.5
     OVERSIZE = 25.0
} TEMP = latch_up
BOOLEAN latch_subtie INSIDE latch_up {
     COMMENT = "Subtie too far from device contact"
} (100;1)
```

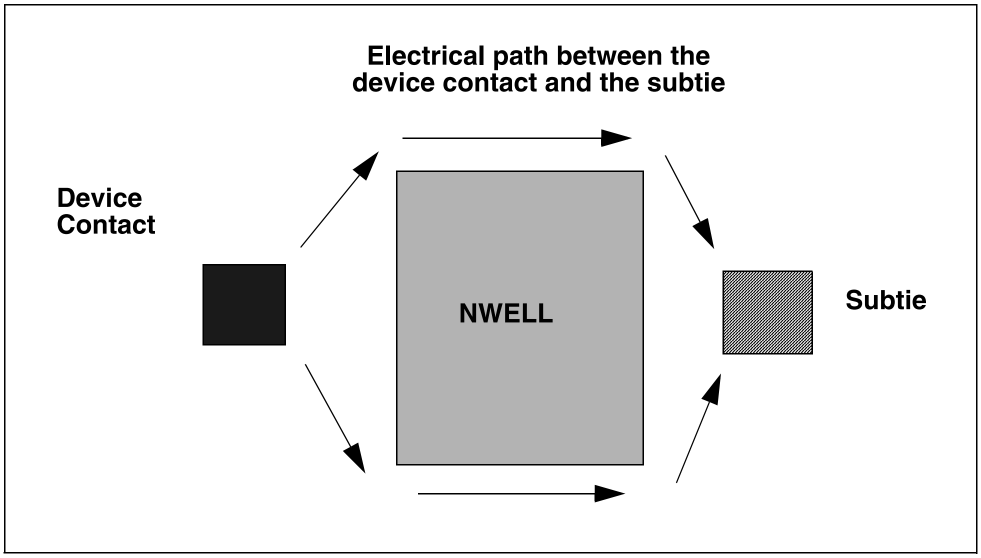
The SIZE INSIDE command can also be used to generate two separate output layers—one corresponding to the default output and one corresponding to the DELETE_NOT output. This is useful when a design contain two different types of subties that can be different distances from the devices to which they are connected. Therefore, rather than performing two independent checks, both of which had to start from the same initial conditions, time can be saved by starting the second check from the point where the first check stopped. Now the normal output from the two-layer SIZE INSIDE command can be used to perform the latch-up check while the DELETE_NOT layer can be used as the starting point for the second SIZE command. Example 5-7 illustrates how this can be done.

Example 5-7 Latchup Check Using Two Different Subties with Different Spacing Rules

```
/* This code identifies subtie_type1 polygons that are
   more than 25 um away from a dev_cont polygon, and
   subtie_type2 polygons that are more than 55 um away
   from a dev_cont polygon. */

SIZE dev_cont INSIDE conductor {
     INCREMENT = 2.5
     OVERSIZE = 25
} TEMP = too_far_one  TEMP = restart
SELECT subtie_type1 INSIDE too_far_one {
     COMMENT = "Type one subties >= 25 um from dev_cont"
} (100;4)

SIZE restart INSIDE conductor {
     INCREMENT = 2.5
     OVERSIZE = 30.0
               /* You only need to go the remaining distance */
} TEMP = too_far_two
SELECT subtie_type2 INSIDE too_far_two {
     COMMENT = "Type two subties >= 55 um from dev_cont"
} (100;5)
```

Sometimes latchup checks require consideration of the electrical path around an intervening obstacle rather than through a conductor. One situation where this might occur (illustrated in Figure 5-18) is when an NWELL sits between a subtie and the device it is protecting.

Figure 5-18 An Intervening NWELL Requires Use of the SIZE OUTSIDE Command to Check for Latch-up Conditions



To address this situation, Hercules provides the SIZE OUTSIDE command. This command grows a connecting polygon that tries to connect the device contact and the subtie, NOTting away that part of the polygon that overlays the intervening barrier polygon. The following example illustrates the command syntax. Further information is available in the *Hercules Reference Manual*, Detailed Commands chapter.

```
SIZE n_dev OUTSIDE NWELL TO Ptap {
    OVERSIZE = 100
    INCREMENT = 20
} TEMP = too_far
```

Identifying Nets/Devices Not Tied to Power or Ground

The gates of all functional MOS devices in a design should be traced back to an input signal, a power rail, or a ground. If they cannot be traced back to those nets or devices, there will not be any way to use those devices as switches. The NET_PATH_CHECK command (described in the *Hercules Reference Manual*, Detailed Commands chapter) is designed exclusively to carry out this type of check. Example 5-8 shows a common use of NET_PATH_CHECK. A series of CONNECT statements (not shown) establishes how the various layers in the design are connected to each other. The NET_PATH_CHECK command is used to locate any nets that cannot trace an electrical path to either a power or ground rail or to an input signal net (indicated by an IO prefix). The input layer set consists of the layers listed in the last CONNECT sequence before this command, and unused devices will be ignored.

Example 5-8 Example NET_PATH_CHECK Command

```
NET_PATH_CHECK {
        NET1 = { VDD* vdd* }
        NET2 = { VSS* vss* }
        NET3 = { IO_* }
        PATH_TO = NONE
        IGNORE_UNUSED_DEVICES = TRUE
        DEVICE = { NMOS* }
 } (100;50)
```

Antenna Violations and Repair Flows

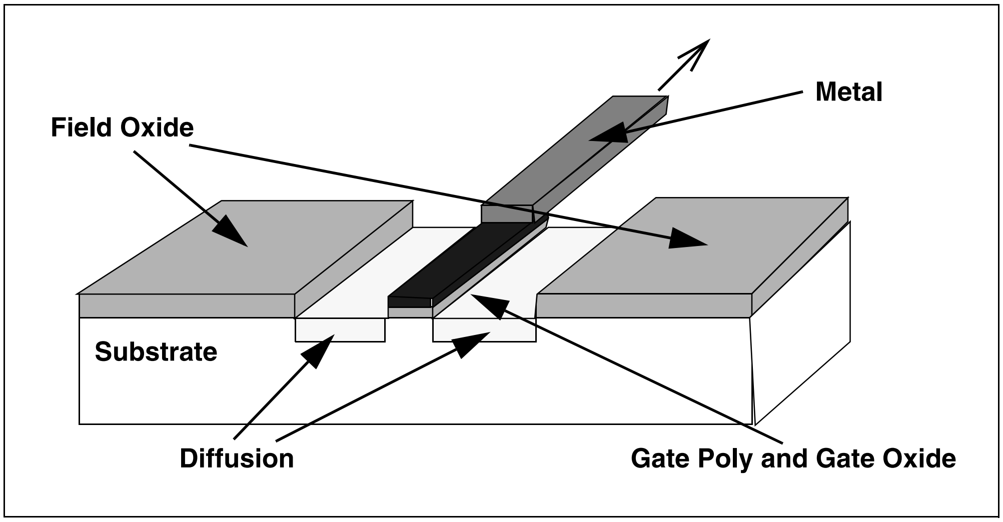
As IC fabrication moves to smaller technologies, charge accumulation on the chip during the manufacturing process becomes a significant problem. If left unchecked, the charge can eventually build to the point where it creates an electrostatic discharge (lightning). This discharge can damage the chip and make it inoperable. The accumulation usually takes place where there are long runs of metal in the design, and so the problem is often referred to as an antenna problem.

Hercules provides a number of tools designed to address the antenna problem, some of which work in conjunction with the Astro Place & Route tool. This section describes the tool used to identify potential antenna violations. It also addresses how these violations can be prevented by stitching (that is, by moving up or down to a different metal, thus limiting the size of the potential antenna layer). "Diode Insertion Using the Astro/Hercules Interface" on page 5-52 describes how antenna violations can be prevented by the insertion of charge-dissipation diodes.

Antenna Problem Defined

Figure 5-19 shows a simplified diagram of a portion of the layers that make up a MOSFET device. It includes the diffusion layers that will make up the source and drain regions of the device, the gate oxide (thin oxide) and polysilicon that will form the device's gate, and the metal wire that connects the gate to other parts of the circuit. (The contact between the polysilicon and the metal is not shown. The arrow extending from the metal indicates that this layer continues for an indeterminate distance.)

Figure 5-19 Antenna Problem Illustrated

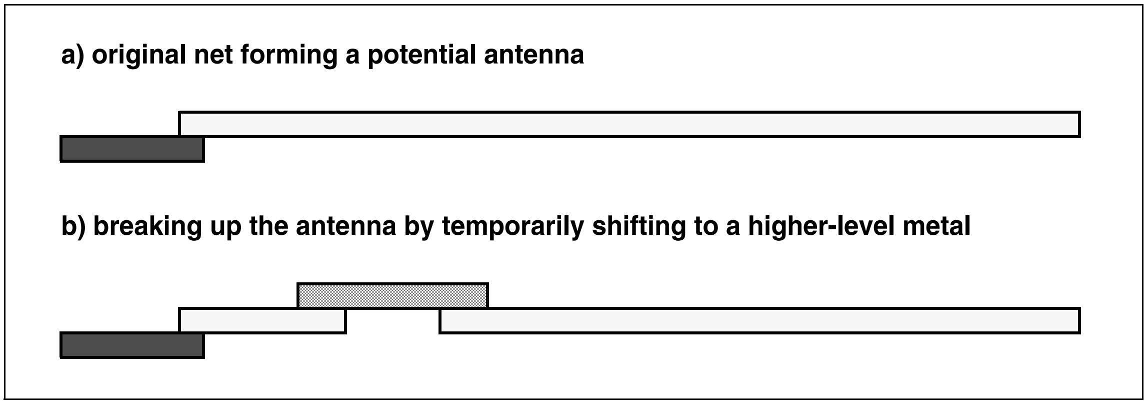


During the IC manufacturing process, the metal layer is exposed to conditions that lead to the build-up of an electrostatic charge. The amount of charge that builds up depends on a number of factors; from an antenna standpoint, the most important is the amount of metal exposed. As more metal is exposed, the maximum charge that accumulates on the net of which the metal is a part also increases. The substrate remains at ground since it is connected to the fabrication device. As a result, a voltage gradient develops across the gate oxide. When this gradient becomes large enough, it is relieved by an explosive discharge (lightning). The problem is more significant at smaller technologies because the damage resulting from the discharge is more likely to extend across the entire length of the gate.

The conditions that lead to antenna formation depend on the technology used to fabricate the chip and must be determined empirically for each process. Once they have been identified, they can be used to define a set of antenna rules, similar to conventional DRC rules, that can be coded into Hercules. When defining antenna rules, a factor to consider is whether the antenna should be based on the “top” area of the metal or on its sidewall area. In an aluminum-based process, charge accumulation occurs during the ETCH step. The top of the metal is protected by a resist during this step, so the antenna rules for this process should be based on the metal sidewall area. In copper-base technologies, charge accumulation occurs during CMP (Chemical-Mechanical Polishing). In this process, the sides of the metal are protected, so the antenna rules need to be based on the metal’s top surface area.

When a potential antenna has been identified in a design, there are two ways to protect the circuit from its effects. One is to insert a reverse-bias protection diode into the circuit as close to the devices being protected as possible. This solution is discussed in detail later in this chapter. The other is to break up the antenna by shifting briefly to a different metal. This is illustrated in Figure 5-20. Illustration (a) shows a gate connected to a long metal layer that forms an antenna violation. In illustration (b), the metal has been broken into two pieces. When this metal layer is fabricated, the long piece on the right is no longer electrically connected to the gate and does not contribute to any antenna effects. When it is eventually connected through the higher-level metal bridge, it is no longer exposed to the charge accumulation and, again, does not contribute to an antenna violation.

Figure 5-20 Shifting a Net to Break Up an Antenna (Vias and Contacts Not Shown)


a) original net forming a potential antenna
b) breaking up the antenna by temporarily shifting to a higher-level metal

Antenna Identification in Hercules

There are two commands in Hercules that can be used to identify antenna violations.

- The RATIO command is intended to provide a relatively fast check on a limited set of antenna rules.
- The NET_FILTER command is used for more complex checks and offers more information (NET_FILTER is required when using the ANTENNA_FIX command.)

All commands are described in detail in the *Hercules Reference Manual*, Detailed Commands chapter.

The following examples demonstrate how the two commands can be used to identify antenna violations. Each example begins with a brief description of the antenna rule being tested. This is followed by the code that might be used to perform the check, including any preceding CONNECT statements that the commands might require. Finally, each example will end with a discussion of any unique aspects of the code whose purpose might not be self-evident.

Example 5-9 illustrates this rule:

> Rule: The total area of any individual metal layer that is on a net connected to a gate should not be more than 100 times greater than the total gate area on that net.

In Example 5-9 these rules are written to check metal2, but similar rules can be written for other metal layers. The checks are restricted to nets that are connected to gate polygons. Without this restriction Hercules would process all nets. Nets that did not contain any gate polygons would generate a divide-by-zero error. Hercules recognizes this situation and ignores these nets in its calculations.

Example 5-9

```
CONNECT {gate BY ipoly}
CONNECT {ipoly METAL1 BY CONT}
CONNECT {METAL1 METAL2 BY VIA1}
...
RATIO METAL2 WITH gate CONNECTED gate {
     RATIO >= 100
     ERROR_COORD = SET2
} (100;1)
...
NET_FILTER {
     LAYER_SET[1] = { METAL2 }
     LAYER_SET[2] = { gate }
     ratio = EV_AREA[1]/EV_AREA[2]
     FILTER_RULE = { ratio >= 100 }
     CONNECTED = { *gate }
     ERROR_SET = { gate }
} (100;1)
```

Example 5-10 illustrates this rule:

> Rule: The total perimeter of all metals, up to and including the current metal, should not be more than 250 times greater than the total gate area connected to a given net.

In Example 5-10, only the applicable RATIO command is included; an equivalent NET_FILTER command can easily be generated for this check. Any nets that already contain protection diodes are excluded from consideration. Power and ground nets are also excluded from consideration on the basis that any power/ground nets connected to the gate of a MOSFET reflect problems that would need to be caught in another check.

Example 5-10

```
CONNECT {gate BY ipoly}
CONNECT {ipoly METAL1 BY CONT}
CONNECT {METAL1 METAL2 BY VIA1}
CONNECT {METAL1 METAL2 BY prot_diode}
...
TEXT {METAL1 WITH METAL1.TEXT}
TEXT {METAL2 WITH METAL2.TEXT}
...
RATIO ipoly METAL1 METAL2 WITH gate
   CONNECTED gate NOT_CONNECTED prot_diode
   NOT_TEXTED_WITH “PWR” “GND” {
     PERIMETER_OVER_AREA > 250
     ERROR_COORD = SET1
} (100;2)
```

Example 5-11 illustrates this rule:

> Rule: The total sidewall area for all metals, up to and including the current metal, should not be more than 250 times greater than the total gate area connected to a given net. The total top area for all metals, up to and including the current metal, should also not be more than 250 times greater than the total gate area on a given net.

Example 5-11 shows how the thickness of each metal layer can be used to estimate the sidewall area of that net. This information must be provided to Hercules manually because there is no way to access the information automatically. Each metal layer has a different thickness. Therefore the sidewall areas of each layer must be calculated individually before adding them together to get the total sidewall area for the net. (The variable perim_ratio, which is calculated in the command and listed in the error report, lists the perimeter-over-area ratio that the RATIO command calculates so that it can be compared to the more rigorous sidewall area calculations.) This example also illustrates how two different filter rules can be combined in the same check. In this case, it does not matter which of the two criteria flag the net as an antenna violation; if either condition is met, the net is reported.

Example 5-11

```
VARIABLE DOUBLE M1_thick = 0.6;
VARIABLE DOUBLE M2_thick = 0.8
...
CONNECT {gate BY ipoly}
CONNECT {ipoly METAL1 BY CONT}
CONNECT {METAL1 METAL2 BY VIA1}
CONNECT {METAL1 METAL2 BY prot_diode}
...
NET_FILTER {
   LAYER_SET[1] = { gate }
   LAYER_SET[2] = { ipoly }
   LAYER_SET[3] = { METAL1 }
   LAYER_SET[4] = { METAL2 }
   LAYER_SET[5] = { METAL1 METAL2 }
   net_area = EV_AREA[5]
   gate_area = EV_AREA[1]
   sidewall = (EV_PERIM[3] * M1_thick) + (EV_PERIM[4] * M2_thick)
   area_ratio = net_area/gate_area
   sidewall_ratio = sidewall/gate_area
   perim_ratio = EV_PERIM[5]/gate_area
   FILTER_RULE = {(area_ratio > 250) || (sidewall_ratio > 250)}
```

```
    CONNECTED = { *gate  !prot_diode }
    ERROR_SET=LAYER_SET[5]
    ERROR_REPORT = { gate_area, area_ratio, sidewall_ratio,
           perim_ratio }
} (100;2)
```

Example 5-12 illustrates this rule:

> Rule: a) The maximum ratio of metal area (for metal 1–5) to gate area (for nets not protected by diodes) is 200.
> Rule: b) When present, protection diodes offer safeguard to additional metal areas equivalent to an area 100 times that of the diode.

Example 5-12 shows a case where protection diodes are assumed to offer limited protection against antenna checks. The first NET_FILTER command checks nets that do not contain a protection diode (this check can also be performed using the RATIO command). The second NET_FILTER command examines nets that do contain a protection diode. Because these diodes only offer limited protection, the command has to determine what conditions are required before the net can be considered an antenna. This check must be carried out using the NET_FILTER command, because the RATIO command does not allow equations in the command.

Example 5-12

```
CONNECT {gate BY ipoly}
CONNECT {ipoly METAL1 BY CONT}
CONNECT {METAL1 METAL2 BY VIA1}
CONNECT {METAL1 METAL2 BY prot_diode}
...
NET_FILTER {
     COMMENT = "Rule a: Nets without protection diodes"
     LAYER_SET[1] = { gate }
     LAYER_SET[2] = { METAL2 }
     ratio = EV_AREA[2]/EV_AREA[1]
     FILTER_RULE = {ratio > 200}
     CONNECTED = { *gate  !prot_diode }
     ERROR_REPORT = { ratio }
} (100;5)

NET_FILTER {
     COMMENT = "Rule b: Nets with protection diodes"
     LAYER_SET[1] = { gate }
     LAYER_SET[2] = {METAL2 }
     LAYER_SET[3] = { prot_diode }
     ratio = EV_AREA[2]/EV_AREA[1]
     diode_protection = (100*EV_AREA[3])/EV_AREA[1]
     FILTER_RULE = { ratio > 200 + diode_protection }
     ERROR_REPORT = { ratio, diode_protection }
     CONNECTED = { *gate *prot_diode }
} (100;6)
```

The Hierarchical Antenna Report (Astro/Hercules Interface)

Layout engineers often save time in the design process by re-using existing elements of previous designs, or by purchasing intellectual property (IP) from third-party vendors. These hard macros are usually ERC-clean with respect to antenna violations, but the process of routing to their ports can sometimes create an antenna. The Astro Place & Route tools address this potential problem with commands that will jog the routing layer to break up the antenna. The success of these commands depends on the accuracy of the information provided to them about the nets in the IP blocks.

The Astro feature known as Hierarchical Antenna Report allows Astro to use Hercules to extract antenna-related properties from hard macros. It is invoked from Astro, and the Hercules aspects of the feature are transparent to the user. It is discussed here so that Hercules users understand how it works and the results it generates.

System Requirements and Data Preparation

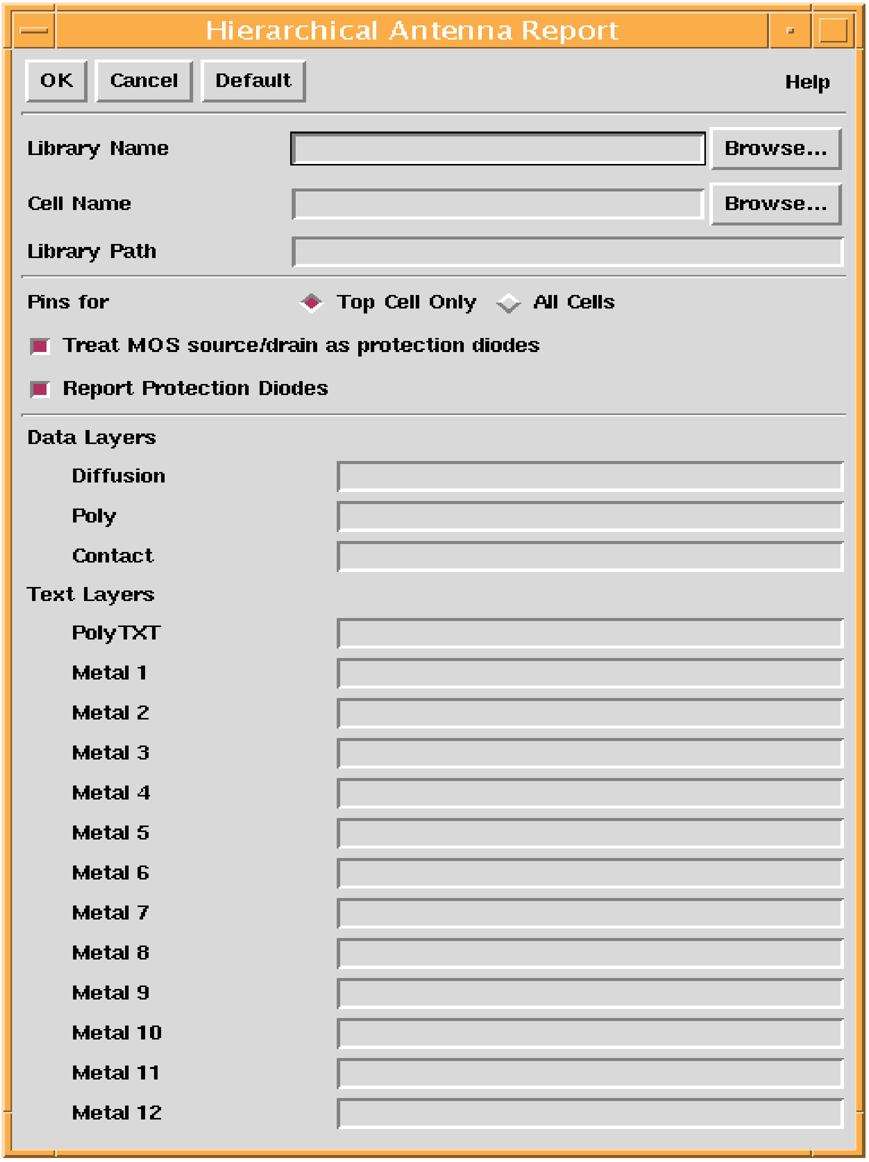
In addition to the basic product licenses, you also need the Astro HPO options and the Hercules advanced ERC license. Hercules must be visible in the path from which you are running your Place & Route tool so that it can be called successfully from that tool.

In order to apply the hierarchical antenna interface, the block to which it is being applied must already be converted into a macro. To check this, look in the Milkyway library's FRAM view and see if the top block of the macro is listed as a cell in this view. If it is not, the Astro menu item Make Macro Abstract needs to be invoked. This brings up a GUI that controls the macro creation process.

Generating the Report

To generate the hierarchical antenna report, enter the hmiHierAntenna command in the Astro command window. There is no menu item for this. Executing this command will bring up the GUI shown in Figure 5-21.

Figure 5-21 hmiHierAntenna GUI



Pins for ◆ Top Cell Only ◇ All Cells

☒ Treat MOS source/drain as protection diodes

☒ Report Protection Diodes

The first section of the GUI is used to identify the macro to which this interface is being applied. Enter the library and cell names and the library path (if necessary) into the appropriate fields.

The next section of the GUI is used to set options that govern how the antenna-related properties will be extracted from the macro. These options operate as follows:

- Pins for Top Cell Only/All Cells—When set to Top Cell Only, Hercules only extracts antenna-related properties for nets connected to pins in the top block of the macro. When set to All Cells, Hercules also extracts the properties of nets whose pins are in lower-level cells.
- Treat MOS source/drain as protection diodes—When selected, this option tells Hercules to treat MOS source/drain regions as equivalent to protection diodes when reporting on the amount of diode protection existing on a net.
- Report protection diodes—When selected, this option tells Hercules to report the total area of all protection diodes on each net. This option must be set if the previous option is selected.

The last section of the GUI is used to identify the layers in the database. The first three entries, diffusion, poly, and contact, are required. Enter the layer numbers for those layers that contain the diffusion, poly, and contact polygons. The remaining fields are required only if there is text for the layer that is not on the same layer as the polygons. Enter the number(s) of the appropriate text layer(s) in these fields. In every field, multiple layer numbers should be separated by a comma.

Once all of the required data in this form have been entered, click OK to create the runset and execute Hercules. The message “Hercules is running” appears in the command window, and a series of dots is displayed to indicate that Hercules is executing. When Hercules is finished, the message “Hercules is done” is displayed. This is the only indication that Hercules is executing.

Verifying Successful Operation

Hercules writes the results of the hierarchical antenna report directly into the Milkyway database. Confirm that antenna-related properties from the macro have been successfully extracted by writing the Cell Library Format (CLF) data to a file. Close and re-open the library containing the macro before doing this to remove any lock files set on that library.

To write the CLF to a file, select Tool > Data Prep in Astro. Select the CLF > Write to File, as shown in Figure 5-22. This will bring up the GUI displayed in Figure 5-23. Enter the name of the file to which the CLF information will be written in the CLF File Name field. Enter the name of the library containing the macro in the Library Name field and then click OK. If property extraction was successful, the file generated will contain entries similar to those in either Example 5-13 or Example 5-14, depending on which Astro licenses you have. If extraction was not successful, this file will be empty.

Figure 5-22 Write CLF to FIle Menu Entry

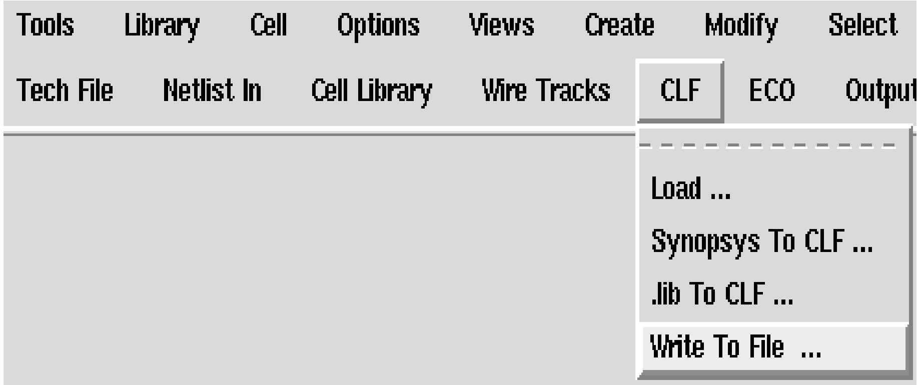
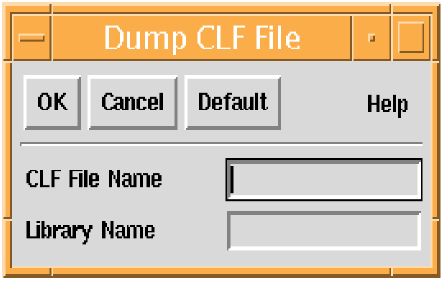


Figure 5-23 Write CLF to File GUI



Example 5-13 Representative Output from CLF

```
defineDiodeProtection "nand_macro" "NET_A" '(895.184 1288.94 1288.94
1288.94 1288.94)
;; "nand_macro" "NET_A" has hier antenna prop
defineDiodeProtection "nand_macro" "NET_B" '(108.835 250.593 256.369
256.369 256.369)
;; "nand_macro" "NET_B" has hier antenna prop
. . .
```

Example 5-14 Alternate Output from CLF

```
defineDiodeProtection "nand_macro" "IN0" '(0 0 0 0 0)
 (defineHierAntennaProp "nand_macro" "IN0" '(
     ("METAL2" 0 6.24 6.24 8.7456 21.648 21.648 30.4656)
     ("METAL3" 0 0 6.24 8.7456 0 21.648 30.4656)
     ("METAL4" 0 0 6.24 8.7456 0 21.648 30.4656)
     ("METAL5" 0 0 6.24 8.7456 0 21.648 30.4656)
```

In Example 5-13 and Example 5-14, notice that the first line in the CLF output (defineDiodeProtection) lists the diode protection areas found by Hercules on the indicated nets (nand_macro is the name of the block, while the other text entry is the name of the net). The first entry indicates the total diode area when the net is traced from the gates through metal-1, the next is when the net is traced through metal-2, and so on. The remaining lines in Example 5-14 list the other antenna-related properties of the indicated net when the net is traced through to the different metal layers. The first number lists the total gate area on the net, while the remaining numbers list the metal area corresponding to the six Astro antenna modes.

If you are getting the summary output shown in Example 5-13, but you need the detailed output shown in Example 5-14, contact your Astro support engineer for assistance.

Assumptions Used in the Hierarchical Antenna Report

When the Hierarchical Antenna Report feature is invoked, Astro generates a Hercules runset to perform the property extraction.This runset makes the following assumptions:

- Gates are formed by the BOOLEAN POLY AND DIFF, where DIFF is all diffusion;
- Source/drain regions are formed by the BOOLEAN DIFF NOT gate;
- Protection diodes are formed from a combination of diffusion, contacts, metal, and vias in an appropriate sequence. Diodes are not formed where there is a source/drain region.

Potential Problems

Problem: The file containing the CLF data is empty.

Solution: The most likely situation is that the block you are trying to extract has not been converted to a macro. Confirm this by looking at the contents of the FRAM directory within the library containing the macro. If there is no file in this directory with the same name as that of your proposed macro, then the macro has not been created. For help in creating macros, consult the Astro documentation or your local Astro support engineer.

Problem: Astro claims that there is no hierarchical antenna information for ports that you know exist in the macro. It may also list other nets as antennas that you do not think should be listed, claiming that they contain more metal than you think is reasonable.

Solution: If your design contains any physical shorts, Hercules combines the shorted nets into one big net, selects one of the texts as the name of the net, and discards any other names attached to that net. This information is currently not passed back to Astro to warn you that there is a short circuit in the design. If these symptoms occur, look for a possible short circuit. For information on short-finding in Astro, check the Astro documentation or contact your local Astro support engineer. For information on short-finding in Hercules, see the *Hercules LVS User Guide*, Analyzing and Debugging for LVS Extraction and COMPARE chapter.

Charge-Accumulation Antenna Repair

Hercules can be used to repair charge-accumulation antennas so they do not damage the devices to which they are connected. The following discussion describes how this is done. However, if you are not familiar with antenna identification in Hercules, we recommend that you review the preceding sections of this chapter before proceeding with this material.

As you go through this material, keep in mind that the ANTENNA_FIX command is not intended to be the primary mechanism used to provide antenna protection and repair to a design. Most antenna issues should be resolved during Place & Route, and the Hercules software antenna-repair functionality should be reserved for those few violations that have somehow been missed during Place & Route. The ANTENNA_FIX command has to do a significant amount of data processing, and, if a large number of nets need repair, runtimes can be excessive. In addition, antenna repairs introduced by Astro are more likely to be retained if the design is modified (this is discussed in more detail later in this section).

Antenna Identification

The procedures used for antenna identification were described above. If this information is to be used to repair antennas, it must be communicated to the antenna repair commands later in the runset (see below). To do this, you will need to add the option OUTPUT_FILE to the NET_FILTER commands that identify the antenna violations. This option causes Hercules to generate a file containing an ASCII list of the nets it has identified as being an antenna. Example 5-15 provides a simple example of a NET_FILTER command that includes this option. Note that the RATIO command cannot be used for antenna repair because it is unable to generate this file.

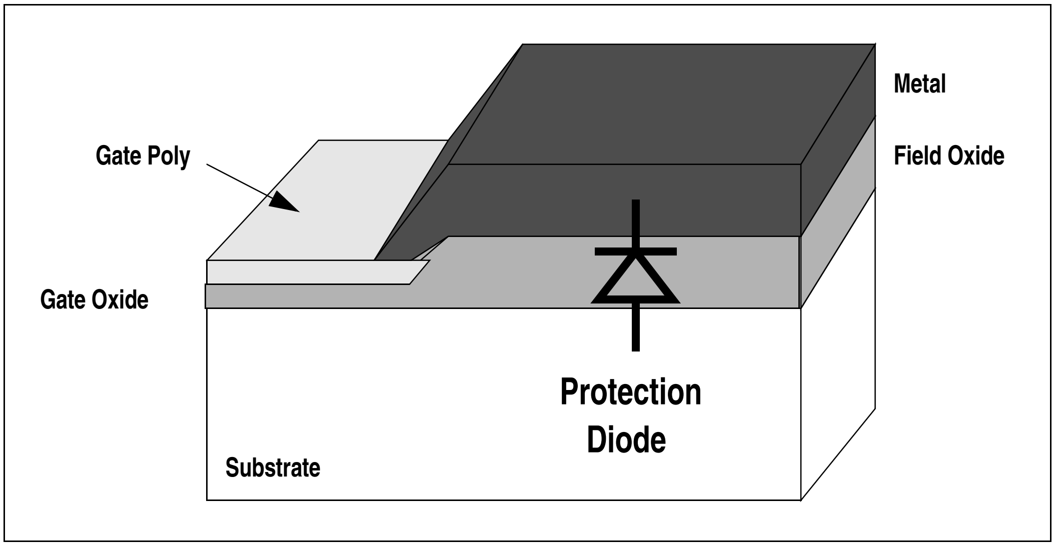
Example 5-15 Example NET_FILTER Command with OUTPUT_FILE Option

```
CONNECT {gate by ipoly}
CONNECT {ipoly METAL1 by CONT}
CONNECT {METAL1 METAL2 by VIA1}
. . .
NET_FILTER {
   LAYER_SET[1]={ METAL2 }
   LAYER_SET[2]={ gate }
   ratio = EV_AREA[1]/EV_AREA[2]
   FILTER_RULE = {ratio >= 100}
   CONNECTED = { *gate }
   OUTPUT_FILE = MET2_NEEDS_DIODE
} (100)
```

Antenna Repair

Antenna repair in Hercules is accomplished by inserting a reverse-bias diode on the violating net as close to the gates being protected as practical, as illustrated in Figure 5-24. During normal chip operation, the reverse bias prevents electrons from flowing from the net through the diode and into the chip's substrate. During fabrication, however, the charge on the net can build to the point where the voltage drop across the diode exceeds its breakdown voltage. This voltage is greater than the normal operating voltage, but less than the voltage at which an electrostatic discharge at the gate can be expected. When this happens, the diode allows electrons to flow from the net to the substrate and thus limits how much charge can accumulate on the net. The process is not destructive, and it is possible that the net could discharge through the diode several times during the fabrication process.

Figure 5-24 Antenna Protection Using a Reverse-Bias Diode



The ANTENNA_FIX command inserts these diodes into the design. This command (described in the *Hercules Reference Manual*, Detailed Commands chapter) has four operations:

Data input
DRC specification
Layer processing
Diode definition

Data Input

Input to the ANTENNA_FIX command is the file generated by the NET_FILTER command containing the list of antenna violations. Typically, each NET_FILTER command generates a separate file, and each is paired with a separate ANTENNA_FIX command. This is done to make the runset more efficient. Assume, for example, that a net has been identified as having an antenna violation on metal 1. If this net is repaired by inserting a diode on metal 1, that diode also protects the net for all higher-level metals. Subsequent NET_FILTER commands can therefore be written to recognize this and not flag the net as an antenna, and subsequent ANTENNA_FIX commands do not have to process the net to add another diode.

DRC Specification

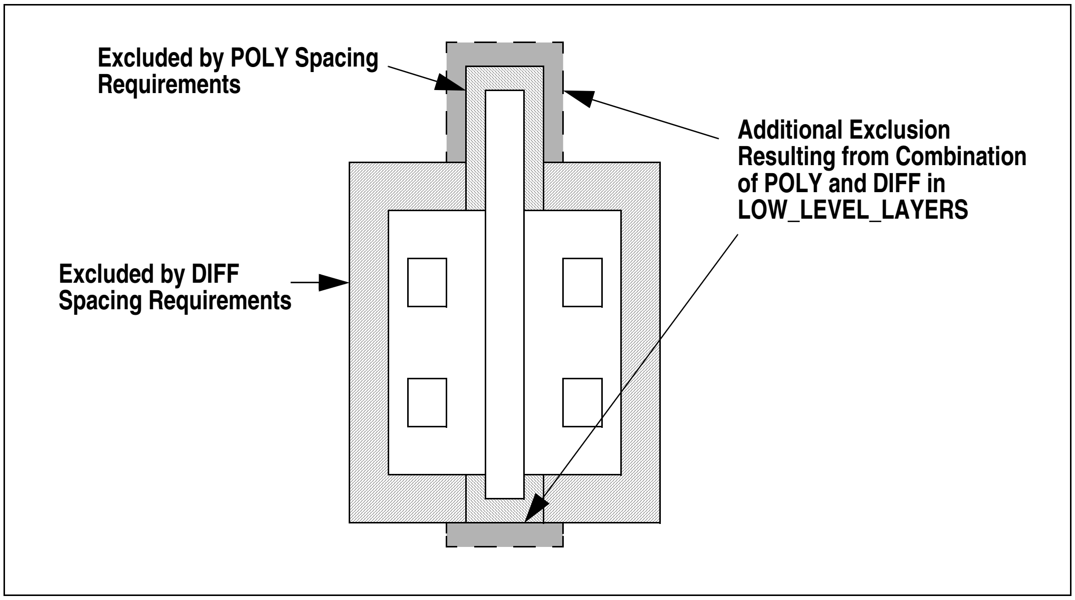
Hercules always tries to insert diodes into a design that obeys the DRC rules specified in the ANTENNA_FIX command. If it is unable to find a location where it can do this, no diode will be created. It is therefore recommended that each ANTENNA_FIX command be followed by a second NET_FILTER check to identify unrepaired antennas so they can be repaired by hand. It is also important to remember that the DRC checks in the ANTENNA_FIX command are EXTERNAL spacing checks only. It is your responsibility to ensure that the diode's design satisfies any other DRC rules (for example, internal dimensions or enclosure requirements).

The number of DRC checks the ANTENNA_FIX command needs to consider can often be reduced (resulting in improved runtimes) by careful consideration of how your design rules are implemented. The vias and contacts in a diode, for example, are generally smaller than and enclosed within the other layers that make up the diode. As a result, locations that are DRC-clean for these other layers are likely to be DRC-clean for the contacts and vias as well, and separate checks for these layers are probably unnecessary (see below). Notice, however, that this assumes that the rest of the design has already passed a full DRC check before being checked for antenna violations.

Layer Processing

When Hercules determines where to place a diode, it does so by identifying regions in the layout where the diode cannot be placed, based on the DRC checks described above. The process can be made more efficient by using the LOW_LEVEL_LAYERS option. This option merges the data in the relevant layers and reduces the number of interactions Hercules needs to consider. Effective use of the option also requires careful consideration of your design. In Figure 5-25, for example, the relevant DRC rules specify that diodes must be at least 0.5 µm from a diffusion (DIFF) layer, but only need to be 0.25 µm from poly. The exclusion zones resulting from these rules are indicated in Figure 5-25 by the diagonal shading. The gray shading indicates the additional exclusion zone that results from combining the two layers and applying the more restrictive design rule to the combination. While this makes it slightly more difficult to place the diode, it also significantly reduces the runtime required by the ANTENNA_FIX command.

Figure 5-25 Effective Use of LOW_LEVEL_LAYERS


Excluded by POLY Spacing Requirements
Additional Exclusion Resulting from Combination of POLY and DIFF in LOW_LEVEL_LAYERS
Excluded by DIFF Spacing Requirements

Diode Definition

The diodes created by the ANTENNA_FIX command are square diodes, with each layer centered around a common point. They are listed in the ANTENNA_FIX command starting with the bottom layer and working up in sequence. Layers designated as being routing layers mark the potential top of the diode, with the actual top being determined by where the diode is placed on the net. Hercules tries to insert diodes as low on the net as it can, as determined by the first Routing layer. For instance, if metal 1, metal 2, and metal 3 are all designated as Routing layers, Hercules tries to place the diode where it can build up to the first metal 1 polygon on the net. If Hercules cannot find a place to do this, it tries to find a place where it can connect to metal 2, and finally to metal 3. If only metal 3 is specified as a Routing layer, however, the ANTENNA_FIX command will skip the metal 1 and metal 2 attempts and will only try to place the diode on a metal 3 part of the net.

Runset Coding Examples

These examples show how the NET_FILTER and ANTENNA_FIX commands can be coded in a runset to identify and repair antenna violations in a design.

In Example 5-16, antennas are defined based on a basic area ratio check. Power and ground nets are excluded from consideration (presumably a previous DRC check has confirmed that these are not connected directly to any device gates), and we are only interested in nets that connect to a device gate. The list of nets needing repair will be written to the file M1_needs_fix, and the nets themselves will be written as graphical output to the error hierarchy.

Example 5-16 Runset Excerpt Illustrating Use of NET_FILTER and ANTENNA_FIX to Insert Protection Diodes in Charge-Collection Antennas

```
ASSIGN {
   NWELL   (10)
   NDIFF   (11)
   PDIFF   (12)
   CP      (13)
   CONT    (15)
   M1      (16)    TEXT (16)
   V1      (17)
   M2      (18)    TEXT (18)
   . . .
}

BOOLEAN CP AND NDIFF { } TEMP=ngate
BOOLEAN CP AND PDIFF { } TEMP=pgate

BOOLEAN ngate OR pgate { } TEMP=all_gate

BOOLEAN CP NOT all_gate { } TEMP=ipoly

BOOLEAN NDIFF NOT ngate { } TEMP=nsd
BOOLEAN PDIFF NOT pgate { } TEMP=psd

BOOLEAN nsd OR psd { } TEMP=all_sd

CONNECT {all_gate BY ipoly}
CONNECT {ipoly M1 BY CONT}
CONNECT {all_sd M1 BY CONT}

TEXT { M1 BY M1.TEXT }

NET_FILTER {
   LAYER_SET[1] = { all_gate }
   LAYER_SET[2] = { M1 }
   ratio = EV_AREA[2] / EV_AREA[1]
   FILTER_RULE = { ratio >= 30 }
   TEXT_LIST = { !VDD !VSS }
   CONNECTED = { *all_gate }
   ERROR_REPORT = { ratio }
   OUTPUT_FILE = M1_needs_fix
} (100)
```

In Example 5-17, the ANTENNA_FIX command creates, where possible, diodes that connect to the nets identified as antennas in the preceding NET_FILTER command. The four layers listed in the LOW_LEVEL_LAYERS option will be tested for DRC violations as a group. Two additional checks will also be made, one ensuring that the diode's diffusion is not placed too near a pre-existing NWELL, and one making sure that contacts are not placed too near a pre-existing via. Since only metal-1 antenna violations have been identified, only M1 has been specified as a routing layer.

Example 5-17

```
ANTENNA_FIX {
   NETS_FILE = M1_needs_fix
   LOW_LEVEL_LAYERS = { NDIFF, PDIFF, CP, CONT }
   DIODE_LAYERS = {
      NDIFF   0.4           TEMP = m1_diode_diff
      CONT    0.32          TEMP = m1_diode_cont
      M1      0.4   ROUTING TEMP = m1_diode_m1
   }
   DRC_TABLE = {
      LOW_LEVEL_LAYERS LOW_LEVEL_LAYERS 0.4
      NDIFF   NWELL   1.2
      CONT    V1      0.5
   }
} (110)
```

There will be four separate outputs generated by the ANTENNA_FIX command shown in Example 5-17. Three are the TEMP layers listed in the DIODE_LAYERS option that define the diodes being placed on the net. The fourth is the error output going to layer 110. Any nets that the ANTENNA_FIX command is unable to repair will be written to this layer and will be identified in the *block*.LAYOUT_ERRORS file. The ANTENNA_FIX command is not able to repair all antenna violations, so it is important to check this output to identify any nets that need to be repaired by hand. (Another way of identifying nets that could not be repaired by the ANTENNA_FIX command is discussed in Example 5-20 on page 5-49.)

In Example 5-18, the NET_FILTER command identifies antenna violations based on the combined areas of both metal1 and metal2. As before, power and ground nets are excluded, as are nets not connected to a device gate. In addition, nets that have already been protected by the addition of a diode on metal1 are excluded. The assumption is that a diode at any level of metal will provide unlimited protection against antenna effects for all higher level metals.

Example 5-18

```
CONNECT { M1 BY m1_diode_m1 }
CONNECT { m1_diode_m1 m1_diode_diff BY m1_diode_cont }
CONNECT { M1 M2 BY V1 }

TEXT { M1 BY M1.TEXT }
TEXT { M2 BY M2.TEXT }

NET_FILTER {
   LAYER_SET[1] = { all_gate }
   LAYER_SET[2] = { M1 M2 }
   ratio = EV_AREA[2] / EV_AREA[1]
   FILTER_RULE = { ratio >= 60 }
   TEXT_LIST = { !VDD !VSS }
   CONNECTED = { *all_gate, !m1_diode_m1 }
   ERROR_REPORT = { ratio }
   OUTPUT_FILE = M2_needs_fix
} (100)
```

Example 5-19 is similar to the previous ANTENNA_FIX command shown on Example 5-18 on page 5-48, except that it has to consider the possibility of being able to place a diode on both metal1 and metal2. If it can do so, Hercules will try to place the diode on metal1 first. If it cannot, it will then try to insert the diode on metal2. To force the diode to be inserted on metal2, you will need to remove the ROUTING designation from metal1.

Note:

In the nomenclature used for the diode layers, the first part of the layer name indicates the level at which the antenna was identified, while the last part indicates the layer being formed in the diode. Thus, m2_diode_m1 is a metal-1 polygon being formed as part of the creation of a diode on a metal-2 antenna.

Example 5-19

```
ANTENNA_FIX {
   NETS_FILE = M2_needs_fix
   LOW_LEVEL_LAYERS = { NDIFF, PDIFF, CP, CONT }
   DIODE_LAYERS = {
      NDIFF    0.4              TEMP = m2_diode_diff
      CONT     0.32             TEMP = m2_diode_cont
      M1       0.32    ROUTING TEMP = m2_diode_m1
      V1       0.32             TEMP = m2_diode_via1
      M2       0.4     ROUTING TEMP = m2_diode_m2
   }
   DRC_TABLE = {
      LOW_LEVEL_LAYERS LOW_LEVEL_LAYERS 0.4
      NDIFF    NWELL    1.2
      CONT     V1       0.5
      CONT     V2       0.5
   }
}
```

Example 5-20 illustrates an alternative way of identifying antennas that have not been repaired by the ANTENNA_FIX command. The CONNECT sequence preceding the command attaches the diodes created by the preceding ANTENNA_FIX command to the nets they are protecting. The NET_FILTER command then identifies those nets identified by the previous NET_FILTER that have not been connected to a diode. The connection criterion is based on the diode diffusion layer for the sake of convenience, because if the selection were based on the diode metal layers, one more layer would have to be listed in the CONNECT option.

Example 5-20

```
CONNECT { M1 BY m2_diode_m1 }
CONNECT { M2 BY m2_diode_m2 }
CONNECT { m2_diode_m1 m2_diode_diff BY m2_diode_cont }
CONNECT { m2_diode_m1 m2_diode_m2 BY m2_diode_via1 }

TEXT {  M1 BY M1.TEXT }
TEXT {  M2 BY M2.TEXT }
```

```
NET_FILTER {
        LAYER_SET[1] = { all_gate }
        LAYER_SET[2] = { M1 M2 }
        ratio = EV_AREA[2] / EV_AREA[1]
        FILTER_RULE = { ratio >= 60 }
        TEXT_LIST = { !VDD !VSS }
        CONNECTED = { *all_gate !m1_diode_diff !m2_diode_diff }
        ERROR_REPORT = { ratio }
        OUTPUT_FILE = M2_after_fix
} (100)
```

Example 5-21 shows how the different diode layers produced from different ANTENNA_FIX commands can be combined to simplify writing the data to the output database. More information on generating output from an ANTENNA_FIX runset is presented in Suggested Hercules-Based Antenna Flows.

Example 5-21

```
BOOLEAN m1_diode_m1 OR m2_diode_m1 { } TEMP = diode_m1
COPY m2_diode_m2 { } TEMP = diode_m2

BOOLEAN m1_diode_diff OR m2_diode_diff { } TEMP = diode_diff
BOOLEAN m1_diode_cont OR m2_diode_cont { } TEMP = diode_cont

COPY m2_diode_via1 { } TEMP = diode_via1
```

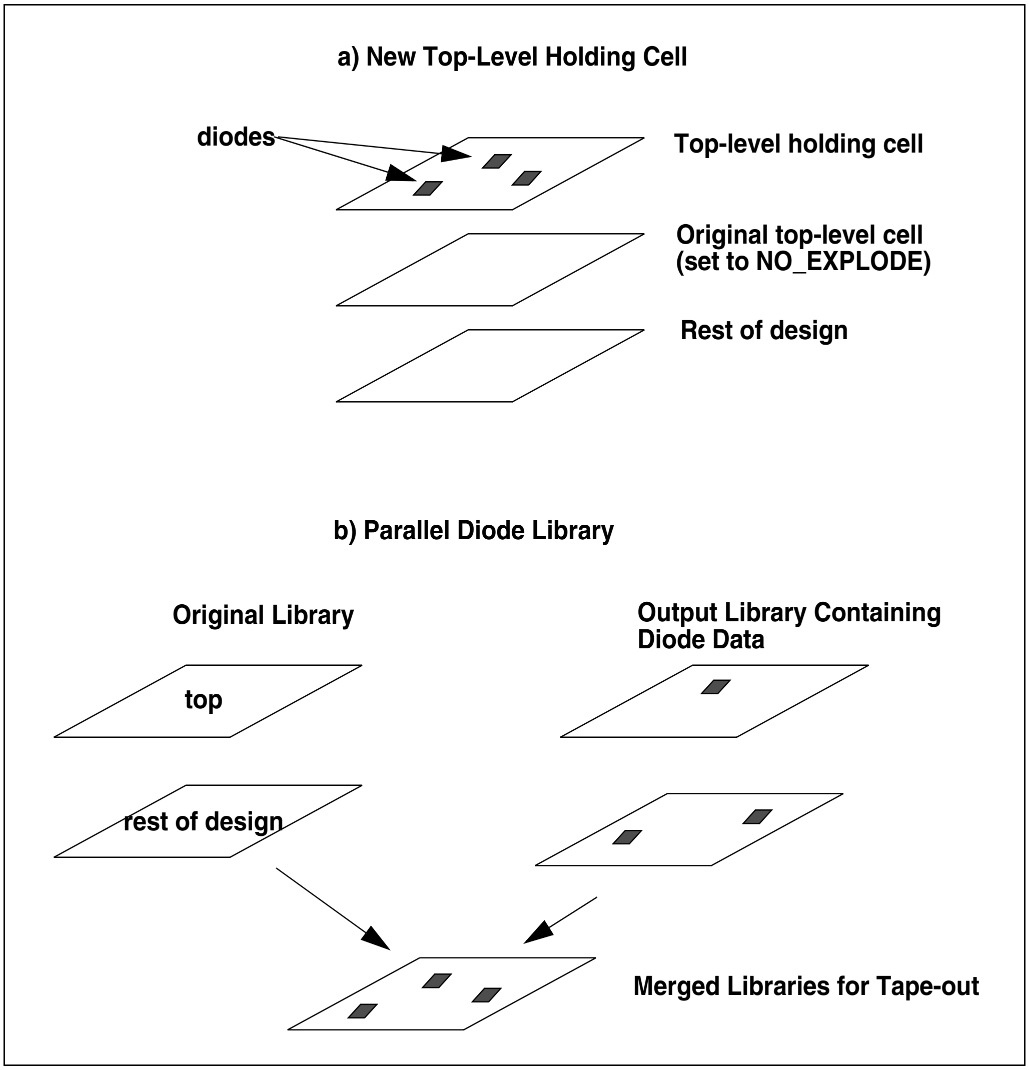
Suggested Hercules-Based Antenna Flows

As much as possible, antenna-related issues should be resolved during Place & Route. (See “Diode Insertion Using the Astro/Hercules Interface” on page 5-52 for how this can be done in Astro.) The reason for this lies in the difference between the way the two tools insert antenna-protection diodes into a design. Hercules does so by placing polygons that define the diode’s structure into the design at a specific location. If an Engineering Change Order (ECO) causes the net that is being protected to move, the diode will be left behind. This means that you need to remove all previously-created protection diodes from the design before you implement the ECO, then reinsert them when the ECO is complete. When Astro inserts a diode into the design, it does so by inserting a diode cell. In the event of an ECO, Astro understands the relationship between that cell and the net it is protecting, and is able to move the diode as necessary to maintain that relationship.

Another reason for using Astro to place these diodes is that Astro can make adjustments to how cells are placed in the design to “make room” for the diode. Hercules is limited to placing the diodes only where room already exists in the design for them. This slows down the diode placement process in Hercules significantly because the command has to follow the net, checking repeatedly to see if it has found a DRC-clean location. In the worst case, Hercules has to check the entire net just to see that there is not enough room anywhere to place a diode.

When Hercules is used for diode insertion, there are two suggested flows that can be used. These are illustrated in Figure 5-26 and described below. In both flows, the objective is to keep the diode data inserted by the ANTENNA_FIX command isolated from the rest of the design so that it can be easily removed as needed and replaced with new diode data.

Figure 5-26 Hercules-Based Diode Insertion Flows



In Figure 5-26(a), a new top-level holding cell is created in the library, and the original top cell is placed as an instance within this cell. To prevent non-diode data from floating up into the new holding cell, the original top cell is set to NO_EXPLODE. The antenna repair runset is run on the holding cell, and the diodes are written to this cell. In the event of an ECO, the holding cell is deleted and the process is repeated. The advantage of this flow is that the diode information is always part of the original library but in a separate location easily removed from the library. On the other hand, you have to repeat the entire cell creation, instantiation, and diode insertion process every time the design changes.

In Figure 5-26(b), the original library is left untouched, and the diode data is written to a parallel output library. Once the design has been finalized and approved, the two libraries are merged for tapeout. When the design changes, the new diode data is simply written to a new output library. The runset can even automatically overwrite the original output. This flow requires less manual effort than the previous flow, but now you have to maintain two separate libraries for the same design.

Diode Insertion Using the Astro/Hercules Interface

As noted earlier, antenna violations should be repaired as much as possible during Place & Route, and not by using Hercules. The following discussion assumes you are using Astro.

Astro is limited in its ability to detect antenna violations because it does not see some of the layers (for example, the polysilicon gates or the diffusions) involved in the antenna. Because Hercules is able to recognize these layers, it is the preferred tool for identifying antenna violations.

Runset Requirements

In order to use the Astro/Hercules diode insertion interface, you must provide an antenna-check Hercules runset. The preparation of such a runset is discussed earlier in this chapter. In order to make the runset compatible with the diode insertion interface, however, the MW_ANTENNA_REPORT, RATIO_MODE, and MW_ANTENNA_LAYER options must be used with the NET_FILTER command.

A description of each option is provided in the *Hercules Reference Manual*, Detailed Options chapter. In addition, the ERROR_REPORT option must list the ratio value corresponding the RATIO_MODE selected and the total area of all the gates on the given net. You will also need to modify the WRITE_MILKYWAY command so that the information Astro requires to perform the diode insertion is correctly added to the database. The results must be appended to the CEL view of the input library, and you must add the option ANTENNA_REPORT to the command. Example 5-22 presents a short illustration of how the runset needs to be set up for this interface.

The first name listed in the MW_ANTENNA_LAYER option gives the Hercules name of the top-level metal checked by this command; that is, the name listed in the ASSIGN section of the runset. The second name gives the Astro name of this layer; that is, the name listed in the technology file. If both names are the same, only one entry needs to be listed in this layer list.

Example 5-22 Example Runset for Diode Insertion Interface

```
NET_FILTER {
        LAYER_SET[1] = { all_gate }
        LAYER_SET[2] = { M1 }
        ratio = EV_AREA[2] / EV_AREA[1]
        FILTER_RULE = { ratio >= 30 }
        TEXT_LIST = { !VDD !VSS }
        CONNECTED = { *all_gate }
        ERROR_REPORT = { ratio, gate_area=EV_AREA[1] }
        MW_ANTENNA_REPORT = TRUE
        RATIO_MODE = 1
        MW_ANTENNA_LAYER = { MT1, met1 }
} (100)
```

No layers need to be listed in Example 5-23 because the ANTENNA_REPORT option identifies the information that the command is supposed to write to the library.

Example 5-23

```
WRITE_MILKYWAY {
        LIBRARY_NAME = INPUT_LIB
        LIBRARY_PATH = .
        VIEW_NAME = CEL
        OUTPUT_MODE = APPEND
        ANTENNA_REPORT = TRUE
}
```

When the diode insertion interface was developed, it was assumed that all antenna checks would be performed in accordance with different RATIO_MODE settings. In practice, however, this assumption is regularly violated. For example, there has been a case where antennas were defined by the number of vias on the net. In situations like this, you can use one criteria as the basis for selecting the net as an antenna, yet report different information to Astro. You could, for instance, select nets that contain an excessive number of metal2 to metal3 vias, but report the ratio of metal2 area to gate area using RATIO_MODE=1.

Sample Diode Insertion Flow Using Hercules and Astro

Figure 5-27 presents a suggested flow for identifying and repairing antenna violations using the combination of Astro and Hercules. It is written as an Astro script, and the options in each command will need to be replaced with an appropriate value in order to use the script. Users not familiar with Astro will need to refer to the Astro documentation for details about the commands; however, the overall purpose of the flow should be evident from the figure.

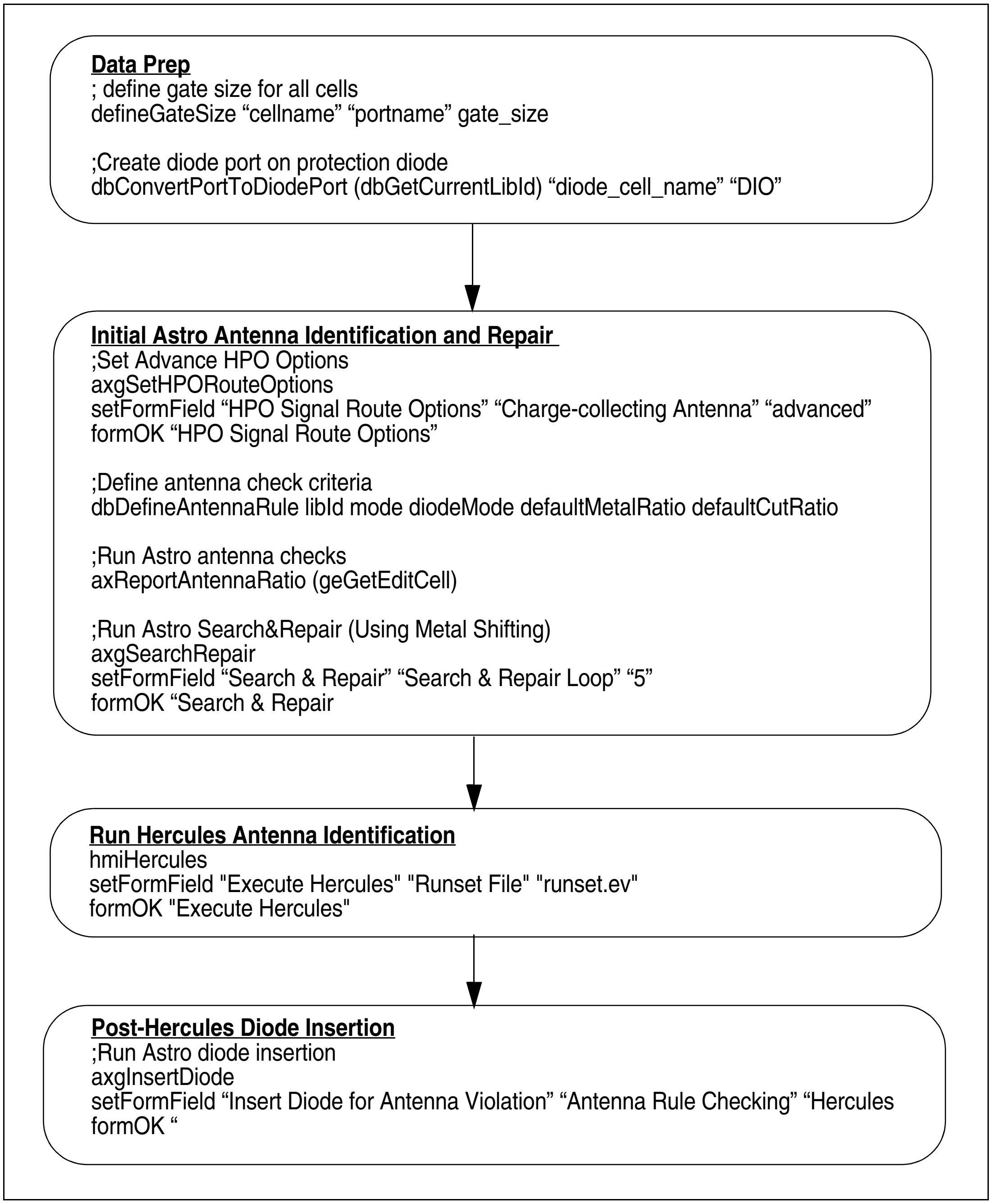
The script assumes that you are in Astro, and that you have already opened the library and cell of interest. The initial portion of the script prepares the library for the rest of the flow. You need to define the gate size for all the cells you are interested in. You can do this either as shown or by loading CLF information into the library. You also need to set up a diode cell so that it can be used as a protection diode. (Protection diodes have pins that are not netlisted to avoid interfering with LVS comparisons.)

The first pass at antenna identification and repair is performed inside Astro. The objective is to reduce the number of antenna violations from a few thousand (typically) to a few hundred through metal stitching. This process can usually be completed in a reasonable amount of time (usually three or four hours, depending on the design). The library is then passed to Hercules for further antenna identification. Hercules results will likely include not only antennas that have already been identified by Astro but have not been repaired, but also antennas that Astro was not able to recognize. Once Hercules has completed its job, the Astro diode insertion routine is invoked to repair as many of the remaining antennas as it can.

Do not expect to eliminate all antenna violations in your design. In fact, some violations will need to be repaired manually. The flow shown in Figure 5-27 should be followed by a second Hercules run used only to identify any unrepaired antennas left in the design. By this time, the number of antennas that will need manual repair should be reasonable.

When writing this follow-up runset, consider setting the option NODE_ERROR=TRUE in the NET_FILTER commands. This bins all errors that belong to the same net and simplifies the manual antenna repair process. The GUI debugging tool will display all of the violations at one time, and a single diode may be able to repair them all.

Figure 5-27 Suggested Astro/Hercules Diode Insertion Flow



Data Prep

```
; define gate size for all cells
defineGateSize “cellname” “portname” gate_size

;Create diode port on protection diode
dbConvertPortToDiodePort (dbGetCurrentLibId) “diode_cell_name” “DIO”
```

Initial Astro Antenna Identification and Repair

```
;Set Advance HPO Options
axgSetHPORouteOptions
setFormField “HPO Signal Route Options” “Charge-collecting Antenna” “advanced”
formOK “HPO Signal Route Options”

;Define antenna check criteria
dbDefineAntennaRule libId mode diodeMode defaultMetalRatio defaultCutRatio

;Run Astro antenna checks
axReportAntennaRatio (geGetEditCell)

;Run Astro Search&Repair (Using Metal Shifting)
axgSearchRepair
setFormField “Search & Repair” “Search & Repair Loop” “5”
formOK “Search & Repair
```

Run Hercules Antenna Identification

```
hmiHercules
setFormField "Execute Hercules" "Runset File" "runset.ev"
formOK "Execute Hercules"
```

Post-Hercules Diode Insertion

```
;Run Astro diode insertion
axgInsertDiode
setFormField “Insert Diode for Antenna Violation” “Antenna Rule Checking” “Hercules
formOK “
```

Index