Calibre Verification User's Manual

Software Version v9.1_5

Calibre 2002.5

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Table of Contents

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

Table of Contents (cont.)

List of Figures

List of Figures (cont.)

List of Figures (cont.)

List of Figures (cont.)

List of Tables

List of Tables (cont.)

List of Tables (cont.)

About This Manual

This manual contains full functionality updates through version 9.1_4 of the Calibre Verification applications. For later functionality, see the *Calibre Verification Release Notes*.

This document is the *Calibre Verification User's Manual*, which explains the concepts and use of the Calibre Verification toolset. You use this toolset to verify the physical and electrical integrity of IC designs. The Calibre Verification toolset consists of:

- Calibre DRC
- Calibre DRC-H*
- Calibre MGC
- Calibre LVS
- Calibre LVS-H*
- Calibre RVE/QDB-H and Query Server

*Includes multi-threaded processing capability.

This application uses Adobe Acrobat Reader as its online help and documentation viewer. Online help requires installing the Acrobat Reader. For more information, refer to the section, "Setting Up Online Manuals and Help" in *Using Mentor Graphics Documentation with Acrobat Reader*.

In This Manual

This manual contains the following chapters:

- Chapter 1, Overview describes Calibre Verification tools and utilities.
- Chapter 2, Invocation describes file requirements and invocation syntax for Calibre Verification tools.
- Chapter 3, Calibre Interactive describes the graphical user interfaces (GUIs) that assist in the invocation of Calibre Verification applications.
- Chapter 4, DRC Concepts describes concepts of Calibre DRC tools: layer specification, rule checks, dimensional check operations, and database manipulation.
- Chapter 5, DRC Execution describes processes that Calibre DRC tools perform.
- Chapter 6, Hierarchical DRC describes concepts specific to hierarchical Calibre DRC tools.
- Chapter 7, Connectivity Extraction describes the connectivity extraction module used by the Calibre Verification toolset.
- Chapter 8, Electrical Rule Checks describes concepts specific to electrical rule check applications.
- Chapter 9, Device Recognition describes the concepts and terminology of device recognition and the built-in language used to define devices.
- Chapter 10, LVS Circuit Comparison describes concepts of Calibre LVS tools including; connectivity comparison, components and names, and device reduction.
- Chapter 11, Spice Format describes the Spice netlist syntax and how Calibre LVS interprets the data.

- Chapter 12, Utilities describes input requirements, invocation, and procedures for EDIF-to-LVS, Verilog-to-LVS, Dracula to SVRF file converter, GDSII database comparison, and rules syntax checker.
- Chapter 13, Hierarchical LVS describes concepts specific to hierarchical Calibre LVS tools.
- Chapter 14, Results describes the Calibre Verification transcripts, reports, and results output.
- Chapter 15, RVE/QDB-H and Query Server describes the input requirements, usage, procedures, and commands for Calibre RVE/QDB-H.
- Appendix A, Application Notes lists the available Application Notes and how to access them.
- Appendix B, Calibre Interactive Files shows examples of files used with Calibre Interactive.
- Appendix C, V2LVS BNF shows the V2LVS binary notation format.

Command Line Syntax Conventions

The notational elements for command line syntax are as follows:

Standard	Standard font indicates literal text. This text should be entered exactly as shown.
Bold	A bold font indicates a required argument.
[ ]	Square brackets enclose optional arguments (in command line syntax only). Do not enter the square brackets.
Italic	An italic font indicates a user-supplied argument.
{ }	Braces enclose arguments to show grouping. Do not enter the braces.

	A vertical bar indicates an either/or choice between items. Do not enter the vertical bar.
…	An ellipsis follows an argument that may appear more than once. Do not enter the ellipsis.

Audience

This manual addresses two basic audiences, as determined by use of the main component of the Calibre Verification toolset—the *rule file*:

- **Users:** Typically, IC Layout Engineers/Specialists who use existing rule files with the verification tools to check the design of an IC layout. Sometimes users also write the rule file.
- **Programmers:** Persons who *write* rule files. Often programmers are members of a support group whose task is to write and maintain rule files. Sometimes programmers are also users of the verification tools.

The primary audience for this manual is *users* of the Calibre Verification toolset. However, programmers will find the information in this manual helpful. Rule file programming is outside the scope of this manual. For information on rule file creation and rule file operation, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Users of this manual should have knowledge of IC layout techniques and procedures, and the data formats that are being used. *Knowledge of verification techniques is not required.*

Related Publications

Calibre Verification Release Notes — contains information on new or changed functionality specific to the Calibre DRC/DRC-H, Calibre LVS/LVS-H, Calibre RVE and Calibre Interactive toolsets.

Configuring and Licensing Calibre/xCalibre Tools Guide—contains information on system configuration and licensing information for the Calibre Verification, Calibre RET, and xCalibre toolsets.

Calibre LITHO Commands Release Notes — contains information on new or changed functionality specific to the Calibre RET toolset and includes a summary of the defect reports that have been addressed by the current release. As appropriate, significant changes to the documentation since the last release are also described.

Calibre RET User's Manual — describes the prerequisites, key concepts, and procedures for Calibre OPCpro, ORC, PRINTimage, PSMgate.

Calibre RET Reference Manual — a dictionary style reference. It contains syntax and descriptions of commands, technology setup file keywords, and edge tagging keywords that are shared by these products: Calibre WORKbench, Calibre OPCpro, Calibre ORC, Calibre PRINTimage, Calibre PSMgate.

Calibre WORKbench Release Notes — contains information on new or changed functionality specific to Calibre WORKbench and includes a summary of the defect reports that have been addressed by the current release. As appropriate, significant changes to the documentation since the last release are also described.

Calibre WORKbench User's Manual — describes the optical and process modeling tool, Calibre WORKbench. It describes the Calibre WORKbench key concepts and procedures to develop accurate optical and process models for your specific manufacturing process.

Using the xCalibre-H Tool — contains key concepts, input requirements, and invocation syntax related to the xCalibre-H hierarchical parasitic extraction tool.

Using the xCalibre PX-C/PX-RC Tool — contains key concepts, input requirements, and invocation syntax related to the xCalibre PX-C/PX-RC flat parasitic extraction tool.

xCalibre Release Notes — contains information on new or changed functionality specific to the xCalibre PX-C/PX-RC and xCalibre-H toolsets.

Using RC-Delay and RC-Reduction — describes the input requirements, invocation usage, and concepts for the RC Delay and RC Reduction toolsets.

Using the xCalibrate Rule File Generator — contains key concepts, process and modeling descriptions, procedures, and reference information that xCalibre toolset users use for calibration, validation, and generation of SVRF rule file capacitance specification statements.

Standard Verification Rule Format (SVRF) Manual — contains key concepts and reference information about rule file statements and operations that are used by Calibre, xCalibre, and ICverify applications.

Using Mentor Graphics Documentation with Acrobat Reader — describes setting up and using the Mentor Graphics Corporation-supplied Acrobat Reader for online viewing of Mentor Graphics PDF-based documentation and help. The manual contains procedures for using Mentor Graphics documentation, including set up for online manuals and help, opening documents, and implementing full-text searches. Also included are tips on using Reader.

Chapter 1
Overview

This chapter introduces the Calibre Verification toolset. It describes each of the Calibre verification tools, the utilities available, and the licensing environment. Refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide* for details on installing, configuring, and licensing Calibre products.

Product Description

The Calibre Verification toolset assists you in verifying the physical and electrical integrity of IC designs.

The Calibre Verification tools operate on rule files written in Standard Verification Rule Format (SVRF). Rule files can consist of design rule (DRC) and electrical rule (ERC) checks, layout versus schematic (LVS) device and connectivity checks, and lithography statements for the Calibre Resolution Enhancement Technologies (RET) applications. For information about SVRF, refer to the *Standard Verification Rule Format (SVRF) Manual*. For information about the Calibre RET applications, refer to the *Calibre RET User's Manual*.

The Calibre Verification tool set consists of Calibre DRC/DRC-H, Calibre LVS/LVS-H, Calibre MGC, Calibre RVE/QDB-H, Calibre Interactive, Calibre Connectivity Interface, Calibre CB and utilities which are described below.

Calibre DRC / DRC-H / MT DRC-H

The Calibre DRC/DRC-H tools perform physical verification of integrated circuit designs in flat, hierarchical, and multi-threaded configurations:

- Flat—Calibre DRC performs design rule checking by reading the input layout database flat and operating on the resulting geometries.

- Hierarchical—Calibre DRC-H performs design rule checking hierarchically, which minimizes redundant processing. It stores, analyzes, and processes data once per cell instead of once for every flat placement of the cell.

- Multi-threaded—This configuration of Calibre DRC-H allows you to take advantage of multiple CPUs.

Calibre LVS / LVS-H / MT LVS-H

The Calibre LVS/LVS-H tools compare layout versus schematic in flat and hierarchical configurations:

- Flat—Calibre LVS performs flat layout versus schematic netlist checking.

- Hierarchical—Calibre LVS-H performs hierarchical layout versus schematic netlist checking, optimizing the process due to the hierarchy. Like Calibre DRC-H, it also stores, analyzes, and processes data once per cell instead of once for every flat placement of the cell.

- Multi-threaded—This configuration of Calibre LVS-H allows you to take advantage of multiple CPUs.

Calibre MGC

The Calibre MGC tool compares layout versus schematic in the flat configuration only. Calibre MGC allows you to use the Mentor Graphics EDDM Design Viewpoint.

Calibre RVE/QDB-H

Calibre RVE/QDB-H is a licensed query database and graphical user interface that allows you to investigate, debug, and highlight Calibre LVS/LVS-H discrepancies and Calibre DRC/DRC-H errors. You can use either the Mentor Graphics ICgraph, Cadence Virtuoso, or Seiko System SX9000 layout editor environments to view results.

The results viewing environment (RVE) is the graphical user interface. QDB-H is the hierarchical query database accessed by query server commands, which returns requested data about a design. RVE uses the query server to return connectivity information.

Calibre Interactive

Calibre Interactive is a licensed user interface environment for the Calibre tool set. It can be invoked from standalone Calibre, through ICgraph, or through third-party layout editors such as Cadence Virtuoso.

Calibre Connectivity Interface

The Calibre Connectivity Interface (CCI) is a set of licensed functionality associated with the query server. CCI is designed to enable conversion of the Calibre LVS SVDB results database into standards-based file formats (GDSII and Spice) that can be tailored for and used by downstream tools that need to access LVS extraction and comparison results (for example, backannotated netlisting).

Calibre CB

Calibre Cell/Block is a license package consisting of flat Calibre DRC and LVS verification tools, Calibre Interactive, Calibre RVE, and Query Server. It is intended for interactive block verification using a variety of layout editors.

Calibre Verification Utilities

The Calibre Verification toolset comes with the following utility programs that increase the capabilities of the Calibre toolset:

- EDIF-to-LVS. EDIF-to-LVS (E2LVS) is a converter that translates an EDIF structural netlist into a Spice-like netlist for use as input to Calibre LVS/LVS-H.

- Verilog-to-LVS. Verilog-to-LVS (V2LVS) is a converter that translates a Verilog structural netlist into a Spice-like netlist for use as input to Calibre LVS/LVS-H.

- Dracula: File Conversion and User Notes. The Dracula converter allows you to convert a Dracula rule file into a Standard Verification Rule Format rule file.

- Compare Two GDSII Databases. The Compare GDS utility allows you to compare two GDSII databases. This utility produces a ASCII DRC results database based on a layer-by-layer analysis.

Chapter 2
Invocation

This chapter describes the file requirements and invocation procedures for the Calibre Verification toolset. For similar information on the utility programs, refer to chapter 12, "Utilities".

Before Invocation

Ensure your Calibre installation and configuration are correct, including installation of all applicable licenses. See the *Configuring and Licensing Calibre/xCalibre Tools Guide* for details.

Before you invoke a Calibre Verification tool, the following data must exist:

- Rule file
- Layout database
- Source database, as applicable

Rule File

Except for invocation arguments on the command line, rule file *Specification Statements* control Calibre verification operations. These specification statements describe the overall "environment" for Calibre tools, such as describing the layout and source databases, and specifying where to store the results and reports. Specification statements also guide internal heuristics.

All Calibre rule files are written in the Standard Verification Rule Format (SVRF) language and are compatible with all Calibre, xCalibre and ICverify tools. SVRF is case-insensitive by default.

Required statements

The following tables show the specification statements required for Calibre DRC/DRC-H, Calibre LVS/LVS-H, and Calibre MGC. Each table shows the names of the required statements, and a description of the statement. For detailed descriptions of these statements, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Table 2-1 shows the required rule file statements for Calibre DRC applications.

Table 2-1. DRC/DRC-H — Required SVRF Rule File Statements

Statement	Purpose
Layout System	Specifies the format of the layout data.
Layout Path	Specifies the location of the layout data.
Layout Primary	Specifies the top-level cell within the layout data.
DRC Results Database	Specifies where to save the results.

Table 2-2 shows the required rule file statements for Calibre LVS and Calibre MGC applications.

Table 2-2. LVS/LVS-H/MGC — Required SVRF Rule File Statements

Statement	Purpose
Layout System	Specifies the format of the layout data.
Layout Path	Specifies the location of the layout data.
Layout Primary[1]	Specifies the top-level cell within the layout data.
Source System	Specifies the format of the source data.
Source Path	Specifies the location of the source data.
Source Primary[1]	Specifies the top-level cell within the source data.
LVS Report	Specifies where to save the report

[1]The Layout Primary and Source Primary statements are not required if your Layout System and Source System statements are set to Spice.

Layout Database

A layout database contains the geometric description of a circuit. Table 2-3 shows the allowed database formats. For a given toolset, the layout database must be one of the following system formats:

Table 2-3. Layout Database Formats

System Format	DRC	DRC-H	LVS	LVS-H	MGC
CIF	X	X	X	X	X
GDSII Stream	X	X	X	X	X
Spice			X	X	X
ASCII	X		X		X
Binary	X		X		X
Cnet database			X		X
V8.x Eddm					X

CIF Database Format

When the layout database format is CIF (Caltech Intermediate Form), you must specify the rule file Layout Path and Layout Primary specification statements. Layout Path specifies the pathname to the CIF symbol file, and Layout Primary specifies the top-level layout cell name within the database—only the top-level layout cell and the cells below it in the layout hierarchy are processed.

The Layout Path statement may be specified with multiple file names and any number of times. This facilitates reading in multiple databases. Multiple input databases are treated as if all symbol definitions are embedded in the first file specified. Each input file is expected to be syntactically complete.

The original Mead/Conway BNF is followed except for the following extensions and limitations:

- The user extension command “9” immediately following a “DS” command will define the cell name associated with the symbol number.

- Implicit commands “P”, “B”, “R”, “W”, “C”, and any implicit user extension commands are not processed. An implicit command is defined as one outside of “DS” … “DF”. A warning or error (depending on the setting of The Layout Error on Input specification statement) is issued for implicit non-user-extension commands.

- Commands “R” (round flash) and “DD” (definition delete) are not processed.

- User extension commands “4N”, “94”, “4M”, and “4X” are interpreted as text objects with the following syntax:

```
4N/94 string sinteger sinteger
4M string integer point point string
4X string integer point integer string string
```

- CIF layer names must be resolvable (that is, defined) in the rule file. Objects will not be added to unresolvable CIF layers. As an example rule file definition using an alias:

```
LAYER METAL1 M1// I really want to use the name METAL1.
LAYER M1 12// The way it’s defined in the CIF file.
```

GDSII Layout Database Format

When the layout database format is GDSII (Layout System GDSII), you must specify the pathname to the database in a Layout Path specification statement, and you must identify the top-level cell in a Layout Primary specification statement—only the top-level layout cell and the cells below it in the layout hierarchy are processed.

The Layout Path statement may be specified with multiple file names and any number of times. This facilitates reading in multiple databases. Multiple input databases are treated as if all structure records are embedded in the first file specified. Each input file is expected to be syntactically complete.

The following GDSII records are processed by the Calibre Verification toolset:

HEADER	BGNLIB	LIBNAME	UNITS

ENDLIB	BGNSTR	STRNAME	ENDSTR
BOUNDARY	PATH	PATHTYPE	WIDTH
BGNEXTN	ENDEXTN	XY	COLROW
LAYER	DATATYPE	SREF	AREF
SNAME	TEXT	TEXTTYPE	STRING
STRANS	MAG	ANGLE	ENDEL

The Calibre verification tools can process and treat BOX and BOXTYPE records as BOUNDARY and DATATYPE records, respectively, by placing the following statement in the rule file:

```
LAYOUT PROCESS BOX RECORDS YES
```

The default is to not process BOX and BOXTYPE records.

You can also specify the layout depth for geometries via the optional rule file Layout Depth specification statement. ALL (the default) specifies that geometries are read from the top-level cell to the bottom of the hierarchy. PRIMARY specifies that geometries are read from the top-level cell only.

Note

GDSII boundaries and paths with zero vertices will generate a fatal read error.

ASCII Layout Database Formats

When the layout database format is ASCII (Layout System ASCII), it appears as a set of polygon files in the form icv_data_*n*, where *n* represents the corresponding drawn layer number. ASCII databases are user-created.

The ASCII format for a polygon file is simply a list of polygons where each polygon is a vertex count followed by the vertices. More precisely:

> <ascii polygon file> -> WS* [<polygon> WS+ [... <polygon>]] WS*
> <polygon> -> <vertex count> WS+ <vertex> WS+ <vertex> [... WS+ <vertex>]

<vertex count> -> positive integer
<vertex> -> <x> WS+ <y>
<x>, <y> -> positive or negative integer

WS* represents zero or more whitespace characters and WS+ represents one or more whitespace characters. The number of vertices for each polygon is given by the <vertex count> field. The polygon vertices are expressed in database units. A two-vertex polygon is understood to represent an orthogonal rectangle.

The ASCII database format does not support text. Text will only be read from Text specification statements in the rule file. Also, this format is used only by flat Calibre DRC applications; it is *not* used by Calibre DRC-H.

Binary Layout Database Formats

When the layout database format is binary (Layout System BINARY), it appears as a set of polygon files in the form icv_data_*n*, where *n* represents the corresponding drawn layer number.

The binary layout database format allows you to compare two databases to ensure database integrity. For instance, you can compare two GDSII databases, two MGC databases, or a GDSII database and MGC database.

The following BNF summarizes the binary polygon format:

```
<bpf file> -> <bpf record> [ ... <bpf record> ] EOF

<bpf_record> -> <node record> | <non-node record>

<node record> -> <node vertex count> <node number> <vertices>
<non-node record> -> <vertex count> <vertices>

<node vertex count> -> <short16 with MSB set>
<vertex count> -> <short16 with MSB unset>
<vertices> -> <vertex> [ ... <vertex> ]

<vertex> -> <x> <y>

<x> -> <int32>
<y> -> <int32>
```

```
<node_number> -> <int32>
```

Source Database

A source database contains the reference information of a circuit. This is also called a source schematic or source netlist. Table 2-4 shows the allowed database formats.

You must use a source database (schematic or netlist) when doing layout versus schematic checks. The Source System statement identifies the reference to be compared. The source database can be in one of the following system formats:

Table 2-4. Source Database Formats

System Format	LVS	LVS-H	MGC
Spice	X	X	X
Cnet database	X		X
V8.x Eddm			X

Calibre supplies two utilities that convert Verilog and EDIF structural netlists into a Spice-like netlist format for use with Calibre LVS applications.

- EDIF-to-LVS (E2LVS)—translates an EDIF structural netlist into a Spice-like netlist.
- Verilog-to-LVS (V2LVS)—translates a Verilog structural netlist into a Spice-like netlist.

Invocation Procedures

Invoking Calibre

Before you invoke a Calibre Verification tool, you must first set the MGC_HOME environment variable to the location of your Mentor Graphics tree. To set MGC_HOME and invoke a Calibre Verification tool, follow the procedures below.

Setting the Environment Variable

1. In a C shell window, enter:

   ```
   setenv MGC_HOME path_to_mgc_tree
   ```

   In a Bourne or Korn shell window, enter:

   ```
   MGC_HOME=path_to_mgc_tree
   export MGC_HOME
   ```

2. In either type of shell window, enter:

   ```
   echo $MGC_HOME
   ```

   This verifies the location of your Mentor Graphics tree that you just set.

Starting Calibre

In the shell window you are using, enter:

```
$MGC_HOME/bin/calibre options
```

For information about the options you can set, refer to the section "Calibre Command Line" below.

Calibre Command Line

This section describes the following Calibre verification tools command line options.

- Calibre DRC/DRC-H
- Calibre LVS/LVS-H/MGC
- Calibre RVE/QDB-H
- Calibre Interactive

Calibre DRC/DRC-H

Usage

```
calibre { -drc [ -writedatabase ][ -cb ]
        || -drc -hier [ -turbo [ number_of_processors ] ]
        [ -turbo_litho [ number_of_processors ] ]
        [-turbo_all]
        }
        [ -nowait ] [-wait n] [-64]
        rule_file_name
```

Description

Calibre DRC/DRC-H performs either flat (calibre -drc) or hierarchical (calibre -drc -hier) design rule checking.

Calibre DRC performs traditional design rule checking by reading the database flat and operating on the resultant flat geometries. You can also use Calibre DRC to perform binary layout translation by using -writedatabase. Binary layout translation allows the input to be used with some third-party tools.

Calibre DRC-H performs hierarchical design rule checking by maintaining the database hierarchy to reduce processing time, memory usage, and DRC result counts. Calibre DRC-H imposes no design restrictions on geometries overlapping cell placements, or overlaps of cell placements.

For accepted database formats refer to the section “Layout Database” above.

When you use Calibre DRC-H for mask preparation, you should output a GDSII DRC results database (see also DRC Check Map in the *SVRF Manual*). For general design rule checks, you should output an ASCII DRC results database. You should follow these guidelines since Calibre DRC-H requires a large amount of internal overhead to generate a GDSII DRC results database.

You can use RVE/QDB-H and Query Server to analyze the DRC results database. If you use Mentor Graphics IC Station, you can load a GDSII DRC results database into IC Station for interactive debugging.

Both Calibre DRC and Calibre DRC-H use the same rule file. This means you do not need to add, remove, or modify any statements, with the following exception: If you include a Layer Directory specification in the rule file for a Calibre DRC-H run, Calibre DRC-H issues a warning since it does not support disk-based layers.

When you perform DRC checking, Calibre selects and runs all DRC rule checks by default. You can override this by using a DRC Select Check (see also DRC Unselect Check) specification statement to run a subset of the rule checks.

Table 2-1 shows the required specification statements for Calibre DRC/DRC-H.

Arguments

To display help information, enter either of the following commands (without arguments):

```
calibre -drc
calibre -drc -hier
```

- [**-drc** | **-drc -hier**]

  This switch selects the type of DRC to run. Possible values are:

  -drc selects flat DRC checking.

  -drc -hier selects hierarchical checking.

- -writedatabase (Calibre DRC only)

  This switch translates a GDSII layout database into binary polygon format. Calibre performs no rule checking in this mode.

  Calibre writes all processed geometric data (driven by the rule file) to the current directory as binary polygon files. These files are equivalent to the binary layout database input format. The name of each file is in the form of icv_data_*n*, where *n* is the layer number.

  The binary polygon files include the simple layers and their geometric subsets that would have been read from the database if the check set had been executed by DRC. Geometries specified in Polygon specification statements are not included.

  Calibre does not perform rule checks if the database is being written in this manner. Geometry flagging of acute angles, skew edges, and offgrid vertices is also disabled by this switch.

- -cb

  This option is discussed under “Calibre CB” on page 2-32.

- -turbo *number_of_processors* (Calibre DRC-H)

  This switch instructs Calibre DRC-H to use multi-threaded parallel processing for all stages except Litho operations. The *number_of_processors* argument is a positive integer that specifies the number of CPUs to use in the processing. If you do not specify a value, Calibre DRC-H runs on the maximum number of CPUs available.

  Calibre DRC-H runs on the maximum number of CPUs available, if you specify a number greater than the maximum available. For example:

  ```
  calibre -drc -hier … -turbo 3 …
  ```

  will not improve performance on a single-CPU machine.

  This switch is not for flat applications. For more information, refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide*.

  You can specify the `-turbo` and the `-turbo_litho` parameters concurrently in a single command line and the respective *number_of_processors* strings can vary between the two parameters.

- -turbo_litho *number_of_processors* (Calibre DRC-H)

  This switch instructs Calibre DRC-H to use multi-threaded parallel processing when performing Litho operations. The *number_of_processors* argument is a positive integer that specifies the number of CPUs to use in the processing. If you do not specify a value, Calibre DRC-H runs on the maximum number of CPUs available.

  Calibre DRC-H runs on the maximum number of CPUs available, if you specify a number greater than the maximum available.

  This switch is not for flat applications. For more information, refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide*.

  You can specify the `-turbo` and the `-turbo_litho` parameters concurrently in a single command line and the respective *number_of_processors* strings can vary between the two parameters.

- -turbo_all (Calibre DRC-H)

  The -turbo_all switch is an optional argument you use in conjunction with the -turbo and/or -turbo_litho switches. This switch halts Calibre tool invocation if

the tool cannot obtain the exact number of CPUs you specified using -turbo or -turbo_litho, or both.

Specifying the -turbo or -turbo_litho switches without specifying a specific number of CPUs is effectively the same as specifying the maximum number of CPUs on the machine. For example, specifying:

```
% calibre -drc -hier -turbo -turbo_all rule_file
```

on an 8-CPU machine for a hierarchical DRC run is the same as specifying:

```
% calibre -drc -hier -turbo 8 -turbo_all rule_file
```

Without -turbo_all, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable. See "-turbo_all Switch" in the *Configuring and Licensing Calibre/xCalibre Tools Guide* for licensing information.

- -nowait

  This switch instructs the MGLS license queueing features to be disabled. This results in Calibre exiting, instead of queueing for a license, if one is not available.

- -wait *n*

  This switch places a limit on the total time in minutes that Calibre will queue for a license. For example, the command:

  ```
  calibre -drc -wait 5 rules
  ```

  will queue on a calibredrc license for 5 minutes. If a license does not become available within 5 minutes, the application will exit with the following message:
  `// Queue time specified by -wait switch has elapsed.`

- -64

  This switch invokes the 64-bit version of Calibre. It is available on the HP and Solaris platforms, which require at least HP-UX 11.0 and Solaris 7, respectively. The default is 32-bit mode.

  The 64-bit executable on HP-UX provides a theoretical process size limit of roughly 1G * 1G / 4 bytes (or 2^{62} bytes) compared to only 4 Gbytes with the

32-bit executable. The 64-bit version of Calibre may, however, consume more memory than 32-bit Calibre executing on the same data.

- ***rule_file_name***

  Pathname of the rule file.

Examples

The following examples show how to run both the flat and hierarchical mode:

```
calibre -drc my_rules
calibre -drc -hier /user/project/bicmos.rules
```

The following example uses the -writedatabase switch for creating a binary database:

```
calibre -drc -writedatabase ./my_rules
```

Calibre LVS/LVS-H/MGC

Usage

Calibre LVS/LVS-H

```
calibre [ -lvs {[ { -tl || -ts } cnet_file_name ]
                [ -nonames ] [ -cell ]
                [-dblayers "name1,..."]
                [ -bpf ] [ -nl ] [ -cb ]
              || [ -hier [ -automatch ] ]
              }
              [ -ixf ] [ -nxf ]
        ]
        [ -spice spice_file_name
          [ -turbo [ number_of_processors ] ]
          [ -turbo_litho [ number_of_processors ] ]
          [ -turbo_all]]
         [ -hcell cell_correspondence_file_name ]
        [ -nowait ][-wait n][-64]
        rule_file_name

calibre -lvs [ -cs  ] [ -cl ] [-nowait] [-wait n] [-64]
        [ -cb ] rule_file_name
```

Calibre MGC

```
calibre_mgc [arguments as above unless otherwise indicated]
```

Description

Calibre LVS/LVS-H/MGC performs either flat or hierarchical layout versus schematic (LVS) checking.

Calibre LVS is a traditional LVS checking tool that flattens the input database and operates on the resulting flat geometries.

Calibre LVS-H is a hierarchical LVS checking tool that maintains and exploits the database hierarchy to reduce processing time, memory usage, and LVS discrepancy counts. Calibre LVS-H imposes no design restrictions on geometries that overlap cell placements or on overlaps of cell placements.

Calibre MGC operates in the same manner as Calibre LVS, in that it is a traditional LVS checking tool that flattens the input database and checks the resulting flat geometries. Calibre MGC primarily performs layout versus schematic (LVS) checking on a Mentor Graphics v8.x Eddm Design Viewpoint (EDDM) database.

Calibre LVS/LVS-H/MGC directly read GDSII and CIF databases and can compare various combinations of design types, such as GDSII to netlist, netlist to netlist, and Cnet to Cnet. The rule file Source System and Layout System specification statements determine which comparison is in effect.

When you specify the Mask SVDB Directory specification statement in the rule file, you can view Calibre LVS/LVS-H results graphically with Calibre RVE/QDB-H. Refer to chapter 15, “RVE/QDB-H and Query Server” for more information on the Query Server.

You can load the results from flat Calibre LVS into IC Station for interactive debugging in ICtrace Mask mode. You can also load and view results with the Mentor Graphics Verification DataPort tool.

When you perform hierarchical circuit extraction and circuit comparison with a single command line (`calibre -lvs -hier -spice …`), Calibre verifies that the source netlist and the LVS report file are specified and accessible before executing the circuit extraction step.

You can use Calibre LVS in one of two methods:

- LVS comparison
- Cnet database translation

Table 2-2 shows the required rule file specification statements for Calibre LVS/LVS-H/MGC.

Calibre LVS applications exit with a non-zero status if they cannot complete any form of requested processing due to fatal error conditions.

Arguments

To display help information, enter any of the following commands (without arguments):

```
calibre -lvs
calibre -lvs -hier
calibre_mgc -lvs
```

- -lvs

  This switch specifies to run Calibre LVS.

  When you use -lvs with -spice, Calibre extracts a Spice netlist (hierarchically) from the layout system. The extracted netlist then serves as layout input to the LVS comparison module, in place of the original layout system.

- [-tl || -ts] (select one)

  This switch determines whether you want to generate a Cnet database called *cnet_file_name* from the layout or from the source. Do not use this option with the -hier switch. Possible values are:

  -tl Selects layout translation. You specify the layout in the Layout Path specification statement.

  -ts Selects source translation. You specify the source in the Source Path specification statement.

- *cnet_file_name*

  The pathname of the file receiving the layout-data-to-cnet translation.

- -nonames (or -non)

  This switch prevents the Cnet writer from generating net and instance name files in the Cnet database. Use it only with -tl or -ts.

- -cell (or -c)

  This switch specifies that the Cnet writer scan only the top level cell (no hierarchical evaluation). Use it only with -tl or -ts and when the original design is an EDDM database.

- -dblayers "*name1*, …" (or -db "*name1*, …")

  This switch controls the layer geometries written to the mask results database. You specify an argument of comma-separated layer names, enclosed in quotation marks. Calibre writes only these layer names to the mask results database. Each name is a layer or a layer number that appears in the rule file.

  If you omit this switch, Calibre writes all relevant layers to the mask results database. The written layers include those that appear in Connect and Sconnect operations, all Device seed and pin layers from the rule file, and all Stamp target layers, with the possible exception of contact layers as specified with the Mask Results Database specification statement. Do not use this option if Mask Results Database NONE is specified in the rule file.

  This option can only select layers that appear in Connect and Sconnect operations, serve as Device seed or pin layers, or serve as Stamp target layers.

- -bpf

  This switch generates a binary polygon format (BPF) and trapezoid segmentation database and a layout cross-reference file. You cannot specify this switch in the same command line containing both the -nl and -spice switches; Calibre returns an error if this occurs.

  You use BPF databases to interface Calibre to third-party tools. The format is described in the "Binary Layout Database Formats" section. You can use this switch in both normal operation and in translate operation (-tl argument). The files have names of the form *lvs_report_name*.*layer_name*.bpf, where *layer_name* is the rule file layer name and *lvs_report_name* results from the LVS Report specification statement in the rule file. By default, Calibre provides all connect and device seed layers as output. You can use the -dblayers argument to explicitly select layers for generation. Do not use this option with -hier; it is not applicable with the -spice option.

The header file, DrcBPFReader.h, is the BPF data reader interface and is located in the $MGC_HOME/shared/include directory. This file provides a C interface to the BPF database, with which you can access and manipulate a BPF file. For additional information concerning this file, refer to the DrcBPF_example.c provided with DrcBPFReader.h.

The layout cross-reference file is named *lvs_report_name*.lxf and lists the internal net number and layout texted name.

This switch also creates a file containing top level port information. The name of this file is *report_name*.ports where *report_name* is the name specified in the LVS Report specification statement. If no LVS Report statement is included, the filename defaults to icv.ports. The file contains one line for each top level port (unattached ports are not output). Each line has the following fields:

```
port_name port_node_number port_node_name
                          port_location port_layer_attached
```

where:

- port_name—specifies the layout name of the port object, for example the GDSII text string when using the Port Layer Text specification statement, or “<UNNAMED>” if the port is not named.
- port_node_number—specifies the layout node number to which the port is connected.
- port_node_name—specifies the layout node name to which the port is connected; or layout node number if the node is unnamed.
- port_location—specifies the location of the database text object when using the Port Layer Text statement, or a vertex on the port polygon marker when using Port Layer Polygon. In the form: X Y; in database units.
- port_layer_attached—specifies the layer of the polygon to which the port got attached. Rule file layer name or rule file layer number if the layer is unnamed. This layer appears in a Connect or Sconnect operation.

- -nl (flat netlist)

  This switch produces a flat netlist from the layout. Nets are identified with numerical IDs only, no names, with the exception of nets that are connected to texted port objects. Such nets are represented in the netlist by the name of the respective port object. If a net is connected to more than one texted port object then one of the port names is arbitrarily chosen to represent the net. The netlist format is affected by the LVS NL Pin Locations specification statement. This parameter operates in flat Calibre LVS only. You use the netlist to interface Calibre to third-party tools. To extract a hierarchical Spice netlist from GDSII, use the -spice switch. You cannot specify this switch in the same command line containing both the -bpf and -spice switches; Calibre returns an error if this occurs.

- -cb

  This option is discussed under “Calibre CB” on page 2-32.

- -hier

  This switch runs the LVS comparison hierarchically. Both layout and source must be Spice, unless you also specified -spice, in which case layout must be GDSII or CIF.

- -automatch (-aut[o]; Calibre LVS-H)

  This switch specifies automatic correspondence *by name* for cells in hierarchical LVS comparison. Calibre compares cells with the *same name* in the layout and source (plus those specified by the -hcell option or in an Hcell rule file specification statement) as hierarchical entities. Calibre pushes all other cells down to the next level of hierarchy (correspondence level).

  Cell names that appear in a cell correspondence file specified by -hcell or in an Hcell specification statement are not automatically matched by name, even when you specify -automatch for the following reason: case-sensitivity in the for hcells is controlled by the LVS Compare Case specification statement. Hcell names are case-insensitive by default; but are case sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES. Top-level cells always correspond, regardless of their names. Note that -automatch has no effect on hierarchical circuit extraction.

- -ixf (for flat runs)

  
  Note

  The cross-reference files generated in flat Calibre LVS, with the use of the -ixf switch, are not equivalent to those generated for hierarchical Calibre LVS, with the use of the -ixf switch.

  This switch generates an instance cross-reference file. The filename is *lvs_report_name*.ixf, where *lvs_report_name* is specified by the LVS Report specification statement in the rule file. This option is not valid with the -tl, or -ts switches. For additional information about instance cross-reference files, refer to the Cross-Reference Files section in x.

  When you specify the -ixf switch and your rule file includes the Mask SVDB Directory specification statement with the QUERY, PHDB, or XDB keywords, Calibre LVS writes the instance cross-reference file to the SVDB directory. This file will not use the LVS Report name from above, but will be in the form *layout_primary*.ixf, where *layout_primary* is from the Layout Primary specification statement, if present in the rule file. If you do not specify the Layout Primary statement, ICV_UNNAMED_TOP is substituted for *layout_primary*.

  This option is not valid with the -tl and -ts options.

- -nxf (for flat runs)

  
  Note

  The cross-reference files generated in flat Calibre LVS, with the use of the -nxf switch, are not equivalent to those generated for hierarchical Calibre LVS, with the use of the -nxf switch.

  This switch generates a net cross-reference file. The filename is *lvs_report_name*.nxf, where *lvs_report_name* is specified by the LVS Report specification statement in the rule file. This option is not valid with the -tl or -ts switches. For additional information about net cross-reference files, refer to the Cross-Reference Files section in the “Results” chapter.

  When you specify the -nxf switch and your rule file includes the Mask SVDB Directory specification statement with the QUERY, PHDB, or XDB keywords, Calibre LVS writes the net cross-reference file to the SVDB directory. This file will not use the LVS Report name from above, but will be in the form *layout_primary*.nxf, where *layout_primary* is from the Layout Primary

specification statement, if present in the rule file. If you do not specify the Layout Primary statement, ICV_UNNAMED_TOP is substituted for *layout_primary*.

This option is not valid with the -tl and -ts options.

- -spice *spice_file_name* (or -spi *spice_file_name*; Calibre LVS-H)

  This switch extracts a hierarchical Spice netlist from the layout system, which must be GDSII or CIF and directs output to *spice_file_name*. When you specify this option with -lvs, Calibre extracts a Spice netlist from the layout system and uses it in place of the original layout system for comparison against the source. When you use the -hcell switch, Calibre preserves hcells as subcircuits throughout circuit extraction.

  
  Note

  When you use source names with xCalibre-H, *spice_file_name* must be an explicit pathname that places the file in the SVDB directory. That is: /*directory_path*/*layout_primary*.sp, where *directory_path* and *layout_primary* appear, respectively, in the Mask SVDB Directory the Layout Primary specification statements in the rule file.

  You can use the -spice switch when you run xCalibre-H (after running Calibre LVS-H) and specify the Mask SVDB Directory specification statement in the rule file using the keyword XCALIBRE. This writes the results of circuit extraction (and device recognition) to a hierarchical database (HDB) and places it in the SVDB.

- -turbo *number_of_processors* (Calibre LVS-H)

  This switch instructs Calibre LVS-H to use multi-threaded parallel processing. The *number_of_processors* argument is a positive integer that specifies the number of CPUs to use in the processing. If you do not specify a value, Calibre LVS-H runs on the maximum number of CPUs available.

  This switch applies only to hierarchical circuit extraction, not to the circuit comparison stage. Therefore, -turbo requires the -spice switch.

  Calibre LVS-H is limited to running on the maximum number of CPUs available. If you specify a number greater than the maximum available CPUs, Calibre LVS-H will run only on the maximum number. For example:

  ```
  calibre -spice -turbo 3 …
  ```

will not improve performance on a single-CPU machine.

This switch is not for flat applications. Refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide* for important considerations.

- -turbo_litho *number_of_processors* (Calibre LVS-H)

  Similar to the -turbo option; specifies multithreaded execution of OPC operations only.

- -turbo_all (Calibre LVS-H)

  The -turbo_all switch is an optional argument you use in conjunction with the -turbo and/or -turbo_litho switches. This switch halts Calibre tool invocation if the tool cannot obtain the exact number of CPUs you specified using -turbo or -turbo_litho, or both.

  Specifying the -turbo or -turbo_litho switches without specifying a specific number of CPUs is effectively the same as specifying the maximum number of CPUs on the machine. For example, specifying:

  ```
  % calibre -lvs -hier -auto -turbo -turbo_all rule_file
  ```

  on an 8-CPU machine for a hierarchical DRC run is the same as specifying:

  ```
  % calibre -lvs -hier -auto -turbo 8 -turbo_all rule_file
  ```

  Without -turbo_all, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable. See “-turbo_all Switch” in the *Configuring and Licensing Calibre/xCalibre Tools Guide* for licensing information.

- -hcell *cell_correspondence_file_name* (Calibre LVS-H)

  
  Note

  You must run Calibre LVS-H with the -hcell switch (or with Hcell specification statements in your rule file) before running xCalibre-H when source names are specified in the rule file. In addition, you must ensure that the Mask SVDB Directory specification statement appears in the rule file. Calibre LVS-H will generate source-to-layout cross-reference files (XREFs) suitable for hierarchical net extraction and place them in the SVDB directory.

  This switch specifies a cell correspondence file for hierarchical LVS comparison. Use of the -hcell switch always preserves hcells as subcircuits. Top-level cells do not need to appear in the cell correspondence file.

  Cell correspondence can also be controlled using the Hcell specification statement in your rule file. You may use -hcell with a correspondence file in addition to any Hcell rule file statements. The lists of hcells will be concatenated.

  The following is an example of a cell correspondence file:

  ```
  ABC DEF
  ABC GHI
  ABC JKL

  UVW XYZ
  RST XYZ
  OPQ XYZ

  UVW GHI
  OPQ DEF
  ```

  You can specify a 1-to-n relationship by placing a layout name in several lines with different source names. From the above example:

  ```
  ABC DEF
  ABC GHI
  ABC JKL
  ```

  Similarly, you can specify a m-to-1 relationship by placing a source name in several lines with different layout names. From the previous example:

UVW XYZ
RST XYZ
OPQ XYZ

However, m-to-n relationships are not allowed. From the above example:

UVW XYZ
RST XYZ
OPQ XYZ
OPQ GHI

OPQ DEF
ABC DEF
ABC GHI
ABC JKL

By default, primitive devices correspond by component type as in the flat mode. You can override this by including their names in the cell correspondence file. The cell correspondence file then exclusively determines the correspondence of the primitive devices.

Warnings are issued for cell names that do not exist in the input data.

See also "Hcells" on page 13-6.

- -nowait

  This switch instructs the MGLS license queueing features to be disabled. This results in Calibre exiting, instead of queueing for a license, if one is not available.

- -wait *n*

  This switch places a limit on the total time in minutes that Calibre will queue for a license. For example, the command:

  ```
  calibre -lvs -wait 5 rules
  ```

  will queue on a calibrelvs license for 5 minutes. If a license does not become available within 5 minutes, the application will exit with the following message:
  `// Queue time specified by -wait switch has elapsed.`

- -64

  This switch invokes the 64-bit version of Calibre. It is available on the HP and Solaris platforms, which require at least HP-UX 11.0 and Solaris 7, respectively. The default is 32-bit mode.

  The 64-bit executable on HP-UX provides a theoretical process size limit of roughly 1G * 1G / 4 bytes (or 2^{62} bytes) compared to only 4 Gbytes with the 32-bit executable. The 64-bit version of Calibre may, however, consume more memory than 32-bit Calibre executing on the same data.

- -cs

  This switch instructs Calibre LVS to read and verify (through a syntax checker) the Spice netlist specified in the Source Path specification statement. Calibre LVS issues any applicable warnings or errors, and also writes them to the LVS report. Calibre LVS reads the Spice netlist hierarchically (as is done with the -hier switch) but does not generate any LVS comparison structures.

  This switch cannot be used with input systems other than Spice netlists and cannot be used with other switches, besides -cl, -nowait, and -64.

  You can combine the usage of -cs and -cl switches. The primary status message in the LVS report is SYNTAX OK if the check succeeded or SYNTAX CHECK FAILED if the check failed.

  Calibre LVS consumes a flat LVS license when using this switch, or both this and the -cl switch at the same time.

- -cl

  This switch instructs Calibre LVS to read and verify (through a syntax checker) the Spice netlist specified in the Layout Path specification statement. Calibre LVS issues any applicable warnings or errors, and also writes them to the LVS report. Calibre LVS reads the Spice netlist hierarchically (as is done with the -hier switch) but does not generate any LVS comparison structures.

  This switch cannot be used with input systems other than Spice netlists and cannot be used with other switches, besides -cs, -nowait, and -64.

  You can combine the usage of -cs and -cl switches. The primary status message in the LVS report is SYNTAX OK if the check succeeded or SYNTAX CHECK FAILED if the check failed.

Calibre LVS consumes a flat LVS license when using this switch, or both this and the -cs switch at the same time.

- ***rule_file_name***

  Pathname of the rule file.

Examples

Calibre LVS

The following two examples show the syntax for running flat layout versus schematic (LVS) comparison. The second example creates a Cnet database without net and instance name files.

```
calibre -lvs my_rules
calibre -lvs -tl mycirc.cnet -nonames  my_rules
```

Calibre LVS-H

The following example extracts a hierarchical Spice netlist from the layout and writes it to file `foo.net`. It does not perform comparison.

```
calibre -spice foo.net rules
```

The following example performs hierarchical LVS comparison between a Spice layout and a Spice source. Cells correspond by name.

```
calibre -lvs -hier -automatch rules
```

The following example performs hierarchical LVS comparison between a Spice layout and a Spice source. The file `cells` specifies cell correspondence.

```
calibre -lvs -hier -hcell cells rules
```

The following example performs hierarchical LVS comparison between a Spice layout and a Spice source. Cells with the same name correspond; the file `cells` specifies additional cell correspondence.

```
calibre -lvs -hier -hcell cells -automatch rules
```

The following example extracts a hierarchical Spice netlist from a layout system (which must be GDSII or CIF), and then compares it to a Spice source. Netlist

extraction is hierarchical and comparison is flat: (This method is not recommended because it will result in a loss of layout locations in the LVS report, as well as less than optimal performance in the netlist input stage. Instead, use -lvs -hier with an empty -hcell file.)

```
calibre -spice foo.net -lvs rules
```

The following example extracts a hierarchical Spice netlist from a layout system (which must be GDSII or CIF) and then compares it to Spice source. Both netlist extraction and comparison are hierarchical. The file `cells` specifies cell correspondence.

```
calibre -spice foo.net -lvs -hier -hcell cells rules
```

The following example extracts a hierarchical Spice netlist from the layout and then compares it to a Spice source. Both netlist extraction and comparison are hierarchical; cells correspond by name:

```
calibre -spice foo.net -lvs -hier -automatch rules
```

The following example extracts a hierarchical Spice netlist from a layout specified in the rule file. The extraction preserves hcells as subcircuits in the extracted netlist, and the hcells are available in the comparison phase:

```
calibre -spice foo.net -hcell cells rules
```

Cell Correspondence Files

The next two examples show cell correspondence files. The first example uses comment lines, which must begin with two slashes (//) that appear at the beginning of a line. In the second example, Calibre LVS-H flattens the design down to the transistor level. This is essentially equivalent to flat Calibre LVS; however, for Spice-to-Spice comparisons, it will often execute faster.

1.

```
BUFF BUFF
BITL BIT
BITR BIT
TOP TOP
// This is a comment
```

2.

```
TOP TOP
```

Calibre MGC

The following example shows the syntax for running flat layout versus schematic comparison:

```
calibre_mgc -lvs my_rules
```

The following example shows the syntax for creating a Cnet database without net and instance names files:

```
calibre_mgc -lvs -tl -nonames my_circ.cnet my_rules
```

Sample LVS Rule File

```
//  Sample rule file to compare GDSII to Spice

    LAYOUT PATH "layout.gds"
    LAYOUT PRIMARY "top"
    LAYOUT SYSTEM GDSII

    SOURCE PATH "source.net"
    SOURCE PRIMARY "top"
    SOURCE SYSTEM SPICE

    LVS ABORT ON SUPPLY ERROR yes
    LVS POWER NAME VDD
    LVS GROUND NAME VSS
    LVS ALL CAPACITOR PINS SWAPPABLE no
    LVS REDUCE PARALLEL MOS yes
    LVS REDUCE PARALLEL BIPOLAR yes
    LVS FILTER UNUSED MOS no
    LVS FILTER UNUSED BIPOLAR no
    LVS REDUCE SERIES CAPACITORS yes
    LVS REDUCE PARALLEL CAPACITORS yes
    LVS REDUCE SERIES RESISTORS yes
    LVS REDUCE PARALLEL RESISTORS yes
    LVS REDUCE PARALLEL DIODES yes
    LVS RECOGNIZE GATES all
    LVS REDUCE SPLIT GATES yes
    LVS COMPONENT TYPE PROPERTY element // ignored for SPICE
    LVS COMPONENT SUBTYPE PROPERTY model // ignored for SPICE
    LVS PIN NAME PROPERTY phy_pin // ignored for SPICE
```

```
LVS IGNORE PORTS no
LVS REPORT "lvs.rep"
LVS REPORT MAXIMUM 50
MASK SVDB DIRECTORY "svdb" QUERY

// Connectivity extraction and device recognition
// operations not shown.
```

Calibre RVE/QDB-H

Usage

calibre -**rve** {***svdb_directory*** [*layout_primary*] || ***drc_db_file***}
[-cb] [-nowait]

calibre -**query** {***svdb_directory*** [*layout_primary*]} [-cb]
[-nowait][-64]

Description

The Calibre RVE/QDB-H commands probe Calibre LVS/LVS-H discrepancies and Calibre DRC/DRC-H results. Calibre RVE/QDB-H consists of two elements, as follows:

- Results Viewing Environment (RVE)—the graphical user interface of Calibre RVE/QDB-H. This interface provides two session windows: one for LVS debugging called LVS RVE, and one for DRC error highlighting called DRC RVE. Calibre RVE accesses the hierarchical query database, or query server, to return connectivity information about a design.

- Hierarchical Query Database (QDB-H)—the command line interface of Calibre RVE/QDB-H, which allows you to use the query server to probe a persistent hierarchical database (PHDB). Generally, you do not interact with the query server directly.

This tool requires that you run Calibre LVS/LVS-H or Calibre DRC/DRC-H prior to execution. It takes a SVDB directory as input to LVS-RVE. This directory must contain a PHDB or XDB. It takes an ASCII DRC results database as input to DRC-RVE.

For Calibre LVS/LVS-H results, Calibre RVE attempts to find discrepancy viewer and Spice files automatically, based on what the SVDB contains. The database stores the SVDB pathname when Calibre created the database. If you rename or copy the SVDB, RVE may not find the Spice files automatically.

Refer to the RVE/QDB-H and Query Server chapter for details on how to use the Calibre RVE interface to probe your Calibre LVS/LVS-H discrepancies and Calibre DRC/DRC-H results.

Arguments

Entering the following with no arguments displays a help line:

```
calibre -rve
calibre -query
```

- **-rve**

  A switch that invokes the results viewing environment (RVE). When you specify this switch without optional arguments, a dialog box displays and prompts you for either the SVDB directory or the DRC results database. This switch sets automatic access to query server functionality.

  
  Note

  To investigate LVS/LVS-H discrepancies with RVE, you must make sure the rule file includes the Mask SVDB Directory specification statement. In addition, source-to-netlist cross-reference files must be present when you are querying source information.

- ***svdb_directory***

  The absolute or relative pathname to the SVDB directory. This path contains the results from Calibre LVS/LVS-H execution: hierarchical database, cross-reference files, and discrepancy-viewer file. When you specify this pathname with the -rve switch, the LVS RVE session window displays.

- *layout_primary*

  The top cell name, as specified by the Layout Primary specification statement in the rule file. If you do not specify this name, LVS-RVE attempts to locate a single primary cell in the SVDB directory. If it finds multiple top cells, a dialog displays and prompts you with the top cells it found.

When you use the -query switch, you must specify this cell name only when the *svdb_directory* contains data from several different primary cells. You can omit this cell name when only one primary cell is present. When you do not specify *layout_primary*, the query server looks for a directory called *.phdb in the *svdb_directory*. If one and only one such directory is found then you must specify; *layout_primary* is the directory name excluding the ".phdb" suffix.

- ***drc_db_file***

  The absolute or relative pathname to an ASCII DRC results database (this includes short-isolation databases created by LVS Isolate Shorts YES). When you specify this pathname, the DRC-RVE session window displays.

- -cb

  This option is discussed under "Calibre CB" on page 2-32.

- -nowait

  This switch instructs the MGLS license queueing features to be disabled. This results in Calibre exiting, instead of queueing for a license, if one is not available.

- **-query**

  A switch that enables command line interaction with the query server only.

- -64 (Calibre QDB-H)

  This switch invokes the 64-bit version of Calibre. It is available on the HP and Solaris platforms, which require at least HP-UX 11.0 and Solaris 7, respectively. The default is 32-bit mode.

  The 64-bit executable on HP-UX provides a theoretical process size limit of roughly 1G * 1G / 4 bytes (or 2^{62} bytes) compared to only 4 Gbytes with the 32-bit executable. The 64-bit version of Calibre may, however, consume more memory than 32-bit Calibre executing on the same data.

Examples

```
calibre -rve svdb top_cell
calibre -rve results
calibre -query svdb
```

Calibre Interactive

Usage

calibre -gui [{-drc | -lvs} [*runset*]]

Description

The Calibre Interactive palette allows you to invoke a Calibre DRC window, Calibre LVS window, or Calibre RVE. You display the Calibre Interactive palette by entering the following command line in a shell:

```
% calibre -gui
```

- **Calibre DRC Window**—You can invoke the Calibre DRC window with or without a runset loaded. A runset is, roughly, a setup file for a specific Calibre run.

  To invoke the Calibre DRC window alone, use the following command line entry:

  ```
  % calibre -gui -drc
  ```

  To invoke the Calibre DRC window with a runset automatically loaded, use the following command line entry:

  ```
  % calibre -gui -drc runset
  ```

  where *runset* is an optional replaceable parameter that specifies the path name of the runset to be loaded.

- **Calibre LVS Window** — You can invoke the Calibre LVS window with or without a runset loaded.

  To invoke the Calibre LVS window alone, use the following command line entry:

  ```
  % calibre -gui -lvs
  ```

  To invoke the Calibre LVS window with a runset automatically loaded, use the following command line entry:

  ```
  % calibre -gui -lvs runset
  ```

where *runset* is an optional replaceable parameter that specifies the path name of the runset to be loaded.

Calibre Interactive can also be invoked through your layout editor user interface if it supports a Calibre menu item. User interfaces which support this feature are discussed in "Calibre Interactive" on page 3-1.

Arguments

- **-gui**

  A mandatory switch for invoking Calibre Interactive. When specified alone, you get the Calibre Interactive Palette. When specified with -drc or -lvs, you get the respective user interface window.

- -drc

  Optionally used with **-gui** to specify the Calibre DRC window.

- -lvs

  Optionally used with **-gui** to specify the Calibre LVS window.

- *runset*

  Optional argument specifying the pathname of a runset.

Calibre CB

The Calibre Cell/Block license package is intended to provide interactive block verification to customers using layout editors. Note it is not a separate tool, but a license package that enables some of the Calibre applications described previously.

The Calibre CB verification license is invoked through the -cb command line switch. The -cb switch causes Calibre to consume a single caldrclvseve license. This license can be used to run DRC, LVS, RVE, or Query Server. DRC and LVS can only be run in flat mode. DRC rule decks can include flat OPC operations provided the appropriate OPC licenses are available. LVS rule decks can include ERC functionality without acquiring additional licenses.

The caldrclvseve license allows users to do only one verification task at a time. Therefore you can do DRC, or LVS, or RVE, or access the Query Server at any

one time with one license. For example, if there is only one caldrclvseve license and the user is executing a DRC run, an additional job cannot be performed until the initial DRC run completes.

The caldrclvseve license cannot be used to form a hierarchical pair. For example, you cannot combine a caldrclvseve license with a hierarchical LVS (calibrehlvs) license to execute hierarchical Calibre LVS.

Calibre Interactive (calinteractive license included in the Calibre CB package) has an option to call this license for DRC, LVS, and RVE runs. When used in this fashion, and when using Calibre Interactive invoked directly from the layout editor, this license provides for doing interactive verification.

Chapter 3
Calibre Interactive

This chapter describes the Calibre Interactive graphical user interfaces (GUIs). Calibre Interactive is designed to simplify the invocation of Calibre DRC, LVS, and RVE applications. This chapter includes a description of the Calibre Interactive GUIs and information on how the tool interacts with other applications, including layout editors—especially Cadence Virtuoso.

Graphical Interface Overview

Calibre Interactive allows you to invoke Calibre applications using settings specified within the GUIs. These applications include Calibre DRC/DRC-H, Calibre LVS/LVS-H, xCalibre PX-C/RC and Calibre RVE.

Calibre Interactive consists of four windows.

- Calibre Interactive Palette—A palette window that allows you to invoke the Calibre DRC and LVS windows, or Calibre RVE.
- Calibre DRC Window—A session window that allows you to specify settings for a Calibre DRC run.
- Calibre LVS Window—A session window that allows you to specify settings for a Calibre LVS run, including the selection of ERC rule checks.
- Calibre PEX Window—A session window that allows you to specify settings for an xCalibre run. See *Using the Calibre Interactive PEX User Interface*.

Graphical Interface Prerequisites

Calibre Interactive has the following prerequisites:

- **Platform support**—Calibre Interactive is available on all supported platforms found in the *Configuring and Licensing Calibre/xCalibre Tools Guide*. Refer to that document for instructions on how to install and configure Calibre software.

- **Licensing**—Calibre Interactive requires a calinteractive license. This is a stand-alone license or part of the Calibre CB package. License requirements are based on the application to be run and are discussed in the *Configuring and Licensing Calibre/xCalibre Tools Guide*.

- **Required files**—The required files for the Calibre DRC and Calibre LVS interfaces are as follows.

  - Calibre DRC—rule file and a layout database.

  - Calibre LVS—rule file, a layout database, and in most cases, a source database (for example, a Spice netlist).

  For both interfaces, you can (optionally) load a runset on the command line or from the **File** dropdown menu. For further information on loading runsets, refer to the section "Runsets".

Command line invocation of these interfaces is covered under "Calibre Interactive" on page 2-31. Setup of layout editors and Calibre Interactive invocation from layout editor user interfaces is covered in "Connections to Layout Editors" on page 3-22. If you already have your layout editor set up to call Calibre Interactive from a Calibre menu, you may invoke the Calibre Interactive user interfaces from the layout editor (this is the preferred method). The basic command line invocation syntax is this:

```
calibre -gui [ -drc | -lvs | -pex [runset] ]
```

The PEX interface is discussed in *Using the Calibre Interactive PEX User Interface*.

Runsets

The description of the user interface refers to a file called a *runset*. We discuss this file here. A runset is a text file created by Calibre Interactive that stores the settings you specify in the Calibre DRC and LVS windows. You can save the current runset by selecting the **File > Save Runset** or **File > Save Runset As...** menu items (discussed below).

Calibre Interactive saves the runset in the location specified in the **Save Runset** or **Save Runset As** dialog box. Calibre Interactive stores a list of your most recently opened runsets in your home directory as .cgidrcdb or .cgilvsdb for Calibre Interactive DRC or LVS, respectively. When invoked, the Calibre DRC and LVS windows automatically load the runset used when the last session was closed. If the rule file specified in the runset has changed since the time it was last loaded, a dialog box is displayed asking if you want to load the new rule file.

You can change the default behavior for runset opening from the **Setup > Options** menu. From this menu you can select a specific runset to open, or not to open a runset upon invocation. The runset filename opened at startup (if no runset is specified on the command line) can also be specified by setting the MGC_CALIBRE_DRC_RUNSET_FILE environment variable for DRC, and the MGC_CALIBRE_LVS_RUNSET_FILE environment variable for LVS. If these environment variables are set, they take precedence over all other runset opening behavior options. If they are set to " " (blank), no runset is open. If the filename specified by the environment variable is not readable, then the tool will select which runset to open (or none) depending on your selection from the **Setup > Options** menu.

Runset files are ASCII text files. As such, you may view and edit them. An example is shown in Appendix B. Runset files only show the settings you make that are different than the default settings. There is a one-to-one correspondence between the name-value pairs in a runset file and fields in the user interface that you change. The complete list of all default settings is found in $MCG_HOME/pkgs/icv/userware/default/cgi/options.tcl.

Graphical User Interface Description

This section describes the Calibre Interactive user interfaces and their interactions with Calibre DRC and LVS applications.

Calibre Interactive Palette

Invoking calibre -gui brings up the Calibre Interactive palette:

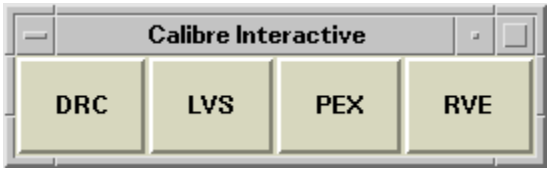


Figure 3-1. Calibre Interactive Palette

From this palette you can invoke the Calibre DRC and LVS windows, and Calibre RVE. Simply select the button of your choice by clicking on it with your left mouse button. It is more likely that you will invoke the DRC, LVS and RVE user interfaces through your layout editor. See "Connections to Layout Editors" on page 3-22. (The PEX interface is discussed in *Using the Calibre Interactive PEX User Interface*.)

Calibre DRC Window

The Calibre DRC window allows you to specify command line options and selected rule file statements that execute the desired Calibre DRC functionality.

You can invoke the DRC window directly from a supporting layout editor, from the command line, or from the palette menu.

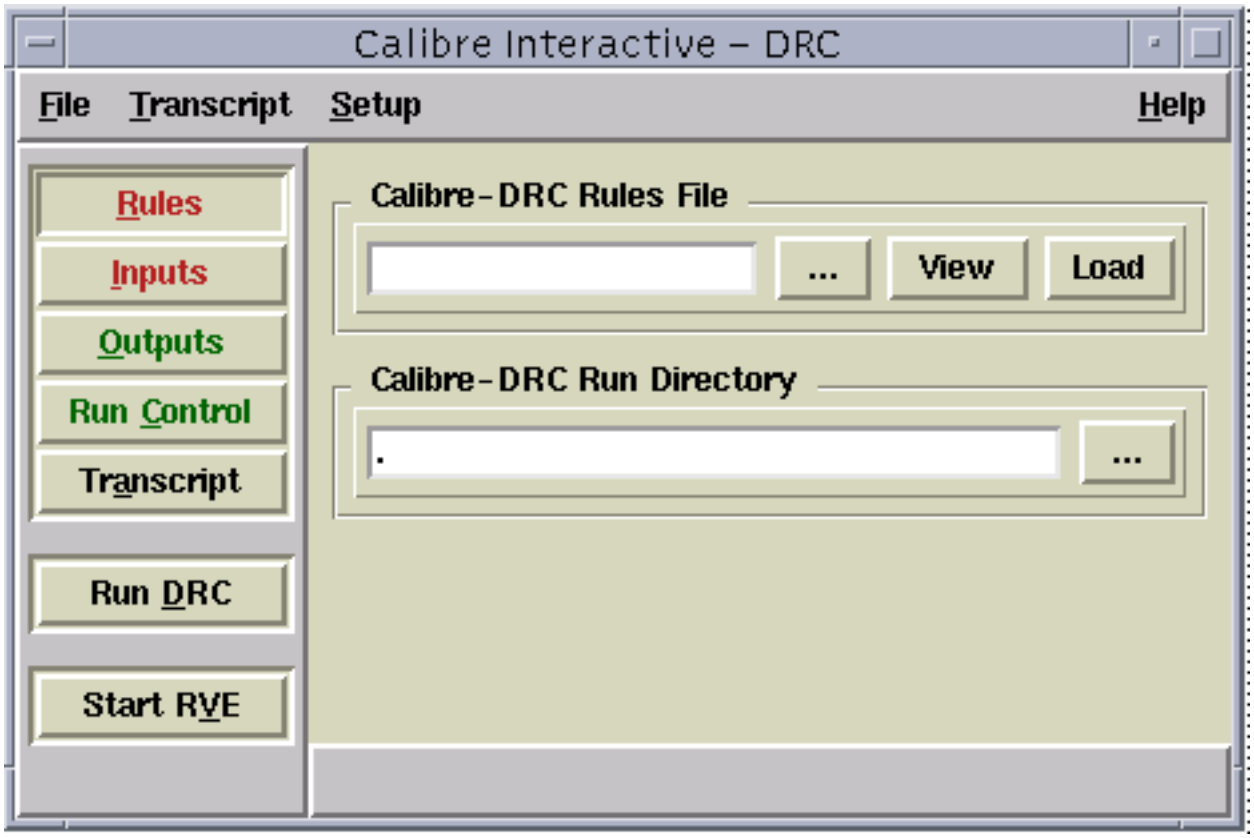


Figure 3-2. Calibre Interactive - DRC Window

Pulldown menus appear across the top of the window. Setup options are accessed from the Setup menu pulldown in addition to the menu buttons along the left side of the interface. Underscored letters indicate keystroke letters to press in conjunction with the Alt button on your keyboard. For example, pressing Alt and *o* simultaneously opens the Outputs menu (this might not work if you run Calibre through Exceed). Red buttons along the left side of the interface indicate menus that require your input before you can execute a DRC run. Green buttons indicate menus that are completely filled in with required information (note that this information may need to be changed by you, depending on your situation). If you start with a new runset (that is, an existing runset is not currently loaded), some menus are pre-populated with default settings and the corresponding buttons appear green upon invocation. The last runset you used (if any) will be loaded by default, unless you have changed this behavior (see "Runsets" on page 3-3).

Balloon help is on by default. This means that when you put your cursor over an item in the GUI for a few seconds without selecting the item, a help balloon opens describing the item. Balloon help is available for all menu buttons and user entry fields. Balloon help can be turned off from the Setup pulldown menu.

Calibre DRC Pulldown Menus

There are four pulldown menus across the top of the interface:

File—allows you to open and save runsets, as well as to exit the application. It contains the following options:

- **New Runset**—restores the GUI to its default configuration.
- **Open Runset...**—opens a dialog box that allows you to select a runset.
- **Open Text File...**—opens a dialog box that allows to select a text file for viewing (for example, a rule file or runset file).
- **Save Runset**—opens a dialog box that allows you to save the current runset.
- **Save Runset As...**—opens a dialog box that allows you to save the current runset under a different name.
- **Recent Runsets**—displays a list of recent runsets that you can choose from.
- **Exit**—exits the application

Transcript—allows you to open, save, and search Calibre DRC run transcripts. It has the following options:

- **Save As...—**opens a dialog box to save the current run transcript. This automatically invokes the Transcript menu button.

- **Echo to File...**—opens a dialog box that enables you to set up copying of run transcripts automatically to a file that you specify.

- **Search...**—opens a dialogue box that allows you to search the current transcript for text that you enter.

Setup—allows you to set up Calibre DRC run options and DRC window help. It has the following options:

- **DRC Options**—selectable button that enables you to set up several DRC specification statements. It also activates a DRC Options button on the GUI. If you load a rule file (or runset containing a rule file) that has specification statements selectable from DRC Options, the DRC Options button will be automatically selected and the current rule file settings will populate the appropriate fields:

  - Output tab—selecting the button is equivalent to specifying DRC Cell Name YES CELL SPACE XFORM in your rule file (DRC-H only).

  - Maximums tab—allows you to specify DRC Maximum Results and DRC Maximum Vertex specification statement settings.

  - Includes tab—allows you to select rule files to include in the DRC run. This is equivalent to the specifying an Include statement in a rule file. Included files are checked for correct syntax. An error is reported for incorrect Include rule files.

  - Area DRC tab—allows you to select the halo size for DRC area checking (See DRC Area Checking below).

- **Set Environment...**— opens a dialog box that allows you to specify environment variables for use in rule files. Such variables can be marked as defined and have values assigned through this dialog. The values specified in the dialog can be saved in a runset file. When the runset is reopened, these environment variables are automatically initialized to the saved values.

- **Select Checks...**—opens a browser that allows you to control which DRC checks to run in a loaded rule file. It allows you to select individual checks

as well as rule check groups found in a Group statement. This is equivalent to specifying the DRC [Un]Select Check rule file statements.

After loading your rule file, the Select Checks dialog box displays all selected checks and groups with a green check box. Rule checks specified in a DRC Unselect Check specification statement are marked with a red X box; these checks are not executable even if you select them in the user interface.

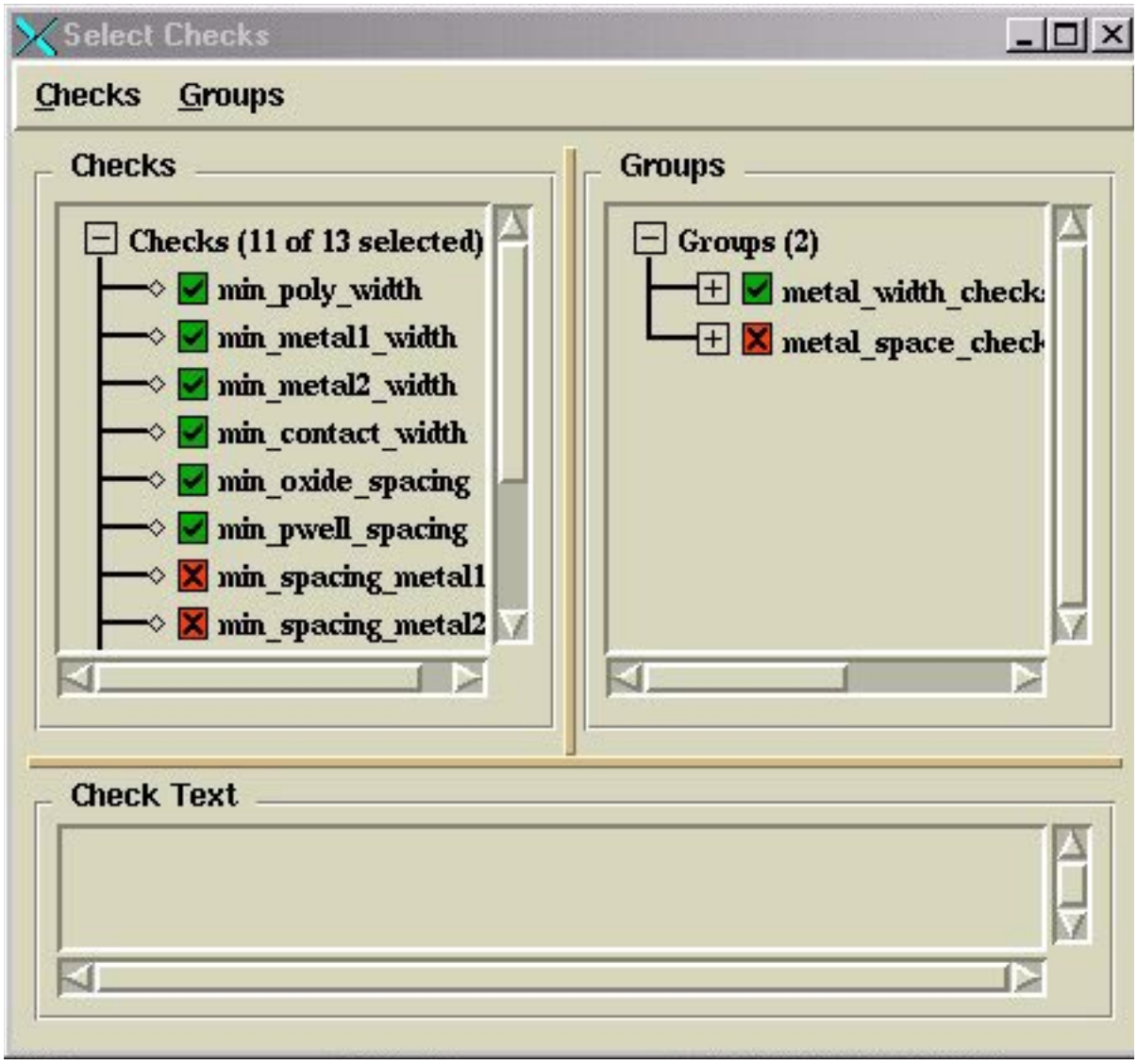


Figure 3-3. Select Checks Dialog

You can toggle the selection of an individual rule check by left-clicking its name in the Checks pane. You can select or unselect a rule check group by right-clicking on its name in the Groups pane.

The Check Text pane displays the check text for the most recently toggled rule check. You can display the check text for any rule check, without toggling its status, by clicking it with the middle mouse button.

The Checks menu allows you to sort the checks in the left pane and to select and unselect all checks. The Groups menu allows you to sort groups and to control how groups are displayed. If you select the Show Groups Hierarchically setting, groups are displayed with their subgroups. Without that setting, groups are displayed *flat*, that is, with their constituent checks only. If the Show Toplevel Groups setting is chosen, only top-level groups (that is, groups that are not part of other groups) are shown at the first level. Without this setting, all groups are shown at the first level.

The settings in the GUI take precedence over the rule file wherever there is the possibility of conflicting settings.

- **Options...**—opens a Setup Program options dialog with a series of tabs.

  - Startup tab—offers Runset opening behavior control. You can have the last runset used open upon Calibre Interactive invocation (the default), open no runset on invocation, or you can specify the pathname of a runset to open.

  - Templates tab—offers control over the settings of various fields when Calibre Interactive is started from a layout editor. If you do not start Calibre Interactive from a layout editor, these fields have no effect. These template fields can contain %l and %s as replaceable parameters for the layout cell name and the source cell name, respectively. If these parameters are present, they are replaced automatically by the tool with the appropriate names.

    The Import Layout Database option is selected by default. If you do not want to import from the layout editor directly, unselect this button. There is a similar feature on the Inputs menu discussed below.

- Misc. tab—allows you to control if a warning is displayed when an open runset is not writable (the default is yes) and to control if Input and Output pane settings are read from the rule file upon loading it (the default is yes).

  You may override any of the settings derived from your most recently loaded rule file that appear in the GUI. The GUI settings take precedence in subsequent Calibre runs. Calibre Interactive offers complete control of inputs and outputs from the Inputs and Outputs menus accessible from the vertical menu bar.

- **Layout...**—opens a dialog box that allows you to specify the socket to use to connect to your layout editor. The setting is 9189 by default and usually the connection can be established automatically when Calibre Interactive is invoked through the editor. You can change the default value. If your layout editor is one that supports Calibre Interactive, but there is no connection established between the tools, setting this manually may resolve the issue. The socket port number may be specified by the MGC_CALIBRE_LAYOUT_SERVER environment variable.

Help—allows you to call up end-user documentation. The **About...** item tells you the version of Calibre Interactive you are using.

Calibre DRC Menu Buttons

On the left side of the GUI you will see a group of menu buttons:

Rules—activates a menu that allows you to select, view, and load your DRC rule file. A rule file must be successfully loaded in this menu to run Calibre. This is equivalent to the ***rule_file_name*** argument on the command line. This menu also allows you to select the Calibre run directory. By default it is the current directory.

Inputs—activates a menu that allows you to select which Calibre DRC license to use (hierarchical, flat, or caldrclvseve; equivalent to -drc -hier, -drc, or -cb command line arguments, respectively). If you invoke Calibre Interactive through a layout editor, the **Import layout database from layout viewer** setting is selected by default (this setting is also controlled from the Setup > Options Templates tab). The current top-level cell in your layout editor will be used. If you do not want to take layout input from the layout editor, unselect that button (it is unselected by default if you invoke Calibre Interactive from the command line). You specify the path to a layout file in the Files field (equivalent to the Layout Path rule file statement). You specify the top-level cell in the Primary Cell field (equivalent to the Layout Primary rule file statement). The latter two items do get read from the loaded rule file, but you can change them. The **Check area** setting allows you to check a rectangular area that you specify (see DRC Area Checking).

Outputs—activates a menu that allows you to specify the pathname and type of DRC Results Database (equivalent to DRC Results Database specification statement), to specify an optional DRC Summary Report file (equivalent to DRC Summary Report specification statement), to view the summary report after the Calibre run, to append the current summary report or replace it, and to start DRC-RVE after the Calibre run.

Run Control—activates a menu that allows you to specify performance and remote queuing options. The Performance tab items control the -nowait, -64, and -turbo command line arguments. The Remote Execution tab controls remote

execution of Calibre. The Calibre Interactive GUI will complete all the text boxes for which it can determine the values. See Support for Distributed Queueing for additional details.

DRC Options—this is discussed under DRC Options on page 3-7.

Transcript—opens the Calibre DRC run transcript. This works in conjunction with the **Transcript** pulldown menu.

Run DRC—executes a Calibre DRC run with the current runset information.

Start RVE—invokes Calibre DRC-RVE.

DRC Area Checking

Calibre Interactive DRC allows you to check a rectangular area of a design. Select the **Check area** setting in the Inputs menu pane and specify the lower-left and upper-right vertex coordinates of the rectangular area. You can also specify the area by first left-clicking on the "..." button and then dragging a rectangle in the layout editor. Right-click on the "..." button to zoom to the area specified in the entry field.

While doing area DRC, the area that is actually checked is the specified area plus a halo region around the area. The size of the halo can be specified through the Area DRC tab of DRC Options menu pane (**Setup > DRC Options**). When the halo width computation is set to **Automatic**, the width is computed as half of the lesser of the width and height of the specified area. You can also specify an explicit width for the halo in this pane. All objects and instances in the primary layout cell are clipped to the area and the halo region. Errors that do not intersect the specified area are removed from the output database. If the **Import layout database from viewer** setting is selected, only the geometries enclosed in the specified area and halo region are exported to Calibre.

DRC area checking uses the Layout Window specification statement. You can see the arguments passed to this statement in the control file created by Calibre Interactive (see "Control Files" on page 3-18).

Support for Distributed Queueing

You can invoke Calibre DRC or LVS on a remote cluster (a distributed queue) using commands supplied by the distributed queue software (or load balancing software). Choose the **Run *application* on remote host** option in the Remote Execution tab of the Run Control pane.

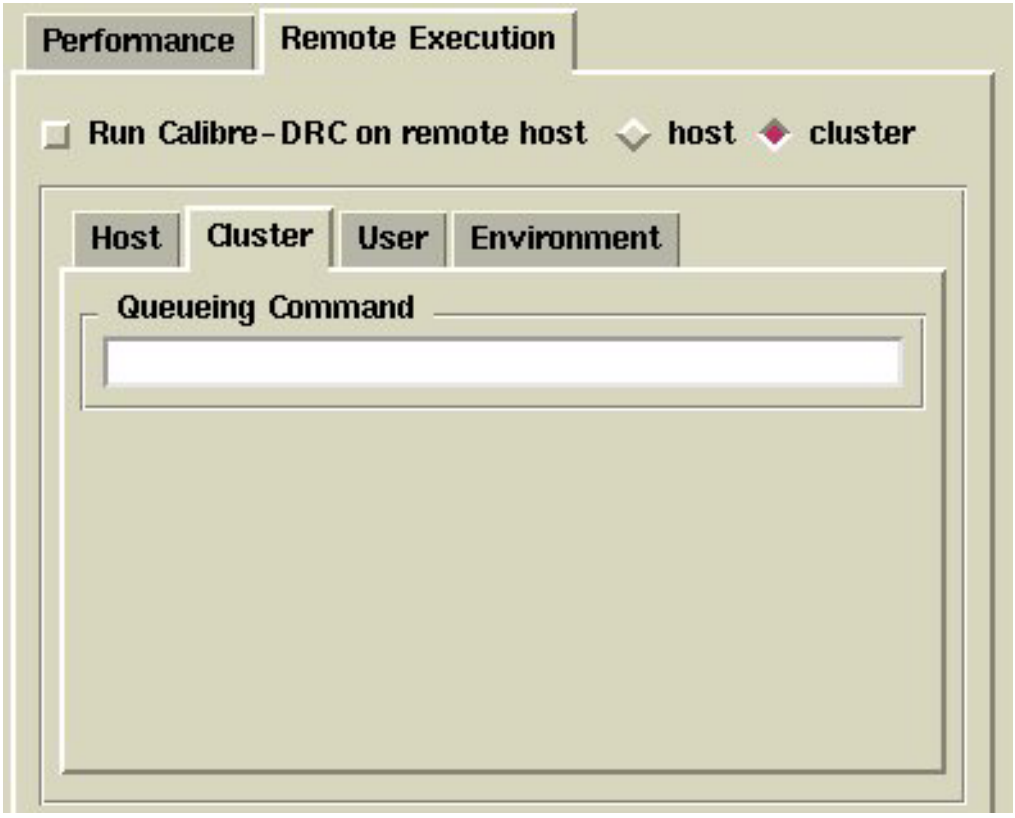


Figure 3-4. Run Control Pane

You can select the **host** or **cluster** button. You can then specify the command to use which assigns the Calibre run to the queue. You can also specify user and environment information for the Calibre run from the associated tabs on the Remote Execution pane. In the text entry fields, you can use the following parameters, which are replaced by the values in the GUI before executing the command:

- %c — is replaced by the Calibre command.
- %o — is replaced by the options to Calibre.

- %C — is replaced by Calibre command and options.
- %u — is replaced by the user name specified in the User tab.
- %m — is replaced by value of the MGC_HOME environment variable specified in the Environment tab.
- %L — is replaced by licensing variable name specified in the Environment tab.
- %l — is replaced by licensing variable value specified in the Environment tab.

For example, you can put “bsub %C” (bsub is a distributed queueing application produced by Platform Computing) for the Queueing Command field and this will submit the Calibre job with the appropriate command and options specified in the GUI to the queue.

Calibre LVS Window

The Calibre LVS window allows you to specify command line options and selected rule file statements that execute the desired Calibre LVS application. It looks and behaves very similarly to the DRC window discussed previously. The LVS-specific functionality that is different from the DRC window is discussed in this section.

Calibre LVS Pulldown Menus

The LVS pulldown menus appear the same as the DRC menus on page 3-6. The **File**, **Transcript,** and **Help** pulldowns have the same features as the DRC window. The **Setup** pulldown appears the same, but has a few different options than the DRC window:

Setup—allows you to set up Calibre LVS run options and LVS window help. It has the following options:

- **LVS Options**—selectable button that enables you to set up numerous LVS specification statements. It also activates an LVS Options button on the GUI. If you load a rule file (or runset containing a rule file) that has

specification statements selectable from LVS Options, the LVS Options button will be automatically selected and the current rule file settings will populate the appropriate fields. Any changes you make to pre-populated fields will take precedence over the rule file you load.

- Supply tab—allows you to control the behavior of LVS Abort On Supply Error, LVS Power Name, and LVS Ground Name specification statements.

- Gates tab—allows you to specify the behavior of the LVS Recognize Gates specification statement.

- Shorts tab—allows you to specify the behavior of the LVS Isolate Shorts specification statement.

- ERC tab—allows you to control the behavior of LVS Execute ERC, ERC Results Database, ERC Summary Report, ERC Maximum Results, and ERC Maximum Vertex statements. The **Select Checks** button activates a browser that controls the behavior of the ERC [Un]Select Check specification statements. The behavior of this browser is very similar to the DRC Select Checks browser discussed on page 3-7.

- Name Connect tab—allows you to specify the behavior of the Virtual Connect Colon, Virtual Connect Name, and Virtual Connect Report specification statements.

- Includes tab—allows you to select rule files to include in the LVS run. This is equivalent to the specifying an Include statement in a rule file.The syntax of included files is checked and errors are reported if the Include files are incorrect.

- Other tab—allows you to control the behavior of the LVS Abort On Softchk, LVS Ignore Ports, LVS Report Maximum, LVS Show Seed Promotions, and LVS Show Seed Promotions Maximum specification statements.

- **Options...**—this is very similar to the Options... item in the DRC interface described on page 3-9. The Templates tab is different, however, in that it deals with LVS templates, which have more options than DRC templates.

Calibre LVS Menu Buttons

These are very similar in appearance to the DRC menu buttons on page 3-11. The **Rules**, **Run Control**, **Transcript**, **Run LVS** and **Start RVE** menus are identical in purpose to their DRC counterparts, with the obvious difference being that they apply to LVS, not DRC. The menus that differ significantly from DRC are the **Inputs**, **Outputs**, and **LVS Options** (discussed above) menus:

Inputs—activates a menu that allows you to:

- select which Calibre LVS license to use (hierarchical, flat, or caldrclvseve; equivalent to -lvs -hier, -lvs, or -cb command line invocation arguments, respectively)
- specify the type of LVS comparison to perform (Layout vs Netlist is GDSII-to-Spice; Netlist vs Netlist is Spice-to-Spice)
- specify whether to extract a Spice netlist from the Layout (equivalent to the -spice command line invocation argument—LVS-H only)
- specify information about the layout and source databases

There are three tabs in this menu. A tab appears red when it requires information from you; green indicates the menu has complete information (although not necessarily correct information for your specific run). Tabs and entry fields are grayed out if they are not required for the type of LVS you plan to run. Information that is read from the loaded rule file populates the corresponding fields in these tabs. If you change an entry, the GUI information overrides the rule file.

- Layout tab—allows you to specify the Layout Path and Layout Primary specification statements information in the Files and Primary Cell fields, respectively. The Layout Netlist field is the pathname of the extracted netlist if you enable the Netlist Extraction radio button (LVS-H only). This field specifies the path of the layout netlist if you enable the Netlist vs

Netlist radio button. The default behavior is to take layout database input from the layout editor, using the current top-level cell.

- Netlist tab—allows you to specify the Source Path and Source Primary specification statements information Files and Primary Cell fields, respectively. You can specify multiple Spice files as the source path. The delimiter is a newline. If multiple Spice files are specified, LVS-GUI creates a file "gui.source.net" that includes the multiple Spice files before running Calibre LVS.

- Hcells tab—allows you to enable the -hcell and -automatch command line arguments. Do not use -automatch unless you are quite certain that there are at least some corresponding source and layout cell names that actually match (see -automatch on page 2-18).

Outputs—activates a menu that allows you to specify the pathnames of the LVS report, the LVS output database, and other LVS output files. Information that is read from the loaded rule file populates the corresponding fields in these tabs. If you change an entry, the GUI information overrides the rule file.

- Report/SVDB tab—allows you to specify the LVS Report and Mask SVDB Directory specification statement pathnames and options, to view the LVS report upon completion of the LVS run, and to start LVS-RVE upon completion.

- Flat-LVS Output tab—allows you to specify Mask Results Database specification statement information, to specify the NOFLAT keyword for the Mask SVDB Directory statement, and to specify the -ixf, -nxf, -bpf and -nl command line invocation arguments.

Run Directory

The run directory is a directory location from which the Calibre application is run and from which all relative path names are resolved. By default, Calibre Interactive sets the run directory to your current directory.

You can change the run directory in the **Rules** dialog box of the Calibre DRC and LVS windows. This may require you to update the paths to input and output files.

Control Files

Before running Calibre DRC or LVS, the Calibre Interactive GUI creates a rule file, called a control (or header) file, based on your dialog box selections that uses an Include statement to call the rule file specified in the **Rules** dialog box. The control file is saved in the run directory with the following naming structure:

```
_rule_file_name_
```

where the name of the specified rule file is enclosed in underscores (_). This control file will be overwritten if you execute another application from the same run directory using a similarly-named rule file.

The settings in the Calibre Interactive control file may override statements that occur in the original rule file. This is not how Include statements usually function in SVRF rule files, but it is allowed in Calibre Interactive.

Text Editing

If you choose **File > Open Text File...** from a DRC or LVS window, you can open a text window like this (without the shown text, of course):

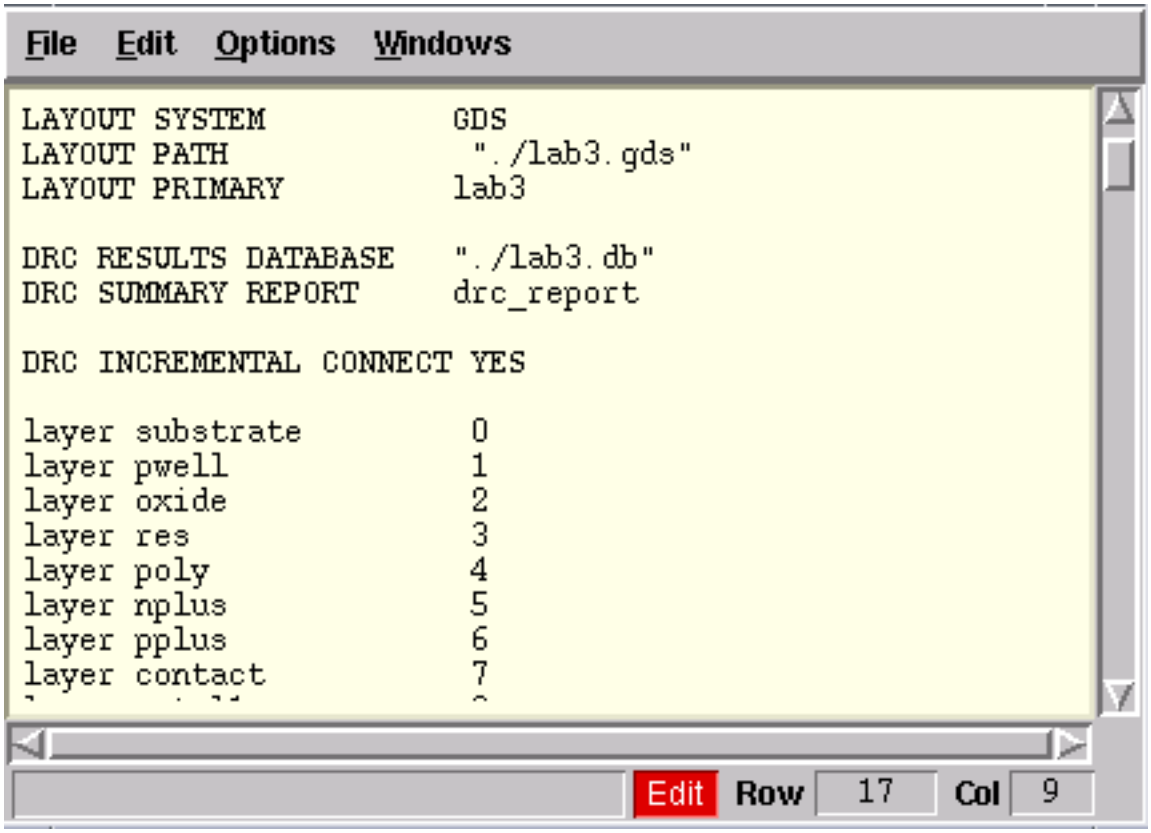


Figure 3-5. Text File Window

Such a window is also invoked by selecting a View button for a text file in GUI menus that have such buttons. The pulldown menus have these selectable options using your left mouse button (pressing and holding the right mouse button opens a flyout menu with these items):

File—allows you to open and save text files and has the following options:

- **Open**—opens a dialog box that allows you to open a text file.
- **Save**—allows you to save a changed file under its current name.

- **Save As...**—opens a dialog box that allows you to save the current file under a different name.
- **Print**—opens a dialog box that allows you to print the text file.
- **Close**—allows you to close the application.

Edit—allows you to perform editing functions.Use of these functions depends upon the selection of text by pressing the left mouse button and dragging through text and/or allowing changes to the file (see **Options >Edits Are**):

- **Cut**—removes selected text from the file and stores it in a memory buffer.
- **Copy**—copies selected text to a memory buffer without removing the text.
- **Paste**—places text from the memory buffer into the file at the current insertion point.
- **Delete**—erases selected text.
- **Select All**—selects all text in the file.

Options—allows you to perform various text search, view, and editing control functions:

- **Search...**—opens a dialog that allows you to search for a text string.
- **Go to Line...**—opens a dialog box that allows you to scroll immediately to a specified line in the file.
- **Line Numbers**—toggling this to true displays the line numbers of text. The default is false.
- **Status Bar**—toggling this to true activates the status bar at the bottom of the text window. The default is true. The status bar displays whether changing the file is allowed or not (the Edit indicator is green if editing is allowed), in addition to the row and column number of your cursor insertion point.

- **Font**—opens a flyout menu that allows you to select font size. The default is Normal.
- **Edits Are**—opens a flyout menu for toggling the alterability status of the file. The default is Not Allowed.
- **Windows**—allows you to select the arrangement pattern of open text windows. The default is “as is.”

Text windows respond to the Page Up and Page Down keys on your keyboard.

Interface to Calibre RVE

Calibre Interactive allows you to invoke Calibre RVE in three ways; from the Calibre Interactive palette, from the button bar of the Calibre user interface windows, or automatically after the completion of a Calibre application.

- Calibre Interactive palette—Click **RVE** to display the Calibre RVE startup screen.
- Automatic—You can instruct Calibre Interactive to invoke Calibre RVE after the Calibre LVS/DRC application finishes, this is done in the Outputs dialog box.

  For DRC, you can set this only if the DRC results database is in ASCII format. For LVS, you can set this only if you select **Create SVDB Database**.
- **Start RVE** button bar item—You can invoke Calibre RVE with the **Start RVE** button. From the Calibre DRC window, this opens Calibre RVE on the specified DRC results database only if it is in ASCII format. From the Calibre LVS window, this opens Calibre RVE with the specified SVDB.

  If you do not specify an ASCII formatted DRC results database or an SVDB directory, the Calibre RVE startup screen is displayed.

You can also access RVE from the Calibre pulldown on supporting layout editors.

For information on Calibre RVE, refer to chapter 15, "RVE/QDB-H and Query Server".

Connections to Layout Editors

Calibre Interactive establishes connections through the following layout editors:

- ICgraph
- Calibre LITHOview
- Calibre WORKbench
- DESIGNrev
- Cadence Virtuoso

The first four editors are Mentor Graphics products. All have a Calibre pulldown menu by default. The Calibre Interactive interfaces are accessible from this pulldown. Virtuoso setup is covered in a following section.

Mentor Graphics Layout Editor Interfaces to Calibre

Mentor Graphics layout editors have a Calibre pulldown menu similar to the following:

Figure 3-6. Calibre Pulldown Menu

Selecting any of the first four buttons activates the corresponding Calibre user interface.

Communications sockets are set up automatically when you invoke Calibre Interactive through the layout editor. The default socket number is 9189. You make this connection manually from the **Setup** menu item.

Calibre Interactive can communicate with any of the previously-mentioned Mentor Graphics layout editors to enable automatic export of GDSII layout databases. Calibre Interactive imports the **Primary Cell** according to the setting in the **Inputs** dialog box. By default, the currently open cell in the layout editor is selected for import.

Cadence Virtuoso Interface

To set up your Virtuoso layout editor with a Calibre pulldown menu, first read the $MGC_HOME/shared/pkgs/icv/tools/queryskl/calibreREADME file. It gives complete instructions on installing the calibre.skl file that resides at the same directory level.

The Calibre menu is added by the Skill Interface as a user menu trigger. Whenever a layout window (of type maskLayout) is opened, the user menu trigger is executed. The Skill Interface then installs any other user menus that may have been defined (either by you or by other applications), and then installs the Calibre menu as the last menu.

If other applications also install menus in layout windows, we recommend that you load calibre.skl after you load these other applications' Skill files. If you wish to load the Calibre menu before other applications' menus, you need to be sure that the other applications' Skill code ensures that the Calibre menu also gets loaded.

If you wish to install the Calibre menu in a different type of window (instead of the default window type maskLayout), you need to set the Skill variable mgcCalibreMenuViewType to that different type before you load calibre.skl. This can be done, for example, in the .cdsinit file as shown in the following code:

```
mgcCalibreMenuViewType = "myLayout"
; instead of "maskLayout"
```

```
load("calibre.skl")
; load calibre.skl
```

Note that if you switch the layout window to another tool (for example, Layout XL orPcell), the Calibre menu will disappear if you are using the default setting for mgcCalibreMenuViewType (= "maskLayout"). To access Calibre, you will have to switch back to the “Layout” tool. The Calibre menu will reappear when you switch to the “Layout” tool. Alternatively, you could set mgcCalibreMenuViewType to the type of the window installed by the tool (for example, “maskLayoutXL” for Layout XL, “maskLayoutParamCell” for Pcell, and so forth). Note that if you do set mgcCalibreMenuViewType, you will not see the Calibre menu for the default layout view provided by the “Layout” tool. To see the type of view installed by a Cadence tool, consult the .ini files (pCellGen.ini for Pcell, lx.ini forLayout XL) in the <cadence-install-dir>/tools/dfII/etc/context directory.

The mgc_load_calibre_menu() Skill command is also provided. This command installs the Calibre menu in the active window. This command can be tied to a bind key if desired.

The Calibre menu in Virtuoso layout windows (this is, windows of type maskLayout) has following items:

- **Run DRC**—starts Calibre Interactive DRC.
- **Run LVS**—starts Calibre Interactive LVS.
- **Start RVE**—starts Calibre RVE.
- **Clear Highlights**—clears RVE highlights in the current cell.
- **Setup**—this is described below.
- **OPC Workbench**—contains a submenu for using Calibre WORKbench, if installed.

The **Setup** menu has the following items:

- **Set RVE Cell Libnames...**—allows you to specify your desired Cadence Library for RVE. Multiple libraries may be entered in a space-delimited list.

- **Socket...**—specifies the socket that Calibre Interactive and RVE use to communicate with Virtuoso. The socket is not initialized until needed. If you wish to initialize the socket during the loading of calibre.skl, set the environment variable MGC_RVE_INIT_SOCKET_AT_STARTUP to any value before loading calibre.skl. The socket number is displayed in the CIW window. The socket port number may be specified by the MGC_CALIBRE_LAYOUT_SERVER environment variable.

- **RVE Highlight Layers**—allows you to define error layer appearance. See Using RVE with Virtuoso.

- **Layout Export**—displays a dialog that controls options used while exporting layout databases to Calibre.

  You can load Streamout template files into the **Layout Export** dialog. The dialog also allows you to save template files.

  Calibre's layout export does not provide control over all the various options available for Streamout in Virtuoso. However, the export procedure utilizes all the supported options during export. The recommended flow is to set up all the applicable options for Streamout export through Virtuoso's **File > Export > Stream** dialog, then save these options in a template file. This file can be loaded into Calibre's **Layout Export** dialog in the Template field.

  You can specify the template file to load at startup by specifying the file name as the Skill variable mgc_calibre_export_layout_template_file before you load calibre.skl. This is equivalent to loading the template file with the Skill load command.

  When you instruct Calibre Interactive to export a layout database from Virtuoso, Calibre Interactive directs Virtuoso to display the Layout Export dialog box. This dialog box allows you to specify a layer mapping file and a Skill template file to be used during the export of the database.

When exporting a layout to GDSII format (using PIPO strmout), several options can be set that govern the export of the layout. These options are stored within Virtuoso as the Skill variable “streamOutKeys”.

Before instructing Virtuoso to export the layout to GDSII format, Calibre Interactive determines whether the streamOutKeys variable has been set within the Virtuoso editor. If streamOutKeys is set, Calibre Interactive creates a text file containing the streamOutKeys list variable values. However, if streamOutKeys is not set, Calibre Interactive initializes streamOutKeys to default values. The created text file is saved in the Run Directory and follows the naming convention:

```
CGI.streamOut.<cell_name>
```

The streamOutKeys variable may be loaded directly in the CIW, or using a trigger function.

Lower and upper-case settings are supported. You can set the streamOutKeys->caseSensitivity variable to "upper", "lower", or "preserve" before exporting the layout.

The **Layout Export** dialog allows you to specify the run directory to use during layout export. Relative pathnames specified in the export dialog are qualified with the run directory pathname. The PIPO.LOG.<cellname> file's location can be controlled by setting the run directory, and by specifying a relative pathname for the log file. The default is the current directory.

Calibre Interactive alters only two options set by existing streamOutKeys within Virtuoso; the case sensitivity and primary cell options. The case sensitivity option is set to *preserve*, and the primary cell option is set to the value specified in the **Inputs > Primary Cell** text box.

After creating the text file based on the streamOutKeys variable, Calibre Interactive instructs Virtuoso to export the layout to GDSII format and continues with the Calibre application.

You can elect to display the **Layout Export** dialog every time before export by checking the “Show dialog before export” check button in the dialog.

- **Netlist Export**—controls options used while exporting Composer schematics to Calibre.

  The LVS Invocation GUI supports automatic import of Composer netlists as the source database. The **Netlist Export** dialog allows you load and save cdl netlisting template files.

  Calibre LVS uses cdl as its source database format when invoked from Virtuoso. Consequently, the netlist export routines use the simulation interface system (“si -batch”) to translate the schematic to cdl. The various settings in the Netlist Export dialog are similar to cdl export settings and can be set through the “cdlOutKeys” global Skill variable. This variable is found in the “si.env” file. This variable may be loaded in the CIW, in a trigger function, or by specifying the mgc_calibre_export_netlist_template_file Skill variable to point to a template file.

  The Calibre Netlist Export Setup dialog has the following features:

  - View name:—this sets the Cadence view of the cell
  - Simulator:—sets the tool doing the netlisting (cdl is typical).
  - View List:, Stop List:—these settings control hierarchy traversal.

    The view and stop lists used for netlist export to Calibre Interactive can be controlled by setting the CDS_Netlisting_Mode variable, either as an environment or as a Skill variable. If the variable's value is set to ANALOG, aucdl views are netlisted. Any other setting (including DIGITAL) netlists cdl views. The corresponding fields in cdlOutKeys are simSimulator, simViewList, and simStopList.
  - Equivalents:—produces *.EQUIV statement in netlist.

- Connects:—produces *.CONNECT statement in netlist. Specify the nets with a "=" sign between them. For example, specifying the following in the dialog entry field:

  VDD=VDD! VCC=VCC!=VCC2

  will produce the following statements in the generated netlist:

```
*.CONNECT VDD VDD!
*.CONNECT VCC VCC! VCC2
```

- Include:—produces *.INCLUDE statement in netlist.

- Check LDD:—produces *.LDD statement in netlist.

- Display pin:—produces *.PININFO statement in netlist.

All settings can be saved to a template file and this creates a cdlOutKeys variable.

The layout and netlist export mechanisms display a dialog if the cell being exported has been modified. The dialog allows the user to optionally save the cell before exporting it. Note that only the top-level cell is checked for modifications.

Using the Virtuoso Interface

The Calibre Interactive allows you to run DRC or LVS on a Virtuoso cell. To do this, start DRC or LVS by clicking on the **Run DRC** or **Run LVS** menu item respectively. The Calibre Interactive will start and the last runset file you were using will be read in. The **Import layout database from layout viewer** check box will be selected. This directs the GUI to communicate with the Skill Interface before DRC or LVS is run. The Skill Interface exports the layout cell in stream format using pipo. Before exporting, it displays a dialog that allows you to specify any layer mapping or Skill file to be used during the translation.

Socket Connections

The Calibre Skill Interface uses a TCP socket to communicate with Calibre RVE and the Calibre Interactive. When you load the calibre.skl file, the code attempts

to initialize a server socket at port 9189. If that port is being used, the code searches for a free socket port in the range 5000-9999.

If the default port number is used, the following message is displayed in the CIW transcript:

```
//  Calibre layout-server initialized successfully at socket
9189.
```

If a different port is found after a search for free ports, the following messages are displayed in the CIW transcript:

```
//  Could not initialize Calibre layout-server socket at port
9189. Trying to find free socket . . .
//  Calibre layout-server initialized successfully at socket
5000.
```

You can control the port number used by setting the environment variable MGC_CALIBRE_LAYOUT_SERVER as follows:

(Bourne Shell)

```
MGC_CALIBRE_LAYOUT_SERVER=[<host_name>:]<port_number>
export MGC_CALIBRE_LAYOUT_SERVER
```

or

(C-Shell)

```
setenv MGC_CALIBRE_LAYOUT_SERVER [<host_name>:]<port_number>
```

The <host_name> parameter is optional and is ignored by the Skill Interface (it is provided for compatibility with RVE, which also reads this environment variable). The <port_number> parameter must be a numerical value.

Skill Trigger Functions

You can now define pre- and post-Calibre Interactive execution trigger functions. The pre-execution trigger is defined by providing the Skill procedure mgc_start_calibre_trigger. The trigger function is called just before Calibre Interactive is launched. The function is called with two parameters: a string that is set to either "drc" or "lvs", and the ID of the window that Calibre Interactive was started from. The trigger function is expected to return t or nil (true or false), and execution proceeds only if t is returned by the function. Here's a sample pre-trigger function:

```
procedure( mgc_start_calibre_trigger(drc_or_lvs win)
prog( ()
    printf( "Starting %s on top cell in window %L...\n"
            upperCase(drc_or_lvs)
            win
          )
    return(t)
))
```

A similar post-execution trigger function is provided as mgc_close_calibre_trigger. It executes when the Calibre Interactive process exits. The return value of the function is not checked. Here’s a sample:

```
procedure( mgc_close_calibre_trigger(drc_or_lvs win)
prog( ()
    printf( "Closing %s on top cell in window %L...\n"
            upperCase(drc_or_lvs)
            win
          )
    return(t)
))
```

Using RVE with Virtuoso

You can start RVE from the **Start RVE** menu item in the Calibre menu in layout windows in Virtuoso. Alternatively, you can also start it from the command line.

- **Startup**—To tell RVE to use Virtuoso as the layout viewer, choose the **Setup** > **Layout** menu item in RVE. Select Cadence Virtuoso as the layout

viewer. In that dialog, you can also specify the name of host that Virtuoso is running on, as well as the socket number that the Calibre Skill Interface has initialized. (This socket number is reported in the CIW transcript when the Skill Interface is loaded. You can also get the socket number by clicking on the **Set RVE Socket...** menu item in the Calibre menu in layout windows).

RVE saves these settings in a .rvedb file in your home directory and restores these settings the next time you run RVE.

If the MGC_CALIBRE_LAYOUT_SERVER environment variable is set when RVE is started, RVE will use the host name and socket port numbers specified by the variable.

If started from the **Start RVE** menu item in the Calibre menu, RVE will run on the same host as Virtuoso. The Skill Interface ensures that RVE will be able to communicate with itself (the Skill Interface) by setting the MGC_CALIBRE_LAYOUT_SERVER environment variable before starting RVE.

- **Highlighting**—RVE allows you highlight DRC errors from DRC databases, and nets, instances, and devices from LVS databases, in Virtuoso. RVE packages the highlight data in a file (go to **Setup > Layout > Send highlight data in file** dialog item to specify the file name) and sends a message to the Skill Interface over the socket directing it to read the file. The Skill Interface interprets the data from the file and accomplishes the highlighting.

  The RVE Highlight Layers dialog (**Calibre > Setup > RVE Highlight Layers...**) allows you to choose which layers RVE uses to do highlighting. The default is layers y0-y9. You can also choose the nine "hilite" layers, the warning marker layer, or the error marker layer. You can set the MGC_RVE_USE_MARKERS (deprecated) environment or Skill variable to direct the use of the (“warning” or “error”) marker layer. The MGC_RVE_HIGHLIGHT_LAYERS variable (either specified as an environment or Skill variable), can also be used to choose from among any of these four choices of highlight layers. The valid values for MGC_RVE_HIGHLIGHT_LAYERS are "y0", "hilite", "warning", and "error". This variable overrides MGC_RVE_USE_MARKERS if both are specified.

If a hierarchical net or device is highlighted, the schematic corresponding to the containing cell is automatically opened and zoomed to the net or device. For example, if you highlight net "X0/net1" in cell "XOR", and cell "XOR" is open in window 1, RVE will descend into the cell corresponding to X0, highlight net "net1" and zoom to it. If a subsequent highlight request is for net "net2" in cell "XOR", RVE will pop back up to the "XOR" cell in window 1 and highlight and zoom to the net. LVS-RVE displays a message in the CIW if a schematic object cannot be successfully highlighted.

When RVE requests highlighting in a particular cell, the Skill Interface checks to see if the cell is already open in a window. If not, it checks the colon-delimited cell libraries path specified in the Set RVE Library dialog (choose the **Set RVE Library** menu item from the Calibre menu) for the cell. If it cannot find the cell in the path, it prompts you with the cell open dialog. When you choose a cell from that dialog from a library not on the path, that library is automatically added to the cell libraries path. One consequence of this is that if a cell is in two libraries, the cell from the first library in the path will always be picked. If you wish to switch designs in the midst of an RVE session, you may want to clear or reset the RVE libraries path explicitly to avoid the Skill Interface picking a cell from the wrong library.

RVE does not open layout and schematic windows on highlight, unless the environment (or Skill) variable MGC_RVE_RAISE_WINDOW is set to any value.

- **Case sensitivity**—Case sensitivity in layout export can be set through the Skill variable streamOutKeys->caseSensitivity. RVE will search for highlight cells in a case insensitive fashion if this setting is insensitive, or if the environment or Skill variable MGC_RVE_LAYOUT_CASE_FOLD is defined. Source cell searching can also be made case insensitive by defining the environment or Skill variable MGC_RVE_SOURCE_CASE_FOLD.

- **Working directory**—If you wish to direct RVE to always start in a particular directory, you can set the environment variable MGC_CALIBRE_DB_DIR to that directory. When RVE starts, it will change its working directory to the one specified in the variable.

Variables Summary

Table 3-1. Calibre Interactive Variables Summary

Variable	Type	Description
MGC_CALIBRE_DB_DIR	env	sets RVE working directory (page 3-32)
MGC_CALIBRE_DRC_RUNSET_FILE	env	sets DRC runset (page 3-3)
MGC_CALIBRE_LVS_RUNSET_FILE	env	sets LVS runset (page 3-3)
MGC_CALIBRE_LAYOUT_SERVER	env	sets communications socket number (page 3-10)
MGC_RVE_INIT_SOCKET_AT_ STARTUP	env	sets socket initiation preference for Virtuoso (page 3-25)
MGC_RVE_USE_MARKERS (deprecated)	env or Skill	sets the use of warning or error layers by RVE in Virtuoso (page 3-30)
MGC_RVE_HIGHLIGHT_LAYERS	env or Skill	sets the use of warning or error layers by RVE in Virtuoso (page 3-30)
MGC_RVE_RAISE_WINDOW	env or Skill	sets the behavior of RVE layout and schematic windows by Virtuoso (page 3-30)
MGC_RVE_LAYOUT_CASE_FOLD	env or Skill	sets the case sensitivity behavior of highlight cells by RVE in Virtuoso (page 3-30)

Table 3-1. Calibre Interactive Variables Summary

Variable	Type	Description
MGC_RVE_SOURCE_CASE_FOLD	env or Skill	sets the case sensitivity behavior of highlight cells by RVE in Virtuoso (page 3-30)
mgc_calibre_export_layout_template_file	Skill	sets layout template file location in Virtuoso (page 3-25)
mgc_calibre_export_netlist_template_file	Skill	sets netlist template file location in Virtuoso (page 3-27)
mgcCalibreMenuViewType	Skill	specifies window type in which to load a Calibre menu (page 3-23)
CDS_Netlisting_Mode	env or Skill	sets netlisting export mode in Virtuoso (page 3-27)

Chapter 4
DRC Concepts

Calibre DRC/DRC-H is an edge-based design rule checking system. It works primarily with the edges of polygons rather than with the regions themselves. Basic geometric data types are polygons and edges of polygons. Edges always have a reference back to the polygon to which they belong. Polygons and edges can also have a reference to an electrical node, as long as you have established the appropriate circuit connectivity.

Figure 4-1 shows the edge-polygon relationship: every polygon edge has an exterior side and an interior side. An edge's side depends upon which side of the edge borders the exterior of the polygon and which side borders the interior of the polygon. In this manual, all figures show the interior side of an edge as the shaded side.

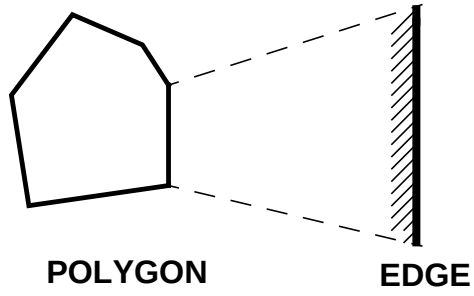


Figure 4-1. Edge-polygon Relationship

Layers

Calibre DRC works on edges located on layers that the IC designer creates. This section discusses the layers that Calibre DRC recognizes and creates, and the types of operations it performs on the layers.

Layer Types

A rule file creates or uses data from four types of layers (see Figure 4-2):

- Original layers
- Derived polygon layers
- Derived edge layers
- Derived error layers

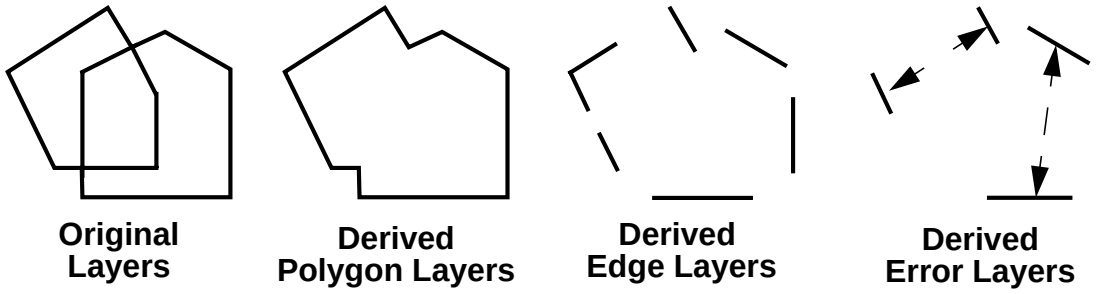


Figure 4-2. Layer Types

Original Layers

Original layers (or drawn layers) are layers that represent original layout data. In a rule file, they are referred to by their name or number. The verification system also allows you to use layer sets. In this case, the layer set behaves exactly as if it were a simple layer with all geometries on the constituent layers combined.

In most layer operations, the DRC system automatically merges the polygon data on an original layer before using that layer. In a merged-data representation, the DRC system merges any polygons that overlap or share an edge into one polygon. The primary exceptions are the one-layer Boolean operations that operate on unmerged original layers. Merged data is normally a more accurate depiction of the true mask than unmerged data.

Derived Polygon Layers

Derived polygon layers represent merged polygons generated as the output of layer operations, such as Boolean functions, area functions, and polygon-directed dimensional check operations. Derived polygon layers can also be derived error layers.

Derived Edge Layers

Derived edge layers represent edges or sub-edges of merged polygons generated as the output of layer operations, such as topological edge operations, and edge-directed dimensional check operations. Derived edge layers can also be derived error layers.

Derived Error Layers

Derived error layers represent clusters of one, two, three, or four edges. Primarily, they hold output of error-directed dimensional check operations for instantiation into the DRC results database. Currently, you cannot use them as input to any operation in the verification system.

Layer Type Summary

You can assign derived polygon and derived edge layers a name to use in another operation. You can also place derived layers into the DRC results database as the output of design rule checks, where the objects on the derived layers become individual results in the database. Any of the three derived layer types can represent output from the DRC system, although only the third type is actually called an “error layer.”

Figure 4-3 shows the global flow of control through the DRC system, in terms of the layer types supported by the system.

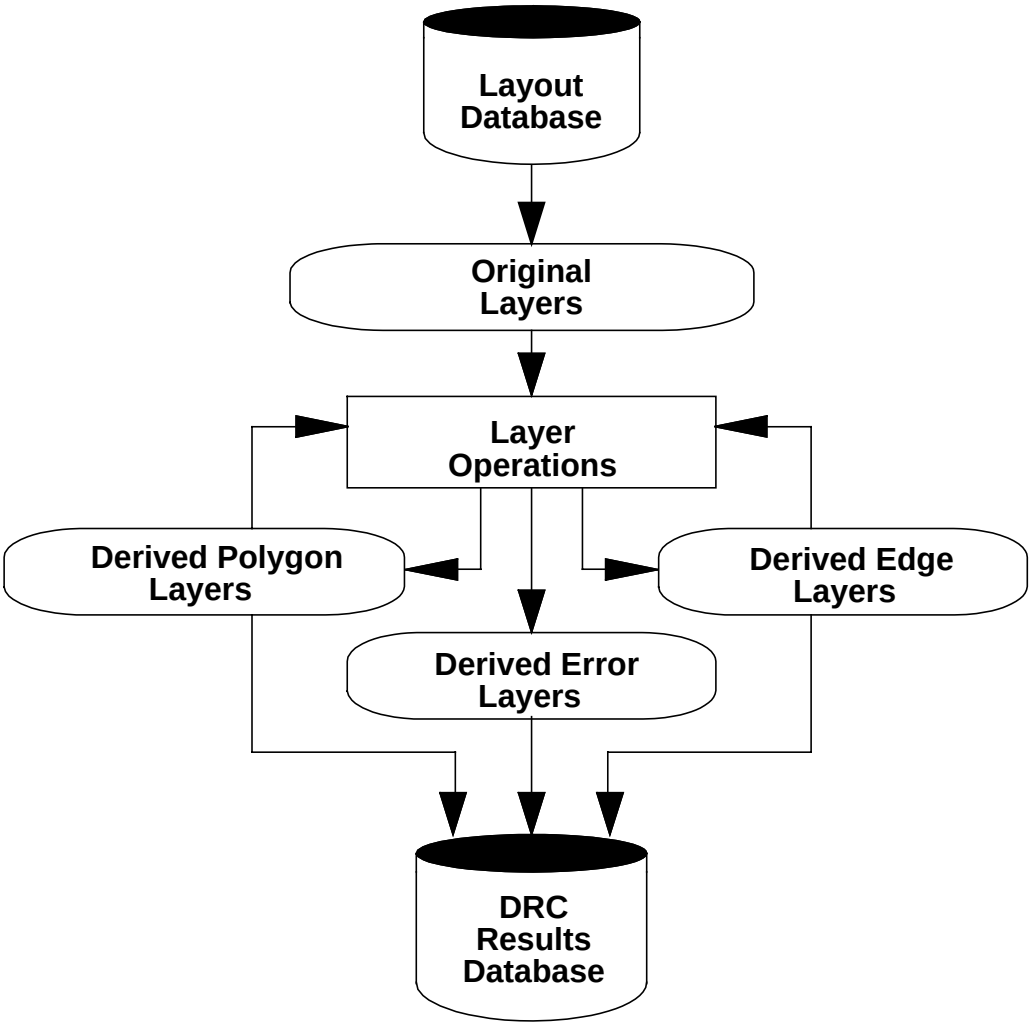


Figure 4-3. Layer Types and Data Flow in the DRC System

Layer Operations

The basic unit within a rule file is the layer operation. A layer operation is any measurement function that creates a derived layer from input consisting of original layers or other derived layers. There are two types of layer operations:

- **Dimensional Check Operations**. Dimensional check operations provide the core design-rule checking capability of the DRC system.

- **Auxiliary Operations**. Any layer operation that is not a dimensional check is an auxiliary operation.

Both types of operations create derived error layers (error-directed layer operations), derived edge layers (edge-directed layer operations), or derived polygon layers (polygon-directed layer operations).

Because no operation currently accepts derived error layers as input, you can use error-directed layer operations only to send results to the DRC results database. You can use the derived layers created by polygon-directed and edge-directed layer operations as input to other operations or as output to the DRC results database. For more information, refer to DRC Results Database in chapter 14.

Layer Definitions

Layer definitions assign names to derived layers. When a layer definition names a derived layer, you can use the derived layer in another rule file operation by referencing its name. A layer definition has the form:

layer_name = layer_operation

The layer operation creates the derived layer and gives it the name *layer_name*.

A layer definition can exist either inside or outside of a rule check statement. A layer definition that is outside a rule check statement is considered global, which you can use in operations anywhere in the rule file. A layer definition that is inside of a rule check statement is local to that statement. A local definition is not available outside the rule check statement.

You can reuse the same local layer names in different rule check statements; however, you cannot use the same local layer name twice within the same rule check statement. You cannot use global layer names twice in one rule file. Any local layer definition supersedes a global definition of the same name within the rule check statement where the global definition appears.

Here is a sample rule file except illustrating layer definitions:

```
// This section defines diffusion regions, transistor gates,
// and source/drain regions.
n_diff = diffusion NOT p_dope      //n+ diffusion
```

```
p_diff = diffusion AND p_dope     //p+ diffusion
n_tap = n_diff NOT OUTSIDE n_well //n-tap areas
not_n_tap = n_diff OUTSIDE n_well //areas which are not n-taps
p_tap = p_diff OUTSIDE n_well     //p-tap areas
not_p_tap = p_diff NOT OUTSIDE n_well //areas not p-taps
n_gate = poly AND not_n_tap       //n-channel gates
p_gate = poly AND not_p_tap       //p-channel gates
nsd = not_n_tap NOT n_gate        //n-source/drain regions
psd = not_p_tap NOT p_gate        // p-source/drain regions
```

Layer operations must always be used *within* a layer definition, except when inside of a rule check statement, as discussed below; in the latter case, they may be “free-standing.”

Implicit Layer Definitions

The syntax for layer definitions is sometimes referred to as an explicit layer definition, because the specification statement explicitly defines the derived layer name. You can also define layer definitions implicitly. Implicit definition avoids having to generate explicit derived layer names and provides an expression capability for layer operations.

An implicit layer definition consists of a pair of parentheses enclosing one layer operation whose input layer(s) can also be implicit layer definitions. You can use an implicit layer definition as an input layer to any operation. Implicit layer definitions allow expression capability in layer operations and free you from having to create names for every derived layer. For example:

```
taps = ( pdiff AND ( bulk NOT nwell ) ) OR ( ndiff AND nwell )
```

In this example, an explicit layer definition defines layer `taps`. The statement implicitly defines the two input layers to the OR operation, as well as the second input layer to the AND operation. Internally, the rule file compiler “unwinds” implicit layer definitions into a sequence of explicit layer definitions.

The only exception to the use of an implicit layer definition as an operation input parameter is in edge-directed output in a dimensional check operation.

Layer Operation Classifications

Layer operations that generate derived edge or polygon layers are either layer constructors or layer selectors. Some layer operations that affect circuit connectivity are net-preserving. The following sections describe these classifications.

Layer Constructors

Operations classified as layer constructors create new polygon data. For example, a two-layer Boolean operation:

```
layer1 AND layer2
```

is a layer constructor because it creates new polygons from the polygon data on both input layers. The following layer operations are layer constructors:

AND	Grow	Ports
Density	[Not] Inside Cell	Push
Device Layer	Internal (polygon-directed check)	Rectangles
Enclosure (polygon-directed check)	Litho	Rectangle Enclosure
Expand Edge	Magnify	Rotate
Expand Text	Merge	Size
Extent	Net Area Ratio [Print]	Shift
Extent Cell	NOT	Shrink
Extents	OR	Snap
External (polygon-directed check)	Ornet	Stamp
Flatten	Pathchk	Topex

Holes	Pins	XOR
Expand Edge	Polynet	

Layer Selectors

Operations classified as layer selectors select existing polygon or edge data from the appropriate input layer. For example, the Coincident Edge operation is a layer selector because it selects edges or edge segments from the first input layer that are coincident with edges from the second input layer. The following layer operations are layer selectors:

[Not] Angle	[Not] Area
[Not] Coincident Edge	[Not] Coincident Inside Edge
[Not] Coincident Outside Edge	Convex Edge
Copy	[Not] Cut
[Not] Donut	Drawn Acute
Drawn Offgrid	Drawn Skew
[Not] Enclose	Enclosure (edge-directed dimensional check)
[Not] Enclose Rectangle	External (edge-directed dimensional check)
[Not] Inside	[Not] Inside Edge
[Not] Interact	Internal (edge-directed dimensional check)
[Not] Length	[Not] Net
Net Area	Offgrid
[Not] Outside	[Not] Outside Edge
Path Length	Perimeter
[Not] Rectangle	[Not] Touch
[Not] Touch Edge	[Not] Touch Inside Edge

[Not] Touch Outside Edge	Vertex
[Not] With Edge	With Text

Layer selector operations select data from the appropriate input layer.

Net-preserving Operations

A net-preserving operation passes connectivity from an input layer through the operation to the derived layer. All layer selector operations are net-preserving because they distinctly select polygon or edge data from a single input layer. Calibre DRC passes connectivity information to the derived layer, if the data contains connectivity information.

The Ornet, two-layer AND, and the two-layer NOT operations (layer constructors) are also net-preserving. The Ornet operation passes connectivity information from both input layers through the operation to the derived layer. The AND and NOT operations pass connectivity from the first input layer through the operation to the derived layer.

Note that net-preserving operations, with the possible exception of edge-directed dimensional check operations, pass connectivity from the first input layer (when two of them are used) to the derived layer.

The order of the input layers affects connectivity information. In the Coincident Edge operation shown in Figure 4-4, connectivity passes from the layer1 polygons (or edges) to the derived layer. When the operation is written:

```
x = COINCIDENT EDGE layer2 layer1
```

the connectivity passes from the layer2 polygon (or edges) to the derived layer `x`. Reversing the layer order does not produce different geometric output. Although both definitions would generate the same geometric edge data, the connectivity attached to the output is not likely the same between the two operations. Connectivity is dependent on the “layer of origin”, which is discussed next.

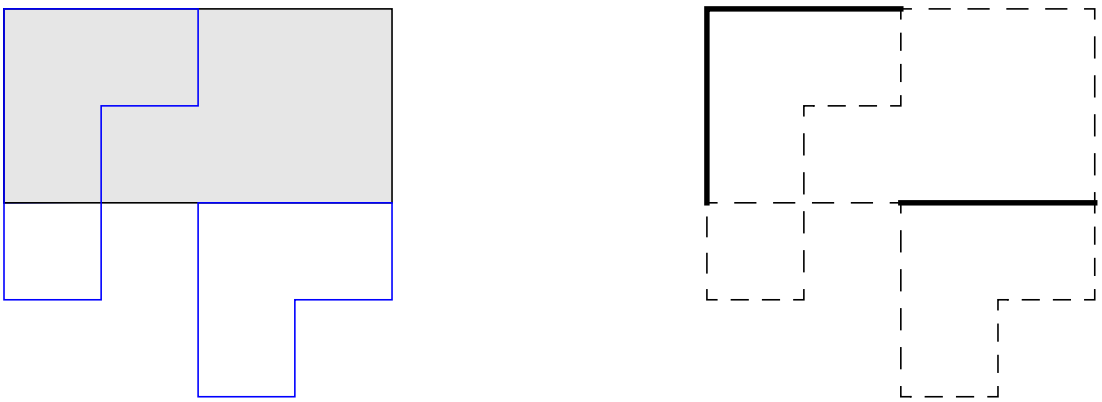


Figure 4-4. Coincident Edge Operation

Layer of Origin

This section describes the concept of a *layer of origin*, which is important for polygon- or edge-directed dimensional check operations. It is also important for net-preserving operations.

Layer operations are either layer constructors or layer selectors. Layer constructors actually create new polygon data. For a listing of these operations, refer to the section "Layer Constructors" and "Layer Selectors" above.

Layer constructor operations generate new polygon data, whereas layer selector operations simply select existing data from the appropriate input layer. For example, polygon-directed dimensional check operations are layer constructors and the edge-directed dimensional check operations are layer selectors.

Note

The layer of origin concept is not applicable to error-directed layer operations, because you cannot currently use their output in other operations.

Given the name X of a layer definition, an arbitrarily long sequence of operations may have produced it. The layer of origin of X is the last layer produced by a layer

constructor from which data within X were derived. Recall that Calibre merges all original layers prior to using them in any layer operation. Because the merge operation is equivalent to the one-layer polygon Boolean OR operation, the layer of origin of every original layer is itself. Also, if no layer constructor operations existed in the layer definition chain for X, then the layer of origin of X is the original layer from which it was initially derived.

For example, consider these layer definitions:

```
X = AND metal contact
Y = AREA X < 4
Z = RECTANGLE Y
```

The layer-of-origin for layer Z (and layer Y) is layer X because it was the last layer that a layer constructor operation generated in the layer definition chain.

An important point regarding layer selector operations: these operations select data from the *appropriate* input layer. It may appear, for example, that the layer definitions

```
X = A coincident edge B
X = B coincident edge A
```

produce the same data in the layer X. Another way of saying this is that it may appear that the Coincident Edge operation is commutative. Although the operation generates the same geometric edge data in both cases, it is also true that polygon (and node) references are passing through the selector operation. Hence, identical geometric data selected from A and B will not, in general, be the same data, because polygon and node references will likely differ. This has important implications for dimensional check operations. For example, the following Internal dimensional check sequence:

```
X = metal coincident edge poly
internal X metal < 3
```

has an entirely different definition from this sequence:

```
X = poly coincident edge metal
internal X metal < 3
```

In the first case, layer X contains geometric data carrying polygon number information from layer `metal`. In the second case, this data carries polygon number information from layer `poly`. That is, in the first case, the layer of origin of the input layers to the Internal operation is the same while in the second case, the input layers have different layers of origin. Thus, the definition of Internal is sensitive to layer of origin.

Rule Check Statements

Rule check statements are specific to Calibre DRC applications (as well as any other Mentor Graphics tool that uses SVRF rule files). These statements specify layer operations within the rule file that instantiate the resulting derived layers into the DRC results database. Refer to Chapter 14, Results, for information on the DRC results database.

Rule check statements are active entities of the system, whereas layer definitions are passive. The output from a rule check statement can consist of derived polygon layers, derived edge layers, or derived error layers, or combinations of the three. A rule check must have output to the DRC results database (that is, it must have at least one standalone layer operation) or it will not compile.

Rule check statements take the following form:

```
name {
   layer_definition | layer_operation
   …
   layer_definition | layer_operation
   }
```

where `name` is the name of the rule check, and each line consists of a layer definition or a layer operation not within a layer definition. Rule check names must be unique. Also note a rule check must occur between curly braces. Oftentimes the braces are omitted in this manual when discussing operations, but you must include the braces in your rule file or it will not compile.

When a Calibre DRC application executes a rule check statement, it places the derived layers in the DRC results database (see Figure 4-3). All derived layers created by all layer operations (and not in layer definitions) within the rule check

statement are placed in the DRC results database. For example, the following rule check statement generates layer definitions and derived error layers:

```
METAL_WIDTH {
 // Metal width check. Metal width must be greater than or
 // equal to 3 microns except where metal length exceeds 5
 // microns; in that case, metal width must be greater than or
 // equal to 4 microns.

long_metal = metal LENGTH > 5     // Layer definition;
                                  // not output to results db
INTERNAL long_metal < 4           // Output to results db
short_metal = metal NOT LENGTH > 5 //Layer definition
INTERNAL short_metal < 3           //Output to results db
}
```

The layer operation `metal LENGTH > 5` defines layer `long_metal`; this layer is then used within the next operation. There is no output to the DRC results database corresponding to the `metal LENGTH > 5` operation, because it is part of a layer definition. Similarly, the third operation is a layer definition. The second and fourth operations are not layer definitions; these operations generate output to the DRC results database under the name `METAL_WIDTH`.

If you wish to see the results of intermediate layer definitions, such as for `long_metal` in the above example, use the Copy operation. For example, inserting

```
copy long_metal
```

into the `METAL_WIDTH` rule check would copy the `long_metal` layer to the DRC results database as an “error layer.” You could then see what `long_metal` will look like. Copy is a very useful debugging tool used in this way. Be sure to comment out any such debugging statements when you finish with them.

Rule Check Comments

You can use two types of comments in a rule check statement. The first is the standard comment, denoted with a double-slash (//). The second is a rule check comment, denoted with an at (@).

The // characters begin a comment that terminates at the end of the line on which they occur. These comments serve only to annotate the rule file.

The @ character begins a rule check comment. All characters from the @ to the end of the line are part of a rule check comment. A rule check comment can appear only within the braces ({ }) that delimit a rule check statement. When a Calibre DRC application executes the rule check, it places all rule check comments within the rule check statement into the DRC results database, along with the output data for the rule check. For example:

```
METAL_SPACING {
  @ Metal to metal spacing errors.
  @ Do not confuse with notch errors;
  @ the spacing rule is 4 microns.
  metal EXTERNAL < 4 space // Check spacing between different
                           // polygons.
   }
METAL_NOTCH {
  @ Metal notch errors.
  @ Do not confuse with spacing errors;
  @ the notch rule is 3.5 microns.
  metal EXTERNAL < 3.5 notch   // Check spacing in the same
                               // polygon.
   }
```

Control of Empty Rule Checks

It is typical in DRC execution that many rule checks do not generate any DRC results (empty rule checks). You may not want these rule checks to take up space in the DRC results database. In some cases, it is the rule check execution that is important, whether or not there were actually results.

Empty Rule Check Suppression in Calibre DRC

In Calibre DRC, suppression of empty rule check instantiation is controlled by the rule file DRC Keep Empty specification statement. If NO is specified, empty rule checks will not be instantiated into the DRC results database. If YES is specified, or the statement is not specified at all, then empty rule checks will be instantiated.

Check Text

Each rule check in the DRC results database stores information in addition to the actual results. This information includes the time and date of its previous execution, and can also consist of text mapped by Calibre DRC applications. This text is known as *check text*. Check text can include either the rule check comment or the entire ASCII text of the rule check statement. Check text can also include the pathname and title of the rule file that last ran the rule check. The DRC Check Text specification statement controls mapping of check text to the DRC results database in Calibre DRC. The default is to map the rule file pathname and title and rule check comments.

Check Text in Calibre DRC

Mapping of check text to the DRC results database in Calibre DRC is controlled by the rule file DRC Check Text specification statement. If NONE is specified, then no check text is mapped. If RFI is specified, then the rule file pathname and title, if present, are mapped. If COMMENTS is specified, then rule check comments are mapped. If ALL is specified, then the entire ASCII text of each rule check is mapped. The default, if not specified, is to map the rule file pathname and title and rule check comments.

DRC Rule Check Result Limits

You can limit the number of DRC results written to the DRC results database for any given DRC rule check by using the DRC Maximum Results specification statement, or the MAXIMUM RESULTS parameter to the DRC Check Map specification statement. This capability can help you avoid the generation of large DRC results databases due, for example, to specifying a large DRC rule value. By default, this maximum result limit, per DRC rule check, is 1000.

Calibre issues a warning whenever the process of writing results for a DRC rule check is limited by reaching the DRC maximum result limit.

As a performance optimization in flat Calibre DRC, this maximum result limit is internally passed to the lowest level utilities which implement the DRC operations External, Internal, and Enclosure. These utilities cease the generation of DRC results accordingly when the results are only to be written to the DRC results

database (not used in a conjunctive setting), and there the tool is certain that the maximum result limit will be reached.

This optimization, referred to as short-circuiting, can save CPU time and memory by not generating and storing results which would later, when writing the DRC results database, be discarded due to reaching the maximum result limit.

For hierarchical Calibre DRC, this performance optimization behaves similarly, except that it cannot determine, with certainty, that the maximum result limit will be reached.

Whenever short-circuiting occurs in the hierarchical External, Internal, and Enclosure operations, Calibre issues the following warning:

```
Output operation <name> abbreviated due to high probability of
exceeding DRC maximum result limit.
```

where <name> is the name of the short-circuited operation. In the event that the maximum result count is not exceeded, for the same operation, it is a result of the short-circuit process halting the process prematurely, this too results in a warning.

Dimensional Check Operations

Dimensional check operations represent the core design rule checking capability of the DRC system. These operations generate derived error layers, derived edge layers, or derived polygon layers by measuring the separation of edges on one or two input layers.

- **Error-directed Dimensional Check Operations**. Error-directed dimensional check operations generate derived error layers consisting primarily of edge pairs whose members mutually meet the constraint of the operation. You can use error-directed dimensional check operations only to generate output within rule check statements; they do not define layer definitions because you cannot provide derived error layers as input to any DRC operation.

- **Edge-directed Dimensional Check Operations**. Edge-directed dimensional check operations generate derived edge layers by creating non-

clustered edge output from an individual input layer. You can use edge-directed dimensional check operations to create layer definitions. This is done using the edge-directed output operators “[ ]” and “( )”.

- **Polygon-directed Dimensional Check Operations**. Polygon-directed dimensional check operations generate derived polygon layers by forming polygons represented by the outline of edge clusters, which their error-directed counterparts would provide as output. You can use polygon-directed dimensional check operations to create layer definitions. This is done using the REGION keyword.

There are three primary dimensional check operations: Enclosure, External, and Internal. The Enclosure operation is for enclosure checks or extension checks. The External operation is for spacing checks. The Internal operation is for width checks or overlap checks.

Figure 4-5 illustrates the edges that the dimensional check operations measure. Note that full rule check syntax is not used in these examples.

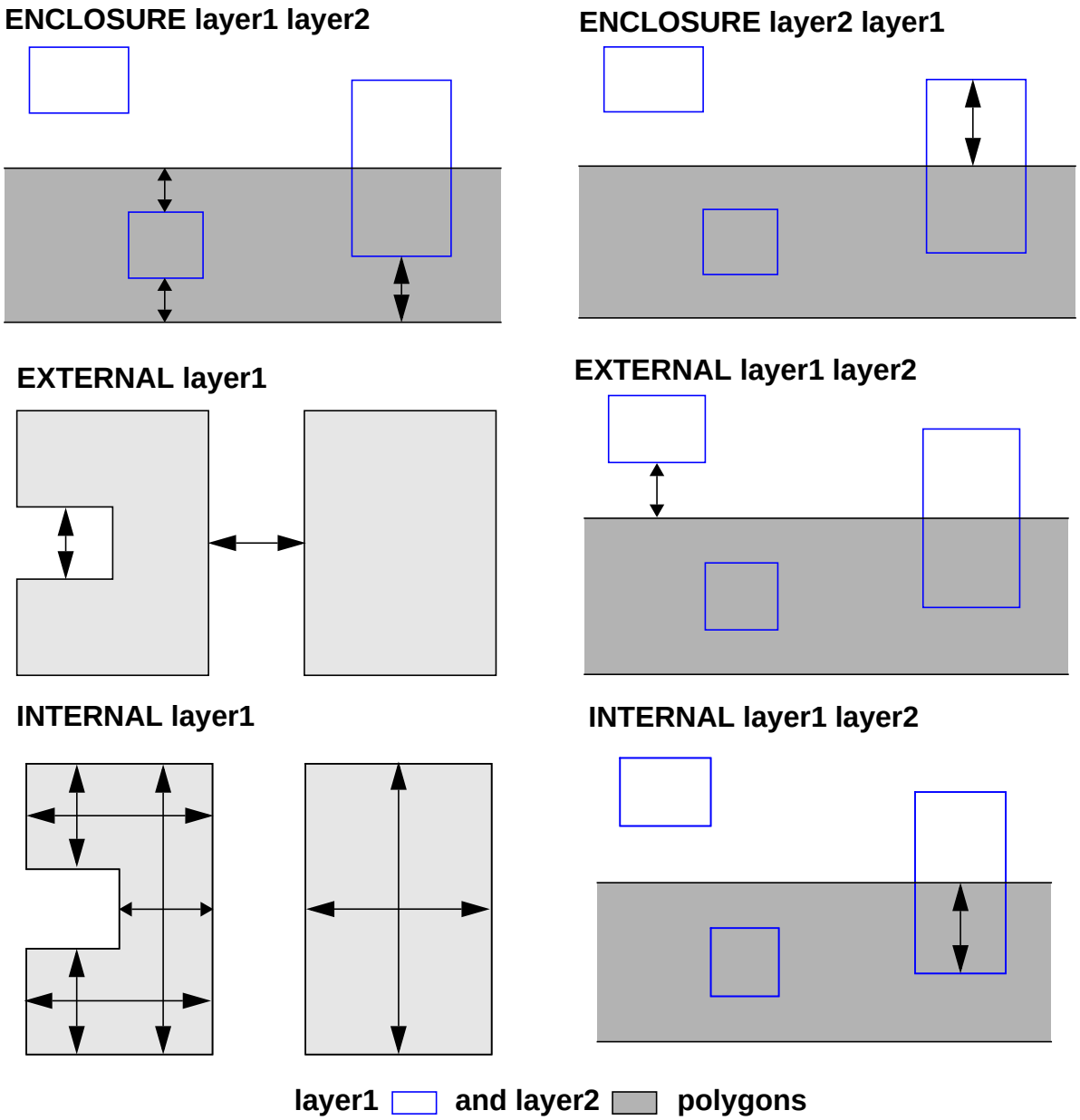


Figure 4-5. Measured Edges in the Dimensional Check Operations

Secondary Keywords

The dimensional check operations, as well as many other statements in the SVRF rule file language, have optional secondary keywords associated with them. These secondary keywords affect the behavior of the operator to which they are assigned. There are many such keywords and they are discussed in detail in the *SVRF Manual*. A number of these secondary keywords are covered in the following sections.

Edge Measurement

The dimensional check operations measure the separation between the insides and outsides of edges, but only if the edges conform to the criteria of the operation. This section describes the criteria of the dimensional check operations and the edge-measurement method that they use.

The edge pairs of dimensional check operations consist only of those portions of measured edges that conform to the measurement constraint. This edge-measurement method proceeds as follows (refer to Figure 4-6).

Assume that the measurement is from the outside of edge A to the outside of edge B and that the operation specifies that edges closer than 3 microns are output.

Using the Euclidean metric (refer to "Metrics" on page 4-21), the operation measures edges A and B by constructing two regions. One region consists of all points in the half-plane on the outside of edge A that are within 3 microns (micron is the default unit of measurement, but you can specify other units) of edge A; a similar region consists of all such points around edge B.

The output is an edge pair consisting of the sub-edge of A that intersects the region around edge B and the sub-edge of B that intersects the region around edge A. The operation provides output of only those portions of the edges that actually conform to the constraint of the dimensional check operation.

This example uses a constraint of 3 microns, but any other type of legal constraint or value will work in a similar fashion.

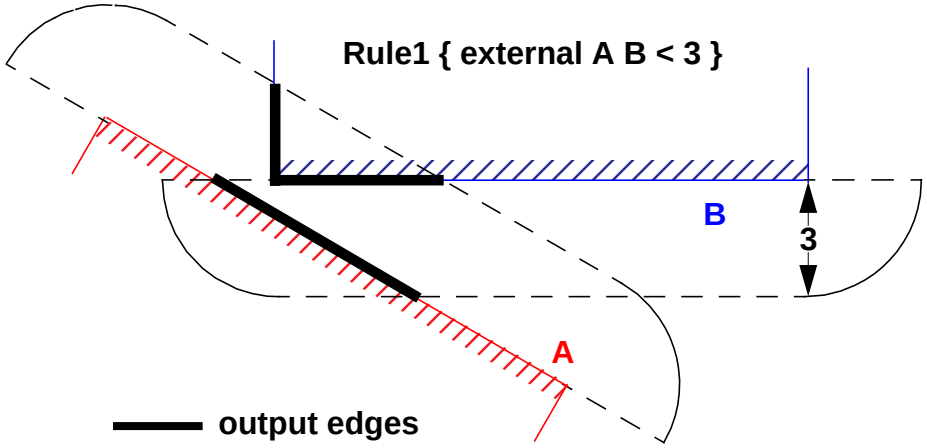


Figure 4-6. Generation of Output Edges

Measurement Region Construction

To construct the region about an edge, a boundary forms on either the outside or inside of the edge, as specified by the operation. This area is referred to as the half-plane. The radius of this boundary is the numeric value within the operation's constraint. The boundary consists of all points in the half-plane whose distance from the edge is the numeric value of the constraint. If the constraint specifies two numbers (an interval), then two such boundaries, each corresponding to one of the numbers, are constructed. Given this boundary, a region is forms that consists of all points in the half-plane in which the boundary was constructed and according to the operation's constraint as follows:

- If the constraint evaluates to the form x < a, then the region consists of all points strictly within the boundary.
- If the constraint evaluates to the form x <= a, then the region consists of all points within and including the boundary.
- If the constraint evaluates to the form x = a, then the region consists of the boundary alone.

- If the constraint evaluates to the form a < x < b, then the region consists of all points strictly outside of the boundary with radius a and strictly inside the boundary with radius b. The other three valid forms of the constraint, a <= x < b, a < x <= b, and a <= x <= b include the boundaries having radii a and b.

Figure 4-7 shows region construction about the outside of edge A. The first region assumes the constraint x < a and the second region assumes the constraint a < x <= b. Note that no points on the line containing edge A are considered to be within the region.

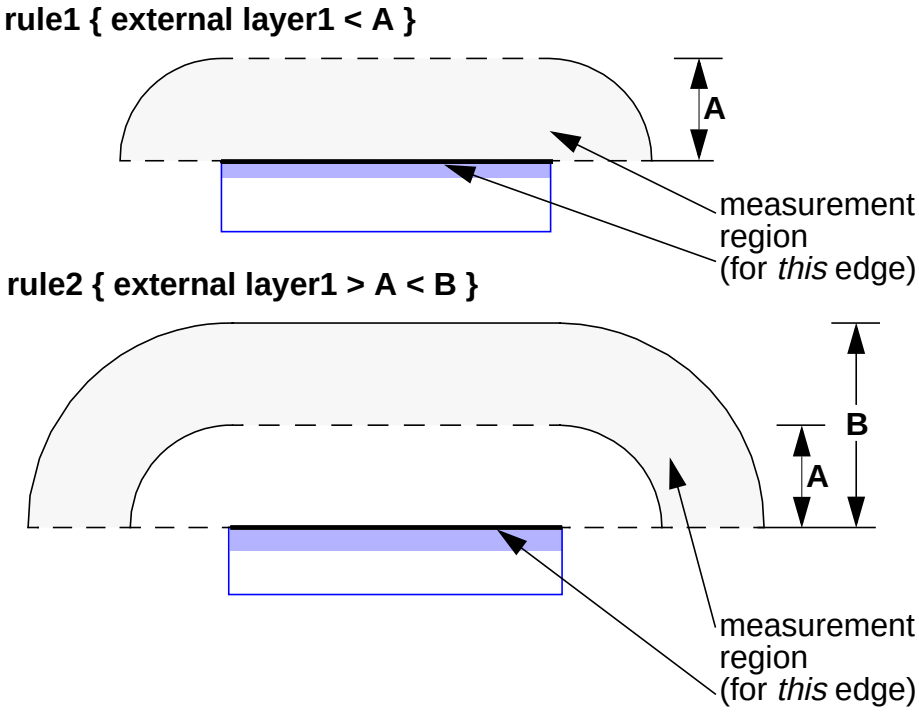


Figure 4-7. Measurement Region Formation

Metrics

There are five forms of metrics you can use with dimensional check operations:

- **Euclidean**. The Euclidean metric forms a region with rounded boundaries at that extend past the corners of the selected edges. This is the default metric.

- **Square**. The Square metric forms a region with right-angle boundaries that extend past the corners of the selected edges.
- **Opposite**. The opposite metric forms a region with right-angle boundaries that do not extend past the corners of the selected edges.
- **Opposite Extended**. The opposite extended metric forms a region with right-angle boundaries that can extend past the corners of the selected edges, dependent upon a value you specify. This metric allows non-commutative measurements to produce output.
- **Opposite Symmetric**. The opposite symmetric metric uses the opposite metric for measurement with post-processing for use with non-parallel edges.

The metrics determine only how the boundaries about the edges are constructed; all other elements of the measurement and output method are unchanged. They are implemented as secondary keywords to the dimensional check operations.

Figure 4-8 shows measurement region construction for four main metric types.

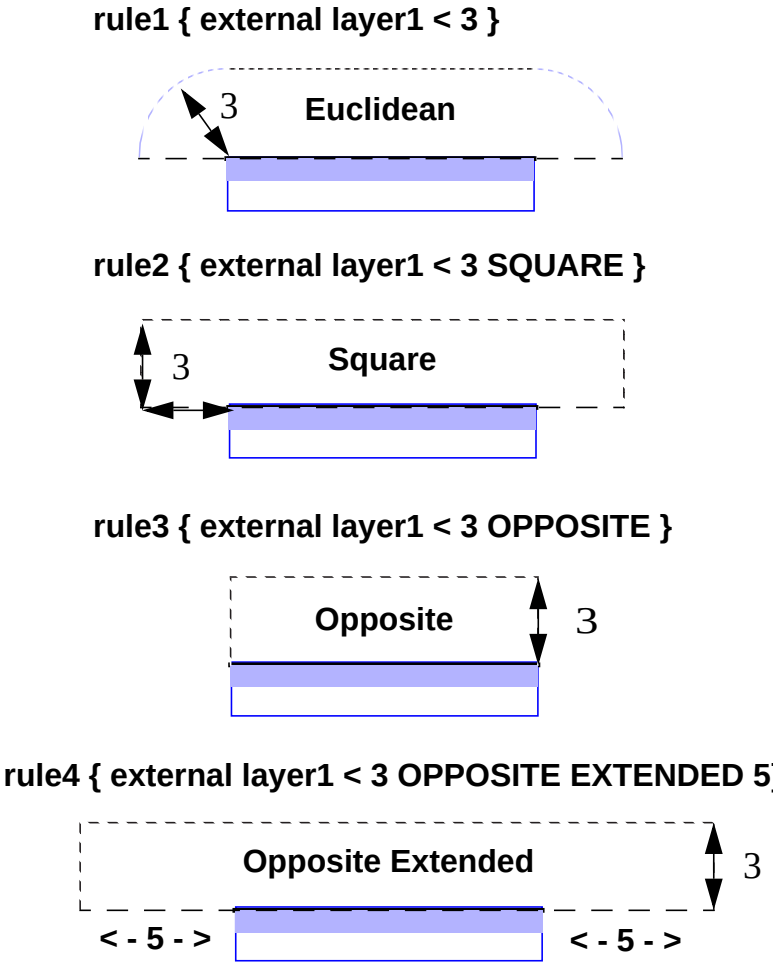


Figure 4-8. Metric Determination of Boundary Formation

The Opposite Symmetric metric uses the Opposite metric for measurement, along with post-processing of the output to achieve better symmetry for non-parallel edges. The following algorithm is used for the opposite symmetric metric, given edges A and B, and Figure 4-9 illustrates the behavior:

1. Apply only the Opposite metric if A and B are parallel, perpendicular, or intersecting.

2. Measure A and B by using the Opposite metric; however do not allow the special treatment for only one non-orthogonal edge, as is done with the Opposite metric, and do not discard the output if the measurement was non-

commutative, as is done with the Opposite metric. This step can result in zero, one, or two outputs from each edge.

3. Discard all trivial output edges. Due to properties within the Opposite metric, there can remain at most one output edge from each input edge.

4. Quit with no output if, after discarding trivial edges, there is no output from A and no output from B. Otherwise, name the resulting output edges OA and OB. Note that either OA or OB may be non-existent.

5. Project edge OA, if it exists, onto B, which forms a subedge PB. Project edge OB, if it exists, onto A, which forms a subedge PA. Discard either PA or PB if they are a result of round-off error.

6. Produce the output of edge A, which is OA+PA. Produce the output from edge B, which is OB+PB.

7. Produce no output for edge A if both OA and PA are non-existent. Produce no output for edge B if both OB and PB are non-existent.

rule5 {external layer1 < 5 OPPOSITE SYMMETRIC }

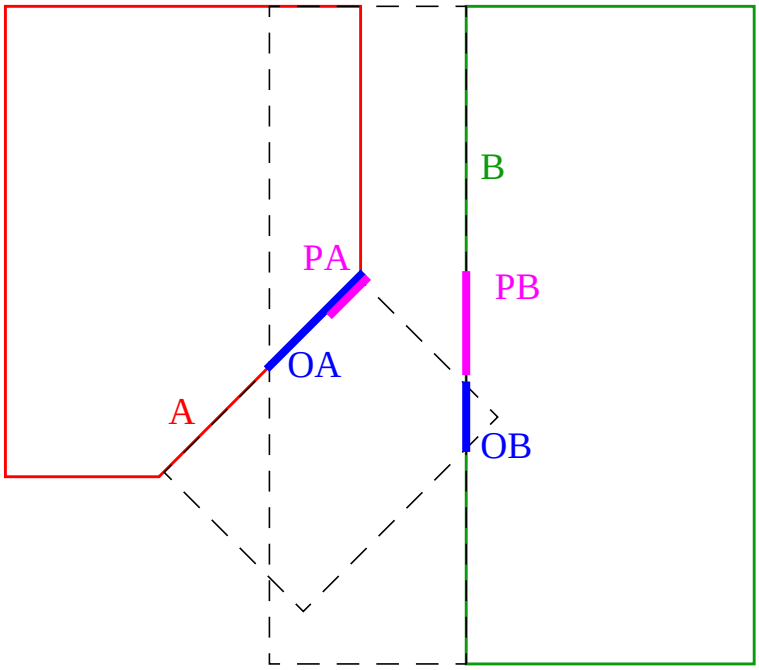


Figure 4-9. Opposite Symmetric Example

The Opposite metrics (Opposite, Opposite Extended, and Opposite Symmetric) treat the constraint a <= x < b as a < x < b, and the constraint a <= x <= b as a < x <=b, unless there is infinite intersection with the top portion of the inner curve of the design rule boundary.

Edge Cluster Generation

The edge measurement methods described above generate edge clusters. These are groups of edges or edge segments that are output by a rule check. Several different types of clusters can be formed. Two-edge clusters are probably the most intuitive and are quite common. However, certain orientations of edges can create a three-edge cluster instead of a two-edge cluster. This primarily occurs because of the side boundary effects of constraints that have two measurement boundaries (for example, “>= 1 < 3”). Figure 4-10 shows the generation of a three-edge output cluster.

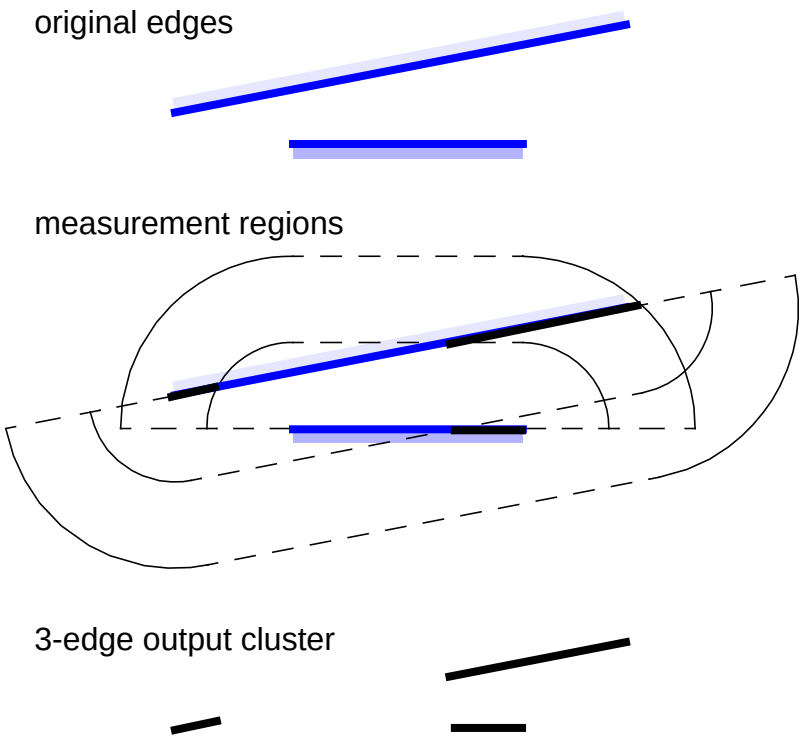


Figure 4-10. Three-Edge Output Cluster

Trivial Edges

The edge measurement regions can create a *trivial edge*, which is an edge consisting of only one point. This is the result of measuring two edges when the constraint specifies that the boundary is to be included in the region (for example, <= a), and one of the edges intersects the region around the other edge at only a single point on the boundary.

The verification system can also force the creation of trivial edges within the measurement method as follows: For two edges, A and B, it is possible that edge A intersects the region about edge B, but edge B does not intersect the region about edge A. This is because in general, the intersection of the measurement regions is not commutative. This is especially true for the non-Euclidean metrics. Therefore, it seems that output from only one edge is required. However, if output is to a derived error layer for results presentation, it is not helpful to have an error consisting of one edge. In this case, the verification system outputs a trivial edge from edge B to represent it. The trivial edge consists of the point on edge B that is

closest to the output edge from edge A and is also on the appropriate side of edge A.

For example, Figure 4-11 shows the generation of a trivial edge. This example uses the square metric for measurement region construction and indicates the trivial edge with an X.

Trivial edges cannot appear on derived edge layers because they are physically insignificant and the primary use of derived edge layers is in conjunctive design rule checking. Therefore, an edge-directed dimensional check operation never generates a trivial edge; it discards trivial edges as if they never occurred.

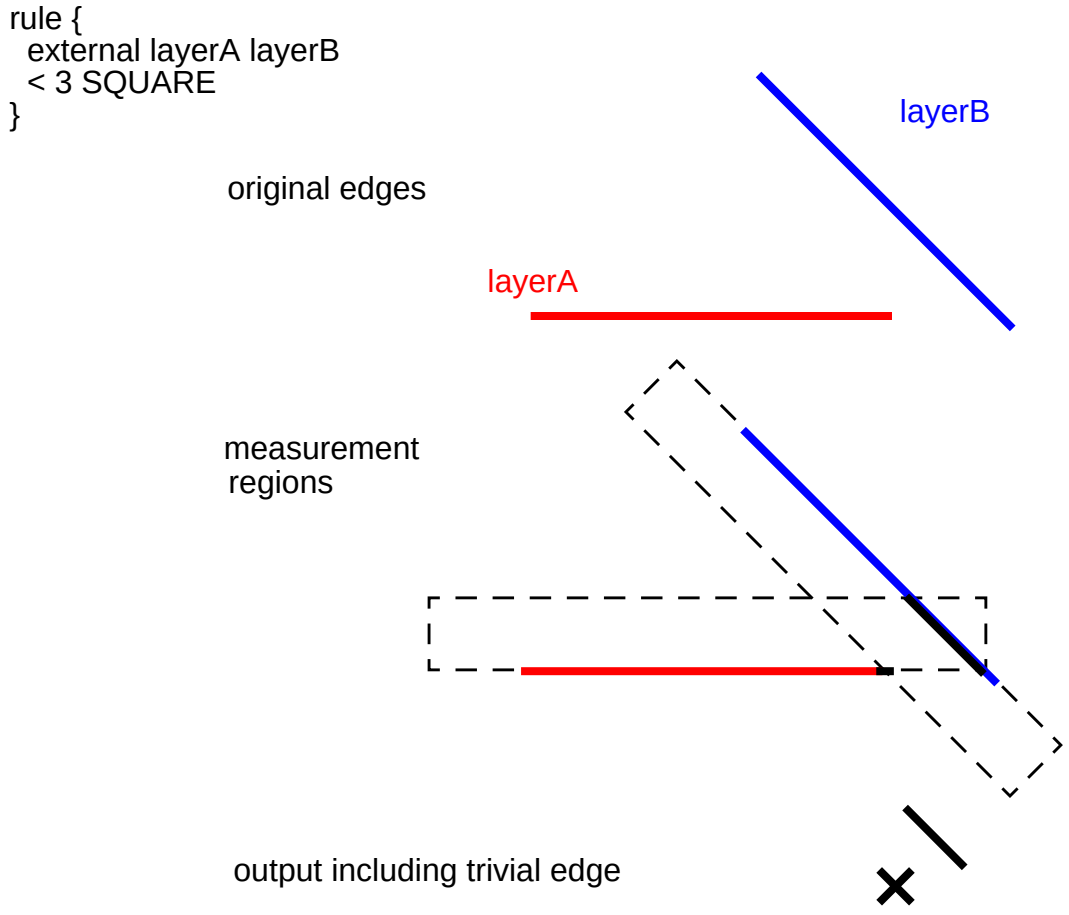


Figure 4-11. Trivial Edge Generation

Four-edge Output Cluster

Trivial edge generation can lead to a four-edge output cluster, which primarily occurs when there are two output sub-edges from edge A but none from edge B, or vice versa. In this case, the tool constructs two trivial output edges from B, each corresponding to a sub-edge from A; the output itself is a four-edge cluster.

For example, Figure 4-12 shows the generation of a four-edge output cluster. This example uses the square metric for measurement region construction and indicates trivial edges with an X.

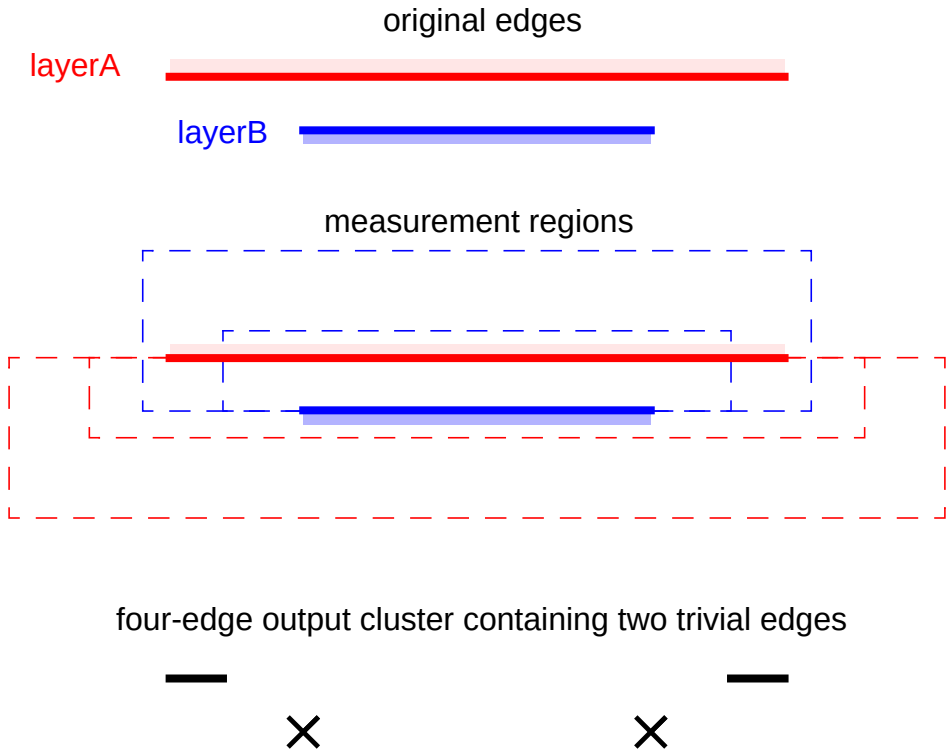


Figure 4-12. Four-Edge Output Cluster

Point-to-point Measurement Output

There is special treatment of true point-to-point output from the measurement process for error-directed and polygon-directed dimensional checks. This output consists of two trivial edges (points) which are generated as a result of the actual measurement process. These are not generated due to measurement region intersections being non-commutative. For example, if your user units are microns

and your Precision is 1000, the situation in Figure 4-13 would result in two output clusters consisting of trivial edges.

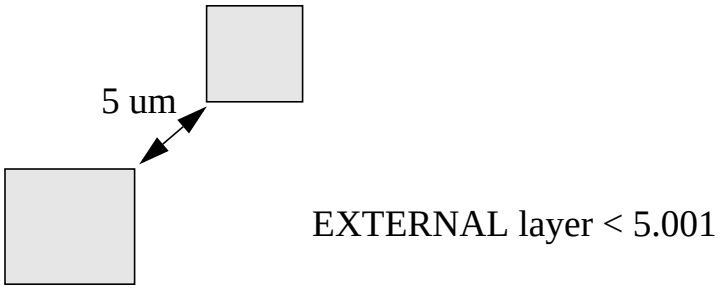


Figure 4-13. Point-to-point Trivial Edge Generation

In order to reduce false measurements and for region formation for polygon directed output, it is necessary to modify true point-to-point output slightly by extending each trivial edge in the cluster by two database units. This extension is along the direction of the original edge. Extension of point to point output is performed only under these circumstances:

- the dimensional check operation must be error-directed or polygon-directed
- the OPPOSITE or OPPOSITE SYMMETRIC metrics are not specified
- the measurement constraint is of the form "< *value*"

Clustered Output Summary

Output from edge measurement can consist of a one-, two-, three-, or four-edge cluster; the two-edge cluster is the most common. A one-edge cluster can result from the Inside Also or Outside Also secondary keyword, or from an Drawn Skew operation. When Calibre sends an edge layer to the DRC results database, the layer becomes an one-edge cluster across that interface.

The concept of edge clustering applies only to derived error layers. The tool sends output edges from an edge-directed dimensional check operation according to the input layer from which they originated. This output does not occur according to any relationships the layers shared with other output edges from edge measurement.

Special Considerations for the OPPOSITE Metric

The algorithms for measurement output generation are slightly different from that discussed above if the OOPOSITE metric has been specified. This is to reduce the number of non-orthogonal edges created when using polygon-directed output.

First, trivial edges are not generated by the OPPOSITE metric in the event of a non-commutative measurement. Non-commutative measurements produce no output if the opposite metric is specified.

Second, if exactly one of the two edges being measured is non-orthogonal (with respect to the database axes), then the measurement region is constructed from the orthogonal edge only. The non-orthogonal edge is then intersected with the measurement region to produce output from that edge. Output from the orthogonal edge is the projection onto the orthogonal edge of output from the non-orthogonal edge. This is illustrated in the figure below:

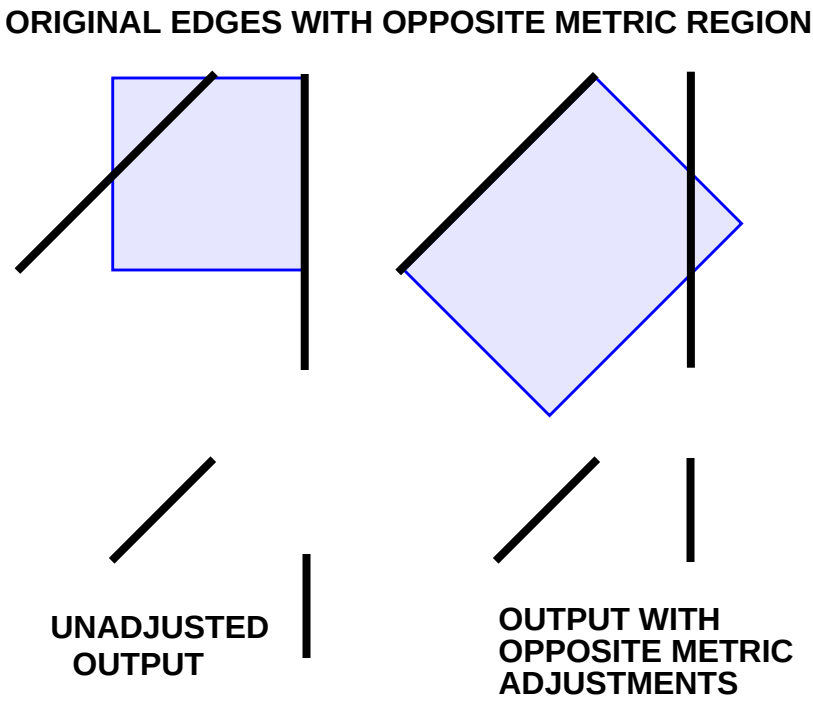


Figure 4-14. Output Adjustments for the OPPOSITE Metric

Interval Constraints for Output Suppression

In conjunctive design rule checking, you may want to suppress redundant errors with interval constraints (containing two numerics) in the final dimensional check operation. For example, consider this rule check statement (refer to Figure 4-15):

```
// Metal spacing must be 3 microns except where metal width is
// less than 3 microns;
//in this case, the metal spacing must be 4 microns.
metal_spacing  {
     EXTERNAL metal < 3
     X = INTERNAL [metal] < 3
     EXTERNAL metal X < 4
     }
```

In this rule check statement, the second External operation provides two edge pairs as output. The first pair is redundant, however, because the first External operation generated a similar error. The cause of the redundant error is that the tool measures an edge on layer X twice.

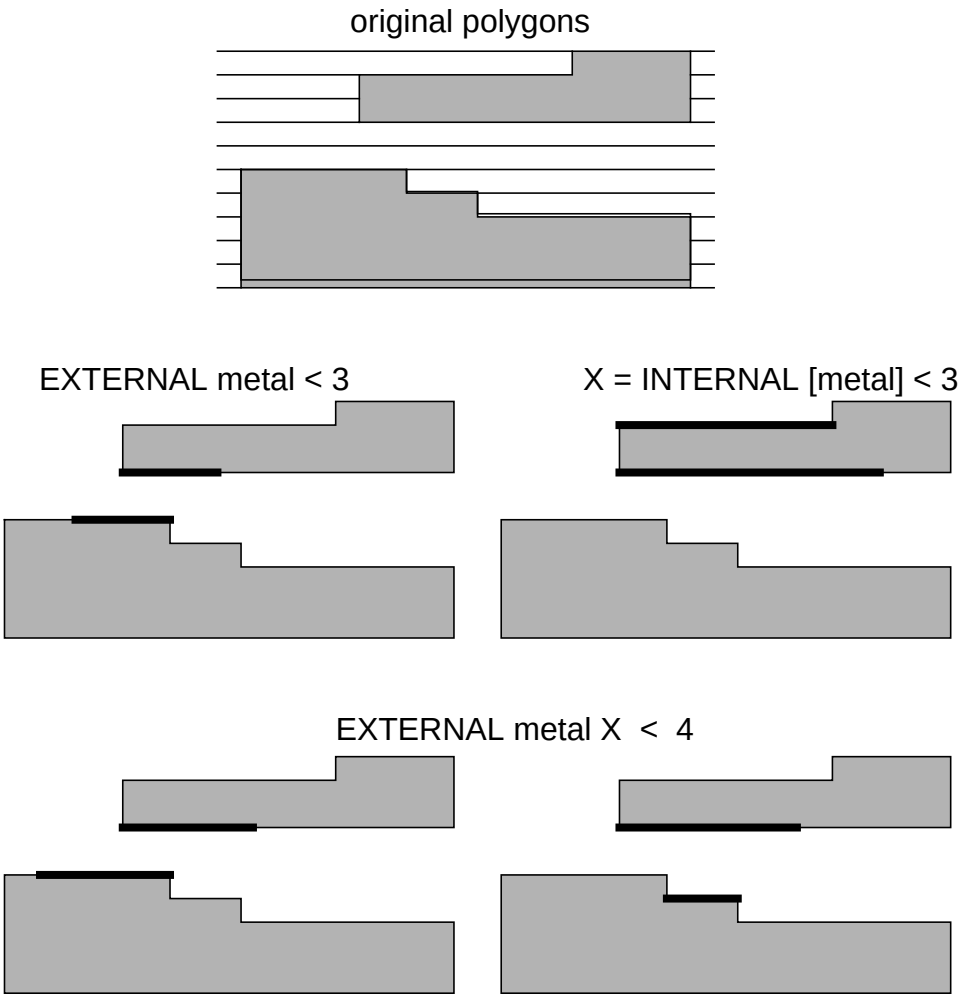


Figure 4-15. Suppressing Redundant Errors (part 1)

One way to suppress this duplication is to use an interval constraint in the second External operation. This prevents generating two errors for the spacing violations

of 3 microns. For example, consider this rule check statement (refer to Figures 4-15 and 4-16):

```
// Metal spacing must be 3 microns except where metal width is
// less than 3 microns;
// in this case, the metal spacing must be 4 microns.
metal_spacing  {
     EXTERNAL metal < 3
     X = INTERNAL [metal] < 3
     EXTERNAL metal X >= 3 < 4 // uses an interval constraint
     }
```

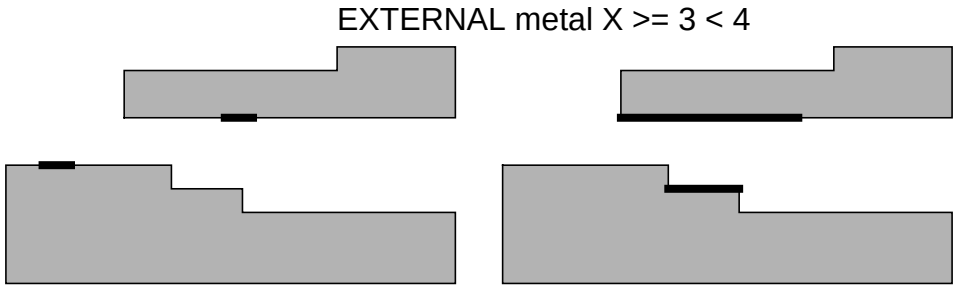


Figure 4-16. Suppressing Redundant Errors (part 2)

Using an interval constraint to suppress redundant errors works for most cases. However, in this particular example, the second External operation still generates the two edge pairs shown in Figure 4-16, with the left-most edge pair being somewhat non-intuitive.

The left-most edge pair occurs because the Euclidean and square metrics measurement regions contain area to the sides when you use interval constraints in the dimensional check operations. This area can correctly cause unwanted intersections (and hence output) which were not supposed to be part of the check.

The Opposite metric generally eliminates this effect because there are no side regions when you use this metric with interval constraints. However, if the Opposite metric is not appropriate, then you must either understand and accept errors such as the first one above, or else do not use interval constraints to suppress possible redundant errors in certain complicated conjunctive DRCs.

The use of interval constraints is applicable to both flat and hierarchical DRC applications. However, when you specify interval constraints in hierarchical applications, without the Opposite keyword, the tool may use a large amount of CPU overhead. It is recommended that the Opposite metric be used whenever you specify interval constraints.

Appropriateness Criteria

The dimensional check operations consider two edges to be “appropriate” for measurement if the corresponding sides of the edges face each other:

- The Enclosure operation measures the separation between the outside of edge A from the first input layer and the inside of edge B from the second input layer only if the outside of edge A and inside of edge B face each other.
- The External operation measures the separation between the outsides of edge A and edge B only if the outsides of the edges face each other.
- The Internal operation measures the separation between the insides of edge A and edge B only if the insides of the edges face each other.

Figure 4-17 illustrates how to make the notion of “appropriateness” for measurement more precise. Given edge A, region IN-A is the half-plane consisting of all points on the same side of the line as the inside of A. Region OUT-A is the half-plane consisting of all points on the same side of the line as the outside of A. Neither IN-A nor OUT-A contains the line determined by edge A.

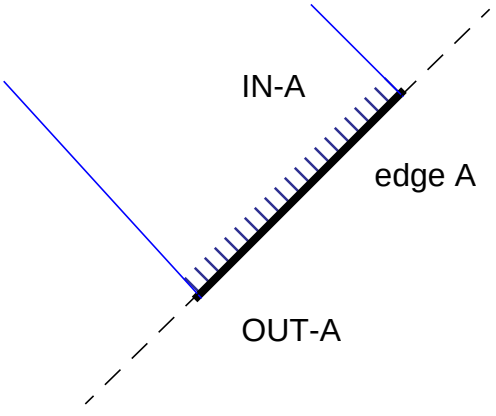


Figure 4-17. Edge Inside and Outside Planes

Using the definition of IN-A and OUT-A for an edge A, the line-of-sight between two edges (A and B) is as follows:

- An outside line-of-sight between an edge A and an edge B is any line segment connecting A and B that intersects both OUT-A and OUT-B.
- An inside line-of-sight between edge A and edge B is any line segment connecting A and B that intersects both IN-A and IN-B.
- An outside-to-inside line-of-sight from edge A to edge B is any line segment connecting A and B that intersects both OUT-A and IN-B.

Note that a line-of-sight of any type does not necessarily exist for any pair of edges A and B. From this fact you can quantitatively define appropriateness as follows:

- Edges A and B are appropriate for measurement in an external check if there is an outside line-of-sight between A and B.
- Edges A and B are appropriate for measurement in an internal check if there is an inside line-of-sight between A and B.
- Edge A, from the first input layer, and B, from the second input layer, are appropriate for measurement in an enclosure check if there is an outside-to-inside line-of-sight from A to B.

From the definition of appropriateness, the dimensional check operations consider two edges to face each other *only if* the angle between the corresponding sides of the edges is less than 180 degrees.

For example, Figure 4-18 shows the angles between the outsides of some edge configurations whose outsides are considered (by the External operation) to face each other.

The angle between the corresponding sides of the edges is called the appropriate angle. The dimensional check operations use orientation filters to govern the

appropriate angle. These filters are discussed in the *SVRF Manual*.

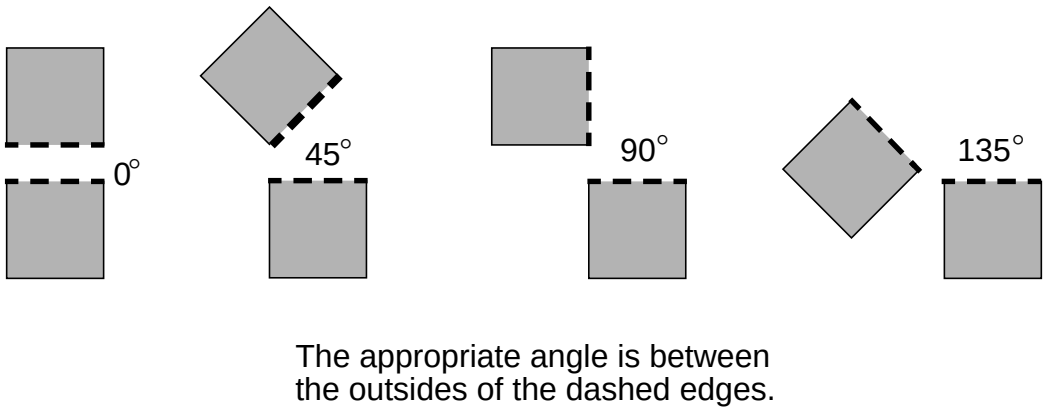


Figure 4-18. Appropriate Angles Between the Outsides of Edges

Intersection Criteria

By default, the dimensional check operations do not measure the separation between the corresponding sides of intersecting edges, even if they conform to the appropriateness criteria. However, this behavior can be altered by using appropriate secondary keywords.

Edge Breaking

Edge breaking occurs during the evaluation of a two-layer dimensional check operation. Calibre DRC breaks edges from each input layer that crosses polygon boundaries of the other input layer into edge segments. This edge-breaking method eliminates many false errors and makes the output from the two-layer dimensional check operations more precise.

Figure 4-19 shows an example of edge-breaking.

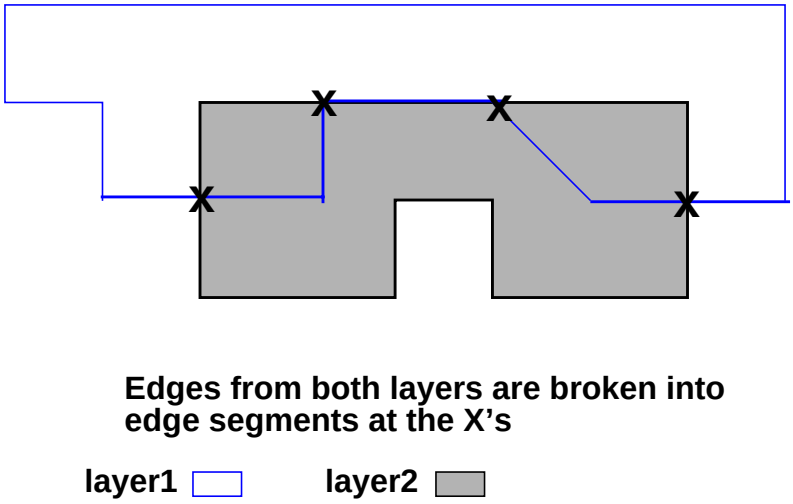


Figure 4-19. Edge Breaking in a Two-Layer Dimensional Check Operation

- Any layer1 edge that lies both inside and outside of a layer2 polygon breaks into edge segments at the point where the layer1 edge intersects the layer2 edge.
- Any layer2 edge that lies both inside and outside a layer1 polygon breaks into edge segments at the point where the layer2 edge intersects the layer1 edge.
- Any coincident layer1 and layer2 edges break into edge segments at the point(s) where they are no longer coincident.

Therefore, after edge breaking, one of the following is true of every layer1 (and layer2) edge:

- The layer1 edge lies completely inside a layer2 polygon, except that one or two endpoints of the layer1 edge can touch the insides of layer2 edges.
- The layer1 edge lies completely outside a layer2 polygon, except that one or two endpoints of the layer1 edge can touch the outsides of layer2 edges.
- The layer1 edge is inside or outside coincident with a layer2 edge.

This edge-breaking method does not fully apply when one or more of the input layers is a derived edge layer. This case modifies the edge-breaking method as follows:

- Any layer2 edge that intersects a layer1 edge at a single point (excluding the endpoint of the layer2 edge) breaks into two edge segments at the point where the layer2 edge intersects the layer1 edge.
- Any layer2 edge that is coincident with a layer1 edge breaks into two or more edge segments at the point were the layer1 and layer2 edges are no longer coincident.

After edge breaking, Calibre DRC applications use the conditions of the layer1 and layer2 edge to determine if an edge conforms to the polygon containment criteria of the dimensional check operations.

Polygon Containment Criteria

The polygon containment criteria apply only to the two-layer dimensional check operations where it provides a more precise and physically meaningful definition of the edge breaking that occurs in two-layer dimensional check operations.

The polygon containment criteria for all dimensional check operations do not fully apply when layer1, layer2, or both are derived edge layers because polygon boundaries of derived edge layers are not computable. This modifies the criteria as follows:

- **Enclosure**. If layer2 is a derived edge layer, then Calibre DRC measures edge A only if it is not coincident with any edge from layer2. If layer1 is a derived edge layer, then Calibre DRC measures edge B only if it is not inside-coincident with any edge from layer2.
- **External**. If layer1 is a derived edge layer, a pair of layer1 and layer2 edges will conform only if the layer2 edge is not outside-coincident with any layer1 edge. If layer2 is a derived edge layer, a pair of layer1 and layer2 edges will conform only if the layer1 edge is not outside-coincident with any layer2 edge.

- **Internal**. If layer1 is a derived edge layer, a pair of layer1 and layer2 edges will conform only if the layer2 edge is not coincident with any layer1 edge. If layer2 is a derived edge layer, a pair of layer1 and layer2 edges will conform only if the layer1 edge is not coincident with any layer2 edge.

The polygon containment criteria for the Enclosure, External, and Internal operations described above address the “looking through the wall” problem of two-layer dimensional check operations. That is, these conditions prohibit the measurement between the edges shown in Figure 4-20, even though the correct sides of the edges face each other.

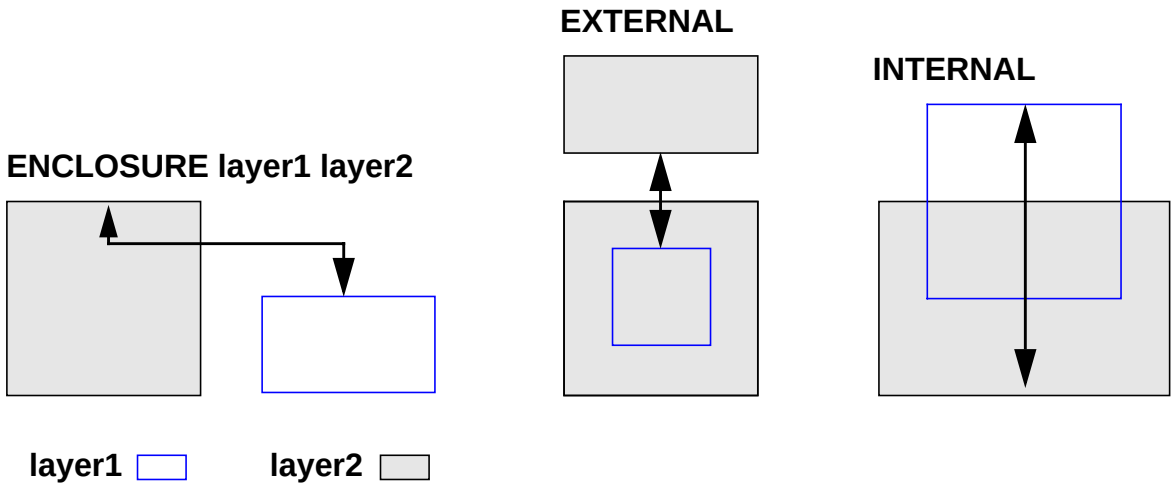


Figure 4-20. Looking Through the Wall Problem

Edge-directed Output

There may be times when you need to have error-directed measurement operations result in edge-directed output. For example the rule:

```
rule {external poly oxide < 4}
```

would result in a derived-error layer containing edge pairs from `poly` and `oxide` that are closer than 4 user units. If you want the output to be a derived edge layer containing only the edge segments pertaining to `poly`, you enclose the layer name in square brackets ([ ]). Enclosing the layer name in [ ] is called positive edge-

directed output. For example:

```
X = external [poly] oxide < 4
```

creates a derived layer of `poly` edges that satisfy the operation.

Enclosing the layer name in parentheses, ( ), is called negative edge-directed output. Negative edge-directed output returns the edge segments that would not normally be returned. For example:

```
X = external (poly) oxide < 4
```

creates a derived layer of poly edges that do NOT satisfy the operation.

Only one edge-directed output specification may appear in a single dimensional check operation. Edge-directed output specifications apply to Enclosure, External, and Internal.

Results from edge-directed output may also be error-directed (sent to the DRC results database). This is done by placing an edge-directed statement into a rule check. For example:

```
rule { enclosure contact [metal1] < .05 }
```

will have positive edge-directed results from `metal1` sent to the DRC results database (as opposed to creating a derived edge layer). The *SVRF Manual* has a number of examples listed under Enclosure, External, and Internal.

Polygon-directed Output

Polygon-directed dimensional check operations generate derived polygon layers by forming the polygon projections between edges in edge-clusters which *would have been* present in the corresponding error-directed dimensional check operation.

For example the rule:

```
rule {external poly oxide < 4}
```

would result in a derived-error layer containing edge pairs from `poly` and `oxide` that are closer than 4 user units. If you want the output to be a derived polygon layer containing a region between the poly-to-metal violations, you include the REGION secondary keyword (often with the OPPOSITE metric specified). For example:

```
X = {external poly oxide REGION OPPOSITE}
```

creates a derived polygon layer of regions between `poly` and `oxide` violations. The *SVRF Manual* has a number of examples.

There are three situations where polygon projections from a polygon-directed dimensional check operation cannot be cleanly formed and special measures must be taken to produce polygonal output. The first case involves a two-edge cluster consisting of coincident edges (most likely from an ABUT==0 secondary keyword). Forming the polygon projection between such output edges would yield a zero-area polygon which would be merged away. Therefore, a region is actually *grown* from the edges. For coincident outside edges (from a two-layer External or Internal operation), the region is grown on the outside of each edge for a distance of approximately 2 database units, yielding a rectangle of width approximately 4 database units with the edge pair running down the middle. For coincident inside edges (from an Enclosure operation), the region is grown on the inside of each edge for a distance of approximately 4 database units, yielding a rectangle of width approximately 4 database units with the edge pair running down one side.

The second case involves output from the INSIDE ALSO and OUTSIDE ALSO options. In an error-directed form, the output from these options consist of one-edge clusters. To form output in the polygon-directed form, these one-edge clusters are converted to polygons by growing a region on the inside of the edge for a distance of approximately 4 database units, yielding a rectangle of width approximately 4 database units with the edge running down one side.

The third case involves true point-to point (see page 4-28) output from the measurement process. Since the polygonal projection would be a zero-area polygon which would be merged away, DRC output using the REGION option could miss true errors. The External, Internal, and Enclosure operations will extend the length of edges output from the measurement process by a small value

(approximately two database units) prior to construction of the REGION option's polygonal projection under the following conditions:

- two *true* trivial edges are the result of the measurement process
- the OPPOSITE or OPPOSITE SYMMETRIC metric was not specified
- the operation's constraint is of the form "< a".

Although primarily intended to construct intermediate layers in conjunctive processes, many users may wish to use the polygon-directed form of a dimensional check operation for DRC results database instantiation. This was illustrated in the previous example. Polygon-directed output from DRC rule checks has three potential benefits:

- Adjacent edge clusters which form multiple errors may be merged together, thereby reducing the error count.
- Spurious errors, especially those associated with the effects of edge breaking and the INSIDE ALSO and OUTSIDE ALSO options may also be merged together, again reducing the error count.
- Polygon-directed output may seem visually more appealing to some users.

To illustrate the first two points, consider the following check, shown in both error-directed form and polygon-directed form:

```
Rule5.3 { ENCLOSURE cont met < 2 ABUT == 0 SINGULAR OUTSIDE
          ALSO }
Rule5.3 { ENCLOSURE cont met < 2 ABUT == 0 SINGULAR OUTSIDE
          ALSO REGION }
```

The following figure illustrates a scenario where the polygon-directed form has reduced the total error count:

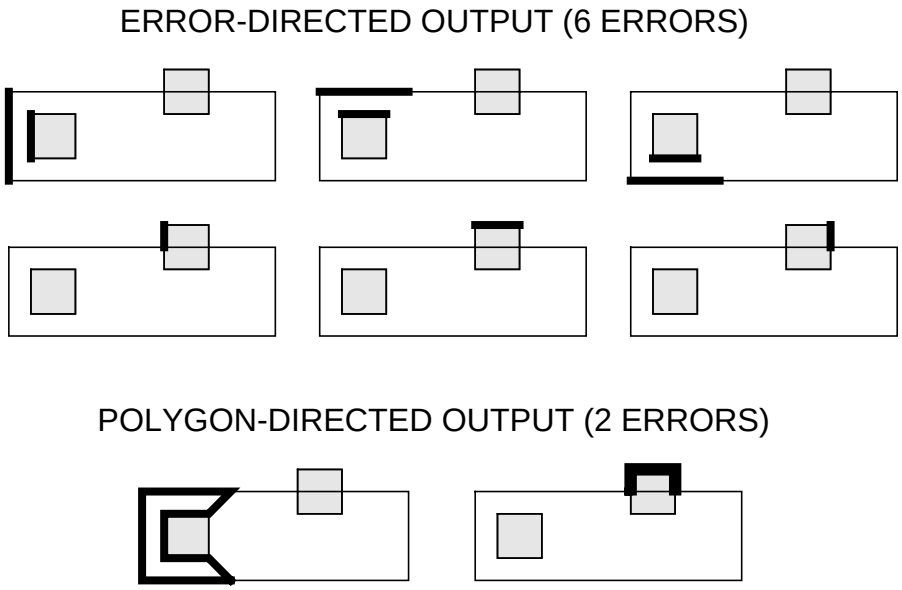


Figure 4-21. Error Reduction Using Polygon-directed Output

Although primarily used for final error output, the SINGULAR keyword has some very interesting applications in conjunctive checks when used along with INTERSECTING ONLY and REGION. Consider the following example:

```
poly_to_diff {
@ Normally, there is no spacing rule between poly and
@ diffusion. However, when either poly or diffusion bends
@ after forming a gate, the spacing must be 0.2 microns with
@ no touching allowed.
x = EXT poly diffusion < 0.2 ABUT == 0 REGION OPPOSITE
// potential error regions
y = EXT [ x ] gate < 0.1 ABUT == 0 SINGULAR
// 0.1 is arbitrary
x WITH EDGE y
// real error regions
}
```

False Measurement Reduction

There are cases where output from DRC measurement is undesirable, or false. False output can occur even though the measurement definitions constructed are followed exactly. For example, the following notch measurement is generally considered false:

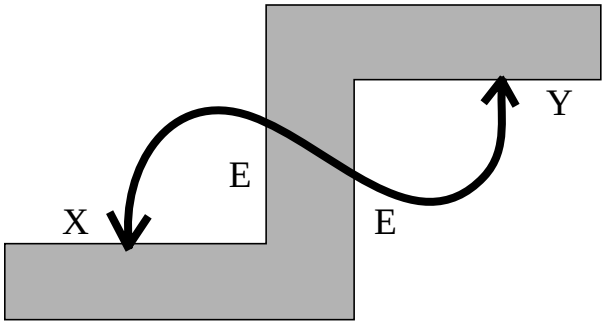


Figure 4-22. False Notch Measurement

as is the following enclosure measurement:

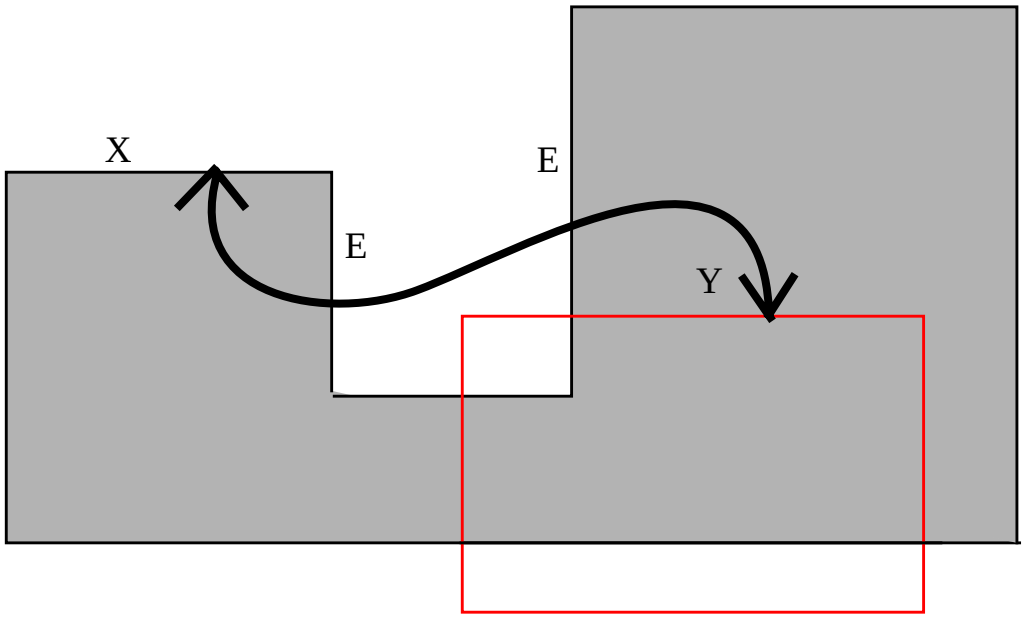


Figure 4-23. False Enclosure Measurement

Note that both of the above measurements obey all previous semantics, including polygon containment criteria for two-layer DRC operations. Since they are almost universally considered false, it is desirable to attempt to avoid them. In both cases the measurement is considered false due to the presence of another edge completely blocking the line of sight between the two edges being measured.

Since false measurement elimination in an edge-based system is expensive, the DRC application only attempts to eliminate the most blatant examples. The following algorithm is used:

A measurement between any two edges X and Y is considered false if the following circumstances are true:

- You did not specify the Opposite metric.
- The edges do not intersect.
- The edges are not orthogonal and projecting.
- The region defining the “violation” (in the sense of that produced by the REGION keyword) is cut completely in half by an edge intersecting an endpoint of X or Y.

Note that any of the edges labeled E in the two examples above render the measurement between X and Y false by this definition. (In the second example, remember that the edges are first subject to edge-breaking).

In Calibre DRC-H it is possible in rare cases that the blocking edges (E) are not available at the correct level of hierarchy when edges X and Y are measured. This can allow the false measurements to slip through. This phenomenon is most often observed with notch measurements.

Error Tolerance Setting

Calibre DRC can generate some warnings that create a possible data integrity issue, such as when it excludes the objects that cause these warnings from processing. The rule file Layout Error On Input specification statement allows you

to convert these warnings into fatal errors. For the following rule:

```
LAYOUT ERROR ON INPUT YES
```

the Calibre DRC warnings listed below become fatal errors:

```
Absolute angle in placement of cell name within cell name not
          yet supported.
Absolute magnification in placement of cell name within cell
          name not yetsupported.
Another cell record encountered for cell name.
Another cell record encountered for symbol# number ( name ).
Cell name is referenced but not defined.
Cell symbol# number is referenced but not defined.
Database precision number in GDSII stream file name is
          inconsistent with number in file name.
Implicit command name outside symbol definition in CIF input
          file name at line number, file offset number.
Invalid box (direction is (0,0) at location point in cell
          name on layer layer.
Invalid call (rotation is (0,0)) of symbol number in cell
          name.
Invalid symbol scale factor in CIF input file  name  at line
          number, file offset number.
Large number-vertex path (>1024 vertices) at location point
          in cell name on layer layer.
Large vertex (>4096) polygon at location point in cell name on
          layer  layer .
Non-orientable or degenerate polygon at location point in
          cell name on layer layer.
Path of absolute width at location point in cell name on
          layer layer not yet supported.
Physical precision number in GDSII stream file name is
          inconsistent with number in file name.
Process precision number is not consistent with GDSII
          precision number.
Problem extending type 4 path at location point in cell name
          on layer layer by number.
Round flash at location  point in cell name on layer layer
          not yet suported.
Trivial (1-vertex) path at location point in cell name on
          layer layer.
```

```
Unprocessed user extension commands present in the CIF input
          database.
Unresolved layers present in the CIF input database.
Unsupported Definition Delete in CIF input file name at line
          number, file offset number.
```

Disk-based Layers

By default, layers created during a verification run use a significant amount of virtual memory. These layers include:

- Named derived layers
- Merged original layers
- Layers representing output operations of DRC rule checks

In general, these layers consume the vast majority of virtual memory used during execution.

For selected flat verification applications, you can specify that layers created during the run use disk-based memory rather than virtual memory. In most cases, this can save large amounts of virtual memory by transferring memory usage to disk files. It also uses of network resources for layer storage.

The amount transferred from virtual memory to disk files when you use disk-based layers depends on the number of large derived layers created during the run that will exist at one time. Because the application reads the database in one scan at the beginning, all original layers will be simultaneously present in memory prior to being written to disk files. In addition, the application first creates each individual derived layer in virtual memory before writing it to a disk file. In some cases, using disk-based layers saves no virtual memory and can increase the resource requirement by the amount of file space used.

You control the use of disk-based layers in DRC by the Layer Directory specification statement.

Use of disk-based layers is not supported in hierarchical Calibre applications.

Disk-based Layers in Calibre

The use of disk-based layers in Calibre is controlled by the presence of the Layer Directory specification statement. If this statement is not specified, then disk-based layers are not used. Otherwise, layers will be based in disk files in the specified directory. This specified directory is created if it does not exist. A sub-directory is then created in the specified directory to hold the layer files. This subdirectory is called icv.<number> where <number> is a time stamp at the resolution of one second. A test open is performed to determine the integrity of this directory. If a directory creation fails or the test open fails, then a warning is issued and virtual memory based layers will be used. File I/O exceptions occurring from that point will cause aborts. Files that hold disk-based layers are named L<number> where <number> is an internal layer number. The files are created and deleted using the same scheduling algorithms as for virtual memory based layers. When execution completes, all directories that were created are removed.

Specialized DRC Applications

Dual Database Capability

Dual database capability refers to the ability of Calibre applications to read and merge two distinct input layout databases. This capability is not a special mode. The merging is transparent and occurs as the application reads the databases. A major application of dual database capability in Calibre DRC is layout-versus-layout (LVL) comparison.

Note

Dual database capability is not the same as allowing multiple file names in the Layout Path specification statement. Calibre applications treat these multiple files as a single input layout database (for example, there is only one top-level cell). In a dual database application, there are two distinct layout hierarchies.

Rule File Specification Statements

In a dual database application, the Layout System, Layout Path, and Layout Primary specification statements specify the first input layout database. Similarly, the corresponding Layout System2, Layout Path2, and Layout Primary2 specification statements specify the second database. These specification statements are required to designate a dual database application. (The Layout Bump2 specification statement is discussed in the next subsection.)

Only the GDSII and CIF layout systems are supported. You can specify Layout Path2 any number of times. All other layout database specification statements apply equally to both databases in a dual database application.

Layer Bump

For a dual database model to have meaningful applications, primitive objects (geometries and texts) must be distinguishable between the two databases. Calibre DRC implements this by incrementing the primitive layer number (before applying any Layer Map specification statements) of all objects in the second database by a constant value. You specify this value in the Layout Bump2 specification statement. Each object in the second database of a dual database application behaves exactly as if its primitive layer number increases by the specified layer bump value. For a CIF system, the rule file Layer specification statement that defines the CIF database layer name provides the primitive layer number of an object.

Layer and Layer Map specification statements apply to each of the two layout databases equally in a dual database model. Thus, you must be careful when constructing original layer definitions in a rule file and any layer mappings in a dual database application.

In the following example, assume you want to compare GDSII databases A and B, each contains layer 2 (diffusion) and layer 45 (metal1):

```
LAYOUT SYSTEM GDSII
LAYOUT PATH a.gds
LAYOUT PRIMARY ATOP

LAYOUT SYSTEM2 GDSII
LAYOUT PATH2 b.gds
LAYOUT PRIMARY2 BTOP       // Could be ATOP also.

LAYOUT BUMP2 100
LAYER DIFF 2
LAYER METAL1 45

LAYER DIFF_2 102
LAYER METAL1_2 145
A { DIFF XOR ( SIZE DIFF_2  BY 0.01 ) }
B { METAL1 NOT METAL1_2 }
C { METAL1_2 NOT METAL1 }
```

The choice of 100 as a layer bump value is arbitrary. The value needs to be larger than 45, which is the highest referenced simple layer number from the first database. A simple layer number refers to the primitive layer number of an object after applying any Layer Map specification statements. If you choose a layer bump value of 20, then an object on layer 25 in the second database would become layer 45 and appear on the original layer `metal1`. This is because Layer specification statements apply equally to each of the two databases.

For dual databases, the tool ignores all objects in the first database whose primitive layer number is greater than or equal to the value of Layout Bump2. Objects on layer 102 in the first database, if one exists, will be placed on the original layer `diff_2`.

Construction of original layer definitions with Layer Map specification statements in a dual database application is more complicated. You need to recall the exact definition of how a layer map works and choose the layer bump value that is greater than the highest simple (not primitive) layer number from the first database. The layer bump value also applies to text objects, including those defined in Layout Text specification statements.

Special Semantics for Hierarchical Applications

For flat dual database applications, the layer bump value from the Layout Bump2 specification statement controls all semantics for combining the two input layout databases. The internal combination process is more complicated in hierarchical applications because they preserve, not flatten, the input hierarchy. The tool does this as follows:

1. Renames all cells in database 1 internally (such as A -> A1).

2. Renames all cells in database 2 internally (such as A -> A2), as in Step 1.

3. Creates a new top level cell and instantiates the top level cells T1$1$ and T2$2$ from databases 1 and 2 with identity transforms. The name of the new top-level cell is that of the top-level cell from database 1.

4. Attempts merging at critical points in the construction of the hierarchical database. Creates a new cell A, if it has not done already so whenever two placements A1 and A2 are in an exact overlap situation (equal extents, transforms, and array characteristics). Places all objects from A1 and A2 into cell A, and replaces the two placements of A1 and A2 with a placement of cell A that has the same transform and array characteristics.

5. Retains internal names of cell templates that were not merged in all placements and keeps them as part of the hierarchy. If the top-level cell from database 1 was merged, then the merged cell retains an internal name to avoid naming conflicts with the new top-level cell.

Processes that require cell names, such as Inside Cell operations, and hcell detection, always use the original cell name (never the internally-generated name). Also, by using the Layout Rename Cell specification statement, you can force the merging of differently, or in some cases similarly, named cells from the two databases.

If the hierarchies of the two databases are not drastically different, the automatic hierarchical database construction processes of dense overlap removal and hierarchical injection make them as similar as possible. This prevents a major impact on performance. If the hierarchies are drastically different, you should consider running the application flat.

Flat Procedure Example

As discussed above, one application of dual database capability is to compare two layout databases on a layer-by-layer basis. The binary layout database format also allows the Calibre DRC-F application to compare databases flat, albeit in a more convoluted fashion. It can be useful, however, when one or both of the layout databases is in IC Station and, for some reason, translation to GDSII is not desirable.

As an example of how this is done, consider comparing an IC Station database where the top-level cell is called TOPCHIP to its corresponding GDSII database called topchip.gdsii. Assume that the design layers are 1-10:

1. Write a rule file called rules.a, which contains DRC rule checks and operations sufficient to act on all of the design layers (this guarantees all layers are actually read in). It can be the real rule file or just one for testing. It should also contain required specification statements for Calibre DRC using GDSII (just so it can be used twice). An easy example is the following:

   ```
   LAYOUT SYSTEM GDSII
   LAYOUT PATH topchip.gdsii
   LAYOUT PRIMARY TOPCHIP
   DRC RESULTS DATABASE drc_results
   LAYER XYZ   1 2 3 4 5 6 7 8 9 10
   rule { AREA XYZ == 0 } // Any operation on XYZ will do.
   ```

2. In IC Station, open a window on cell TOPCHIP, load the rule file rules.a, and run CHECK DRC with the WRITEDATABASE option. Make sure to run it flat (the default) to cover the entire hierarchy. This will create binary polygon files

   ```
   icv_data_1  ...  icv_data_10
   ```

   representing the IC Station database in the current directory.

3. Rename all of the binary data files just created to files with different numbers. A simple way is to change icv_data_1 to icv_data_100,and so on. The purpose is not to overwrite them when we write out the GDSII database.

   ```
   icv_data_100  ...  icv_data_110
   ```

   now represent the IC Station database in the current directory.

4. Execute Calibre DRC-F with the -writedatabase command option to write the GDSII database out to binary polygon files. The same rule file, rules.a, may be used since it contains the necessary stand-alone specification statements:

   ```
   calibre -drc rules.a -writedatabase
   ```

   This will create binary polygon files

   ```
   icv_data_1  ...  icv_data_10
   ```

   representing the GDSII database in the current directory.

5. We now are ready to XOR the two databases. Create a new rule file, call it rules.b, which contains DRC rule checks to do the XORs and specification statements to use the binary polygon files. One example is:

   ```
   LAYOUT SYSTEM BINARY
   DRC RESULTS DATABASE drc_results

   diff_1 { XOR 1 100 }
   diff_2 { XOR 2 102 }
   diff_3 { XOR 3 103 }
   diff_4 { XOR 4 104 }
   diff_5 { XOR 5 105 }
   diff_6 { XOR 6 106 }
   diff_7 { XOR 7 107 }
   diff_8 { XOR 8 108 }
   diff_9 { XOR 9 109 }
   diff_10 { XOR 10 110 }
   ```

6. Now run Calibre DRC-F using rules.b. This will execute all of the XOR operations:

```
calibre -drc rules.b
```

   The ASCII DRC results database "drc_results", as specified in rules.b, will contain all of the differences, if any, between the two databases.

Comparing other types of databases is similar. If the binary polygon files are too big to exist together simultaneously, the steps above can be repeated for each individual design layer with appropriate modifications. These binary polygon files can also be placed around the network and accessed by links.

GDSII DRC Results

The DRC Check Map specification statement provides a way to designate the destination (layer,datatype) coordinate for DRC rule check output into GDSII-type DRC result databases. The statement is much more complex than that, however, and additionally allows an m->n mapping between DRC RuleChecks and DRC results databases, as well as AREF compaction capability for rectangles, and per-rule-check specification of the maximum result count.

The m->n mapping is of primary interest. For any given Calibre DRC run, a set of unique DRC rule checks $\{ R_1, ..., R_n \}$ $(n > 0)$ is executed according to selection semantics described earlier. The output from these DRC RuleChecks is sent, by default, to the DRC results database specified in the DRC Results Database specification statement. Using DRC Check Map specification statements, you may expand this output so that a set of DRC result databases $\{ D_1, ..., D_m \}$ $(m > 0)$ are generated from the Calibre DRC run such that:

[1] Any individual DRC rule check R_j may have output directed to any number of different DRC results databases in the set $\{ D_1, ..., D_m \}$.

[2] Any individual DRC results database D_j may contain output from any number of different DRC RuleChecks in the set $\{ R_1, ..., R_n \}$.

[3] The set { D 1 , ..., D m } does not have to have uniform type, that is, ASCII, binary, or GDSII.

Attribute Specification

Note that the DRC Check Map specification statement allows local (that is, per-rule-check) specification of a DRC results database and its associated type (ASCII, binary, GDSII). These attributes are globally-specified in the DRC Results Database specification statement. The DRC Check Map statement also allows local specification of the DRC Maximum Results specification statement value. Finally, the statement allows locally-specified (layer,datatype) and AREF output specifications for GDSII-type DRC results databases.

The following DRC results database attributes cannot be locally-specified in the DRC Check Map specification statement and, hence, always apply globally when applicable:

[1] The check text mapping, as specified in the DRC Check Text specification statement.

[2] The maximum vertex count for result polygons, as specified in the DRC Maximum Vertex specification statement.

[3] Whether to retain empty rule checks (those with no DRC results), as specified in the DRC Keep Empty specification statement.

[4] Whether to output cell names and transforms in ASCII-type DRC results databases, and leave coordinates untransformed, as specified in the DRC Cell Name specification statement.

[5] Whether pseudo-hierarchy is to be retained, as specified by the PSEUDO parameter in the DRC Results Database specification statement.

[6] The append string for cell names in GDSII-type DRC results databases, as specified following the GDSII parameter in the DRC Results Database specification statement.

[7] Transfer of input layout database text to GDSII-type DRC results databases, as specified in the DRC Map Text specification statement.

[8] The magnification factor for the DRC results database, as specified in the DRC Magnify Results specification statement.

[9] The DRC results database precision, as specified in the DRC Results Database Precision specification statement.

Mapping Algorithm for Output

This section describes the mapping algorithm for multiplexing of DRC rule check output in detail. Given the set of unique DRC rule checks $\{ R_1, ..., R_n \}$ to be executed in a Calibre DRC run, a set of unique DRC result databases $\{ D_1, ..., D_m \}$ is determined which will encompass the cumulative DRC output from the application. Each results database D_j has an associated (and unique) type (ASCII, binary, GDSII). The mapping is n->m: Any given DRC RuleCheck R_i may map to multiple members of $\{ D_1, ..., D_m \}$ and any given DRC results database D_j may be mapped into by multiple members of $\{ R_1, ..., R_n \}$.

In order to allow completely unambiguous and non-conflicting construction of the above mapping, the compiler will enforce strict rules on the DRC Check Map specification statement:

[1] It is a compilation error for the *file_name* parameter to be specified in another DRC Check Map specification statement with a different type (ASCII, binary, GDSII) or to be globally specified (in the DRC Results Database specification statement) with a different type.

[2] If *file_name* is not specified in a DRC Check Map specification statement, then the type (ASCII, binary, GDSII) if the statement must match the global type specified (or defaulted) in the DRC Results Database specification statement. This is because the default *file_name* parameter is the global DRC Results Database file name.

[3] Multiple specification of the (*rule_name*,*file_name*) coordinate in the set of DRC Check Map specification statements is a compilation error. In particular, each of the three scenarios below is an error:

DRC CHECK MAP ruleR ... fileF
DRC CHECK MAP ruleR ... fileF

DRC CHECK MAP ruleR
DRC CHECK MAP ruleR

DRC RESULTS DATABASE fileF
DRC CHECK MAP ruleR ...
DRC CHECK MAP ruleR ... fileF

The second and third scenarios are errors since the *file_name* parameter of a DRC Check Map specification statement defaults to the global (that is, that specified in the DRC Results Database specification statement) DRC results database file name.

[4] If Maximum Results is specified in a DRC Check Map specification statement for DRC rule check R then it must be specified with exactly the same value in all DRC Check Map specification statements for rule check R. That is, no individual DRC rule check may have different maximum results specifications. (This is an internal limitation).

Given any DRC rule check R, the set of DRC results databases into which it maps is determined by the following algorithm. This algorithm, along with the compilation checks described above, also insures unambiguous determination of the MAXIMUM RESULTS values and AREF parameters for any R_i -> D_j map. A conceptual pseudocode of this is given as follows:

```
if R is not in any DRC Check Map specification statement {

  The DRC results database for R is that specified by the global
  (file_name,type) coordinate

}

else {

  for each DRC Check Map statement associated with R {

    if  file_name is specified in the DRC Check Map statement  {
```

The (*file_name*,*type*) coordinate of the DRC Check Map statement becomes a DRC results database for R

}

else {

The global (*file_name*,*type*) coordinate becomes a DRC results database for R

}

}

AREF Output

In Calibre DRC-H, the user may request the application to create GDSII AREF structures in a GDSII-type DRC Results database from arrayed rectangles. This can save considerable space in the output database when large numbers of rectangles (which each require 60 bytes in standard GDSII format) are converted to AREF structures. It is especially valuable in conjunction with output generated from the Rectangles operation. AREF output is specified by the optional AREF parameters in the DRC Check Map specification statement:

DRC CHECK MAP ***rule_name*** **... [AREF** ***cell_name width length*** **...**
[AREF ***cell_name width length*** **]]**

Recall that AREF cannot be specified if the DRC results database type of the statement is not GDSII.

The AREF keyword(s) instruct Calibre DRC-H to attempt to create GDSII AREF structures (that is, array placements of the specified *cell_name*) from rectangles of the given *width* and *length* (specified in user units) which constitute output of the given DRC rule check. For example:

```
output_active { // Planarize the active layer to create the
                // final active mask.
```

```
    x = ( EXTENT ) NOT ( SIZE active BY 1 ) // planarized area
    y = ( RECTANGLES 2 4 1 ) INSIDE x       // planarization
                                            // rectangles.
    active OR y                             // final active
                                              // layer.
}
…
DRC CHECK MAP output_active 2 AREF rect24 2 4
```

In this example, output polygons from DRC rule check output_active which are not 2 x 4 rectangles are sent directly to the DRC results database as before. Calibre DRC-H will then attempt to create GDSII AREFs from 2 x 4 output rectangles according to a complex set of internal heuristics intended to maximize the size and number of AREFs created and minimize the number of rectangles not array referenced. For all 2 x 4 output rectangles (where the dimension along the x-axis is 2), the system attempts to recognize as many array patterns as possible (where each pattern is as large as possible). For each pattern so recognized, an AREF of cell “rect24” of the appropriate dimension and pitch is output, instead of the rectangles themselves. The process is then repeated for 4 x 2 output rectangles (that is., the dimension along the x-axis is 4); the only difference is that the output AREFs will have a rotation component. Rectangles which are not array referenced in the above algorithm are output to the DRC results database as before. If any AREF is created, then a GDSII structure record is also output representing cell “rect24”; this structure contains only a single boundary on layer #2 representing the rectangle from (0,0) to (2,4).

Conversion of rectangles into AREFs is completely independent of all other DRC Results Database semantics including maximum result limit, GDSII multiplexing, and so on. Of course, all AREFs are created inside the output GDSII cell structures where the corresponding rectangles would have been output. Calibre DRC-H will not allow an AREF cell name to duplicate any structure name in the input layout database (or any pseudo-cell name) since the latter structure also has the possibility of being output; a warning is issued and AREF output is suppressed for that DRC Check Map AREF component.

Although not common, multiple AREF outputs may be specified in the same DRC Check Map specification statement. This allows AREF creation for output rectangles of various sizes from the same DRC rule check statement. The restrictions are that [1] an AREF cell name may not appear twice in the same DRC

Check Map specification statement, and [2] no two AREFs in the same DRC Check Map specification statement can have their *width* and *length* parameters equal in either order. Restriction [2] is intended to prevent output ambiguity. These restrictions are checked at rule file compilation time. An AREF cell name in a DRC Check Map specification statement is allowed to duplicate an AREF cell name in another DRC Check Map specification statement providing that the output layers are equal, the output datatypes are equal, the widths are equal, and the lengths are equal (again, this restriction is checked at compilation time); this allows AREFs of the same cell to be created by multiple DRC rule check statements.

AREFs with less than 16 elements are never created.

Incremental Connectivity and Antenna Checks

This section discusses some details involving incremental connectivity in the context of antenna checks. These methods may be employed in any problem involving incremental connectivity, however.

Antenna checks are a broad category of design checks that are intended to check for interconnect paths of sufficient surface area that can accumulate excessive charge during the fabrication process. These paths are called antennas and can adversely affect yield in the fabrication process. For the purpose of this discussion, we will limit ourselves to three layers of metal deposition.

Antenna checks for the metal i deposition stage must ignore connectivity created by metal j, for j > i. Since Mask mode Connect operations are executed as a single unit (at the beginning of the executive module), layers must be copied so as to effectively partition the connectivity of the design for each layer of metal. As an example, consider the rule file flow for simple antenna checks on a three-metal layer process:

```
diode = contact AND diff // Diffusion diodes for all levels.
cp1 = COPY poly   // Copy layers for first-level check.
cg1 = COPY gate  // This copying only needs to be done if the
                  //"standard" set of connect operations are
cm11 = COPY met1  // also present in the rule file.
cc1 = COPY contact
CONNECT cp1 cg1   // Connect for first-level check.
```

```
CONNECT cm11 cp1 by cc1
cp2 = COPY cp1    // Copy layers for second-level check.
cg2 = COPY cg1   // Note that we copy the previous copies at
                  // each stage. This insures that the layers
cm12 = COPY cm11 // at each stage are truly different since
                 // the rule file compiler combines identical
cc2 = COPY cc1    // operations.
cm22 = COPY met2
cv12 = COPY via1
CONNECT cp2 cg2   // Connect for second-level check.
CONNECT cm12 cp2 BY cc2
CONNECT cm22 cm12 BY cv12
cp3 = COPY cp2    // Copy layers for third-level check.
cg3 = COPY cg2
cm13 = COPY cm12
cc3 = COPY cc2
cm23 = COPY cm22
cv13 = COPY cv12
cm33 = COPY met3
cv23 = COPY via2
CONNECT cp3 cg3 // Connect for third-level check.
CONNECT cm13 cp3 BY cc3
CONNECT cm23 cm13 BY cv13
CONNECT cm33 cm23 BY cv23
// First level antenna check:
cdiode1 = cm11 AND diode // Diffusion diodes.
m1_check = NET AREA RATIO cm11 cdiode1 == 0 // Check only m1
                                             //not connected
rule1 { NET AREA RATIO m1_check cg1 > 300 } // to a
                                            //diffusion diode.
// Second level antenna check:
cdiode2 = cm12 AND diode                  // Diffusion diodes.
m2_check = NET AREA RATIO cm22 cdiode2 == 0 // Check only m2
                                              //not connected
rule2 { NET AREA RATIO m2_check cg2 > 300 }  // to a diffusion
                                               //diode.
// Third level antenna check:
cdiode3 = cm13 AND diode                  // Diffusion diodes.
m3_check = NET AREA RATIO cm33 cdiode3 == 0 // Check only m3
                                              //not connected
rule3 { NET AREA RATIO m3_check cg3 > 300 } // to a
                                            //diffusion diode.
```

Functionally this approach is correct since all layers in each “set” of Connect operations are disjoint from all layers in any other set. The copying of layers gets around the fundamental characteristic that all Connect operations are executed together and effectively partitions the circuit into independent collections of nets, thus insuring correct modeling of connectivity for antenna checking. That is, the nets created at each stage of metal deposition are completely disjoint from those created at any other stage.

From a performance standpoint, however, it is a bad solution since it requires numerous layer copies and connect operations—this consumes much memory space. In fact, some very large designs with many metal layers (for instance, 6) cannot be checked in a single run and the user is forced to put each antenna level in a different rule file (and a different run).

From a performance standpoint, the solution to the problem of efficient antenna checking is to support *incremental connectivity*. This is the ability to execute a sequence akin to the following:

[1] Execute some of the Connect operations.

[2] Execute the layer operations where connectivity requirements are derived only from the Connect operations executed in step [1].

[3] Execute more of the Connect operations.

[4] Execute the layer operations where connectivity requirements are derived only from the Connect operations executed in steps [1] and [3].

...

[n] Execute the remainder of the Connect operations.

[n+1] Execute the layer operations where connectivity requirements are derived from all of the Connect operations.

Hence, with incremental connectivity, the flow for the above example could be as follows:

```
DRC Incremental Connect Yes
diode = contact AND diff
```

```
// First level antenna check:
CONNECT poly gate
CONNECT m1 poly BY contact
CONNECT m1 diode
m1_check = NET AREA RATIO m1 diode == 0
rule1 { NET AREA RATIO m1_check gate > 300 }
// Second level antenna check:
CONNECT m2 m1 BY v1 // This changes connectivity of poly,
                      // diode, contact, and gate also.
m2_check = NET AREA RATIO m2 diode == 0
rule2 { NET AREA RATIO m2_check gate > 300 }
// Third level antenna check:
CONNECT m3 m2 BY v2 // This changes connectivity of poly,
                     // diode, contact, gate, m1, and v1 also.
m3_check = NET AREA RATIO m3 diode == 0
rule3 { NET AREA RATIO m3_check gate > 300 }
```

(We did not connect the diode in the first example simply to minimize the number of copies and connects required). Note that there is no copying of layers and the number of Connect operations is dramatically fewer. However, this method relies on the rule file being *order dependent*, which it is not by default.

Incremental connectivity is triggered by the specification statement DRC Incremental Connect YES. (The default is NO). If DRC Incremental Connect YES is specified, then DRC execution will view the rule file as having a partial front-to-back ordering as follows:

<layer operations> <- Connectivity zone #0
<connect operations>
<layer operations> <- Connectivity zone #1
<connect operations>
<layer operations> <- Connectivity zone #2
...
<connect operations>
<layer operations> <- Connectivity zone #N

Operations requiring connectivity in connectivity zone #0 are not allowed. Those requiring connectivity in connectivity zone #i, for i > 0, treat the connectivity as if only the Connect statements prior to connectivity zone #i have been executed. Operations requiring connectivity in connectivity zone #i, i > 0, are not allowed if

that connectivity can only be established by Connect statements after connectivity zone #i. Mask mode Label Order and Sconnect operations are not allowed (due to internal limitations). All errors resulting from this schema are flagged at compilation time.

Only the DRC applications will support incremental connectivity. Other applications will ignore the DRC Incremental Connect specification statement (at runtime, not compilation time) and treat the rule file as order-independent (and Connect statements as global), as usual. The rule file developer has the responsibility, as always, of mitigating connectivity conflicts, if any, in rule files covering multiple applications. Of course, this may require greater care with the addition of incremental connectivity.

Verification of connectivity becomes much more complicated with the presence of DRC Incremental Connect YES. Layer operations requiring connectivity of their parameter(s) are [Not] Net, Net Area, Net Area Ratio, Stamp, Ornet, constrained polygon topological operations with BY NET specified, and nodal dimensional check operations. Briefly, connectivity of the appropriate parameters originates from their presence in a Connect or Sconnect operation or their derivation via a sequence of node-preserving operations from a Connect or Sconnect parameter. With incremental connectivity, the addition of an order-dependency to the rule file means that connectivity of a layer may only be established using Connect operations which appear prior to its reference in the rule file. More precisely, incremental connectivity adds the following rules for connectivity verification:

[1] A layer operation defined in connectivity zone #i may not have a forward reference to connectivity zone #j, for j > i, that is, may not have a parameter defined in connectivity zone #j.

[2] A Mask mode Connect parameter must be defined prior to the Connect operation.

[3] Mask mode Label Order and Sconnect operations are not allowed.

[4] Connect operations after connectivity zone #i are not used to verify connectivity of a layer referenced in connectivity zone #i.

[5] A node-preserving layer derivation may not cross connectivity zones. For example, the following is illegal in the presence of DRC Incremental Connect YES:

```
CONNECT m1 poly BY contact
x = AREA m1 > 3
CONNECT m2 m1 BY via
rule { NET AREA x > 10 } // Connectivity of x verifies in a
                            // different connectivity zone.
```

while the following is valid:

```
CONNECT m1 poly BY contact
x = NET AREA m1 > 3 // Connectivity of m1 verifies in this
                       //zone.
CONNECT m2 m1 BY via
rule { AREA x > 10 } // Connectivity of x is not required.
```

These stricter connectivity verification rules given in [1] - [5] above allow two existing connectivity restrictions to be removed in Calibre DRC applications when DRC Incremental Connect YES is specified. First, a Connect layer may be derived from an operation requiring connectivity in an incremental connect environment if the layer's definition appears before the Connect operation (less restrictive than rule [2] above); this is always prohibited in a non-incremental environment. For example, the following construction (which can be a key capability for certain specialized DRC checks) is legal:

```
DRC INCREMENTAL CONNECT YES
CONNECT metal1
x = NET metal1 VDD
CONNECT x // Starting a new zone. Ok that x is derived from an
          // operation requiring connectivity
```

Second, a non-Connect layer requiring connectivity may exist in a Connect layer's derivation tree in an incremental connect environment. For Example:

```
DRC INCREMENTAL CONNECT YES
CONNECT x
y = AREA x > 2 // Layer y requires connectivity
rule { EXT y < 3 CONNECTED }
z = AREA y > 3 // Layer z requires layer y
```

```
CONNECT z     // Layer z in a CONNECT; ok - starting a new
               // CONNECT zone
```

This removal of the above two connectivity restrictions is only for Calibre DRC applications when DRC Incremental Connect YES is specified. It is not supported in ICrules (even though ICrules supports incremental connectivity).

The compiler will also disable operation equality optimizations across connectivity zones. This is essential for incremental connectivity support so that diode1 and diode2 in the following example are not optimized into the same operation:

```
CONNECT m1 poly BY contact
diode1 = contact AND diff
...
CONNECT m2 m1 BY via
diode2 = contact AND diff
```

The reason is that the contact layer in the first zone is not necessarily the same as the contact layer in the second zone. Finally, concurrency threads will not cross connectivity zones in order to prevent inadvertent creation of layers with incorrect connectivity.

The DRC execution sequence will also, of course, change if incremental connectivity is specified. The default execution sequence is briefly described as follows:

[1] Produce layer parameters for all Connect operations.

[2] Execute Connect operations, node-annotate all Connect layers which require it, and perform net naming.

[3] Produce data for all DRC RuleCheck output operations.

With specification of DRC Incremental Connect YES, the execution sequence will change to:

For each connectivity zone from first to last {

[1] Produce layer parameters for all new Connect statements which defined the zone.

[2] Execute all new Connect operations which defined the zone, appending new connectivity to existing connectivity. Node annotate all Connect layers at or prior to the zone which require it. Perform net naming based upon existing connectivity if there are [Not] Net operations in the zone.

[3] Produce data for all DRC rule check output operations in the zone.

[4] Produce data for all referenced connectivity layer operations in the zone

}

Notice the requirement that referenced connectivity layer operations in the connectivity zone must be executed (if they have not been already) while in the zone. This is to capture the connectivity of the zone and correctly satisfy backward references to the zone later.

The connectivity extraction operation Disconnect allows total deletion of the existing connectivity model in an incremental connect sequence. That is, the presence of a Disconnect operation causes the current connectivity “build-up” to be discontinued; the next Connect operation will begin the new connectivity build-up.

Disconnect may appear any number of times. It is ignored in non-DRC applications and in DRC applications where DRC Incremental Connect YES is not specified. That is because, in non-incremental connect flow, all Connect operations are executed as a single block and, hence, there is nothing to “disconnect.”

A Disconnect operation does not define a new connectivity zone; rather, the presence of a Disconnect operation in connectivity zone #i causes all existing connectivity to be deleted prior to execution of the first Connect operation past

zone #i. Appropriate modifications of the compile-time connectivity verification algorithms for layers in an incremental connect setting, discussed previously, are made in the presence of Disconnect operations. For example, connectivity of a layer cannot verify across any Disconnect operation.

Judicious sequencing of Connect operations in an incremental connectivity flow along with, perhaps, careful copying of Connect layers, makes the Disconnect operation rarely useful. It is, however, indispensable in certain very advanced DRC checks where any existing text attachment must be completely discarded.

Soft Connection Checks

DRC application can check for soft connections with the External operation. Calibre LVS checks for soft connections with the LVS Softchk specification statement, which you cannot use with DRC applications.

In the following examples, the INSIDE ALSO and NOT CONNECTED secondary keywords of the External operation check for soft connections in DRC applications.

The following examples show a pair of statements LVS applications use to check for soft connections, followed by the associated DRC rule:

- Example 1

```
SCONNECT upper_layer lower_layer
LVS SOFTCHK lower_layer LOWER  ALL

SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED
   lower_layer NOT OUTSIDE REJ_UPPER
   }
```

- Example 2

```
SCONNECT upper_layer lower_layer
LVS SOFTCHK lower_layer CONTACT
```

```
SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED
   upper_layer NOT OUTSIDE REJ_UPPER
   }
```

- Example 3

```
SCONNECT upper_layer lower_layer
LVS SOFTCHK lower_layer CONTACT ALL

SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED
   CONF_LOWER = lower_layer NOT OUTSIDE REJ_UPPER
   upper_layer NOT OUTSIDE CONF_LOWER
   }
```

- Example 4

```
SCONNECT upper_layer lower_layer ABUT ALSO
LVS SOFTCHK lower_layer LOWER ALL

SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED ABUT == 0
   lower_layer INTERACT REJ_UPPER
   }
```

- Example 5

```
SCONNECT upper_layer lower_layer ABUT ALSO
LVS SOFTCHK lower_layer CONTACT

SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED ABUT == 0
   upper_layer NOT OUTSIDE REJ_UPPER
   }
```

- Example 6

```
SCONNECT upper_layer lower_layer ABUT ALSO
LVS SOFTCHK lower_layer CONTACT ALL
SOFTCHK {
   REJ_UPPER = EXTERNAL lower_layer upper_layer == 0
      INSIDE ALSO REGION NOT CONNECTED ABUT == 0
   CONF_LOWER = lower_layer INTERACT REJ_UPPER
   upper_layer INTERACT CONF_LOWER
   }
```

- Example 7

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer LOWER  ALL
SOFTCHK {
   USED_CONTACT = upper_layer AND CONTACT_LAYER
   REJ_CONTACT = EXTERNAL lower_layer USED_CONTACT == 0
      INSIDE ALSO REGION NOT CONNECTED
   lower_layer NOT OUTSIDE REJ_CONTACT
   }
```

- Example 8

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer CONTACT

SOFTCHK {
   USED_CONTACT = upper_layer AND contact_layer
   REJ_CONTACT = EXTERNAL lower_layer USED_CONTACT == 0
      INSIDE ALSO REGION NOT CONNECTED
   contact_layer NOT OUTSIDE REJ_CONTACT
   }
```

- Example 9

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer UPPER
```

```
SOFTCHK {
   USED_CONTACT = upper_layer AND contact_layer
   REJ_CONTACT = EXTERNAL lower_layer USED_CONTACT == 0
      INSIDE ALSO REGION NOT CONNECTED
   upper_layer NOT OUTSIDE REJ_CONTACT
   }
```

- Example 10

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer CONTACT ALL

SOFTCHK {
   USED_CONTACT = upper_layer AND contact_layer
   REJ_CONTACT = EXTERNAL lower_layer USED_CONTACT == 0
      INSIDE ALSO REGION NOT CONNECTED
   CONF_LOWER = lower_layer NOT OUTSIDE REJ_CONTACT
   USED_CONTACT NOT OUTSIDE CONF_LOWER
   }
```

- Example 11

```
SCONNECT upper_layer lower_layer BY contact_layer
LVS SOFTCHK lower_layer UPPER ALL

SOFTCHK {
   USED_CONTACT = upper_layer AND contact_layer
   REJ_CONTACT = EXTERNAL lower_layer USED_CONTACT == 0
      INSIDE ALSO REGION NOT CONNECTED
   CONF_LOWER = NOT OUTSIDE REJ_CONTACT
   CONF_CONTACT = USED_CONTACT NOT OUTSIDE CONF_LOWER
   upper_layer NOT OUTSIDE CONF_CONTACT
   }
```

GDSII Datatypes and Texttypes in Calibre

If the layout database format for Calibre is GDSII, then datatypes and texttypes are ignored by default. However, processing of drawn geometry and text based on datatypes and texttypes is possible via Layer Map specification statements.

As an example, assume that datatypes 7-32 on layer 0 are to comprise a separate original layer, call it "forks". Then the following rule file statements would correctly define layer "forks":

```
LAYER forks 108
// Really layer 0, datatypes 7-32. Just picked 108.
LAYER MAP  0  DATATYPE  >= 7 <= 32  108
```

As with all rule file entities, there are no restrictions on the relative ordering of the Layer and Layer Map specification statements.

When the DATATYPE keyword is present, we call a Layer Map specification statement a *datatype map*. When the TEXTTYPE keyword is present, we call it a *texttype map*. Texttype maps work for GDSII texttypes and layers specified in Text Layer specification statements in the same way that datatype maps work for GDSII datatypes and original layers defined in Layer specification statements.

Specifically, for datatype maps, when reading a geometry G on layer L with datatype D, the GDSII stream reader will proceed as follows. If for any datatype map

(source-layers, source-types, target-layer)

L is in source-layers and D is in source-types, then for each datatype map

(source-layers, source-types, target-layer)

such that L is in source-layers and D is in source-types, geometry G will be treated as if it were on target-layer; if G is so mapped, then it is no longer treated as being on layer L unless L is a target-layer in one of G's datatype maps. If G is in no datatype map, then it is treated as being on layer L regardless of datatype.

Datatype maps are completely not the same as layer sets.

As another example, assume that all geometries having datatype 1 are to be ignored regardless of layer. Then, assuming that layer number 1000 is not used otherwise, the following effectively discards any geometry having datatype 1:

```
LAYER MAP  > 0  DATATYPE  1  1000
```

However, now assume that metal1 is layer 9 and that geometries with datatype 1 are to be ignored for all layers except metal1. Then

```
LAYER metal1 9
...
LAYER MAP  > 0  DATATYPE  1  1000
LAYER MAP  9  DATATYPE  1  9
```

A common problem when using layer maps are target layer numbers created for the sole purpose of the mapping which are, in fact, non-empty. For example, consider:

```
LAYER metal 100
LAYER MAP 6 DATATYPE 1 100
```

The intent is that metal is on layer 6, datatype 1, and the number 100 is "made up" because there needs to be a target layer number. Problems can arise if layer 100 in the input layout database contains geometry not intended to be used, but which will be placed on layer 100 by default as per the mapping algorithm described above. A simple solution is to map objects on layer 100 to an unused layer. Since layer maps are not always that carefully designed, a warning will be issued for any GDSII layer which:

1. is required by the execution flow
2. is the target layer of a datatype map
3. contains objects which themselves are not mapped by any datatype map.

GDSII/CIF Input Control in Calibre

GDSII input. The following GDSII records are processed by Calibre applications for a GDSII-type input layout database:

HEADER BGNLIB LIBNAME UNITS ENDLIB
BGNSTR STRNAME ENDSTR
BOUNDARY
PATH PATHTYPE WIDTH BGNEXTN ENDEXTN
XY COLROW

LAYER DATATYPE
SREF AREF SNAME
TEXT TEXTTYPE STRING
STRANS MAG ANGLE
PROPATTR PROPVALUE
ENDEL

BOX and BOXTYPE records can be processed and treated exactly like BOUNDARY and DATATYPE records, respectively, if the specification statement Layout Process Box Record YES is present in the rule file.

CIF input. The original Mead/Conway BNF is adhered to except for the following extensions and limitations:

- The user extension command "9" immediately following a "DS" command will define the cell name associated with the symbol number.
- Implicit commands "P", "B", "R", "W", "C", and any implicit user extension commands are not processed. An implicit command is defined as one outside of "DS" ... "DF".
- Commands "R" (round flash) and "DD" (definition delete) are not processed.
- User extension commands "4N", "94", "4M", and "4X" are interpreted as text objects with the following syntax:

  4N/94 string sinteger sinteger
  4M string integer point point string
  4X string integer point integer string string

- CIF layer names must be resolvable (that is, defined) in the rule file. Objects will not be added to unresolvable CIF layers. As an example rule file definition using an alias:

```
LAYER METAL1 M1   // METAL1 is the name I really want to use.
LAYER M1 12       // The way it's defined in the CIF file.
```

Layout Depth Control. If the input layout database format is GDSII/CIF, the layout depth for geometries may be specified via the Layout Depth specification statement.

Handling Duplicate Cells

The Layout Path specification statement may be specified with multiple file name parameters and any number of times. This allows multiple GDSII/CIF input databases to be used. Multiple input databases are treated as if all the structure records (for GDSII) or symbol definitions (for CIF) were embedded in the first file specified. Each input database file, however, is expected to be syntactically complete.

Whether or not multiple files are specified for the input layout database, multiple records for the same layout cell are not allowed by default. All records after the first are discarded (with a warning or error). The user may control this behavior with the Layout Allow Duplicate Cell[s] specification statement.

If YES is specified, then multiple cell records are treated as if the constituent data were concatenated into a single record and there is no warning or error. This is useful, for example, when the database is split into multiple files by layer, not cell.

Wildcards in Layout Primary

The Layout Primary (and Layout Primary2, discussed later) specification statement argument may contain one or more wildcard (*****) characters. This allows the system to attempt a degree of auto-recognition of the top-level cell in a GDSII or CIF input layout database. The recognition semantics are as follows: If there is a literal match between a cell name and the Layout Primary parameter, then that cell becomes the top-level cell. If there is no literal match and the Layout Primary parameter contains one or more wildcards, then the system assembles a list of candidate top-level cells in order of their appearance in the input stream. A candidate top-level cell is defined as any unplaced (that is, unreferenced) structure. The first such candidate whose name matches the Layout Primary value according to the usual rules of cell name wildcard matching will be selected as the top-level cell; a warning is issued in this event. If there is still no match, then a fatal error is issued.

Database Pre-merging

A layout database which originated from a module generation program, for example, may often contain massive numbers of overlapping geometries on a single layer within each cell. If merged, however, this number may be dramatically reduced. It is often desirable to merge on a per-layer, per-cell basis prior to merging the flat representation, which is done automatically by the verification system for essentially every original layer. For example, assume that on layer L in cell C there are N geometries, which, if merged, would be M geometries where M << N. If cell C has P flat placements, then the total number of geometries due to the flattening of layer L from cell C is NP. If P is large and M << N, then this can be reduced considerably, to MP. Merging of the flat layer will then be much more efficient and memory will be saved. This is accomplished with the Layout Merge On Input YES specification statement.

The input layout database format must be GDSII or CIF for the merging to occur. In cases such as the one above, the time for merging of the flattened layers in flat applications can be considerably reduced, at the expense of slightly more time to read the input layout database. The default is NO, that is, merging will not occur. YES should generally not be specified if the input layout database lacks the aforementioned characteristics.

Cell Renaming

The user may also specify cells that are to be renamed as the input layout database is being read. This is done with the Layout Rename Cell specification statement. This is especially useful in establishing cell correspondences for dual database capability in hierarchical applications. This is discussed later in this chapter.

Cell Exclusion

All Calibre applications have the capability of excluding one or more layout database cells from processing. This is important in excluding alignment markers, trademarks, memory cores, and so on. Excluding a cell means that no objects from any placement of the cell will be processed by the application; this includes all hierarchy within the placement.

In Calibre, cell exclusion is controlled by the presence of Exclude Cell specification statements. These collectively specify the set of cells to be excluded. If no such specification statements are present, then no cells are excluded. Cell exclusion is not supported for binary and ASCII input layout database formats for Calibre. The wildcard character "*" may be used within the cell names.

Area-based Filtering in Calibre

In Calibre applications, area filtering of layout database objects is supported via the following three specification statements: Layout Window, Layout Windel, and Layout Window Clip. See the *SVRF Manual* for details.

Flagging and Snapping Original Geometries in Calibre

Original geometry *flagging* causes warnings to be generated when original geometries which fail certain constraints are read from the layout database (the original geometries are still processed, however). Original geometry *snapping* aligns vertices of original geometries on specified grids.

Flagging of original geometries is controlled by the rule file specification statements Flag Acute, Flag Skew, Flag Offgrid, Flag Nonsimple Polygon, and Flag Nonsimple Path. The default for all statements is NO. They are fully discussed in the *SVRF Manual*.

The error-directed auxiliary operations Drawn Acute, Drawn Skew, and Drawn Offgrid provide an alternative method to flag original geometries than that provided by the corresponding Flag... statements. Each method checks the same set of original database geometries and flags the same problems. The operations produce DRC results which can be visually realized by DRC results presentation commands. The warnings generated by the FLAG... statements, however, include layer and cell name information, which is not attached to DRC result database objects.

Remember that original geometry flagging, whether done via error-directed auxiliary operations or by FLAG...YES (or both), only checks geometries on original layers which are actually read from the database in the verification run. These original layers are only those actually required by the application (that is, layers which are required for layer derivations and rule checks), not necessarily

the entire set of original layers referenced in the rule file. In particular, resolution overrides for original layers specified in Layer Resolution specification statements do not apply unless the original layer is literally required by the application—layer sets should not be built simply to override the default resolution for a group of layers.

If Snap Offgrid YES is specified in the rule file, then original layer geometries read by Calibre are snapped to the grid specified in the Resolution specification statement or, if present, the grid specified in the Layer Resolution specification statement for the given original layer. Snapping occurs prior to any acute, skew, or off-grid flagging. Snapping preserves 45-degree angles on edges between two orthogonal edges if the snap resolution has equal x- and y-values.

For hierarchical applications, placements are also snapped to grid if Snap Offgrid YES is specified. Geometries are then snapped on a per-cell basis. The resolution for placement snapping is the least common multiple of all grid values specified in applicable Resolution and Layer Resolution specification statements. This insures that geometries can be snapped on a per-cell basis and cannot become off-grid (in the flat view of the input layout database) due to an off-grid placement. For resolutions where the x- and y-values are unequal, snapping of geometries is always to the least common multiple of the two values.

Finally, in DRC-H, DRC Map Text objects are snapped to the grid specified in the Resolution specification statement if Snap Offgrid YES is specified. These text objects are treated as single-point geometries and snapped in the same manner as geometries, as described previously. No other text objects are snapped.

Input Layout Database Magnification

The input layout database may be magnified as it is read into Calibre applications via the Layout Magnify specification statement. This statement specifies that the input layout database is to be magnified by the given value as it is being read into the Calibre application. The magnification algorithm simply multiplies all coordinate data (including placement base points—for GDSII and CIF—and array pitches for GDSII) by the given value, regardless of its hierarchical position. This algorithm is completely equivalent (disregarding mathematical round-off error) to placing the entire input layout database at the given magnification into a new top-level cell; the difference is that the new cell is not explicitly created.

In certain applications, magnification of the input layout database is used as a tool to increase the database precision. In such usage, it is recommended that the rule file precision be modified accordingly so that dimensioned quantities in the rule file agree with the new precision. For example:

```
LAYOUT MAGNIFY 10
// Increase database precision for some reason
PRECISION 10000
// Was 1000. Better increase that also.
```

Binary Layout Database Writing

DRC applications can be directed to write the layout database into binary polygon files in the current directory. (These binary polygon files are equivalent to the binary layout database input format for Calibre DRC, each having the file name

```
icv_data_<number>
```

where <number> represents the corresponding drawn layer number.)

The layers which are written are exactly those simple layers, and the appropriate geometric subset thereof, which would have been read from the layout database had the check set been executed (except for geometries specified in Polygon specification statements). Each simple original layer read becomes a binary polygon file. No rule checks are actually executed if the database is being written in this manner. Geometry flagging of acute angles, skew edges, and offgrid vertices is also disabled.

This is performed by specifying the command line switch -writedatabase. This switch is not supported in Calibre DRC-H.

Chapter 5
DRC Execution

The following command lines invoke Calibre DRC and Calibre DRC-H, which operate on the specified rule file:

```
calibre -drc [-writedatabase] rule_file_name
calibre -drc -hier rule_file_name
```

For more information on command usage of Calibre DRC / DRC-H, see "Calibre DRC/DRC-H" in chapter 2, "Invocation".

Rule File Compilation

You must specify a rule file to invoke Calibre DRC applications. To do this, you provide the name of the rule file as an argument to the invocation command. Calibre DRC automatically compiles the rule file as the first step of executing the command. Compilation errors abort the program. Compilation errors are discussed in the *SVRF Manual*.You can check syntax of statements in the rule file before running DRC by using the rules_syntax_checker utility. For more information on this utility, refer to the section "Rules Syntax Checker" in chapter 12.

Rule Check Selection

You can select one or more rule check statements from a rule file, which Calibre DRC will run as a unit. This unit is called the *check set*.

By default, Calibre DRC selects all rule checks in the rule file during compilation. You can use the DRC Select Check and DRC Unselect Check specification

statements to control compile-time inclusion of any rule checks. Calibre DRC selects rule checks when it compiles the rule file as follows:

1. If there are no DRC Select Check specification statements in the rule file, Calibre DRC selects all rule checks. Otherwise, Calibre DRC selects only those rule checks specified in DRC Select Check specification statements.

2. Calibre DRC does not select any rule checks specified with DRC Unselect Check specification statements in the rule file.

General Execution Characteristics

This section discusses some of the general characteristics of Calibre DRC execution. Much of the discussion applies in general to all verification applications.

Concurrency

Calibre DRC performs many of the layer operations concurrently. This means that they run together, when required, simultaneously. Whenever Calibre DRC performs layer operations, it locates all required layer operations within the set that it can run concurrently and executes them as a single group.

Calibre executes the following layer operations concurrently:

- All dimensional check operations with one input layer that have the same input layer.

- All dimensional check operations with two input layers having the same two input layers in either order.

- All unconstrained polygon topological operations having the same two input layers in either order, such as:

  (Not) Cut — (Not) Enclose

  (Not) Inside — (Not) Interact

  (Not) Outside — (Not) Touch

- All constrained polygon topological operations having the same two input layers in a given order, such as:

  (Not) Cut (Not) Enclose

  (Not) Interact (Not) Touch

- All (Not) Rectangle operations having the same input layer.
- All other polygon measurement operations having the same input layer.
- All Expand Edge operations having the same input layer.
- All Path Length operations having the same input layer.
- All Convex Edge operations having the same input layer.
- All topological operations having the same two input layers in either order:

  (Not) Inside Edge (Not) Outside Edge

  (Not) Coincident Edge (Not) Coincident Inside Edge

  (Not) Coincident Outside Edge (Not) Touch Edge

  (Not) Touch Inside Edge (Not) Touch Outside Edge

- All (Not) Length, and (Not) Angle, operations having the same input layer.
- All (Not) With Edge operations having the same two input layers in a given order.
- All (Not) Net operations having the same input layer.
- All Net Area operations having the same input layer.
- All Net Area Ratio operations having the same set of input layers, not including the ACCUMULATE layer, and, if specified, the same ACCUMULATE layer.
- All Density operations that have the same input layer, boundary, window, and step parameters.

Calibre DRC generally execute rule checks in the order they occur in your rule file. When the executive routine “sees” that it is finished with a particular layer (that is, no more rule checks are to be performed on a layer), that layer is flushed from memory. You can optimize your rule file and your run times by placing all rule checks for given layers sequentially in the rule file. This is discussed in more detail in the following sections.

Redundancy Elimination

Calibre DRC combines all identical layer operations during execution. Operations are identical if they have the same keyword, arguments and input data. For example, in the following rule check statements, Calibre DRC recognizes that the operations defining X and P are identical as well as those defining Y and Q:

```
ABC {
   X = poly and diff
   Y = X inside metal
   …
   }

DEF {
   P = diff and poly
   Q = P inside metal
   …
   }
```

Note that you can duplicate layer operations within a rule file without affecting compilation or execution. This is true whether the duplicated operations are within the global scope (outside of rule check statements), the local scope (inside of rule check statements), or both. For Calibre DRC applications, you do not need to use the global scope to avoid duplicate execution of layer operations.

Layer Operation Scheduling

Calibre DRC applications automatically schedule all layer operations necessary to satisfy any data requirements of a layer operation.

For example, assume that a layer operation required by a check set refers to some layer A defined in the rule file. Calibre DRC locates the layer definition defining

A and evaluates the defining operation. This is a recursive process that proceeds through the entire tree of operations that Calibre DRC must evaluate to produce A. The process stops when Calibre DRC retrieves data for the original layers from the database.

Calibre DRC schedules all involved layer operations for concurrency and to undergo redundancy elimination, as described above. If a layer operation is required to evaluate a check set, Calibre DRC does not perform that operation more than once during execution of the check set. Calibre DRC performs only those operations necessary to satisfy the data requirements of a check set.

Maximizing Capacity and Minimizing Execution Time

DRC applications produce derived layer data during the course of operation and delete all original and derived data when they are no longer required. Some systems produce all derived layer data “up front” and delete all of the data when the run is complete. Calibre DRC attempts to maximize capacity by delaying data production until it is required and by deleting data when it is no longer required.

DRC applications proceed as follows:

1. Perform one database scan and generate all original layers and layer sets required by the check set.

2. If the check set requires connectivity extraction, perform the Mask mode Connect and Sconnect operations in the order of their appearance in the rule file.

3. Execute the rule check statements in the check set in the order of their appearance in the rule file. Executing a rule check statement is equivalent to generating the derived layers represented by all output operations within the rule check statement (in order within the rule check), and mapping each derived layer to the DRC results database.

Given the above order, the following facts apply to generating derived layers:

- Calibre DRC never generates a derived layer, including one produced by an output operation within a rule check statement, until the layer is actually

needed. The exception to this is if Calibre DRC generated the layer earlier according to Concurrency.

- All layers, original or derived, persist only until they are no longer required, at which point Calibre DRC deletes them.

These data generation guidelines provide you with latitude in arranging the order of rule check statements in the rule file so as to maximize capacity.

Conjunctive Checks

Calibre DRC performance depends on the amount of data Calibre DRC must use at any given time. In designing certain conjunctive checks, for example, you can reduce execution time. Consider the following design rule:

> All metal contacts must be inside of metal and, in addition, inside of either polysilicon or diffusion.

This rule states that metal contacts connect metal to polysilicon or metal to diffusion. One way to check this rule is as follows:

```
metal_contact_check {
   x = poly or diff
   y = x and metal
   contact not inside y
   }
```

The problem with this approach is that, in a large design, Calibre DRC may generate potentially large amounts of data in the intermediate layers x and y. As a result, the AND and Not Inside operations are slower than necessary. Although correct, you can alter this check to speed up the process by decreasing the amount of intermediate data generated. The method below achieves this and is as accurate as the above check:

```
metal_contact_check {
   bad1 = contact not inside metal
   x = contact not inside diff
   bad2 = x not inside poly
   bad1 or bad2
   }
```

The OR operation unites any duplicate objects in the error layers `bad1` and `bad2`. Since objects in `bad1` and `bad2` are errors (these layers should be very small, or empty) the final OR operation will be negligible. The only other intermediate layer is x, whose size is probably on the order of 1/3 of the size of the original contact layer. (In practice, however, the Outside Also option for these type of contact checks is much more efficient).

Concurrency Checks

One of the most straightforward ways to reduce execution time is to use concurrency. In general, if Calibre DRC performs N operations concurrently, run time is close to that for one of the operations. For example, consider the following:

```
regular_contact = contact outside pad
pad_contact = contact not regular_contact
```

This involves two potentially expensive operations. An equivalent method introduces concurrency and reduces the time required for one of the above operations:

```
regular_contact = contact outside pad
pad_contact = contact not outside pad
```

Concurrency can introduce the greatest savings in dimensional check operations, which are usually the most time-consuming and because the amount of concurrency is open-ended. For example:

```
gate = poly AND diff
rule_A {
   external poly diff < 3
   }
rule_B {
   enclosure gate poly < 2
   }
```

The output operations in `rule_A` and `rule_B` represent two potentially time-consuming operations. However, because of polygon containment criteria, there is no reason to use the gate layer in `rule_B`. The layer `diff` would work as well and introduce concurrency, as follows:

```
rule_A {
   external poly diff < 3
   }
rule_B {
   enclosure diff poly < 2
   }
```

Rectangle Checks

Another general rule for avoiding excessive run time is never to use a very large design rule constraint in a dimensional check operation on a layer whose average feature size is much smaller than the rule's constraint, or where the layer has a high density of polygons. This can slow DRC down by orders of magnitude. For example, consider the design rule:

> Contacts must be rectangles of width greater than or equal 1.2 and less than or equal to 90.

One way to check this is the following:

```
contact_size {
   internal (contact) >= 1.2 <= 90    // Output edges where
                                      //  width is bad.
   }
```

The combination of the huge constraint value of 90 compared with the large contact density will cause the dimensional check operation to take excessive time. A better way to check the design rule is as follows:

```
contact_size {
   not rectangle contact >= 1.2 <= 90 by >= 1.2 <= 90
   }
```

Pad Checks

Another way to reduce run time (applicable mostly to flat, not hierarchical DRC applications) is in the area of pad checks on a large design. For example:

```
Rule_5 {
   external pad metal2 < 32
   }
```

Flat Calibre DRC is not optimized to ignore the potentially large number of `metal2` shapes that are in the middle of the circuit and that can have no possible interaction with the pads. Because a polygon topological operation is faster than a dimensional check operation and because there are generally very few pad shapes, the following rule can offer significant speed improvement (at the expense of additional lines of code):

```
Rule_5 {
   X = pad size by 32.1 //make pads larger by 32.1 units
   Y = metal2 not outside X //any metal 2 touching X?
                            //probably not much
   external pad Y < 32      //now do the external check
   }
```

Operation Execution Time

The following is a rough order of comparing the run time of layer operations for flat verification.The order below is slightly different for hierarchical verification, as discussed in the section "Hierarchical Operation Efficiency" in chapter 6, "Hierarchical DRC". This order is not 100 percent applicable for all data configurations, but it does give a general idea of the relative speed of these operations. The slowest operations are at the top, the fastest at the bottom:

- Density (can run faster depending on design)
- Internal, External, Enclosure
- (Not) Touch Edge, (Not) Touch Inside Edge, (Not) Touch Outside Edge
- (Not) Inside Edge, (Not) Outside Edge, (Not) Coincident Edge, (Not) Coincident Inside Edge, (Not) Coincident Outside Edge, (Not) Enclose, (Not) Touch, (Not) Interact, (Not) Cut (Constrained), (Not) With Edge (different layer of origin), Convex Edge
- AND (two layer), OR (two layer), NOT, XOR (two layer), (Not) Inside, (Not) Outside, (Not) Cut (unconstrained), Stamp, (Not) With Text
- Size, Snap, Expand Edge, Holes, Extents, (Not) Length, (Not) Angle, AND (one layer), OR (one layer), XOR (one layer)

- Path Length, Perimeter, Vertex, (Not) Donut, (Not) Area, Net Area, Net Area Ratio, (Not) Rectangle, (Not) With Edge (same layer of origin)
- Copy, Polynet, Rectangles, Shift, Magnify, Rotate, Extent, (Not) Net, Offgrid

Polygon Segmentation

Calibre DRC places no limit on the vertex count for polygons on original layers or on derived polygon layers. For some database formats, you should limit the vertex count of polygon results. For example:

- The IC Station template database has a vertex limit of 4096.
- The GDSII database format has a vertex limit of 200.

Calibre DRC contains a polygon "segmenter" that breaks up result polygons whose vertex count exceeds a certain value. The segmenter breaks up a polygon vertically, producing polygons with sufficiently small vertex counts. The merged representation of the segments is equal to the original polygon. Whenever a polygon result is segmented, Calibre DRC issues a warning—once per rule check.

By default, Calibre DRC segments any DRC polygon result whose vertex count exceeds 4096. You can control the maximum vertex count for polygon segmentation with the DRC Maximum Vertex specification statement.

Calibre DRC uses the specified value even though some database formats are specified to contain polygons with fewer vertices. This may present a problem if you use such a database in IC Station.

Calibre DRC computes DRC result counts after applying any polygon segmentation. These result counts appear in the DRC results database or in various DRC messages.

Polygon Segmentation in Calibre DRC

By default, Calibre DRC segments any DRC polygon result whose vertex count exceeds 4096. You can control the maximum vertex count for polygon

segmentation via the rule file DRC Maximum Vertex specification statement. ALL simply denotes a very large number (2147483647 to be exact). For example, if the DRC results database type is GDSII, then DRC MAXIMUM VERTEX 199 can be desirable since the GDSII vertex "limit" is 200 and GDSII format requires the last vertex in a polygon to duplicate the first one. (If the DRC results database type is GDSII, a warning is issued if there is no DRC Maximum Vertex specification statement in the rule file or, if present, its value exceeds 199; however, polygons will not be segmented simply on the basis of the results database format being GDSII). The user can even segment DRC polygon results into quadrilaterals by using DRC MAXIMUM VERTEX 4

A word of caution. If the value specified in DRC Maximum Vertex exceeds 4096, Calibre DRC will use this value even through ASCII DRC results databases, for example, are specified not to contain polygons with more than 4096 vertices. This would definitely present a problem if such a DRC results database was later loaded into IC Station (via REStore DRc Results, for example).

Layout Database End Segment Warning

The Calibre layout data input module will issue a warning for any non-extended path type such that either end segment length is less than 1/2 of the path width. The warning is issued because the expanded path may have a notched corner near the short end segment. The warning includes the path's layer, cell, and one of the end segment's coordinates. This applies to GDSII and CIF databases, and is issued in both flat Calibre DRC and DRC-H.

Chapter 6
Hierarchical DRC

Calibre DRC-H is a *fully-hierarchical* DRC application. Unlike flat verification applications, which completely flatten the input database and operate on the resulting flat geometries, Calibre DRC-H maintains the database hierarchy. This exploits the hierarchy to reduce processing time, memory usage, and DRC result counts.


Note

Calibre LVS and xCalibre also have hierarchical versions. Much of the information in this section applies to those applications also.

Calibre DRC-H uses the same rule file as its flat counterpart, Calibre DRC. Therefore, you do not need to add or remove statements. Calibre DRC and Calibre DRC-H are nearly identical, except for hierarchical processing algorithms and the reduction of design rule errors through hierarchical error suppression.

Calibre DRC-H imposes no design restrictions concerning geometry overlapping cell placements or overlaps of cell placements.

Theory of Operation

Flat verification applications work from a flat database representation only. For GDSII and CIF input, Calibre flattens the layout database up front and maintains no record of the original database hierarchy.

Hierarchical Calibre applications, by contrast, do not flatten the input database. Instead, Calibre maintains the database hierarchy throughout processing. It stores geometry (both original and derived) only once in the cell with which it is associated instead of replicating every flat placement of the cell. Each operation uses this hierarchical information to minimize the redundant processing that

occurs in a flat system when Calibre analyzes data within a cell repeatedly for all flat placements of the cell.

For each cell, every operation determines which portion of the data Calibre can analyze, independent of the placements of the cell. Calibre analyzes this subset of data only once, regardless of the number of placements of the cell, and promotes the remaining data up the hierarchy, until accurate analysis within context is ensured.

Storing, analyzing, and processing data once per cell, instead of for every flat placement of the cell, can generate significant performance improvements and greatly reduce memory requirements. Figure 6-1 provides a simple illustration of the idea behind hierarchical processing, using a two-layer Boolean AND operation

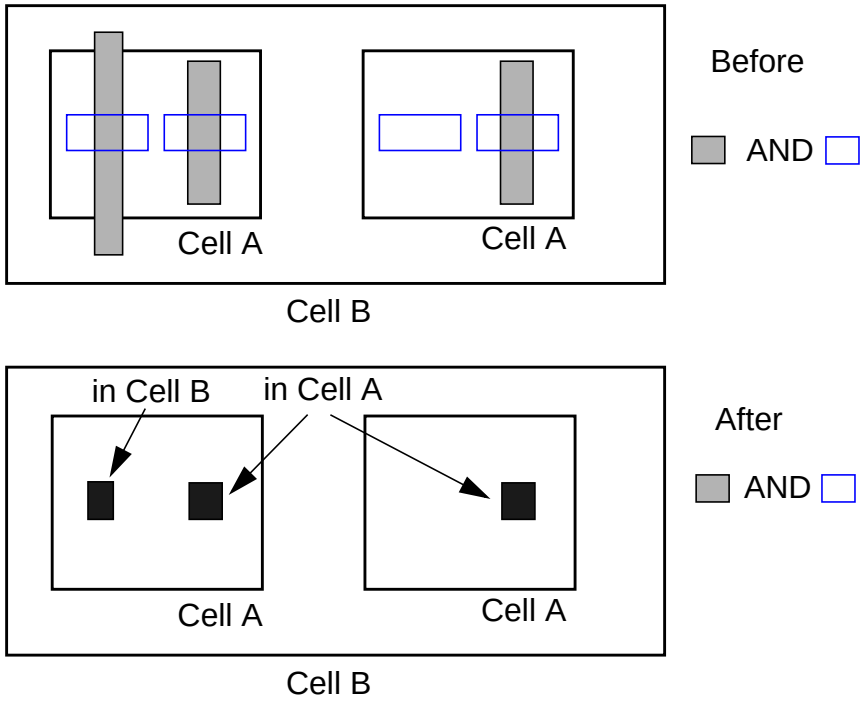


Figure 6-1. Hierarchical AND Operation

Cell B contains two placements of cell A and a geometry that overlaps one placement of cell A. You can combine the two right-hand geometries in cell A with an AND operation on a cell-specific basis in cell A. Calibre stores the result

once in cell A, irrespective of the fact that cell A is placed twice. The context of a placement of cell A affects only the left-hand geometry in cell A. Calibre promotes this geometry to ensure an accurate AND operation, and the result will become a geometry in cell B

The performance of hierarchical Calibre applications, as compared with their flat counterparts, depends on the amount of repetition within the design. In addition, determining per-cell data subsets that Calibre can independently analyze versus those that it must process in context, represents overhead not present in the flat system. On a majority of designs, you can realize significant speed increases. This ranges from 2x for many MPU designs to orders of magnitude for memory designs. Hierarchical processing can similarly reduce memory requirements because Calibre stores data once per cell, instead of replicating it for every flat placement of the cell.

DRC Data Storage

An important corollary of hierarchical processing is that Calibre DRC-H maintains data (original or derived layer geometry) at the lowest possible hierarchical level. For original layer geometries, this means that Calibre stores the data once with the cell (just as in the original user design) as opposed to flattening to the top level, that is, replicating for every flat placement of the cell.

Layer operations that generate derived layer geometries attempt to analyze them within each cell, promoting only when necessary to examine data in context. Calibre then creates the derived layer geometries at the lowest hierarchical level and, as with original layer geometries, stores geometries once within the cell in which they were created. Because DRC results are elements of derived layers mapped to the DRC results database, this implies that Calibre DRC-H incorporates a natural error suppression device.

For example, consider the design in Figure 6-2.

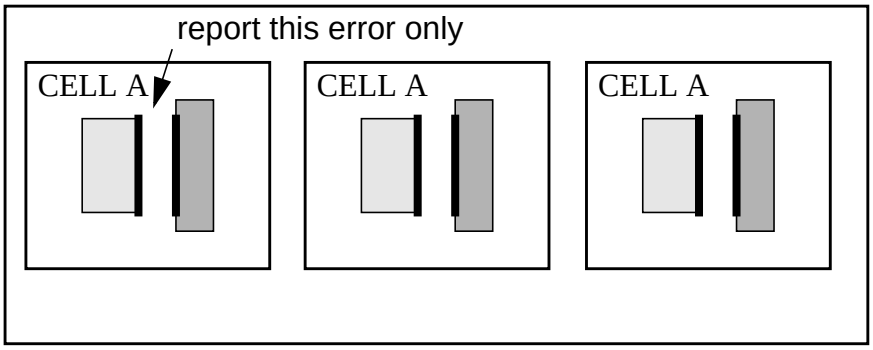


Figure 6-2. Hierarchical Error Suppression

The design consists of three placements of cell A, which contains a spacing error between the two shapes. Calibre DRC-H generates the error when analyzing cell A; the error is independent of context. Therefore, Calibre DRC-H stores the error geometry only once, namely in the cell template of A and reports the error only once. A non-hierarchical DRC application flattens cell A, resulting in six geometries in the top-level cell and three separate errors. In this example Calibre DRC-H implicitly recognizes that two errors are essentially repeated errors.

Calibre DRC-H generates an ASCII or binary DRC results database with the same format as its flat counterparts. Calibre DRC-H also transforms all DRC results into the top-level cell space, consistent with the flat applications. Errors suppression occurs as follows:

- Assume that DRC result R is a derived geometry in cell A.
- If cell A is the top-level cell, Calibre DRC-H writes result R to the DRC results database exactly as in the flat systems.
- If cell A is not the top-level cell, then Calibre DRC-H chooses one placement of cell A throughout the entire hierarchy. Calibre DRC-H transforms R to the coordinate space of the top-level cell using the to-world transform of that placement and then writes R to the DRC results database.
- The placement of A chosen is the one that yields the lower-leftmost placement of cell A in the flattened hierarchy.

The only difference in the ASCII or binary DRC result databases that Calibre DRC-H and Calibre DRC generate is that, in the former case, there will normally be fewer DRC results.

Flat Instantiations

Calibre DRC-H supports layers in both hierarchical and flattened form. In the hierarchical form, Calibre DRC-H stores geometry, both original and derived, once in the cell where it is associated, instead of replicating for every flat placement of the cell.

A layer can exist in one of three forms:

- As an exclusive hierarchical instantiation. The layer exists in hierarchical form only.
- As an exclusive flat instantiation.The layer exists in flattened form only.
- As a dual instantiation. The layer exists in both flat and hierarchical form simultaneously.

Calibre DRC-H creates flat instantiations for layers in one of three ways:

- The result layer of the Flatten operation has an exclusive flat instantiation.
- The result layer of a layer operation that is not supported hierarchically, or has an input layer with a flat instantiation, has an exclusive flat instantiation.
- If an input layer to an operation is not supported hierarchically, or another input layer has a flat instantiation, Calibre DRC-H converts to a dual instantiation. Another possibility is that Calibre DRC-H may create the input layer a temporary flat copy.

Note that any flattening required to support non-hierarchical layer operations or coexistence of flat and hierarchical instantiations is completely automatic.

Calibre DRC-H explicitly flattened a layer with the Flatten operation.

Hierarchical Operation Efficiency

The following differences exist in the efficiency of all layer operations in the verification suite and in hierarchical Calibre applications:

- Some of the more rarely-used operations including (Not) With Edge, Holes, Extents, Perimeter, Vertex, (Not) Donut, (Not) Enclose Rectangle, and the Size operation with OVERLAP ONLY specified, internally create a merged copy of the input layer, resulting in a merged form of the output layer. These commands can be relatively slow if an input layer contains large polygons distributed throughout the hierarchy.

- Calibre DRC-H performs Magnify, Rotate, Density, Polynet, Ornet, the one-layer Boolean operations AND (with a constraint other than >= 1 or > 1) and XOR, flat. This results in an exclusive flat instantiation of the output layer.

- Two-layer Boolean operations are generally faster than polygon topological operations.

- Calibre DRC-H also performs Angle flat whenever the measurement constraint includes zero (0) but not 90, and vice-versa.

- The Shift operation can be performed in a cell if all of the placements of that cell, in the flat viewpoint of the design, have a consistent rotational/reflectional transformation component. Otherwise, input geometry must be promoted up out of the cell to the lowest hierarchical level having the aforementioned characteristics. This insures that the result of the Shift operation is correct from the flat viewpoint. However, the hierarchical implementation of the operation can be exceedingly slow in designs where transformational consistency is reached only near the top level. It may be more efficient in such cases to flatten the input layer and perform the operation flat.

- Dimensional check operations External, Internal, and Enclosure are slower when using a constrained PROJECTING filter, an interval constraint without OPPOSITE, a NOT PROJECTING filter, a NOTCH or SPACE filter, or a CONNECTED or NOT CONNECTED filter.

- The Rectangles operation is more complex in hierarchical mode as compared to flat mode, and is one of the more costly hierarchical operations.

False Notch Error Suppression

In rare cases, Calibre DRC-H can, report a false notch violation in a one-layer external check that flat Calibre DRC would not report. Calibre DRC-H generally reports this false notch error across a bend in a polygon that traverses hierarchy. The flat system uses a post-processing algorithm to find such false notch errors. In the hierarchical system, this post-processing algorithm might not find a false notch error if the geometry producing the error has certain hierarchical characteristics.

EXCLUDE FALSE NOTCH is a secondary keyword to the one-layer External operation. This keyword instructs the hierarchical operation to apply extra effort to minimize or eliminate the possibility of false notch errors. This adds approximately 10-20 percent extra runtime to the operation. You should use this instruction *only* to suppress false notch errors that Calibre actually generates. That is, because the possibility of false notch errors is low, you should not increase runtime unless required. In fact, a minimal number of false notch errors can be acceptable in certain cases. The DRC Exclude False Notch specification statement essentially applies the EXCLUDE FALSE NOTCH secondary keyword to each one-layer External operation.

Layer Area Printing

When first generating the layer, flat applications print the total area of a polygon-type layer in the transcript with other related statistics. Hierarchical applications do not do this by default because of the time it requires. However, you can specifically request this information with the DRC Print Area specification statement.

This statement directs hierarchical Calibre applications to print the flat area of a layer when generating the layer. This prints the area, along with other relevant

statistics for the layer. Each <layer> can be the name, not number, of an original layer or derived polygon layer.

Text Mapping

You can use the DRC Map Text specification statement to cause Calibre DRC-H transfer *all* text objects in the input layout database to a GDSII-type DRC results database. This transfers all text objects in the input layout database to a GDSII-type DRC results database. The text objects have the same hierarchy in both the input and results databases, unless Calibre DRC-H expanded or flattened a placement during hierarchical processing. In that case, Calibre DRC-H moved the text up the hierarchy, as appropriate.

Calibre applications do not retain TEXTTYPE properties from GDSII input layout databases, and all mapped text in the DRC results database has a TEXTTYPE of 0. If Calibre maps text with the Layer Map specification statements, then text in the DRC results database will be on the target layer(s) of the mapping.

Additional Hierarchy-specific Statements

There are additional specification statements that hierarchical applications use exclusively: Expand Cell, Flatten Cell, Layout Top Layer, Push Cell, and Layout Base Cell. By using one or more of these statements, you can dramatically improve the performance of the hierarchical Calibre application.

Layout Top Layer is highly recommended, and has the benefit of being process-specific, not design-specific. This statement enumerates a set of “high-level” original layer names. On designs with many levels of interconnect, the Layout Top Layer specification statement can enhance internal decision making related to the design style and similar optimizations. You should include all metal and via layers from first metal up. You should also include everything except core device layers and contacts, such as; pads, solder bumps, and cell boundaries.

For example:

```
LAYOUT TOP LAYER M1 V1 M2 V2 M3 V3 M4
LAYOUT TOP LAYER PAD BUMP BNDRY
```

When appropriate, include any new original layers defined for dual database capability as Layout Top Layer.

DRC Use of Hcells

The Hcell specification statement is used by Calibre LVS-H to denote the correspondence between cells in the layout and cells in the source (schematic, for example).

The initial phase of all hierarchical Calibre applications involves construction of an internal hierarchical database from the original input layout database. The original database is modified in a number of ways for optimal performance of Calibre's algorithms. Most notably, certain cell placements are automatically expanded and new cells and placements are automatically created. Automatic placement expansion, in particular, occurs for many reasons, all of which are intended to optimize the hierarchy for Calibre algorithms. However, expansion of a layout hcell can obviously introduce problems for Calibre LVS-H. Therefore, the hierarchical database construction phase for Calibre LVS-H will not automatically expand any layout cell in a rule file Hcell specification statement regardless of the performance cost.

In many application flows for Calibre DRC-H, the purpose is database modification, not necessarily traditional DRC checking. The output database(s) created by Calibre DRC-H may be subject to future LVS verification. In that event, it is not desirable that the DRC-H portion of the process expand cells which may later be designated in the LVS portion as hcells. Therefore, DRC-H will also inhibit expansion of any layout cell in a rule file Hcell specification statement, again regardless of the performance cost. The Hcell specification statement should only be in a rule file for a DRC-H execution if the desired flow is as described herein, that is, future presentation of the results to LVS. This statement should not be used simply to prevent automatic cell expansion by Calibre DRC-H.

Chapter 7
Connectivity Extraction

Connectivity extraction recognizes electrically connected regions in the layout called nets. Nets are recognized from layout geometries by analyzing the relations between layout shapes and other objects on various layers. The analysis is driven by rules you specify in the rule file. Connectivity extraction results are used internally by various components of the Calibre tool set.

Each electrical path, or net, is given a unique node number for identification during connectivity extraction. In addition to the node number, the net may also be named. The results of connectivity extraction are used by Calibre LVS to compare connections appearing in the layout against the schematic netlist, by Calibre DRC to perform rule checks involving connectivity, and Calibre RVE to display nets in your layout editor. Connectivity extraction also checks several connectivity-related criteria in the layout and issues error messages if such criteria are violated.

The connectivity extraction operations are:

- Attach
- Connect
- Label Order
- Sconnect
- Stamp

Establishing and Verifying Connectivity

Many operations require that the circuit connectivity of the design be established prior to executing the operation so that the correct connectivity information will be present on the input layers as needed.

For example, nodal dimensional check operations (that is, those having a CONNECTED or NOT CONNECTED filter), measure edges only if they belong to the same electrical nets or different electrical nets, respectively. In order to execute such an operation, the verification system automatically computes circuit connectivity beforehand using the connectivity extraction operations in the Mask set (discussed below). Parasitic extraction operations and device recognition also require circuit connectivity to be available or computable.

Mask Connectivity Extraction

Mask connectivity extraction is available in Calibre. It analyzes a layout hierarchy and extracts connectivity from mask-level geometries. It is only invoked as an internal subsystem by other components of the layout verification system: Mask LVS, Mask parasitic extraction, and Mask DRC. Mask LVS and Mask parasitic extraction store extracted connectivity in their own database on disk, thus allowing graphic highlighting of nets. DRC only uses connectivity information internally for connectivity-dependent checks.

Mask LVS, Mask parasitic extraction, and flat DRC completely analyze the layout hierarchy. Connectivity is then extracted from completely flat geometries. Only external geometries (typically pins) of cell instances are processed. Instance pins contribute to connectivity extraction as explained in the section "Instance Pins with Multiple Shapes". Ports of the top level cell also participate in Mask connectivity extraction. Ports contribute to connectivity extraction as explained in the sections "Ports with Multiple Shapes" and "Must-connect Groups".

Mask connectivity extraction is driven by connectivity rules from the Mask set in the rule file.

Mask connectivity extraction internally merges overlapping shapes and paths on any single database layer. References to the original database objects are not maintained.

Connectivity and Rule File Compilation

The rule file compilation verifies that any operation requiring connectivity will have, if and when the operation is executed, the requisite connectivity on the appropriate input layer(s). This consists of verifying that an input layer conforms to one of the requirements outlined in the following subsections on connectivity extraction.

Mask Mode Connectivity Extraction. A layer must conform to one of these conditions to have its connectivity established by connectivity extraction in the Mask verification mode:

- Appear directly in a Connect or Sconnect operation that belongs to the Mask set.
- If the previous case does not apply, the layer may not appear in a derivation tree of any layer parameter in a Connect or Sconnect operation in the Mask set.
- Be derived (via a sequence of net-preserving operations) from a layer appearing directly in a Connect or Sconnect operation that belongs to the Mask set.
- Be derived (via a sequence of net-preserving operations) from a Stamp operation. The Stamp operation's second input layer (layer2) must conform to one of the conditions outlined in this list.
- Be derived (via a sequence of net-preserving operations) from a Polynet operation.


Note

DRC Incremental Connect YES relaxes some of the connectivity derivation criteria. See the *SVRF Manual* for details.

Recognizing Electrical Nets

The rules that connectivity extraction use to recognize electrical nets are described in the following sections.

Shapes on a Single Layer

Abutting or overlapping polygons on a single interconnect layer are always considered to be part of a single net (see Figure 7-1). Interconnect layers are layers that appear in Connect or Connect By operations in the rule file. Polygons that touch only at corners are not considered to be part of the same net.

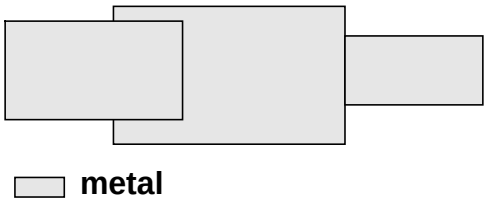


Figure 7-1. Connected Shapes on a Single Layer

Connect

Polygons on different layers can be connected directly by overlap or abutment, as specified in Connect operations in the rule file. For example:

```
connect polya polyb
```

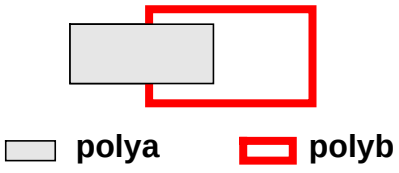


Figure 7-2. Polygons Connected Directly

Connect By

Polygons on two interconnect layers can be connected to each other by mutual intersection with a third polygon on a contact layer specified in a Connect By operation in the rule file. For example:

connect metal poly by cont

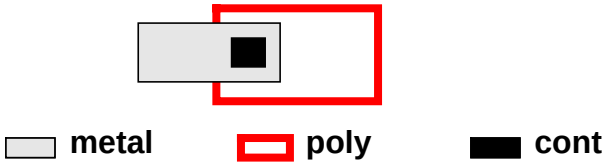


Figure 7-3. Polygons Connected By Contact

Sconnect

Sconnect performs soft connections from an upper layer to a lower layer. You may also establish connectivity by specifying a contact layer and multiple lower layers. Connectivity is unidirectional and is passed from the upper layer to lower layers. The lower layers and contact layer (if specified) receive node numbers from the upper layer, never in the other direction. Positive area overlap between the upper and lower layers with the contact layer (if specified) must exist. For example:

sconnect metal1 poly by contact // node numbers passed from metal1 to //contact and poly

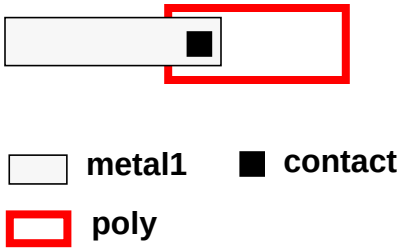


Figure 7-4. Sconnect Operation

You can specify polygons from lower layers which are involved in conflicting soft connections by using the LVS Softchk operation. Depending on the options you

specify, you can output upper, lower, contact, or all layers involved in conflicting soft connections to a lower layer. In addition, LVS Report Option S turns on detailed reporting of Sconnect conflicts in the session transcript. This option is ignored in the DRC applications.

You can also cause Calibre LVS to abort after detecting any Sconnect conflicts, regardless if LVS Softchk statements are present or not, by using the LVS Abort On Softchk specification statement. If you wish to abort on detection of conflicts, you would specify LVS Abort On Softchk YES in your rule file.

Stamp

Stamp is a layer operation that creates a derived layer which receives its connectivity information from a "stamping" layer. Connectivity is passed unidirectionally from the stamping layer to the derived layer. The input layers used in a Stamp operation are not affected by it. For example:

```
x = stamp polyc by metal1 // x is a derived layer with metal1 connectivity
                          // information
```

Ports and Pins

A port is an entry point to a cell. Ports of a cell form the external interface of the cell. A port becomes an instance pin (or simply "pin") when the cell is placed at a higher level of the layout hierarchy.

In Calibre LVS applications, ports can be specified for the top level cell using Port Layer Text and Port Layer Polygon specification statements in the rule file. A port can consist of any number of shapes and paths on any number of layers. Port shapes and paths always have both an internal and an external aspect in the cell.

Each port geometry is represented in the connectivity extractor by a (x,y) location, a layer, and an optional port name. Connectivity extraction operates only on port geometries whose layers appear in Connect or Connect By operations, or as first parameters in Attach operations in the rule file. Information on ports and pins can be transferred from original layers to derived polygon layers by means of the Attach operation (see the section "Attach Operation"). The effects of ports and pins on connectivity extraction are described in the following sections.

The process of assigning label names to port objects is based on the label location and layer. It is optionally controlled by Attach and/or Label Order operations in the rule file.

Port Text and Polygon Objects

Port layer specification statements allow you to specify ports in GDSII and CIF layout databases. In Calibre LVS and xCalibre applications, text and polygon objects on port layers can be read and treated as ports. The Port Layer Text specification statement supports text objects where the port's layer, location, and name are the same as the layer, location, and value of the text object, respectively. The Port Layer Polygon specification statement supports geometries where the port layer is the geometry's layer, and the port location is the center of the geometry's extent (the port has no name). Text objects and geometries defined in the rule file (with the Text and Polygon specification statements) cannot become ports.

In flat applications, port objects are read from the top level cell only (regardless of a Text Depth specification statement setting). In hierarchical applications, port objects are read from all levels of hierarchy and are used locally in the cells where they appear.

Hierarchical Processing of Port Text and Polygon Objects

Port objects are read from all levels of hierarchy and are used locally in the cells where they appear. Port objects in the top-level cell are output as subcircuit pins by the hierarchical Spice netlister and thus participate in hierarchical LVS. This behavior is consistent with flat LVS. Port objects at lower levels of hierarchy are not output by the hierarchical Spice netlister (calibre -spice) and do not specify cell pins for LVS. They are nevertheless used by xCalibre. For more information on hierarchical parasitic extraction with xCalibre refer to the *xCalibre User's Manual*.

Port text and polygon objects at any level of hierarchy can overlap geometries at lower levels of the hierarchy and can be attached to those lower level geometries. The hierarchical connectivity extractor will form any pins in lower level cells that are necessary to connect to port objects higher up in the hierarchy. This is similar to how text is handled hierarchically.

Hierarchical Netlisting of Port Text and Polygon Objects

For the top-level cell, the hierarchical Spice netlister outputs ports in the pin list of the top .subckt statement. Of course, .subckt pin names are the same as the respective net names within the subcircuit. Ports contributed by Port Layer Text and Port Layer Polygon statements can be named (text ports) or unnamed (polygon ports). The name chosen for netlisting is the port name or the original net name, whichever is present. If both the port and the respective net are texted, then the original net name prevails. If neither is texted then a system-generated number is used. If several ports with different names are attached to a single net and the net itself is unnamed, then one of the port names is chosen arbitrarily.

You may specify the hierarchical depth for fetching port objects from the layout database for use in the top-level cell by using the Port Depth specification statement. Port objects that come from lower levels of the hierarchy are transformed to the top level coordinate space and are replicated according to the hierarchical structure of the design.

Notes:

- These statements are not used by the Calibre DRC application.
- These statements are not used in BINARY or ASCII layout database modes.
- Reading of text ports does not depend on Text Layer or Text Depth statements and reading of polygon ports does not depend on whether the layer is or is not referenced by other operations.
- Unless Layout Merge On Input YES is specified in your rule file, port polygons use unmerged data with centers computed after path expansion.
- Port polygons are flagged for non-orientable and non-simple objects but do not participate in acute, skew, or offgrid flagging (unless the specified layer is referenced by other operations that cause such flagging).

Ports with Multiple Shapes

A port can consist of a group of disjoint shapes or paths that are connected by geometries internal to the cell. Such configurations are sometimes called *feedthroughs*. They allow routing of signals through instances of the cell at higher levels of the hierarchy, without actually routing on top of those instances.

Shapes and paths that belong to a single port of a cell are always considered to be part of a single net when extracting the connectivity of the cell.

In addition, connectivity extraction verifies the following two rules for cell ports:

- All shapes and paths of a single port must be physically connected in the cell. Physical connections are connections formed by abutment or overlap between geometries, connections formed by geometries of a single instance pin. This rule ensures that connectivity, due to instance pins with multiple shapes at higher levels of hierarchy, is correct.

- The shapes or paths of two distinct ports must not be physically connected inside the cell. This rule ensures that connectivity, due to instance pins with multiple shapes at higher levels of hierarchy, is complete, implying that connections that are really present are not missed.

Errors are reported when these rules are not followed.

For example, in Figure 7-5, port A consists of two metal shapes (drawn with thick lines): shape 1 and shape 2. The two port shapes are connected through a third metal shape. Extraction would place the two port shapes in a single net, even if the third metal shape was not present, however a warning would be issued.

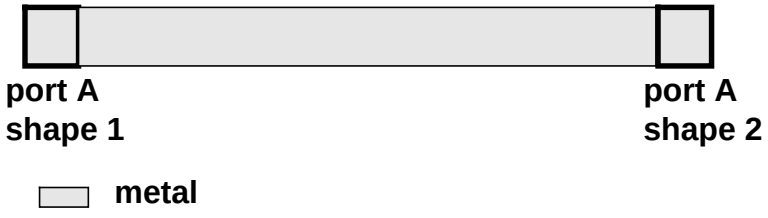


Figure 7-5. Port With Multiple Shapes

In Mask extraction, it is possible for a net to be formed entirely of shapes or paths of a single port. This occurs when a port is not connected to any other geometries in the cell.

Instance Pins with Multiple Shapes

A port of a cell becomes an instance pin when the cell is placed at a higher level of the layout hierarchy. Instance pin geometries participate in connectivity extraction of the containing cell.

In Mask extraction, internal geometries of some or all instances are not processed. Instead, only external geometries of those instances (typically, pin shapes, and paths) are processed. The following is done for instances when their internal geometries are not processed. All shapes and paths of a single instance pin are considered to be part of a single net in the cell that contains the instance. This is true even if the shapes or paths do not physically touch or overlap. The pin shapes and paths can be on multiple layers. This feature allows for recognition of connectivity formed by signals that are routed through cell instances (feedthroughs).

For example, figure 7-6 shows how a net is routed through a cell instance. The figure shows an instance with a single pin A. Pin A consists of two metal shapes; shape 1 and shape 2 (drawn with red lines). The example also shows two metal interconnect shapes, each one touching one shape of pin A. All four metal shapes are placed in a single net by connectivity extraction.

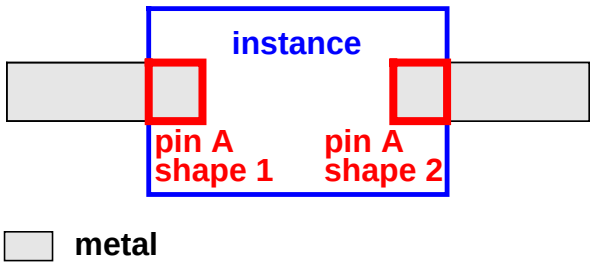


Figure 7-6. Connection Through a Pin with Multiple Shapes

In Mask extraction, a net can be formed entirely from shapes or paths of a single instance pin. This happens when an instance pin is not connected to any other geometries in the cell.

In Mask extraction, a net can be formed by the direct abutment of two or more instance pins. Such net can be formed entirely from pin shapes or paths.

Must-connect Groups

The shapes and paths of all ports in a single must-connect group in a cell are considered to be part of a single net when extracting the connectivity of the cell. All other geometries in the cell physically connected to any one of those ports are also considered to be part of the same single net.

For example, figure 7-7 shows two ports (drawn with thick lines); port A consists of a metal shape, and port B consists of a poly shape. Both ports belong to a single must-connect group; group 1. Also shown are two interconnect shapes, one on metal and one on poly, touching the shapes of port A and port B, respectively. All four shapes will be placed in a single net by connectivity extraction.

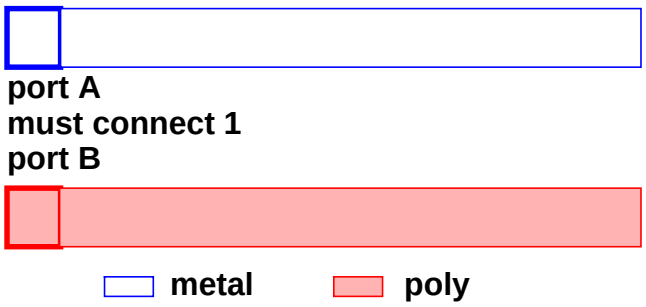


Figure 7-7. Connection by Means of a Must-Connect

Ports in a must-connect group can have identical names, but this is not required. A port can belong to one must-connect group only. A must-connect group can be specified as either conditional or unconditional.

Must-connects are used when a cell contains disjoint parts of a single electrical signal. The intention is that those parts be physically connected at some higher level of the layout hierarchy.

See the section "Virtual Connect Statements" on specifying virtual connections with net names.

Verifying Must-connect Conditions.

One of the following must hold for any two pins of an instance that belong to a single unconditional must-connect group in the instance's cell template:

- The pins must be directly connected in the cell that contains the instance.
- The pins must be connected to ports of the containing cell, which also belong to a single must-connect group.

An error is reported if neither of these conditions is satisfied.

For a conditional must-connect group, if a connection is made within the containing cell to one or more of the pins in the must-connect group, then the group must obey the same conditions as an unconditional must-connect group. However, if no connections are made to any of the pins, then there are no conditions to be satisfied and no error is reported.

For example, figure 7-8 shows an instance with six pins in three must-connect groups. The must-connect condition between pin A and pin B is satisfied, because they are interconnected in the containing cell. The must-connect condition between pin C and pin D is also satisfied, because they are connected to port X and port Y of the containing cell, respectively, and these two ports belong to a single must-connect group in the cell. The must-connect condition between pin E and pin F is not satisfied.

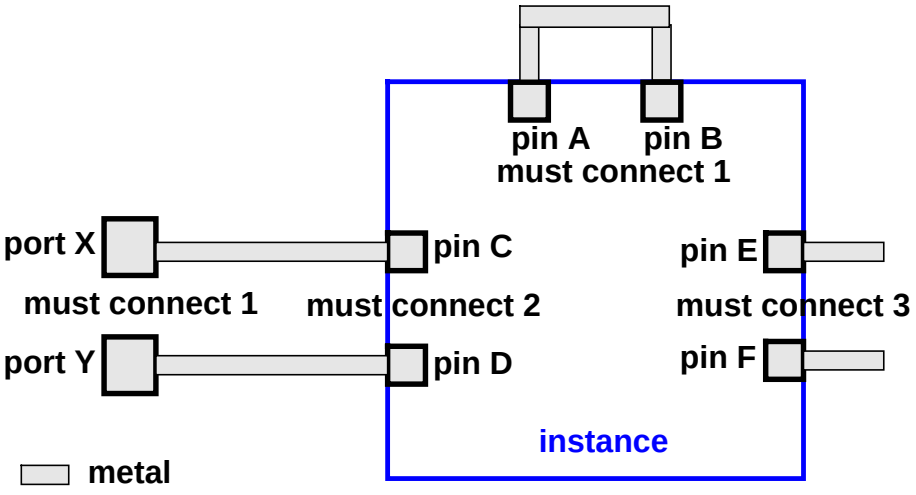


Figure 7-8. Verifying Must-Connect Conditions

- Polygons on merged polygon layers inherit port, pin, and must-connect information from shapes and paths on corresponding original database layers (refer to section "Transferring Logical Information to Merged Layers").

- Port, pin, must-connect, and overflow information can be transferred from original database layers to other layers by means of the Attach operation (refer to section "Attach Operation").

- Connectivity information can be transferred between layers by means of the Stamp operation.

Transferring Logical Information to Merged Layers

Layout verification applications that use Mask connectivity extraction merge original database layers internally before using them for any other operations. Polygons on those merged layers inherit port, pin, and must-connect information from shapes and paths of the corresponding original layers. Namely:

- A polygon on an internally merged layer that originated from a shape or path of a port is considered to belong to the port for the purpose of connectivity extraction.

- A polygon on an internally merged layer that originated from a shape or path of an instance pin is considered to belong to the instance pin for the purpose of connectivity extraction.

Attach Operation

Attach is a connectivity extraction operation that "transfers" connectivity information (net names, port locations, pin locations, and must-connect information) from one layer to another layer:

- Net names are discussed in detail in the section "Net Name Specification".

- Port locations are transferred from port geometries on layer1 to overlapping polygons on layer2. Port names and port must-connect information are transferred along with port locations.

- In the same manner, pin locations are transferred from shapes and paths that are pin members on one layer, to overlapping polygons on another layer. Pin must-connect information is transferred along with pin locations.

In the above cases, the layer1 shapes and paths must be completely covered by the layer2 polygons. The locations and associated information of ports and pins are transferred only from those that actually participate in the particular layout verification application that is being executed.

You may specify a Label Order specification statement that determines the order in which connectivity extraction looks for shapes that intersect a label location (or other significant location). Label Order operates on net labels and on port objects.

Net Name Specification

There are two ways to specify net names in connectivity extraction:

- Text and Layout Text specification statements in the rule file.

- Layout database text objects (GDSII and CIF).

In these cases, the connectivity extractor generates internal objects called labels to represent the textual information. A label has three attributes: a location, a layer, and a name, which is a string. The rules that govern how names are assigned to nets are similar in all cases, and are based on the label location and layer.

Text Specification Statements

This section describes how net names are set with the Text and Layout Text specification statements.

- **Text Specification Statements**. The Text specification statement allows free-standing text to be specified directly in the rule file. It also allows text objects read from the layout database to be edited.

- **Layout Text Specification Statements**. This statement is applicable only when the layout system is GDSII or CIF. The Layout Text specification statement allows free-standing cell-based text to be specified directly in the rule file. The statement specifies a text object that behaves exactly as if it were in the layout database in the specified cell at x,y location in the cell coordinate space.

  A Layout Text specification statement results in a label object being generated in the connectivity extractor. The label has the specified *name*, *xy location*, and *layer*. Refer to "Label Attachment" for a description of how label names are assigned to nets.

  Like any database text, layout text objects can be used both in hierarchical and flat applications. Layout text is particularly useful in hierarchical LVS for specifying local net names in cells. All verification applications use layout text objects as top level text when specified by Text Depth statements.

Listed below are differences between text objects in Text and Layout Text specification statements. Text objects and Layout Text objects refer to text from the Text specification statement and Layout Text specification statement, respectively:

- Text objects are always in world coordinates and have no cell association.
- Text objects can "edit" existing database text, where Layout Text objects behave exactly as existing database text.
- Layout Text objects observe Text Layer and Text Depth requirements (as with any database text used for connectivity extraction), where Text objects do not.
- Text objects are only used for connectivity extraction. Layout Text objects can be used for With Text operations.
- Layout Text objects can have TEXTTYPES, and obey Layer Map statements, where Text objects do not.

- Only simple layer numbers can be associated with Layout Text objects. Text Objects can have original layer names.

Layout Database Text Objects

With this method, GDSII or CIF database text objects determine net names. The database text must be placed at some location on the net.

A database text object results in a label object being generated in the connectivity extractor. The label *location* is the location of the database text object. The label *layer* is the layer of that object, and the label *name* is the value of that object. The section "Label Attachment" describes how label names are assigned to nets.

Reading of text objects from the layout database to support connectivity extraction is guided by the specification statements Text Layer and Text Depth statements the rule file.

Text Layer, Text Depth and Expand Cell Text Specification Statements

The Text Layer and Text Depth specification statements in the rule file guide the reading of text objects from a layout database to support connectivity extraction. They control database text objects only, namely:

- GDSII and CIF text objects
- Text objects entered with Layout Text specification statements (which behave just as if they were database text).

These statements do not apply to text objects entered With Text specification statements.

The Text Layer statement does not influence text objects used by With Text operations in the rule file. When reading text objects from a layout database, the connectivity extractor uses only those text objects whose layers appear in Text Layer specification statements. Therefore, if there are no Text Layer specification statements in the rule file, then no database text objects will be used by the connectivity extractor.

The Text Depth statement specifies the hierarchical depth for taking text objects from the layout database. The Text Depth ALL statement selects text objects from throughout the hierarchy and The Text Depth PRIMARY statement selects only text objects from the top-level cell. The syntax of the statement allow you to precisely control the text depth.

Text objects that come from lower levels of the hierarchy are transformed to the top-level coordinate space, and are replicated according to the hierarchical structure of the design. Such text objects then behave as if they originated at the top level; this is true in flat as well as hierarchical applications.

In flat applications and in the hierarchical DRC application, only those database text objects selected by a Text Depth statement are used in the connectivity extractor.

In hierarchical LVS, text objects from *all* levels of the design hierarchy are used as *local text* in the cells in which they appear, regardless of the Text Depth specification statement; text objects selected by this statement serve as top-level text *in addition* to any local role they can perform.

The Text Depth statement supports connectivity extraction only; it does not influence text objects used by With Text operations in the rule file.

The Expand Cell Text specification statement allows you to add text from placements of a cell of origin to a target cell (or cells) higher up in the layout hierarchy. You specify target cells either explicitly or implicitly. The added text objects are transformed to the coordinates of the target cell (or cells). This statement operates on connectivity extraction text and port text, and does not affect With Text operations.

Label Attachment

As described above, the connectivity extractor generates internal label objects to represent text data from various sources: rule file Text specification statements, and GDSII and CIF text objects. Each label is represented by a *location*, a *layer* and a *name*. A *location* can be a simple (x,y) point, or it can be the whole area of a database shape or path.

The process of assigning label names to nets is based on the label location and layer. It is optionally controlled by Attach and Label Order operations in the rule file.

- The Attach operation transfers connectivity information, including values of net properties, from a specified original database source layer to a specified original or derived target layer that appears in a Connect operation.
- The Label Order operation determines the order in which connectivity extraction looks for polygons that intersect a label location when the label layer does not appear in a Connect operation, and is not attached to any specific layer.

Connectivity extraction attaches labels to nets as follows, in the order given:

1. **Explicitly**. For example, if the rule file contains the operation:

   ```
   ATTACH A B
   ```

   where A is the label *layer* (or A is a layer set that contains the label layer). The connectivity extractor looks for a polygon on B that intersects the label *location*. If found, the label *name* is assigned to the net that contains that polygon.

   A label location can encompass the area of a complete shape or path, and it can happen that the label location intersects two or more polygons on B that belong to two or more distinct nets. In that case, one of those nets is chosen arbitrarily. The label *name* is assigned to that net and a warning is issued.

   If no polygons on B overlap the label location, then the label is ignored and a warning is issued.

   The rule file can contain more than one Attach operation for the label layer, such as:

   ```
   ATTACH A B1
   ATTACH A B2
   …
   ATTACH A Bn
   ```

In this case, the connectivity extractor looks for polygons on any one of the target layers B1, …, B*n* that intersect the label location. If exactly one polygon is found, then the label *name* is assigned to the net that contains that polygon. If more than one polygon is found, one is chosen arbitrarily, and a warning is issued. If no polygons are found, the label is ignored and a warning is issued.

Example 1:

```
connect metal poly by contact
attach poly.txt poly
```

Example 2:

```
connect metal poly by cont
attach text metal
attach text poly
```

2. **Implicitly**. For example, if the rule file contains the following operation:

   ```
   CONNECT … A …
   ```

   where A is the label *layer* (or A is a layer set that contains the label *layer*). Then the connectivity extractor looks for a polygon on A that intersects the label *location*. If found, the label *name* is assigned to the net that contains that polygon.

   Example:

   ```
   connect metal poly by cont
   ```

3. **Freely**. For example, if the rule file contains the following operation:

   ```
   LABEL ORDER … B1 B2 … Bn
   ```

   operation. Then, the order specified in the Label Order operation is used. The connectivity extractor looks for polygons on any one of the layers specified in the Label Order operation that intersect the label *location.* The polygon whose layer appears first in the Label Order operation is chosen. The label *name* is assigned to the net that contains this polygon.

If no Label Order operation is present in the rule file, or if no polygon on any of the Label Order layers intersects the label location, then the label is ignored and a warning is issued.

Example:

```
connect metal poly by cont
label order metal poly
```

Labels on layer text will be attached to metal if metal is present at the location of the label. Otherwise, text will be attached to poly if poly is present.

Additionally, if two or more different names are found on a single net, the alphabetically least name is chosen and the other names are discarded. A warning is issued.

Also, if an attempt is made to assign the same name to two or more nets, one of the nets is arbitrarily chosen; the other nets are unnamed. A warning is issued.

The following examples further illustrate some of the techniques for naming nets in connectivity extraction.

- **Implicit attachment**: The original database layers poly and metal appear in Connect operations in your rule file.

- **Explicit attachment**: The connectivity of source/drain regions of MOS transistors in your rule file is determined in terms of a derived layer src_drn as follows:

  ```
  src_drn = NOT diffusion polysilicon
  CONNECT src_drn metal BY contact
  ```

  Diffusion and polysilicon are original database layers. Include the statement:

  ```
  ATTACH diffusion src_drn
  ```

in your rule file. To assign a net name directly to a source or drain region, place a text object on the diffusion layer over this region.

- **Free attachment**: You use the database layers metal1, metal2, and poly for interconnect and you want to use another database layer text to specify net names. Have a statement:

```
LABEL ORDER metal1 metal2 poly
```

in your rule file. To assign a name to a net, place a text object on the text layer over some metal1, metal2, or poly region of the net.

You can place the text object or the shape over a region where several different nets are present on metal1, metal2, or poly respectively. The layer that appears first in the Label Order operation will be chosen.

See also the Layout Property Text and Layout Rename Text specification statements in the *SVRF Manual*.

Virtual Connect Statements

The *virtual connect* paradigm in layout verification is the capability of the layout connectivity extractor to form a single net from two or more disjoint nets by virtue of the fact that the net segments share the same name. Virtual-connect is triggered by the rule file Virtual Connect Colon and Virtual Connect Name specification statements. You can instruct Calibre to report virtual connections using the Virtual Connect Report YES specification statement.

Virtual Connect Colon

Virtual Connect Colon can be specified at most once. If YES is specified, then the connectivity extractor first strips off all characters from the first colon to the end of the label names. Next, the extractor forms a virtual connection between any two labels that have the same name and originally contained a colon. Names are compared after colon suffixes have been stripped off, such that names are considered identical if they are identical up to the first colon. Colons can appear anywhere in the name with the exception that a colon at the beginning of a name is treated as a regular character (that is, it has no special effect).

Labels can be entered with rule file Text statements, or GDSII or CIF text as described in section "Label Attachment" above. A virtual connection between labels causes a virtual connection between the net segments to which those labels are assigned, regardless of whether or not the label name becomes the final name of the net segment.

If NO is specified, or the statement is not specified at all, then there is no special treatment of colon characters in label names. In particular, colon suffixes are not stripped off and no virtual connections are performed based on the presence of colon characters per se.

Another effect of the rule file statement Virtual Connect Colon YES is to strip off colon suffixes from node names in .GLOBAL statements in a Spice netlist. For example, these are equivalent specifications:

```
.GLOBAL VCC:P VSS:XYZ VDD:
.GLOBAL VCC VSS VDD"
```

For LVS Box cells, the Virtual Connect Box Colon specification statement works similarly to Virtual Connect Colon. It performs virtual connections by colon within LVS Box cells. It also connects respective pins. Note that LVS Connect Box Colon YES does not strip off colon suffixes from node names in .GLOBAL statements in Spice netlists. This statement only applies to Calibre LVS-H.

Virtual Connect Name

Virtual Connect Name can be specified any number of times. Each *name* is a (case-insensitive) net name and can be optionally enclosed in quotes. The connectivity extractor forms a virtual connection between any two labels having the same name such that the label name appears in a Virtual Connect Name specification statement in the rule file. Note that if Virtual Connect Colon YES is also specified, then Virtual Connect Name operates on names after all colon suffixes have been stripped off.

For LVS Box Cells, the Virtual Connect Box Name specification statement works similarly to Virtual Connect Name. This statement only applies to Calibre LVS-H.

Examples:

```
VIRTUAL CONNECT BOX NAME "?"
```

This connects net segments with identical names in LVS Box cells. It also connects together respective pins. In Figure 7-9, if A, B and C are LVS Box cells then the points marked 1, 2, 3 and 4 will all belong to the same net.

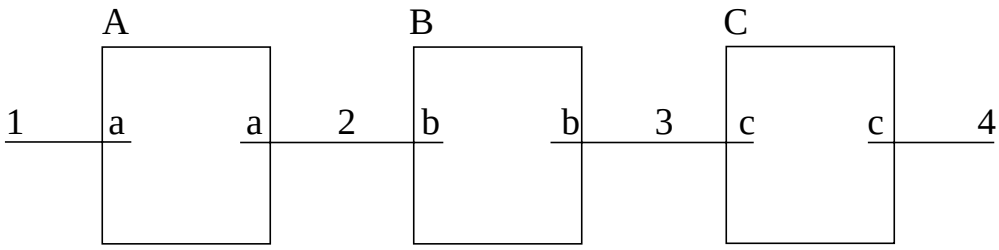


Figure 7-9. Example of Virtual Connect Box

```
VIRTUAL CONNECT BOX NAME "VCC" "VSS"
```

This connects net segments with identical names in LVS Box cells, provided that the names are either VCC or VSS. It also connects respective pins.

Short Isolation

When two nets are shorted together, LVS sees, essentially, one net with two different text names. Normally, finding the short is difficult, especially if the shorted nets include a power or ground net. To simplify the process of isolating shorts, you can specify the LVS Isolate Shorts specification statement in the rule file. This statements isolates the short by finding the shortest path between two or more pieces of conflicting text. The results are output to an ASCII DRC database format file.

When short isolation is enabled, it is executed:

- in flat LVS or layout-to-Cnet translation

- during hierarchical circuit extraction (calibre -spice).

For example, any of these commands can be used to perform short isolation:

```
calibre -lvs rules
calibre -lvs -tl lay.cnet rules
calibre -spice lay.net rules
calibre -spice lay.net -lvs -hier -hcell cells rules
```

The first two commands execute short isolation in the flat mode. The last two commands execute short isolation hierarchically. All short isolation results are returned in the coordinate space of the top level cell. In hierarchical operation, results from lower level cells are shown only once; the lowest leftmost placement of a cell in the design is used as representative for shorts occurring in that cell.

In flat and hierarchical systems, short isolation in the top level cell operates on the text at that level only, subject to Text Depth specification statement. Short isolation in lower level cells operates on text objects that are present locally in each particular cell, independent of Text Depth.

You can view short isolation results as soon as they are created, even before the run terminates. You can accomplish this by altering the command line in the following way:

```
calibre -lvs … | tee logfile_name
```

An example of short isolation results are as follows:

```
SHORT ISOLATION started.
SHORT ISOLATION completed. CPU TIME = 0  REAL TIME = 0  ...
SHORT ISOLATION RESULTS DATABASE = lvs.rep.shorts
```

In Calibre LVS-H, short isolation is performed as part of hierarchical circuit extraction. In this mode, circuit extraction is performed hierarchically, the short isolation algorithm operates hierarchically, and geometries are no longer flattened.

When more than one path exists between a pair of conflicting text points, the hierarchical short isolation algorithm prefers paths that consist of geometries at higher levels of hierarchy over those that pass through cell placements. For this reason, the indicated path in the results file may not be the shortest in terms of

polygon count. Thus, different paths may be chosen when operating hierarchically and flat. Within a given level of hierarchy, paths with fewer polygons are preferred.

The order of execution is this:

1. Hierarchical connectivity extraction performed
2. If shorts exist, short isolation algorithm operates hierarchically and geometries are not flattened
3. Short isolation results are written out to disk
4. Hierarchical device recognition, hierarchical Spice netlist generation and, if requested, hierarchical LVS comparison.

In LVS-H, the short isolator does not process target layers specified by the Sconnect statement unless those layers contain texted geometries that are involved in a short. Therefore, if you use Connect statements to form connections to the substrate, replacing them with Sconnect statements can increase processing efficiency.

Connectivity Extraction Errors and Warnings

- The errors, warnings and notes reported by connectivity extraction are summarized in this section. The hierarchical Calibre circuit extraction module (calibre -spice) reports this information in the session transcript. Flat Calibre LVS reports this information both in the session transcript and in the LVS report file. WARNING: Port contains unconnected shapes. It may contribute false feedthroughs on instances of the cell.

  Issued for a port with multiple shapes or paths that are not physically connected inside the cell. The port name is reported.

- WARNING: Direct connection between different ports.

A direct connection was made between distinct ports of the cell. The port names are reported. LVS Report Option P disables this warning. This warning is reported only in flat execution and is not reported in hierarchical execution.

- ERROR: The unconditional must-connect condition between the following instance pins has not been satisfied.

  Pins of an instance which belong to a single unconditional must-connect group in the instance's template are not connected in the cell that contains the instance, and cannot be connected at any higher level. The pin names are reported in the form: <instance_name>.<pin_name>.

- ERROR: The conditional must-connect condition between the following instance pins has not been satisfied.

  Pins of an instance which belong to a single conditional must-connect group in the instance’s template are not connected in the cell that contains the instance, and cannot be connected at any higher level. This error is reported only if a connection was made to one of the pins. The pin names are reported in the form: <instance_name>.<pin_name>.

- ERROR: A must-connect condition exists on instance pins but is not defined for the corresponding ports.

  Pins of an instance which belong to a single must-connect group in the instance's template, and which are not connected in the containing cell, are connected instead to ports of the cell; however, the ports do not belong to a single must-connect group in the cell. The instance pin names in the form <instance_name>.<pin_name> and the corresponding port names are reported.

- WARNING: Short circuit—Different names on one net.

  This warning is issued when several different names are found on a single net. One name is chosen and the other names are ignored. Conflicts are resolved in favor of, in order of precedence: (1) power names, if specified, in rule file order; (2) ground names, if specified, in rule file order; and (3)

the alphabetically least name. Power and ground names are specified with optional LVS Power Name and LVS Ground Name statements in the rule file and are subject to the LVS Compare Case statement. All names found on the net are reported, along with their locations and layers. The name actually assigned to the net is indicated. In Mask mode extraction, the net id is also reported.

- WARNING: Open circuit - Same name on different nets.

  This warning is issued when an attempt is made to assign the same name to two or more different nets. One of the nets is arbitrarily chosen and the other nets are left unnamed. The name is reported, along with all locations and layers on which it was found. In Mask mode extraction, the report also includes the net IDs on which the name was found, and the net ID to which the name was actually assigned.

- WARNING: Ambiguous label attachment - One label intersecting different nets.

  This warning is issued when a label intersects the regions of two or more different nets and an attempt is made to assign the label to those nets by means of Attach or Label Order statements in the rule file. One of the nets is arbitrarily chosen. The label name and its location and layer are reported. In Mask mode extraction, the report also includes the IDs of all nets involved in the conflict, and the ID of the net to which the label was actually assigned.

- WARNING: Unattached label.

  This warning is issued for a label that can not be assigned to any net. The label name, location and layer are reported.

- WARNING: Unattached must-connect pad; must-connect condition ignored.

  This warning is issued when a must-connect condition on an instance pin cannot be used. This happens when an Attach statement in the rule file cannot be applied to the must-connect because the target layer of the Attach

statement is not present at the location of the must-connect. The must-connect number, pin name, layer and location are reported.

- WARNING: Unattached port pads; port ignored.

  This warning is issued when a port cannot be associated with any net. This happens when the port layer does not appear in Connect, Attach or Label Order statements; or, at the port location there is no geometry that the port can be attached to. The port name, layer and location are reported.

- NOTE: Virtually connected...

  Indicates a name-based virtual connection, for example, from the specification statements Virtual Connect Name or Virtual Connect Colon. For each pair of labels that cause a virtual connection, the label name and both label locations are indicated. This reporting is enabled with the specification statement LVS Report Option V.

  Example:

```
NOTE:  Virtually connected "D#2" at (-16,-19) and (19,-19)
NOTE:  Virtually connected "D#2" at (-16,-19) and (26,-19)
NOTE:  Virtually connected "E#2" at (-16,-25) and (26,-25)
```

Chapter 8
Electrical Rule Checks

Electrical rule check (ERC) operations in Calibre perform tasks related to electrical rule checking. In many respects, ERC is similar to DRC. However, because ERC is perceived as a separate task and (unlike DRC) performed as part of an LVS run, ERC has a separate results database and a dedicated set of specification statements to control execution and result generation, many of them are similar to their respective DRC specification statements.

ERC operations, specifically the Pathchk operation, can generate derived layers, which in turn can be manipulated by other operations. ERC checking within Calibre LVS can utilize the entire set of Calibre operations and is not limited to *dedicated* ERC operations.

ERC operations are layer constructors and can be used to derive layers for rule check output or for other operations. In addition, ERC operations may have side effects, in the form of generating auxiliary result files. Generally, ERC operations can be used to construct layers, to create auxiliary result files, or both.

ERC Statements and Operations

Two types of Standard Verification Rule Format statements apply to ERC; ERC operations and ERC specification statements. Table 8-1 lists the specification statements related to ERC execution.

Table 8-1. ERC Specification Statements

ERC Results Database	ERC Keep Empty
ERC Maximum Results	ERC Select Check
ERC Maximum Vertex	ERC Unselect Check

Table 8-1. ERC Specification Statements

ERC Check Text	LVS Execute ERC
ERC Cell Name	ERC Pathchk
ERC Summary Report	

Table 8-2 shows the ERC operations.

Table 8-2. ERC Operations

Pathchk	Device Layer

ERC operations may contain PRINT secondary keywords that are responsible for the production of auxiliary files. The PRINT secondary keywords and the auxiliary files that they create are discussed in the section "ERC Output Files".

For detailed information on each statement, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Execution of ERC Operations in LVS

ERC execution within Calibre LVS (including PRINT clauses) is controlled by the ERC Select Check and LVS Execute ERC specification statements.

If any rule checks are selected, DRC licenses are consumed in addition to LVS licenses. For further information about licensing refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide*.

All rule check statements in the rule file are candidates for selection; rule checks may contain any layers and are not restricted to ERC operations. For a detailed description of rule check statements refer to the section "Rule Check Statements", chapter 4, "DRC Concepts". ERC is executed in the LVS circuit extraction stage.

Execution of all selected rule checks in Calibre LVS can be suppressed by the LVS Execute ERC specification statement. The secondary keyword NO instructs

Calibre LVS to ignore ERC rule checks. DRC licenses are not consumed if ERC execution is disabled by this statement. If this statement is not present, the default value of YES is assumed, and all selected rule checks are executed.

Some operations, for example DRC measurement operations, may execute slower and/or consume more memory when executed by ERC within Calibre LVS, compared to similar execution within Calibre DRC.

The following commands execute ERC in Calibre LVS:

- `calibre -spice … rules`

  For this command line syntax, ERC is executed hierarchically.

- `calibre -lvs rules`

  For this command line syntax, ERC is executed flat, and the Layout System statement must specify a format that consists of geometries, such as GDSII and CIF.

- `calibre -lvs -tl … rules`

  For this command line syntax, ERC is executed flat, and the Layout System statement must specify a format that consists of geometries, such as GDSII and CIF.

All these commands consume DRC licenses in addition to LVS licenses if ERC rule checks are selected and ERC execution is not disabled.

Execution of ERC PRINT Options

Side effects of ERC operations, such as the PRINT POLYGONS and PRINT NETS secondary keywords, are executed in LVS applications whenever the respective ERC operation is performed. PRINT options do not force an ERC operation to be performed—this is done by specifying the rule check name in an ERC Select Check statement, as shown in the following example:

```
X { PATHCHK !POWER PRINT POLYGONS “file1”}
ERC SELECT CHECK X                  // triggers execution of X
```

You can perform a PRINT option without writing the information to the ERC results database by performing an AND operation on the ERC operation and an empty layer. In the following example, the rule check X creates “file1” but does not write any data to the ERC results database (subject to the use of the ERC Keep Empty statement):

```
X {
   (PATHCHK !POWER PRINT POLYGONS “file1”) AND 999
   }                                        //layer 999 is empty
ERC SELECT CHECK X                  // triggers execution of X
```

ERC operations are not executed in xCalibre applications.

Rule Check Selection in LVS

For LVS applications the following process is followed for rule check selection:

1. The tool unselects *all* rule checks.
2. The tool selects rule checks from ERC Select Check specification statements.
3. The tool unselects rule checks from ERC Unselect Check specification statements.

Execution of ERC operations in DRC

Calibre DRC executes all selected rule check statements subject to the DRC rule check selection mechanism described below. ERC operations produce empty result layers and a warning is generated. PRINT options are not executed.

Note that ERC execution within DRC has little effect and essentially creates no results. The LVS Execute ERC specification statement has no effect in Calibre DRC.

Rule Check selection in DRC

For DRC applications the following process is followed for rule check selection. This is the regular mechanism applied in the DRC application:

If DRC Select Check statements are present in the rule file:

1. The tool unselects *all* rule checks.
2. The tool selects rule checks from DRC Select Check specification statements.

otherwise

1. The tool selects *all* rule checks.
2. The tool unselects rule checks from ERC Select Check specification statements.
3. The tool unselects rule checks from DRC Unselect Check specifications statements.

ERC Output Files

In addition to derived layers, ERC execution can produce two different forms of output: an ERC results database and auxiliary files resulting from the use of PRINT keywords in ERC operations. The following sections describe these types of output.

ERC Results Database

ERC execution can generate a results database similar to the type generated during a DRC run, which is discussed in section "ASCII and Binary DRC Results Databases", in chapter 14, "Results".


Note

You can create only an ASCII format database for an ERC run.

ERC Auxiliary Files

ERC operations contain secondary keywords that instruct Calibre to produce auxiliary files during ERC execution.

These secondary keywords contain the keyword PRINT. The following sections describe the auxiliary files created with the PRINT keyword, specifically for the Pathchk operation.

- **PRINT POLYGONS *filename***—Calibre creates an auxiliary file when you include this syntax in a Pathchk operation. The file contains polygon output of all nets that satisfy the Pathchk condition.

  Calibre writes the polygons in DRC ASCII format with the following syntax:

  ```
  PATHCHK condition [ in cell cellname ] [ (layer_name) ]
  ```

  where:

  - *condition*—specifies the condition of the Pathchk statement, such as "!POWER" or "GROUND && !POWER"
  - in cell *cellname*—specifies the name of the cell that contains the reported net. Calibre includes this information when you specify BY CELL in the Pathchk operation.

  - (*layer_name*)—specifies the name of the layer that contains the reported net, enclosed in parentheses. Calibre includes this information when you specify BY LAYER in the Pathchk operation.

- **PRINT NETS *filename***—Calibre creates an auxiliary file when you include this syntax in a Pathchk operation. The file includes a list of nets that satisfy the Pathchk condition.

  Hierarchical Calibre writes the nets to *filename* in text format with the following syntax:

```
PATHCHK REPORT for layout_primary
  PATHCHK condition
    cell name (placement): net1, net2, …
    cell name (placement): net1, net2, …
    …
```

  and flat Calibre writes the nets to *filename* in text format with the following syntax:

```
PATHCHK REPORT for layout_primary
  PATHCHK condition
    net1, net2, …
```

  where:

  - *layout_primary*—specifies the top-level cell as defined in the Layout Primary specification statement.

  - *condition*—specifies the condition of the Pathchk statement, such as “!POWER” or “GROUND && !POWER”

  - cell *name* (*placement*)—specifies the *name* of the cell that contains the reported nets, followed by the *placement* of that cell, enclosed in parentheses.

  - *net1*, *net2*, …—lists the net names or numbers in the specific cell that satisfy the Pathchk condition.

ERC Examples

The following examples show how you can use the Pathchk operation to create a layer or generate an auxiliary file. You can also do both of these actions during a single run.

- **Example 1** — Creation of a layer with the Pathchk operation, and how different invocations on the same rule file affects the output.

  Assume the following rule file excerpt exists:

```
ERC RESULTS DATABASE “ercdb”
ERC SELECT CHECK E1 E2
CONNECT ....                          //Connect operations.
DEVICE ....                           //Device definitions.
Z = PATHCHK !POWER          //Nets with no path to power.
E1 { Z AND MET1 }           //MET1 with no path to power.
E2 { COPY XXX }                         //Entire layer XXX.
E3 { COPY YYY }                         //Entire layer YYY.
```

  and the following invocation command:

```
calibre -spice z.net rules
```

  Calibre executes rule checks E1 and E2, and outputs the results to an ERC results database named “ercdb”. Rule check E1 requires the execution of the Pathchk statement, which generates derived layer “Z”. For rule check E2, Calibre outputs layer “XXX”, and can involve any ERC operations. Calibre does not execute rule check E3 because it does not appear in the ERC Select Check statement.

  If you add the statement LVS Execute ERC NO to the rule file, then Calibre does not execute any rule checks, and the Pathchk operation is not executed.

  Assume the same rule file, but the following invocation command instead:

```
calibre -drc rules
```

Calibre executes rule check E3, but not rule check E1 or E2 because they appear in the ERC Select Check statement.

- **Example 2** — Generating Auxiliary files with the Pathchk operation.

  Assume the following rule file statement exists:

  ```
  ERC RESULTS DATABASE “ercdb”
  ERC SELECT CHECK X
  X {PATHCHK !POWER PRINT POLYGONS “file1”}
  ```

  and the following invocation command:

  ```
  calibre -spice z.net rules
  ```

  Calibre executes the Pathchk operation and writes the auxiliary file directly to “file1”. In addition, the PATHCHK output layer is written to rule check X in the ERC results database “ercdb”.

- **Example 3**— Generating Auxiliary files with the Pathchk operation.

  Assume the following rule file statement exists:

  ```
  ERC RESULTS DATABASE “ercdb”
  ERC SELECT CHECK X
  X {(PATHCHK !POWER PRINT POLYGONS “file1”) AND 999}
  ```

  and the following invocation command:

  ```
  calibre -spice z.net rules
  ```

  Calibre executes the Pathchk operation and writes the auxiliary file directly to “file1”. No data is written to the ERC results database “ercdb” and rule check X is not created. We assume that there are no geometries on layer 999.

Chapter 9
Device Recognition

Device recognition is responsible for recognizing *instances* of devices from collections of shapes in the layout, computing specified properties of these instances, and preparing the results for use by other processes.

General devices can be modeled using placements of cells from lower in the hierarchy.

Device recognition deals strictly with geometric shapes. Calibre flattens all device placements internally to their underlying geometry before recognizing them as instances.

The Calibre Verification toolset incorporates device recognition into its process. The executive module manages the entire process, which includes:

- Obtaining required data.
- Generating data.
- Running various processes, such as connectivity extraction and device recognition.

This chapter describes only those portions of the system directly related to device recognition.

Connectivity extraction precedes device recognition in the chain of processes. Connectivity extraction recognizes the electrical nets of the layout and annotates layer geometries with net numbers. For further information refer the chapter "Connectivity Extraction". Device recognition uses these net numbers to associate the pins of recognized devices with the nets to which they connect, which enables the forming of an internal layout netlist.

Following device recognition are other processes that are clients for the information produced by device recognition. Which of these clients is active depends on the command being performed. One client is Mask mode LVS for which device recognition prepares a netlist and related information. LVS then compares this layout netlist to a second netlist, usually a schematic, and reports discrepancies. A second device recognition client is parasitic extraction (PEX) for which device recognition prepares a list of pin and port locations. The extractor then analyzes the parasitic resistance and capacitance on the interconnect between these points.

A set of operations in the rule file guides device recognition. These operations govern how devices are recognized and classified, and how properties, such as resistance and capacitance, are computed.

The main rule file operation that identifies devices is the Device statement.

Device Rule Overview

The Device statement:

- Names and classifies a device.
- Specifies how instances of the device are to be recognized.
- Names the pins and their swap groups explicitly, or by default.
- Indicates which, if any, auxiliary layers are associated with the device.
- Defines the properties associated with the device and how they are to be computed.

It has many parts and parameters that supply this information and is discussed in detail in the *SVRF Manual*. Not all of the parts or parameters need be present in any given statement.

The following examples illustrate how this statement is used:

```
DEVICE D diode_layer diode_pin1 diode_pin2
DEVICE R resistor_layer metal_1 metal_2 [1.1]
DEVICE R resistor_layer <implant_layer> metal_1 metal_2 [1.6]
DEVICE R resistor_layer metal_1 metal_2 substrate_layer(SUB)
       //3-pin resistor
DEVICE C (CP) diff_and_poly poly diff_not_poly (POS NEG)
                                                  [1.6 0.07]
DEVICE C (CM) m_cap metal_1 metal_2 (POS NEG) [1.4 0.095]
DEVICE MP (PMOS) diff_and_poly poly diff_not_poly
                                        diff_not_poly [0.5]
DEVICE Q (BJT) base coll base emitt substrate
DEVICE QE2 (BJT2) base emitt1 (E1) emitt1 (E2) base (B)
                                                   coll (C)
DEVICE ARB (MX43) dev_lay  pin_lay_1 (G) pin_lay_2 (H)
                                                pin_lay_3 (I)
     [ property A, P
       A = area(dev_lay)
       P = perim(dev_lay)  ]
```

The examples above show the statements for recognizing a diode, three resistors, two capacitors, a PMOS transistor, a bipolar transistor with a substrate pin, a bipolar device with two emitters, and an arbitrary device having three pins and two user-computed properties.

Concepts and Terminology

Device recognition receives a set of device definitions from the rule file. Each definition defines a particular type of device which is to be recognized. Given these definitions, device recognition analyzes the layout geometry and locates instances of these devices. It is important to recognize the distinction between a device and an instance:

- A device is an abstract template that describes how a collection of shapes can be recognized as an instance of that device.
- An instance is a collection of specific shapes in specific locations that form a physical realization of the abstract device.

A Device statement has many parameters, including:

- **Element name**: Identifies the specific kind of device which corresponds to the element name (also known as component type) in a schematic. Examples are MN and MP for MOS N and MOS P-type transistors, and C for a capacitor.
- **Model name**: Corresponds to the component subtype (by default the MODEL property) in the schematic.
- **Device layer:** Layer which contains device or *seed* shapes.
- **Pin layers:** Layers on which device pins are to be found.
- **Pin names**: Labels the pins of the device. One or more pins can be present.
- **Swap lists:** Identify pin swap groups which allow interchangeable connectivity of device pins.
- **Pin recognition algorithm**: Specifies the algorithm used for device recognition. Choices are By Net (default) and By Shape. Both are discussed in this chapter.
- **Auxiliary layers**: Layers containing shapes that are not pins which aid in the recognition of device instances.
- **Text model layer**: Specifies a text layer in the layout to be used for determining model names for device instances.
- **Properties**: Specify values such as area, perimeter, capacitance, resistance, width, or length.
- **Property specification**: Optional user-defined program that describes how the values of properties are to be computed. This program overrides default property computations for built-in device types.

Device instances are classified based upon the “templates” provided by Device statements. What this means is, the device recognition algorithms will attempt to match device instances based upon Device statements you provide. In this sense,

Device statements are patterns, or templates, which the Device recognition algorithms use to attempt to classify your device instances.

Recognition Logic

The recognition of device instances is the process of identifying sets of interacting device and pin shapes that match the specifications in a Device operation. The process proceeds as follows:

- Scan each layer that appears as a device layer in one or more Device operations, together with all relevant auxiliary and pin layers.
- Identify, for each seed shape, an initial set of auxiliary shapes and pins to which it is connected.
- Classify the device if the resulting pattern of auxiliary shapes and instance pins match one of the Device operations; otherwise attempt to fill in missing pins from initial pins to obtain a unique match.
- Consider the device ill-formed if the pin fill algorithm fails to find a match.

Layer Relations

Device recognition analyzes each layer that appears as a device layer in one or more statements. At the same time that the device layer is examined, a set of auxiliary and pin layers is examined. This set consists of all the auxiliary and pin layers from all Device operations containing the given device layer. Different statements sharing the same device layer compete for classification of each seed shape on that layer.

In the example below, the three statements compete for each shape on layer `dev_lay`.

```
DEVICE D1 dev_lay pin_lay_1(A) pin_lay_2(B)
DEVICE D2 dev_lay pin_lay_1(A) pin_lay_2(B) pin_lay_2(C)
DEVICE D3 dev_lay pin_lay_3(X) pin_lay_4(Y)
```

Since the combined set of auxiliary and pin layers is examined, the interpretation of a Device operation depends on the existence of other operations sharing the same device layer.

In the example above, assume there is a shape on layer dev_lay that touches one pin on each of layers pin_lay_1, pin_lay_2, and pin_lay_3. This shape will fail to match any of the three operations and will be classified as a “bad” device. However, if the operation for Device D3 were eliminated, the shape would be classified as a D1 device. The reason is that in the absence of the D3 operation. Only pin layers pin_lay_1 and pin_lay_2 will be scanned for pins, and the contact with a potential pin on pin_lay_3 would not be noticed.

For device recognition to classify a device shape as an instance the shape must satisfy both auxiliary layer relationships and pin relationships. These relationships are based in part on the notion of shapes *touching* (overlapping or abutting). Contact at a single point such as a corner does not count as touching. The device recognizer has no concept of the third dimension, therefore geometries are treated as planar shapes in the x-y coordinate system. Shielding by other layers is not taken into account.

To satisfy the auxiliary layer relationships, two things must happen.

- The device shape must touch one or more shapes on each auxiliary layer given in the Device operation. It does not matter how many auxiliary shapes the device shape touches on each of these layers, as long as there is at least one on each listed auxiliary layer.

- The device shape must not touch any shapes on auxiliary layers which are not listed in the Device operation if there are other device statements sharing the same device layer and they list these auxiliary layers.

Pin relations are considered if a given device shape passes the auxiliary layer relationship test for one or more Device operations. If not, then the device is classified as ill-formed.

Pin Relations

During the scan of device layers, device recognition forms *initial pins* when a device shape touches one or more shapes on a pin layer. All pin layers associated with the device layer are considered.

- Each shape on a pin layer is treated as a separate pin if the secondary keyword BY SHAPE is specified in the Device operations.
- The nets assigned to the shapes, not the shapes themselves, on a pin layer determine the initial pins on a layer if BY NET is specified either explicitly or by default.

While searching for initial pins on a layer, device recognition classifies all shapes attached to the same net as being part of the same pin. Therefore, two or more shapes on the same layer, having the same net number, form a single pin.

This “one net equals one pin” rule applies only to a single pin layer, and only to the formation of the initial pin set. Shapes on two different layers that are connected to the same net will count as two pins, one pin on each layer. Additional pins for a net on a given layer can also be generated during the “fill in” phase.

Note that in both cases—BY SHAPE and BY NET—if a pin shape touches a device shape in more than one place, it is still counted as only one pin.

Once the initial pin set is formed, the device statements that passed the auxiliary layer test are examined for an exact pin match. The device shape is classified as an instance of that device if the set of initial pins and layers exactly match the pins and layers specified in a Device operation.

The exact match rule allows different devices sharing the same device_layer to be recognized on the basis of the layers on which their pins appear. The rules compiler prevents two Device operations from having identical sets of pins and layers, so an exact match is guaranteed to be unique.

The instance is classified as ill-formed if the exact match rule fails and BY SHAPE is specified.

Device recognition uses a fill-in algorithm to attempt a match if BY NET is specified, either explicitly or by default.

Fill-in Algorithm

This algorithm tries to find a unique match to an operation by duplicating existing pins to fill in for missing pins. More precisely, a *provisional match* is made between the initial pin set and any device definition for which the following three conditions hold:

- The pins and layers of the initial set can be matched with a subset of the pins and layers in the operation.
- The initial pin set contains at least one pin on each pin layer of the operation.
- The initial pin set contains exactly one pin on each pin layer of the operation for which there are any missing pins.

Device recognition classifies an instance as a device and assigns the missing pins on each layer to the same net as the initial pin on that layer if exactly one Device operation generates a provisional match.

The instance is classified as ill-formed if more than one Device operation generates a provisional match.

Ill-formed Devices

When device recognition classifies a shape on a device layer as ill formed, the following actions occur:

- Device reduction ignores any pin and auxiliary shapes the ill-formed shape touches.
- The LVS report lists the ill-formed devices by the (x,y) location of the lower-left corner of the failed device shape, along with the element names from all Device operations using that device layer.

Recognition Example

As an example of several Device operations competing to classify a device shape, consider the following operations for recognizing four different types of lateral pnp-bipolar-junction transistors:

```
DEVICE LPC1 base coll(C) base(B) emitt(E)
DEVICE LPC2 base coll(C1) coll(C2) base(B) emitt (E)
DEVICE LPC3 base coll(C1) coll(C2) coll(C3) base(B) emitt(E)
DEVICE LPE2 base coll(C) base(B) emitt(E1)  emitt(E2)
```

These statements do not specify any auxiliary layers, so device recognition is based on pin relationships.

Shapes on layer base enclose all other pins of the device, therefore they are used as both the device shape and as the base pin shape.

Analysis of each base shape determines how many shapes it touches on layers coll and emitt.

- Shapes that touch exactly one shape on coll, and one on emitt are LPC1 devices.
- Shapes that touch exactly one shape on emitt, and two on coll are LPC2 devices.
- Shapes that touch exactly one shape on emitt, and three on coll are LPC3 devices.
- Shapes that touch exactly one shape on coll, and two on emitt are LPE2 devices.
- Shapes that touch some combination of shapes on coll and emitt, not covered by the above cases, are considered ill-formed.

In this example, the fill-in algorithm will not find a provisional match because of the combinations covered by the four statements.

Property Computation

This section introduces and distinguishes some of the ways in which the computation of properties can be specified. Later sections go into more detail.

LVS uses computed property values when comparing the layout netlist to the schematic netlist. The Trace Property specification statement identifies the property names associated with a device. These properties have nothing to do with parasitic properties associated with interconnect and computed by xCalibre.

After LVS recognizes a device instance, it then computes the property values for the device instance. The property specification and the element name determine the properties to be computed.

In the Device operation, square brackets ([ ]) enclose the user-defined property specification. The specification can consist of either one or two floating point numbers, or of a short program written in a special property computation language.

For each Device operation there are three choices for the property specification:

- No specification given.
- One or two numbers.
- A short program.

For certain reserved element names, some of these choices are invalid.

The element names D, C, R, MN, MP, MD, and ME represent known devices for which default property computations are available. The element name Q represents a bipolar transistor, for which there are no default property computations available, therefore it is treated the same as a user-specified element name.

The interaction between the element name and the property specification is summarized as follows:

- The program form of the property specification can be used with any element name, both reserved and non-reserved. It specifies the properties to be computed and how they are computed. The default properties and default property computations associated with reserved element names are suppressed when the program form is used.

- The list of floating point numbers form of the property specification can only be used with the reserved element names; C, R, MN, MP, MD, and ME. In these cases, a default property computation is used. The element name determines the names and number of properties to be computed. The numbers in the list act as parameters for the computation. An error will result if you use a list of floating point numbers as a property specification with any element name other than those identified above.

- When the property specification is omitted:

    - The properties are computed by the default method for the element names D, MN, MP, MD, and ME.

    - The property values are zero for the element names C and R.

    - No properties are computed for any other element name.

The following section, "Default Property Computations", discusses the default property computations available for element names C, R, MN, MP, MD, and ME. The sections "Built-in Language Details" through "Debugging Property Computations" discuss the use of the special property computation language.

Default Property Computations

Default property computations are available for the reserved element names shown in the following table. You select the default computation by either

omitting the property specification and its square brackets completely, or by supplying the appropriate number of numeric parameters between the brackets.

name	meaning	properties	property parameters	property names
MN	transistor	width, length	effective_width_factor	W, L
MP	transistor	width, length	effective_width_factor	W, L
MD	transistor	width, length	effective_width_factor	W, L
ME	transistor	width, length	effective_width_factor	W, L
D	diode	area, perimeter	(none)	A, P
C	capacitor	capacitance	area_cap perim_cap	C
R	resistor	resistance	resistivity	R

For each Device operation containing a reserved element name from the table above, you can supply appropriate property parameters. An exception is the diode, which requires no parameters. The property parameters are enclosed in a pair of square brackets within the operation. If the property parameters are omitted, they default to zero. The effect of a zero parameter value is discussed in the following subsections.

The actual default internal property computations for M*, C, D, and R devices are given in the “Examples” section of Device in the *SVRF Manual*.

The examples below illustrate the four possible property_parameter setups for reserved element names. Namely, the cases of diode, resistor, capacitor, and MOS transistor. The following subsections will cover the meaning of the parameters for each of these classes in turn.

```
DEVICE D dio_layer pin_lay_1 pin_lay_2
DEVICE R res_layer pin_lay pin_lay [ 1.1 ]
DEVICE C cap_layer pos_layer neg_layer [ 1.6 0.07 ]
DEVICE MP dev_lay gate source drain [ 0.5 ]
```

- **Diodes**. Area and perimeter are computed for diodes. These are the geometric area and perimeter of the device shape on the device layer. The values are expressed in square meters and meters, respectively. Since no

parameters are required for this computation, no property parameters are specified in the operation.

- **Resistors**. Resistance is computed for resistors. One parameter is needed; the resistivity in units of unit_resistance/square. This parameter can be a number, or it can be a reference to a Process variable that contains a numeric value. This parameter must evaluate to a non-negative number. If omitted, the value defaults to zero, and a resistance of zero is returned. The unit of resistance defaults to the ohm, but can be set to some other value by the Unit Resistance specification statement in the rule file. For example, the following statements define the resistivity to be 0.1 kohm/square:

  ```
  UNIT RESISTANCE   kohm
  DEVICE R res_layer pin_lay pin_lay [ 0.1 ]
  ```

  The formula is:

  resistance = r * (L / W)

  where, L is the length and W the width of the resistor, and r is the resistivity parameter supplied by the property parameter in the Device operation.

  The width W is computed to be half of the total perimeter of the POS and NEG pins lying on or within the device shape.

  The length L is computed as A / W, where A is the true area of the device shape.

  This computation is generally valid only for the restricted case of rectangular resistors whose pins exactly abut the ends of the rectangle. Resistors with other shapes, or whose pins contact the interior of the resistor, require a different computation. In the general case, it can be necessary to use appropriate layer operations to alter the geometry prior to computation and then specify the computation using the built-in property programming language.

- **Capacitors**. Capacitance is computed for capacitors. Two parameters are needed; the proportionality constants for area and perimeter capacitance.

The area parameter is in units of unit_capacitance per unit_length squared. The perimeter parameter is in units of unit_capacitance per unit_length. If one parameter is present, they must both be present and the area constant must appear first. These parameters can be numbers, or they can be references to Process variables that contain numeric values. Both numbers must evaluate to non-negative numbers. Specifying either value as zero has a small effect on execution time, since it will eliminate the need to compute the area or perimeter for each instance of the device. If both are omitted, they default to zero and a capacitance of zero is returned. The unit_capacitance defaults to the picofarad, but can be set to some other value by the Unit Capacitance specification statement in the rule file. The unit_length defaults to the micron, but can be set to some other value by the Unit Length specification statement in the rule file. For example, the following statements define the area capacitance to be 300 nanofarads/square mil and the perimeter capacitance to be 10 nanofarads/mil:

```
UNIT CAPACITANCE  nf
UNIT LENGTH  mil
DEVICE C cap_layer pos_layer neg_layer [ 300 10 ]
```

The formula is:

capacitance = (ca * area + cp * perimeter)

where area and perimeter are computed as for diodes, and ca and cp are the parameters supplied by the property parameters in the Device operation. Although the area and perimeter are computed, they are not made available as property values.

- **MOS Transistors**. For MOS transistors, the effective width and effective length of the device are computed with compensation for bends in the device area. One parameter can be specified; the width effect (weffect) constant. This parameter can be a number, or it can be a reference to a Process variable that contains a numeric value. If omitted, it defaults to zero, which indicates that bend compensation is not required. The parameter is a proportionality constant, therefore unitless. If you do not want angle compensation, omit the parameter or supply a value of zero.

The computation proceeds as follows:

- Compute the area A of the device.
- Compute the width W of the device. This value is half of the total perimeter of the source and drain pins lying on or within the device shape.
- Compute the length L of the device. This value is A / W.

The next process depends on whether a non-zero weffect parameter was specified in the property parameter of the Device operation.

- The parameter is zero: W and L are computed as above and are delivered as the width and length properties of the device.
- The parameter is non-zero: The following computation is performed
  - Compute the internal angle I of the device shape. This value is the amount of bend in the centerline of the device shape, and is expressed in units of “right angles.”

    For example, a rectangle has I = 0 and a right-angle “L” shape has I = 1. More formally, I can be computed using the following formula:

    $$I = S / 90$$

    where S is computed by setting it to zero then traversing the boundary of the shape, keeping the interior of the shape on the left. At each vertex where the boundary turns right, add the number of degrees of turn to S. At each vertex where the boundary turns left, do nothing to S.
- After computing I, the width or length is adjusted depending on the proportions of the device shape.

  If W > L, then L is not adjusted, and W is set to the value:

  $$W - weffect * I * L$$

If W <= L, then W is left unadjusted, and L is set to the value:

L - weffect * I * W

The resulting values of W and L are returned as the width and length properties of the device.

Built-in Language Details

The aspects of the property specification language (hereafter called the built-in language), is described in the following subsections.

The property specification, written in the built-in language, is that portion of a Device operation enclosed in square brackets ([ ]).

Only one property specification is allowed per Device operation. It lists the names and types of the properties to be computed for that device and specifies the method of computation from available data. Property specifications cannot be shared by multiple Device operations.

Built-in Language Example

Previously, the only properties available for a PMOS transistor were the effective width (W) and length (L) of the gate. A typical Device operation might have looked as follows:

```
DEVICE mp (pmos ) gate  gate (G) diff (S) diff (D) [ 0.5 ]
```

Suppose you want to compute not only W and L, but also the areas of the source (AS) and drain (AD) pins. If we choose to do this, we must not only add the new computations for AS and AD, but must duplicate the old computations for W and L since it is not possible to combine the old and new methods of computation. Using the built-in language, the altered Device operation might look as follows:

```
1  device mp ( pmos ) gate gate ( G ) diff ( S ) diff ( D )
2  [
3     property  W, L, AS, AD
4     weffect = 0.5
5     AS = area(S)
6     AD = area(D)
```

```
 7    W = ( perim_co (G,diff) + perim_in (G,diff) ) / 2
 8    L = perim_outside(G, diff) / 2
 9    if ( bends(gate) > 0 )
10    {
11       if ( W > L )
12          W = W - weffect * bends(gate) * L
13       else
14          L = L - weffect * bends(gate) * W
15    }
16  ]
```

This example illustrates many of the features of the language. The line numbers on the left are an aid to discussion and are not part of the example text. The language is case insensitive. In this example we have chosen to capitalize only the property and pin names.

- Line 1 contains all of the Device operation, except the property specification, which is contained between the matching square brackets on lines 2 and 16.

- Line 3 contains the property statement that declares that there will be four numeric properties computed and gives their names.

- Line 4 assigns value 0.5 to the temporary variable “weffect”. This variable is later used in the adjustment of W or L on lines 12 and 14.

- Line 5 computes the value of property AS as the area of the pin named S. The pin S is declared on line 1. The “area( )” function is one of the built-in functions available for delivering summarized geometric data.

- Line 6 computes the value of property AD as the area of the pin named D.

- Line 7 computes the physical width of the gate as one half of the perimeter of the gate pin, G, which is coincident with or inside of the perimeter of shapes on the diff layer. The commands perim_co and perim_in are abbreviated forms of perimeter_coincide and perimeter inside, respectively.

- Line 8 computes the physical length of the gate as one half of the perimeter of the gate pin, G, which is outside of the perimeter of shapes on the diff layer.

- Line 9 tests whether there are any bends in the gate pin. If there are, then a further test on line 11 determines whether the gate pin is wider than it is long. If wider, an adjustment to the effective width, W, is made on line 12, otherwise an adjustment to the effective length, L, is made on line 14.

The built-in language appears similar to C or AMPLE, without the semicolons. Semicolons were left out for stylistic compatibility with the remainder of the rule file.

Notational Conventions

The notational conventions used in preparing built-in language statements and functions are identified in Table 9-1.

Table 9-1. Built-In Language Statements

Syntactical Element	Description
Properties	The built-in language is restricted to computation of numeric-valued properties. All values and computations are double-precision floating point. The question of which units to use for representing physical quantities such as length and area is covered in the subsection "Units of Measurement" later in this section.
Style	The built-in language is a blend of the expression and computational statement style common to AMPLE and C, together with the conventions common to the rule file. To this were added necessary declarations and functions.
Structure	The built-in language is structured as a sequence of statements. First there is an optional DEBUG statement, next a required PROPERTY statement and finally a sequence of one or more property computation statements. Each property computation statement is either an assignment statement, an IF statement, or an IF … ELSE statement. There are no loop statements in the language.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description
Statement placement and continuation	Semicolons or other separating devices are not used between statements. The DEBUG, PROPERTY, and IF statements are recognized by their keywords, which must be the first item in the statement. Assignment statements are recognized by the “=” operator, which is always the second item in the statement. Statements can be continued onto multiple lines by breaking them at any white space. Continuation characters are not employed. Common practice dictates that each statement begins on a new line, but this is not required.
Reserved keywords	The keywords of the language such as DEBUG, PROPERTY, IF, and ELSE are reserved. If a word is surrounded by single (‘) or double (“) quotes, it is not taken to be a keyword, but rather a normal identifier for a property, pin, layer, and so forth; or possibly a string valued argument to a function. Therefore, the reserved nature of keywords places no restrictions on property, layer, or pin naming.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description
Optional keywords and function spellings	Long keywords and function names often have a shortened form consisting of a set of letters from the complete name. In this document, the letters in the shortened form are shown uppercase with the remaining letters lowercase. Thus, PROPerty indicates that the spelling is either PROPERTY or PROP. If any of the optional letters are used, they must all be used. Thus, PROPER is not a valid shortening of PROPerty. Some of the function names are compound and include underscores. For example, PERIMeter_INside. In this case, the individual words in the name can be shortened independently. Thus, of the following is an acceptable name for the PERIMeter_INside function: PERIMETER_INSIDE PERIMETER_IN PERIM_INSIDE PERIM_IN
Case sensitivity	The language is case insensitive. All uppercase letters are converted to lowercase for internal purposes. Thus, a variable can be referred to as “weffect”, “Weffect”, or “WEFFECT” interchangeably within the same program. The use of mixed uppercase and lowercase letters in this document is used only to show allowed abbreviations.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description
Data sources	Data used in the computation can come from any of these sources: 1. **Numeric Constants**. You can use numeric floating point constants directly within the property specification. 2. **Process Variables**. These values are accessed by using the Process variable just as any other variable within the property specification. The Process variable must exist and have a numeric value at the time the rule file is loaded, which will be used in all property computations while that rule file is loaded. 3. **Instance Data**. Geometric and connectivity data associated with an instance are accessed by the provided built-in functions. Examples are the AREA and PERIMeter functions. A complete list of the functions and their definitions is given later.
Comments	The comment conventions for the built-in language are the same as for the rest of the rule file. Any text beginning with a double slash (//) through the end of the line is ignored.
Commas	All commas shown in the language are required.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description

DEBUG statement	Tracing of the property computation can be useful in finding errors during the development of new property computation code. Tracing is controlled by the DEBUG statement, which is an optional statement. If present, it must be the first statement. It has the following form:
	
	`DEBUG range1 [ , range2 , … rangek ]`
	
	where each range is either a device instance number, or a pair of device instance numbers separated by a hyphen "-". At least one range must be specified although it can contain only a single instance. For example, the statement:
	
	`DEBUG  0-2, 50`
	
	causes a step by step report of the property computation to be printed for each of the device instances 0,1,2, and 50. The trace can produce much output, so the use of small ranges is recommended. The instance numbers can be obtained from LVS discrepancy reports. See also the INSTance() built-in function below for another method of determining the instance number of a particular device. A more complete discussion of the DEBUG statement with examples is given in the subsection "Debugging Property Computations" later in this section.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description

PROPerty statement	Declares the names of the properties to be computed. This statement is required and, with the exception of the DEBUG statement, must be the first statement. It has the following form:
	
	`PROPerty <prop_name_1>,<prop_name_2>, …`
	
	In the following example, four properties are declared with names W, L, SA and DA:
	
	`PROPERTY W, L, SA, DA`
	
	Property names are treated as variables within computational statements. The final value assigned to the property name is the value of the property associated with the instance.

Numeric constants	Constructed as valid C or AMPLE integer, float, or double constants. Examples: 3 3.0 -2.5 4.6e-10 5e8 5E9 0 1
Local variables	Intermediate values can be assigned to local variables within a property specification. For compatibility with rule file style, it is neither necessary nor possible to declare such variables before use. Any name can be used that does not conflict with property names or Process variables. However, it is best to avoid names beginning with “temp” or “init” since these names are used in debugging output to identify temporary variables. Although the compiler and interpreter will not be confused by your use of these names, you can become so when reading the debugging output.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description

Operators	A subset of the operators of AMPLE and C. These have the same meaning and precedence as in AMPLE and C. They are listed below in order of decreasing precedence.
	`- (unary minus)`
	`! (logical negation)`
	`* (multiplication)`
	`/ (division)`
	`+ (addition)`
	`- (subtraction)`
	`< <= == >= > != (relational operators: lt, le, eq, ge, gt, and ne)`
	`&& (logical AND)`
	`|| (logical OR)`
	`= (assignment)`

Parenthesis	Used in expressions to override normal operator precedence.
Numeric expressions	The arithmetic operators - (unary), +, -, *, and / are used to build up numeric expressions from constants, variables, and function references.
Logical expressions	Basic logical expressions are formed by combining numeric expressions using the relational comparison operators. Logical expression can be further combined using parenthesis together with the logical AND (&&) and logical OR (| |) operators. Logical expressions can only appear within the parenthesis following the keyword IF.
Assignment statements	The assignment statement has the following form: `<local_variable_or_property_name> = <numeric_expression>` You can specify only one “=” operator per assignment statement.

Table 9-1. Built-In Language Statements [continued]

Syntactical Element	Description

Flow control	Flow control is provided by the IF and IF ELSE statements, and have the same meaning and use as in AMPLE and C. These statements have the following forms:
	`IF ( <logical-expression> ) <statement>`
	`IF ( <logical-expression> ) <statement>`
	`ELSE <statement>`

Statement grouping	Curly braces ({}) can be used to group one or more statements into a single statement as in AMPLE and C. For an example of their use see the subsection "Built-in Language Example" previously in this section.

Data Retrieval Functions

A variety of built-in functions are provided for access to geometric and connectivity information about an instance. These functions can be used within numeric expressions, or as an argument to a relational operator.

Most of the functions take one or more arguments that are references to either a layer or a pin. When referencing a pin, the compiler uses the name specified in the Device operation. When referencing a layer, the compiler uses the name or number of the layer, only if it matches the name or number in the layer list of the Device operation. If the reference could be interpreted as either a pin or a layer, the pin interpretation is chosen.

When referencing a layer, you do not reference all shapes on the layer, but only those shapes of the layer that touch or overlap the seed shape of the current device instance. For example, the statement:

```
AREA(lay1)
```

returns the total area of all shapes on layer lay1 that touch or overlap the current instance's seed shape.

When a pin name is used, the function returns summary information about all shapes that are part of that pin. Since auxiliary layers have no pin names, you can reference them only by layer name or number.

The shapes on layers involved in device extraction have been merged prior to being presented to the device extractor. For example, if a pin had been formed by two overlapping shapes drawn on original layer lay5, the perimeter returned by "PERIMETER(lay5)" would be the perimeter of the merged shape, not the sum of the perimeters of the two original overlapping shapes.

Perimeter Functions

The six perimeter functions, described in table 9-2, deal with perimeter length. The function PERIMeter(*pin_or_layer*) gives a simple perimeter while the other five functions report the length of portions of perimeters on one pin or layer as they relate to shapes on a second pin or layer. All values are expressed in meters.

Consider two overlapping shapes on layers A and B as shown in Figure 9-1.

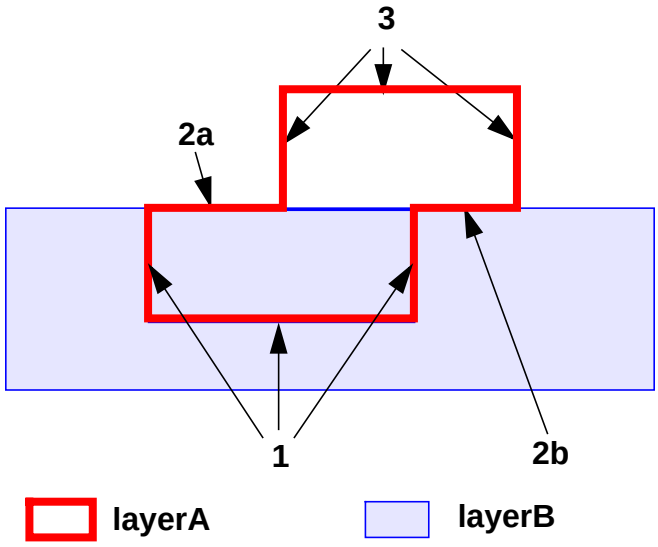


Figure 9-1. Perimeter Relationships

The perimeter of layer A can be divided into three portions depending on its relation to the shape on layer B. These three portions have been labeled with numbers as follows:

1: PERIMETER of A is strictly INSIDE the shape of B.
2: PERIMETER of A and of B COINCIDE.
3: PERIMETER of A is strictly OUTSIDE the shape of B.

The portion labeled 2 can be further subdivided into two portions depending on the relation of the shapes where the perimeters are coincident. These have been labeled as follows:

2a: PERIMETER of A and of B COINCIDE, and shape A lies INSIDE shape B.

2b: PERIMETER of A and of B COINCIDE and shape A lies OUTSIDE shape B.

Use of the three keywords INside, COincide, and OUTside as illustrated above gives rise to the following six perimeter functions:

Table 9-2. Perimeter Functions

Function Name	Portions Totaled
PERIMeter	1 2a 2b 3
PERIMeter_INside	1
PERIMeter_COincide	2a 2b
PERIMeter_OUTside	3
PERIMeter_COincide_INside	2a
PERIMeter_COincide_OUTside	2b

Determining Bends

The bends value can be found by summing the angle, in degrees, by which the perimeter changes direction at all concave vertices, and dividing by 90 to convert to units of “right angle bends”. The sum is over all shapes in the specified pin, or

on the specified layer. The shapes in Figure 9-2 are shown with their concave vertices emphasized and their respective bends values indicated. Note carefully that the change in direction of the perimeter at a vertex is not the same thing as the angle formed by the perimeter at the vertex. For example, in the right most shape in Figure 9-2, the angle formed by the perimeter at the indicated concave vertex is 135 degrees, but the change in perimeter direction as you walk along the perimeter is 45 degrees or 1/2 of a right angle. Thus the bends value for this shape is 1/2.

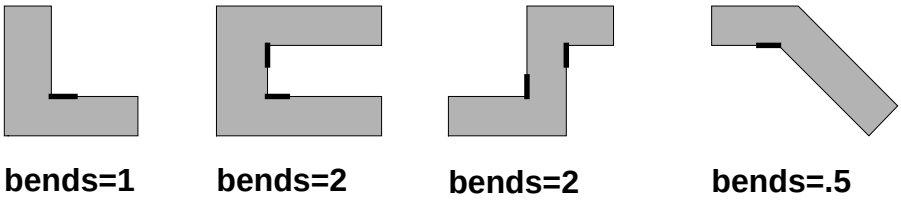


Figure 9-2. Computation of Bends

Built-In Functions

Table lists the data retrieval functions available for the built-in language, along with a description of their use.

Table 9-3. Built-in Functions

Function	Description
Area functions:	
Area	The format is: `AREA(`*`pin-or-layer`*`)` Returns the total area of shapes that are part of the specified pin, or on the specified layer. This value is expressed in square meters.
Area Common	The format is: `AREA_COMmon(`*`pin-or-layer`*`,`*`pin-or-layer`*`)` Returns the total area that is common to shapes on both of the pins or layers referenced. This value is expressed in square meters.

Table 9-3. Built-in Functions [continued]

Function	Description
Bends	The format is: `BENDS(`*`pin-or-layer`*`)` Returns the total bends in the shapes of the specified pin or on the specified layer. The result is expressed in units of right angles. For information about how bends are calculated, refer to "Determining Bends" above.
Count	The format is: `COUNT(`*`pin-or-layer`*`)` Returns the total number of shapes on the specified layer or in the specified pin. In device extraction, pins on a given layer are determined by the nets to which they attach, not by the number of shapes present, so it is possible for a pin to contain more than one shape. The layers are merged before being presented to device extraction, so the shapes counted will not touch or overlap.
Perimeter functions:	
Perimeter	The format is: `PERIMeter(`*`pin-or-layer`*`)` Returns the total length of the perimeter of the shapes in the specified pin or on the specified layer.
Perimeter Inside	The format is: `PERIMeter_INside(`*`pin-or-layer`*`,` *`pin-or-layer`*`)` Returns the total length of the parts of perimeters on the first pin or layer that lie strictly inside shapes of the second pin or layer. Corresponds to portion 1 in figure 9-1 above.

Table 9-3. Built-in Functions [continued]

Function	Description
Perimeter Outside	The format is: `PERIMeter_OUTside(`*`pin-or-layer`*`,` *`pin-or-layer`*`)` Returns the total length of the parts of perimeters on the first pin or layer that lie strictly outside shapes of the second pin or layer. Corresponds to portion 3 in figure 9-1 above.

Perimeter Coincide	The format is:
	`PERIMeter_COincide`
	`(`*`pin-or-layer`*`, `*`pin-or-layer`*`)`
	Returns the total length of the parts of perimeters on the first pin or layer which lie strictly inside shapes of the second pin or layer. Corresponds to portions 2a and 2b in figure 9-1 above

Perimeter Coincide Inside	The format is:
	`PERIMeter_COincide_INside`
	`(`*`pin-or-layer`*`, `*`pin-or-layer`*`)`
	Returns the total length of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer, and where the shape of the first layer is inside the shape of the second layer. Corresponds to portion 2a in figure 9-1 above.

Perimeter Coincide Outside	The format is:
	`PERIMeter_COincide_OUTside`
	`(`*`pin-or-layer`*`, `*`pin-or-layer`*`)`
	Returns the total length of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer, and where the shape of the first layer is outside the shape of the second layer. Corresponds to portion 2a in figure 9-1 above.

Table 9-3. Built-in Functions [continued]

Function	Description
Instance	The format is: `INSTance( )` Returns the instance number of the current device instance. Each device extracted has a non-negative instance number that is unique over all devices of all types extracted during that extraction run. LVS reports the instance numbers of devices as part of the property discrepancy report. This function allows these numbers to be turned into a property value so that they can seen in other contexts.
Net functions:	These functions are useful for detecting pins that are connected to special nets such as power or ground.
Named Net	The format is: `NAMED_NET(`*`net-name`*`)` Returns the number of a net. It requires a net name enclosed in quotation marks (“ ”) as its argument and returns the number of the net that has that name. If there is no net with that name, it returns the special value 0, which is never a valid net number and in this case, indicates that there is no net with the given name.

Pin Net	The format is:
	`PIN_NET(`*`pin-name`*`)`
	Returns the number of a net. Requires a pin name as its argument and returns the number of the net attached to the pin. As an example, the following construct will execute case 1 if the D pin is connected to VDD and will execute case 2 otherwise:
	`if ( pin_net(D) == name_net( “VDD” ) )`
	`  // case 1`
	`else`
	`  // case 2`

Table 9-3. Built-in Functions [continued]

Function	Description
Numeric functions	
Absolute Value	The format is: `ABS(`*`numeric-expression`*`)` Returns the absolute value of the specified numeric expression.
Exponent	The format is: `EXP(`*`numeric-expression`*`)` Returns the value of *e* (Napier's constant, the base of natural logarithms) raised to the power of the numeric expression.
Logarithm	The format is: `LOG(`*`numeric-expression`*`)` Returns the natural logarithm of the specified numeric expression. The value of the numeric expression must be greater than zero.
Power	The format is: `POW(`*`numeric-expression`*`,` *`numeric-expression`*`)` Returns the value of the first expression raised to the power of the second expression. If the first expression is zero, the second must be positive. If the first expression is negative, the second expression must be an integer.
Square Root	The format is: `SQRT(`*`numeric-expression`*`)` Returns the square root of the specified numeric expression. The value of the numeric expression must be zero or greater.

Table 9-3. Built-in Functions [continued]

Function	Description

Truncate	The format is:
	`TRUNC(`*`numeric-expression`*`)`
	Returns the result of truncating the fractional part of the specified numeric expression. Truncation is toward zero. The absolute value of the numeric expression must not exceed 2,147,483,647. For example:
	`TRUNC(2.1)  //returns 2`
	`TRUNC(2.9)  //returns 2`
	`TRUNC(-2.1)  //returns -2`
	`TRUNC(-2.9)  //returns -2`

Unit functions:	
Precision	The format is: `PRECISION( )` Returns the current value of the process precision (1000 by default). The precision is the number of database units per user unit.
Unit Time	The format is: `UNIT_TIME( )` Returns the current value of the unit-time as specified in the rule file by the UNIT TIME statement. The unit-time is the size of the user time unit expressed in seconds.
Unit Resistance	The format is: `UNIT_RESistance( )` Returns the current value of the unit-resistance as specified in the rule file by the UNIT RESISTANCE statement. The unit-resistance is the size of the user resistance unit expressed in ohms.
Unit Length	The format is: `UNIT_LENgth( )` Returns the current value of the process unit-length, normally 1E-6. The unit-length is the length of a user unit expressed in meters.

Table 9-3. Built-in Functions [continued]

Function	Description
Unit Capacitance	The format is: `UNIT_CAPacitance( )` Returns the current value of the unit-capacitance as specified in the rule file by the UNIT CAPACITANCE statement. The unit-capacitance is the size of the user capacitance unit expressed in farads.

X-Y location	The formats are:
	`X_LOCation(`*`pin-or-layer`*`)`
	`Y_LOCation(`*`pin-or-layer`*`)`
	Returns the x and y coordinates in user units associated with the given pin or layer. This coordinate information can be used to help locate a pin or device instance. For seed layers, the point whose coordinates are reported is the lowest of the leftmost of the points of the seed shape. For pin layers and auxiliary layers, the point whose coordinates are reported is an arbitrarily chosen point on the lowest of the leftmost of the edges common to the given pin or layer and to the seed shape.

More Built-in Language Examples

```
////////////////////////////////////////////////////////////
//    Example rule file for Computing Device Parameters
//             NRS, NRD, AS, AD, PD, PS, L, W
////////////////////////////////////////////////////////////
//------------------------------------------------------------
//This section contains example layer definitions, layer
//derivations, and connect statements.
//------------------------------------------------------------
layer ipoly      4
layer diff       5
layer contact    6
layer metal1     8
layer pwell      10
ngate   = diff AND ipoly
```

```
nsd     = diff NOT ngate
connect metal1 nsd ipoly by contact
connect pwell
nsd_rs1 = nsd not contact
nsd_rs2 = nsd_rs1 coincident edge ngate
nsd_rs3 = nsd_rs1 coincident edge contact
nsd_rs  = int nsd_rs2 nsd_rs3 < 100 parallel opposite region
//Note, instead of 100, the above command should use the
//actual largest distance a source/drain contact would ever
//be from the edge of a gate, in your process.
////////////////////////////////////////////////////////
//Note, there are two device descriptions below. The first
//addresses ordinary device configurations. The second
//addresses more exotic device configurations, particularly,
//  a. devices in which both source and drain regions have no
//     contacts (example: three or more devices in series)
//  b. devices in which no contact resides within 100u (or
//     whatever distance you specify) from gate edge
//  c. devices in which no contact has any edge facing the
//     gate edge.  If you find any computation wherein you
//     require more accuracy, please contact Mentor Graphics
//     Customer Support.
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
//                  Device Description Example 1
//
//Note, we have added the auxiliary layers “diff” and
//”nsd_rs” to this first device description; we'll use diff
//in the AS, AD, PS, PD property computations below to
//account for shared source/drain; we'll use nsd_rs in the
//NRS, NRD property computations below.
////////////////////////////////////////////////////////
device mn ngate ipoly(G) nsd(S) nsd(D) pwell(B) <diff>
                                                <nsd_rs>
[
 property W, L, AD, AS, PD, PS, NRS, NRD
 bend_effect = 0.5
//-------------------------------------------------------
//This section of the property computations measures gate
//length and width. The “if” clause accounts for any bends
//which can exist in the gates.
//-------------------------------------------------------
 W = perimeter_coincide(ngate, nsd) / 2
```

```
 L = (perim(ngate) - perimeter_coincide(ngate, nsd)) / 2
   if (bends(ngate) > 0)
   {
     if (W > L)
       W = W - bend_effect * bends(ngate) * L
     else
       L = L - bend_effect * bends(ngate) * W
   }
//-------------------------------------------------------------
//This section of the property computations measures Area of
//Source and Area of Drain, even in cases of shared
//source/drain. Note, Since the capacitance effects of AS and
//AD are a function of source/drain area and perimeter, AS
//and AD are not affected by bends in the source/drain
//regions.
//-------------------------------------------------------------
 AS = area(S) * (W / perimeter_inside(S, diff))
 AD = area(D) * (W / perimeter_inside(D, diff))
//-------------------------------------------------------------
//This section of the property computations measures
//Perimeter of Source and Perimeter of Drain, even in cases
//of shared source/drain. Note, since the capacitance effects
//of PS and PD are a function of source/drain perimeter, PS
//and PD are not affected by bends in the source/drain
//regions.
//-------------------------------------------------------------
 PS = perimeter(S) * W / perimeter_inside(S, diff)
 PD = perimeter(D) * W / perimeter_inside(S, diff)
//-------------------------------------------------------------
//This section of the property computations measures Number
//Resistance Squares in Source and Number Resistance
//Squares in Drain, in terms of a first order approximation.
//Note,
//1. The following calculations use edges of contacts,
//   instead of centers of contacts. That is, NRS =
//   average_distance_from_gate_to_contacts' _nearest_edges /
//   width_of_gate
//2. The following calculations fully account for relative
//   placement of contacts to gate and to each other, with
//   the single exception that contacts which have no edges
//   that face the gate edge are not involved in the
//   calculation
//3. Calculations assume all contacts are equally sized.
```

```
 SUM_S_LENGTH = perimeter_inside(nsd_rs, S) -
                                perimeter_coincide(nsd_rs, S)
 COUNT_S = trunc((count(nsd_rs) *
                                perimeter_coincide(nsd_rs, S)
  / perimeter_coincide(nsd_rs, G)) + 0.5)
  IF (COUNT_S != 0)
  {
    NRS = SUM_S_LENGTH / COUNT_S / W / 2   }
  ELSE {
    NRS = AS / (W * W)
  }
 SUM_D_LENGTH = perimeter_inside(nsd_rs, D) -
  perimeter_coincide(nsd_rs, D)
 COUNT_D = count(nsd_rs) - COUNT_S
   IF (COUNT_D != 0)
   {
     NRD = SUM_D_LENGTH / COUNT_D / W / 2   }
   ELSE {
     NRD = AD / (W * W)
   }
]
////////////////////////////////////////////////////////
//                 Device Description Example 2
//
//This second device description should be used along with
//the first, if your design has any of the following device
//configurations:
//  a. devices in which both source and drain regions have no
//     contacts (example: three or more devices in series)
//  b. devices in which no contact resides within 100u (or
//     whatever distance you specify) from gate edge
//  c. devices in which no contact has any edge facing the
//     gate edge example: contact resides in source/drain
//     “dog-leg”
//Note, we have added the auxiliary layer “diff” to this
//device description; we'll use diff in the AS, AD, PS, PD
//property computations below to account for shared
//source/drain.
////////////////////////////////////////////////////////
device mn ngate ipoly(G) nsd(S) nsd(D) pwell(B) <diff>
[
//  property W, L, AD, AS, NRD, NRS, PD, PS
//  The line above is commented out until NRD and NRS
```

```
//  computations are added, to prevent syntax error upon
//  loading rule file.
property W, L, AD, AS, PD, PS, NRS, NRD
bend_effect = 0.5
//-------------------------------------------------------------
//This section of the property computations measures gate
//length and width. The “if” clause accounts for any bends
//which can exist in the gates.
//-------------------------------------------------------------
W = perimeter_coincide(ngate, nsd) / 2
L = (perim(ngate) - perimeter_coincide(ngate, nsd)) / 2
  if (bends(ngate) > 0)
  {
    if (W > L)
      W = W - bend_effect * bends(ngate) * L
    else
      L = L - bend_effect * bends(ngate) * W
  }
//-------------------------------------------------------------
//This section of the property computations measures Area of
//Source and Area of Drain, even in cases of shared
//source/drain. Note, since the capacitance effects of AS and
//AD are a function of source/drain area and perimeter, AS
//and AD are not affected by bends in the source/drain
//regions.
//-------------------------------------------------------------
AS = area(S) * (W / perimeter_inside(S, diff))
AD = area(D) * (W / perimeter_inside(D, diff))
//-------------------------------------------------------------
//This section of the property computations measures
//Perimeter of Source and Perimeter of Drain, even in cases
//of shared source/drain, Note, since the capacitance effects
//of PS and PD are a function of source/drain perimeter, PS
//and PD are not affected by bends in the source/drain
//regions.
//-------------------------------------------------------------
PS = perimeter(S) * W / perimeter_inside(S, diff)
PD = perimeter(D) * W / perimeter_inside(S, diff)
//-------------------------------------------------------------
//This section of the property computations measures Number
//Resistance Squares in Source and Number Resistance Squares
//in Drain. Note, in the case where neither the source or
//drain has any contacts,
```

```
//NRS = area of source / width / width
//That is,
//average_length_of_AS = AS / W
//number_resistance_squares = average_length_of_AS / W
//-------------------------------------------------------------
NRS = AS / (W * W)
NRD = AD / (W * W)
]


////////////////////////////////////////////////////////
//                    Device Description Example 3
//
//This example computes source and drain areas for transistors
//that share a common source or drain.
//
//The figure below illustrates two MOS transistors that share
//a common pin. The source pin of the top transistor is also
//the drain pin of the bottom transistor. In the device
//property computation, the data retrieval function area (S)
//of the top transistor would give the same area as area (D)
//of the bottom transistor. If both area() functions were
//reported, say to a netlist, then obviously, the common pin
//area is over-calculated.
//
//The solution would be to report only a portion of the common
//area to each transistor. The suggested solution assigns the
//larger transistor the larger area, proportionally. For
//example, the source area of the top transistor might be
//calculated as:
//
//          AS = area(S) * (W1 / (W1 + W2))
//
//It is easy to to calculate the width of the transistor. For
//example, W1 of the top transistor can be realized as:
//
//          W1 = perimeter_coincide( gate, sd ) / 2
//
//Where gate = poly and diff; sd = diff not gate. Note that W2
//could be calculated the same way.
//
//The question might be asked is how to compute (W1 + W2)
//while the top transistor is being recognized. We can
```

```
//compute the total “width” length of all transistors that
//share the common pin using a device property computation.
//In this example, (W1 + W2) can be evaluated as:
//
// shared_source = perimeter_inside( S, diff )
//
// where shared_source equals W1 + W2.
//
//         |             |
//    _____|___________D|____
//         |            G|
//    _____|_____________|____poly
//         |     W1    S|
//        \              |
//         \             |
//          \   diff     |
//           \           |
//            \          |
//             |         |
//        ______|__W2__D|____
//             |       G|
//        ______|________|____poly
//             |       S|
//             |        |
//
```

Units of Measurement

You should consider the selection of appropriate units when computing numeric properties that represent physical quantities.

Consider, for example, the case of computing the capacitance for a capacitor. The basic formula might be

C = perim_factor * perim(device) + area_factor * area(device)

To compute a proper value for C, the following seven questions must be answered:

1. In what units is perim(device) delivered?

2. In what units is area(device) delivered?

3. In what units should C be expressed?

4. In what units should area_factor be expressed?

5. In what units should perim_factor be expressed?

6. What is the appropriate value for area_factor?

7. What is the appropriate value for perim_factor?

The standard units for representing the values of time, length, area, capacitance, and resistance are:

```
          time:  seconds
        length:  meters
          area:  square meters
capacitance:  farads
  resistance: ohms
```

The questions above are answered as follows:

1. The value returned by perim(device) would be expressed in meters. For example, if the device had a perimeter of 4 microns, the number returned by perim(device) would be 4E-6, since 4 microns equals 4E-6 meters.

2. The value returned by area(device) would be expressed in square meters. If the device had an area of one square micron, the number returned by the area function would be 1E-12, since one square micron equals 1E-12 square meters.

3. The value of C depends on the intended use of the property. If it is to be used for comparison by LVS with a property value in a schematic, then the units used in the schematic must be known. Typically for capacitance, the schematic units used are farads, although they can appear to be picofarads. The reason they can appear to be picofarads is that one is likely to encounter a property such as “C = 5p”,where the capacitance being represented is certainly five picofarads. However, the actual number being

presented is 5E-12 because p is a scaling factor of 1E-12. Therefore, the actual units being used are farads because this number that looks like “5 picofarads” is really 5 x 1E-12 farads. The “p” allows the reader to think in picofarads while actually using farads. Since the schematic uses farads, it is obligatory for the property evaluation formula to compute the result in farads.

4. The units of perim_factor should be expressed in farads/meter, since the perimeter is in meters and the capacitance is in farads.

5. The units of area_factor should be expressed in farads/square meter, since the area is in square meter and the capacitance is in farads.

6. The value of perim_factor can be answered if information about the process is available. You must be careful to give the value in the units determined in question 4. For example, suppose that the perimeter capacitance factor is 0.05 pf/u. This value must be re-expressed in units of farads/meter before it is used in the formula. Now:

   0.05 pf/u = 0.05E+6 pf/m = 0.05E-6 f/m = 5E-8 f/m

   Therefore, the value 5E-8 should be used in the formula.

7. The value of area_factor can be answered in a similar fashion to question 6.

This example shows the basic issues involving units that will be encountered with most problems:

- Knowing the units in which information is available.
- Knowing the units in which information must be presented to LVS.
- Knowing the units in which constants must be expressed.
- Determining the constant values when expressed in the appropriate units.

Property Computation Structure

At the time a rule file is loaded, each Device operation containing a property computation causes the generation of a *value array*.

The value array is an array of double-precision numbers indexed from zero through some maximum number. There is only one value array for each Device operation. The entries in this array are used during the computation of properties for each instance of the device. The same array is used over again for each instance. Some of the entries in the array represent property values that are to be computed for the instance, some contain constants or the values of Process variables, some represent data values of the instance such as areas or perimeters, some represent local variables in the program, and some represent temporary variables that are needed during the course of the computation.

Table 9-4 shows a sample listing of a value array:

Table 9-4. Value Array Listing

Index	Type	Name (as shown in debugging output)
0	property value	w
1	property value	l
2	property value	as
3	property value	ad
4	local variable	weffect
5	constant	0.5
6	data value	area(S)
7	data value	area(D)
8	data value	perimeter_coincide(G, diff)
9	data value	perimeter_inside(G, diff)
10	constant	2
11	data value	perimeter_outside(G, diff)

Table 9-4. Value Array Listing

Index	Type	Name (as shown in debugging output)
12	data value	bends(G)
13	constant	0
14	temporary variable	temp1
15	temporary variable	temp2

After device recognition loads the rule file, it creates the array, zeroes all entries, and sets the values of constants and Process variables into the array. The constants and Process variables are never changed. Each time a layout verification command using device recognition is issued, an initiation computation is performed, followed by an evaluation computation for each instance recognized.

The initialization computation computes values that are independent of instance specific data, and stores the results into the value array. The results can be stored in property variables, local variables, or temporary variables. The determination of which parts of the program are instance independent is done automatically by the rule file compiler. There can be no instance independent parts of the program, in which case the initialization computation does nothing.

As each instance is recognized, the data values associated with the instance are loaded into the data value positions in the array and the evaluation computation is performed. When the evaluation computation terminates, the property values are retrieved from the array and stored with other information about that specific instance.

Efficiency Considerations

Since property computations can be performed for each of a large number of device instances, it is important to consider efficiency when writing property specifications using the built-in language. This section contains a few tips for writing efficient computations. You are assumed to have read the section "Property Computation Structure" for an understanding of the value array, and the initiation and evaluation phases of computation.

How the Compiler Optimizes

The rule file compiler seeks to move parts of the computation from the evaluation phase to the initiation phase. This is because the initiation computation is performed only once per run, whereas the evaluation computation is performed once per device instance recognized.

The parts of the computation that can be moved to the initiation phase are those that are instance-independent. That is, they do not depend on data from a recognized instance. Since they do not depend on instance data, they can be performed only once during initiation and the results used many times during evaluation.

The source of instance-dependent data is the following functions:

All AREA functions	X_LOCATION()
All PERIMETER functions	Y_LOCATION()
COUNT()	PIN_NET()
BENDS()	INSTANCE()

All other functions, including NAMED_NET(), deliver instance-independent data.

The compiler uses two strategies for moving computations to the initiation phase.

1. Locate the first line in the program that contains a call to an instance dependent function. This is considered the maximum initial portion of the program.

2. Move all preceding lines to the initiation phase, provided this does not split an IF/ELSE structure between initiation and evaluation. If necessary it will move fewer lines to initiation to avoid splitting an IF/ELSE computation between phases.

Once it has moved the initial instance-independent portion of the program to the initiation phase, it examines the remaining expressions in the program and locates all subexpressions that are instance-independent. It moves the computation of these subexpressions to the initiation phase and stores the results in temporary

variables whose names begin with “init.” During the evaluation computation, the values stored in these variables are used in place of the original subexpressions.

In examining subexpressions the compiler treats constants, Process variables, and instance independent function calls as instance-independent. However, it is currently not clever enough to determine if a reference to a local variable or a property variable is instance-independent, so it treats them as instance-dependent.

The following example illustrates some of the above concepts. In this example you should assume that P and Q represent pin names, and that “normal_adjustment” is a Process variable.

```
 1  [
 2     property ap, aq, inst
 3     power_adjustment = 0.04 + normal_adjustment
 4     power_net = named_net(“VDD”)
 5     if ( pin_net(P) == power_net )
 6        pin_adjustment = power_adjustment
 7     else
 8        pin_adjustment = normal_adjustment
 9     ap = area(P) + pin_adjustment
10     aq = area(Q) + 2 * normal_adjustment
11     inst = instance()
12  ]
```

In the example above, lines 3 and 4 are instance-independent, but line 5 is not. Therefore, the computations performed in lines 3 and 4 are done during the initiation phase. That is, the values of the variables power_adjustment and power_net are computed and placed in the value array during initiation. The evaluation phase will access these variables directly from the value array. Also, the subexpression 2 * normal_adjustment on line 10 is instance-independent. A variable “init1” will be created in the value array to store the value of this subexpression. Line 11 will then be treated as if it read aq = area(Q) + init1.

You should note that the compiler’s two strategies do not find all of the instance-independent computations that could be moved. For instance in the example above, if the statement inst = instance() had been moved from line 11 to just ahead of line 3, then there would have been no instance-independent initial segment of the program and the only optimizations the compiler could perform would be on the two instance-independent subexpressions 0.04 + normal_adjustment and

2 * normal_adjustment. However, by following the programming tips of the next section, you can insure that absolutely all instance independent computations are computed only once in the initiation phase.

Efficient Code Examples

Given the knowledge of how the value array works and how the compiler optimizes, here are some tips on writing clear and efficient property specification programs. The tips include things to do, as well as things you do not have to worry about.

- **Create an instance-free initial segment**. Try to create an initial set of statements in the program in which all of the instance-independent computations are performed and stored in local variables. This portion will be executed only once in the initiation phase. If, instead, you intermix independent and dependent portions, the compiler will still attempt to move independent sub-expressions to the initiation phase, but it is not currently clever enough to move entire assignment statements.

- **Use variables for constants**. You can assign constants to mnemonic variables in the initial instance independent section of your program without paying a performance penalty. This allows you to give the constants meaningful names. For example the following two programs are equally efficient in the evaluation phase. The second one performs the assignment bend_effect = 0.5 during initiation. Then during each evaluation, it accesses the bend_effect value just as efficiently as the first program accesses the constant 0.5 value.

```
[ // Use of unnamed constant. (OK)
   property w
   w = perimeter_coincident(G,pin_layer) - 0.5*bends(G)
]
[ // Use of named constant. (Just as fast, but possibly
                                        //more meaningful)
   property w
   bend_effect = 0.5     // This takes place only once at
                                        //initiation time.
   w = perimeter_coincident(G,pin_layer) -
                                     bend_effect*bends(G)
]
```

- **Parenthesize instance-independent subexpressions**. You might have to use parenthesis to get the compiler to notice certain instance-independent expressions. For example, consider the following assignment statement where var is a Process variable:

  ```
  a = area(P) + 0.5 + var
  ```

  The compiler will not find the instance independent subexpression 0.5 + var in the above since by default it does additions from left to right and hence treats it as if were parenthesized as follows:

  ```
  a = ( area(P) + 0.5 ) + var
  ```

  To optimize the original statement you can introduce your own parenthesis as follows:

  ```
  a = area(P) + ( 0.5 + var )
  ```

  You are telling the compiler to compute 0.5 + var before adding in area(P). In this case it recognizes (0.5 + var) as being instance-independent and will compute it only once during initiation.

  A second way to handle this situation is to use the “variables for constants” trick of the previous tip by computing 0.5 + var and storing it in an appropriately named local variable in the initial portion of your program.

- **Use data functions directly**. The values of any instance data functions, to which you refer, are computed only once per instance and stored in the value array before the property computation is executed for that instance. That is, if you refer to the same instance data function with the same arguments in more that one place in your program, you are simply accessing the precomputed value, not causing it to be computed from scratch each time you reference it. Therefore, you do not have to store the value away in a local variable. Indeed the assignment to a local variable will be an extra step and will slow the evaluation.

For example, the first of the following two programs will take more time to evaluate since it contains an extra assignment statement, which must be executed at evaluation time.

```
[ // Use of function values indirectly through local
                                        //variable. (SLOWER)
  property w, l
  common_perim = perimeter_coincident(G, pin_layer)
  w = common_perim
  l = perim(G) - common_perim
]

[ // Use of  function values directly. (FASTER)
  property w, l
  w = perimeter_coincident(G, pin_layer)
  l = perim(G) - perimeter_coincident(G, pin_layer)
]
```

- **Avoid expensive geometry functions (if possible**). Some of the geometric instance data functions are more expensive (time consuming) to compute than others. In some cases an expression involving more expensive functions can be rewritten to use less expensive functions.

  - Functions that request information about geometry on the boundary of, or within the interior of the seed shape are the least expensive. For example AREA(device_layer) or AREA_COMMON(device_layer, pin_layer) fall in this class.

  - Functions that require information about a single pin or auxiliary layer or about geometry lying strictly outside the seed shape are more expensive. For example AREA(pin_layer) or PERIMETER_OUTSIDE(pin_name, device_layer) are in this class.

  - The most expensive functions are those that ask for information about geometric interactions between different pins, pin layers, or auxiliary layers. For example AREA_COMMON(pin_1, pin_2) or PERIMETER_OUTSIDE(pin_1, auxiliary_layer) are in this class.

  Shown below is a list of the functions classified according to expense. In the list dev_lay represents the device layer or the device as a pin, and

pin_lay, pin_lay_1, … represent other pins or layers including auxiliary layers.

- **Least expensive functions** (geometry on or within seed shape):
  area(dev_lay)
  area_common(dev_lay, pin_lay)
  area_common(pin_lay, dev_lay)
  perimeter(dev_lay)
  perimeter_inside(dev_lay, pin_lay)
  perimeter_inside(pin_lay, dev_lay)
  perimeter_coincide(dev_lay, pin_lay)
  perimeter_coincide(pin_lay, dev_lay)
  perimeter_outside(dev_lay, pin_lay)
  perimeter_coincide_inside(dev_lay, pin_lay)
  perimeter_coincide_inside(pin_lay, dev_lay)
  perimeter_coincide_outside(dev_lay, pin_lay)
  perimeter_coincide_outside(pin_lay, dev_lay)
  bends(dev_lay)
  x_location(dev_lay) & y_location(dev_lay)
  x_location(pin_lay) & y_location(pin_lay)

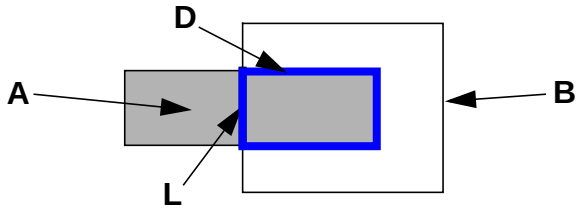
- **More expensive functions** (single pin or layer geometry, or geometry outside of seed shape):
  area(pin_lay)
  perimeter(pin_lay)
  perimeter_outside(pin_lay, dev_lay)
  count(pin_lay)
  bends(pin_lay)

- **Most expensive functions** (interaction between non-device layer shapes)
  area_common(pin_lay_1, pin_lay_2)
  perimeter_inside(pin_lay_1, pin_lay_2)
  perimeter_coincide(pin_lay_1, pin_lay_2)
  perimeter_outside(pin_lay_1, pin_lay_2)
  perimeter_coincide_inside(pin_lay_1, pin_lay_2)
  perimeter_coincide_outside(pin_lay_1, pin_lay_2)

The following is an example that illustrates the above ideas. Suppose the devices to be recognized have a pin “P”, which is know to always lie strictly within the area of the seed shape on layer dev_lay; therefore the two functions shown below are numerically equivalent. However, the area requested in the first function is not known by the compiler to lie entirely within the seed shape. In the second function the area to be computed is guaranteed to lie entirely inside of the seed shape on dev_lay and is thus less expensive to compute.

```
AREA(P)
AREA_COMMON(P, dev_lay)
```

For another example, in Figure 9-3 A and B are two rectangular shapes that intersect to form a third rectangular shape D. In the device rule, D is the device shape and A and B are pin shapes. In the property computation we wish to find the length of the vertical boundary at the left of the D shape. Either of the two expressions shown will do the job, but the first involves the interaction of two pins and the second involves a pin and the device shape. Thus the second will be more efficient.



L=perim_in(B,A) //More expensive
L=perim_in(D,A) //Less expensive

Figure 9-3. Efficient Function Choice

Finally, in some situations, you might have a choice of which layer to use for the seed layer. If so, you might consider using a seed layer whose shapes contain most of the other shapes involved so that the “geometry on or within seed shape” rule will apply to more of the functions that must be computed.

Debugging Property Computations

This section discusses how to use the DEBUG statement to track down errors in a property computation that has been specified by use of the built-in language. We assume you have already read the section above section "Property Computation Structure" on how the property computations are structured.

Suppose you have just written a new Device operation containing a property specification written in the built-in property computation language. It is possible you will have made some mistakes within the specification.

The first mistakes seen will probably be errors detected by the rule file compiler. The rule file compiler will generate an error message for the first such mistake it finds. After correcting this error, reload the rule file to find the next error. Repeat this process until the rule file loads successfully.

Having successfully loaded the rule file, you now use the rule file in an LVS comparison of the layout versus the schematic (LVS). Errors can remain that cause a PROPERTY ERRORS section to appear in the LVS report. Some of these are apparent by studying the property specification in light of the numbers produced. Others might not be so apparent. It is at this point that the DEBUG statement is useful. By using the DEBUG statement you can obtain, for any device instances you choose, a detailed analysis of the computation, what values went into it, what values came out of it, and a step by step analysis of how the computation was performed.

Debug Example

This section is devoted to a single example. The Device operation from a rule file is shown below. The statement appeared on lines 23-38 of the file. The line numbers shown below are for reference and are not part of the Device operation or the file itself. However, the line numbers are used in the debugging output to identify which line of the program is responsible for each step in the step by step analysis.

```
23  device mp(pmos) gate gate(G) diff(S) diff(D) base(B)
24  [
25     property W, L, AS, AD
26     bend_effect = 0.5
```

```
27     AS = area(S)
28     AD = area(D)
29     W = (perimeter_coincide(G, diff) +
                                perimeter_inside(G, diff) ) / 2
30     L = perimeter_outside(G, diff) / 2
31     if ( bends(G) > 0 )
32     {
33       if ( W > L )
34         W = W - bend_effect * bends(G) * L
35       else
36         L = L - bend_effect * bends(G) * W
37     }
38   ]
```

Property Error Report

Now suppose the rule file containing the statement above is loaded and used in a flat LVS comparison that results in the following PROPERTY ERRORS section of the LVS discrepancy report:

```
*************************************************************
                        PROPERTY ERRORS
DISC#  LAYOUT                              SOURCE           ERROR
*************************************************************
  1    5 (1258.000,390.000) (mp)      m5
       ad: 5.6e-11                    ad: 6e-11             7%
       as: 7.2e-11                    as: 6e-11            21%
        w: 31u                         w: 30u               4%
```

We see that the property values for AD, AS, and W do not compare properly. Having failed to resolve the discrepancy in any other way, we begin to suspect the computation itself and decide that we must use the DEBUG statement to get more information about the computation for this instance.

The first thing we must do is identify the device instance associated with the discrepancy as well as the Device operation associated with the instance.

The device instance is identified on the first line of the discrepancy, where:

- “1” is the discrepancy number.
- “5” is the device instance number.

- “1258.000” is the x-coordinate.
- “390.000” is the y-coordinate.
- “mp” is the element name of the device.

In any single run of flat device recognition, the device instances are numbered consecutively beginning with zero. This numbering is over all device types and subtypes. Thus the 5 uniquely identifies the device instance we must debug.

Debug Statement Placement

The next step is to determine which Device operation in the rule file was responsible for generating instance 5. We know in this case that the device element name is “mp”. If this narrows it down to a single Device operation, put the DEBUG statement in that statement. However, if more than one Device operation qualifies, put a DEBUG statement in each one that qualifies. Since the DEBUG statement specifically identifies instance 5, and since instance 5 is unique to only one of the rules, the evaluation computation is only traced for that one instance. However, the initiation computation will be traced for all rules in which DEBUG was placed.

In this case, assume there is only one Device operation for an “mp” device, namely the one beginning on line 23 of the rule file. Insert the DEBUG statement into that Device operation. To avoid any possible confusion from line renumbering, we add the DEBUG statement to the end of line 24. Recall that the DEBUG statement if present must precede the PROPERTY statement. The altered Device operation now looks as follows:

```
23  device   mp(pmos)   gate   gate(G)   diff(S)   diff(D)
    base(B)
24  [ debug 5
25    property W, L, AS, AD
26    bend_effect = 0.5
27    AS = area(S)
28    AD = area(D)
29    W = (perimeter_coincide(G, diff) +
                              perimeter_inside(G, diff) ) / 2
30    L = perimeter_outside(G, diff) / 2
31    if ( bends(G) > 0 )
```

```
32    {
33      if ( W > L )
34        W = W - bend_effect * bends(G) * L
35      else
36        L = L - bend_effect * bends(G) * W
37    }
38  ]
```

Debug Output

Now, reload the rule file and rerun LVS. It is important to note that you must not alter the layout or any of the Device operation that would cause a change in the number of devices recognized, as this could alter the instance number of the device instance we are trying to debug.

The debugging output does not appear in the LVS report, but appears as comments in the transcript of the program that executed device recognition. The output placed in the transcript by the modified rule file is shown below and on the following pages interspersed with paragraphs of comments.

First is a statement identifying the beginning of the debugging output and the name of the rule file that generated it.

```
//  BEGIN EXTRACTED DEVICE PROPERTY COMPUTATION DEBUG OUTPUT
//     Rule file: rules
//
```

Next appears the output generated by the initiation computation. The first line identifies this as initiation output, gives the device type and subtype, and identifies the particular Device operation by giving its starting line number. It then displays the value array before the initiation computation (Values before), the computation itself (Interpreter called), and the value array after the computation (Values after).

```
//  INITIATION: Device: mp(pmos)  (line 22 of rule file)
//     Values before:
//         0:  w   0
//         1:  l   0
//         2:  as   0
//         3:  ad   0
//         4:  bend_effect  0
//         5:  0.5   0.5
```

```
//         6:  area(S)   0
//         7:  area(D)   0
//         8:  perimeter_coincide(G, Diff)   0
//         9:  perimeter_inside(G, diff)   0
//        10:  2  2
//        11:  perimeter_outside(G, diff)  0
//        12:  bends(G)  0
//        13:  0  0
//        14:  temp1  0
//        15:  temp2  0
//     Interpreter called.
//        26: bend_effect
//              = 0.5
//              = 0.5
//     Values after:
//         0:  w   0
//         1:  l   0
//         2:  as   0
//         3:  ad   0
//         4:  bend_effect   0.5
//         5:  0.5   0.5
//         6:  area(S)   0
//         7:  area(D)   0
//         8:  perimeter_coincide(G, Diff)   0
//         9:  perimeter_inside(G, diff)   0
//        10:  2   2
//        11:  perimeter_outside(G, diff)   0
//        12:  bends(G)   0
//        13:  0   0
//        14:  temp1   0
//        15:  temp2   0
//
```

Each value line above contains the following:

- The index position of the value in the array.
- The name of the value.
- The value itself.

Just after a rule file is loaded, these values will all be zero except for the constants and Process variables. However, if this is not the first device recognition run after

loading the rule file, the value array can contain values left over from prior computation. These left-over values will not affect the course of future computations, since they will be overwritten before they are used. However, it can be easier to debug, by loading the rule file just prior to running each debug computation. Note that the name of a constant is the same as its value.

The output above consists of a single step. The rule file compiler determined that the assignment statement on line 26 could be executed at initiation time since it was independent of any instance data values. The display shows this assignment taking place. See the section "How the Compiler Optimizes" above for more information on how the compiler optimizes the computation by placing portions of it into the initiation phase. A more complete discussion of the format of the step by step trace shown above, follows the evaluation output below.

The next output is the trace of the evaluation for the desired instance. The first line identifies this evaluation output, and as before identifies the device type, subtype and line of the rule file containing the Device operation causing the evaluation. The second line identifies the instance by instance number and (x,y) coordinates. Next are displayed the values in the value array before the evaluation (Values before) followed by a step by step trace of evaluation computation itself (Interpreter called) followed by the values in the array after the computation (Values after). The format of the value displays was discussed above. Note that most of the values in the array are no longer zero. This is because they contain values leftover from computation of properties for prior instances of this device type. The format of the step by step trace is discussed following the output.

```
// EVALUATION: Device: mp(pmos)  (line 22 of rule file)
//   Instance: 5  x: 1258  y: 390
//   Values before:
//      0:  w 2.9999999999999991e-05
//      1:  l 1.9999999999999985e-06
//      2:  as 5.9999999999999982e-11
//      3:  ad 5.9999999999999982e-11
//      4:  bend_effect 0.5
//      5:  0.5 0.5
//      6:  area(S) 7.1999999999999975e-11
//      7:  area(D) 5.5999999999999978e-11
//      8:  perimeter_coincide(G, Diff) 6.3999999999999976e-05
//      9:  perimeter_inside(G, diff) 0
//      10:  2 2
```

```
//      11:  perimeter_outside(G, diff) 3.9999999999999973e-06
//      12:  bends(G) 1
//      13:  0 0
//      14:  temp1 5.9999999999999982e-05
//      15:  temp2 0
//   Interpreter called.
//      27: as
//        = area(S)
//        = 7.1999999999999975e-11
//      28: ad
//        = area(D)
//        = 5.5999999999999978e-11
//      29: temp1
//        = perimeter_coincide(G,Diff)
                                    +perimeter_inside(G,diff)
//        = 6.3999999999999976e-05 + 0
//        = 6.3999999999999976e-05
//      29: w
//        = temp1 / 2
//        = 6.3999999999999976e-05 / 2
//        = 3.1999999999999988e-05
//      30: l
//        = perimeter_outside(G, diff) / 2
//        = 3.9999999999999973e-06 / 2
//        = 1.9999999999999985e-06
//      31: ?
//        bends(G) > 0
//        1 > 0
//        TRUE
//      33: ?
//        w > l
//        3.1999999999999988e-05 > 1.9999999999999985e-06
//        TRUE
//      34: temp2
//        = bend_effect * bends(G)
//        = 0.5 * 1
//        = 0.5
//      34: temp1
//        = temp2 * l
//        = 0.5 * 1.9999999999999985e-06
//        = 9.9999999999999928e-07
//      34: w
//        = w - temp1
```

```
//        = 3.1999999999999988e-05 - 9.9999999999999928e-07
//        = 3.0999999999999996e-05
//   Values after:
//      0:  w   3.0999999999999996e-05
//      1:  l   1.9999999999999985e-06
//      2:  as   7.1999999999999975e-11
//      3:  ad   5.5999999999999978e-11
//      4:  bend_effect   0.5
//      5:  0.5   0.5
//      6:  area(S)   7.1999999999999975e-11
//      7:  area(D)   5.5999999999999978e-11
//      8:  perimeter_coincide(G, Diff) 6.3999999999999976e-05
//      9:  perimeter_inside(G, diff)   0
//      10:  2   2
//      11:  perimeter_outside(G, diff) 3.9999999999999973e-06
//      12:  bends(G)   1
//      13:  0   0
//      14:  temp1   9.9999999999999928e-07
//      15:  temp2   0.5
//
```

Each step of the computation above is displayed on three or four lines. The first line gives the rule file line number of the language statement that is causing the step followed by a colon (:). The colon is followed by the name of a value array entry to which an assignment is about to be made, or by a question mark (?) if this step is a relational test. The second line shows the expression that is about to be computed or tested using the value array names of the values involved. The third line shows the same expression with the corresponding numeric values substituted. The fourth line shows the numeric result of the expression or, in the case of a test, one of the values TRUE or FALSE. In the case of a simple assignment, there is no fourth line because the third line already displays the value to be assigned.

The numeric values are shown to about 17 significant digits so numerical errors due to roundoff or to the inexact representation of decimal fractions on a binary machine can be better seen. This leads to lots of trailing 9’s in the output, but is worthwhile. For example with only a few digits you might wonder how the test “1.3 < 1.3” could return TRUE, but with 17 digits it becomes clear that “1.29999999999999996 < 1.29999999999999998” is indeed TRUE.

In evaluating a compound logical expression such as (a < b || c < d) the interpreter does not explicitly perform logical operations NOT (!), AND (&&), and OR (||). Rather it makes a sequence of relation tests in the proper sequence as determined by the results of those tests. In the example here it would first perform the test (a < b). If the result of that test were FALSE, it would then perform the test (c < d), otherwise it would go to the next appropriate step. The action of the OR (||) is implicit in the sequence of tests performed.

Each step of the trace represents only a single relation test or a single numeric operation followed by an assignment. Therefore complex statements in the program will generate several steps in which intermediate results are stored in temporary locations in the value array. These locations are given names of the form temp*n*, where *n* is a small integer. In the example above the statement on line 34 of the rules is:

```
34 W = W - bend_effect * bends(G) * L
```

In the trace, this breaks down into three separate steps, each labeled with line number 34, which use the temporary variables temp1 and temp2 to store intermediate values of subexpressions.

The temporary variables are reused from one statement to the next. Note that “temp1” was also used by the steps for the statement on line 29.

Although this example doesn't show it, there is a second kind of temporary variable whose name is of the form init*n*, where *n* is a small integer. These temporaries are used to store the results of subexpressions that are instance independent. Their values are computed during the initiation computation and then used repeatedly for the evaluations computations.

Finally the end of the debugging output is noted by the following line:

```
//  END EXTRACTED DEVICE PROPERTY COMPUTATION DEBUG OUTPUT
```

Debug Analysis

By examining the value arrays displayed, and the step by step trace, it should be possible to determine the cause of the discrepancy. Note that the geometric data values such as perimeter_inside(G, diff) are clearly displayed in the value array

within the trace. These values can be checked against the layout to see if the proper data functions have been used.

In the example covered above, only one Device operation contained a DEBUG statement and that operation referenced a single instance only. The output had the simple structure of initiation and evaluation. If more than one instance had been referenced by the DEBUG statement, then the sequence would have been initiation, evaluation, … , evaluation.

If you placed DEBUG statements in a second Device operation, the order of the output would depend on whether the Device operations used the same or different device (seed) layers. If they used the same device layer, then both initiations would appear, followed by a mixture of evaluations for both types of device. This occurs since Device operations for a given device layer are processed simultaneously. If they used different layers, then the output for one of the layers would precede the other. Since each initiation or evaluation output begins by identifying the device by type, subtype, and rule file line number, it should be easy to keep the output straight. Note also that each Device operation has its own value array.

Hierarchical Debugging

The previous discussion of debugging procedures focused on flat analysis. This is the easiest to do and this is often the technique employed when building a rule file. Hierarchical extracted netlists are more difficult to debug as you cannot always know which device instances to debug. There are two ways of approaching this problem:

1. Go into the rule file and insert something like the following into the device property computation:

   [PROPERTY *prop1 prop2* ... inst
   ...
   inst = instance()
   ]

   Rerun LVS and you can find the flat instance number of the device you need to debug in the extracted netlist. This is perhaps the more elegant solution.

2. Rerun the design (or a portion of the design) in flat LVS. This is the brute force method.

Property Specification Error Messages

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR1	a definition for this device name may not have a numeric parameter set specification: <element name>	The only element names which may take a numeric parameter set in square brackets are D, C, R, MN, MP, MD, and ME. This element name is not one of them.
DPR2	the numeric parameter set specification for this device definition must contain a single number: <element name>	The element names R, MN, MP, MD, and ME take only a single number as a numeric parameter set. For R it is the resistivity and for the others it is the effective width factor. No additional numbers are allowed.
DPR3	the numeric parameter set specification for this device definition must contain one or two numbers: <element name>	The numeric parameter set specification for element name C must contain either just the area capacitance factor, or the area capacitance factor followed by the perimeter capacitance factor. No additional numbers are allowed.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR4	a statement was expected at this point (did you forget a comma?) : <syntax element>	The syntax element displayed was found where a valid assignment statement, IF statement, or compound statement beginning with a left brace (**{**) was expected.
DPR5	this was found where the second number of a debug range was expected: <syntax element>	In a DEBUG statement, the first number and hyphen (-) of a debug range was found, but the second number was missing. A debug range must be either a single number or a pair of numbers separated by a hyphen.
DPR6	this was found where the first number of a debug range was expected: <syntax element>	A number must appear immediately following the DEBUG keyword and immediately following each comma in the DEBUG statement.
DPR7	this was found where a right brace, "}" was expected: <syntax element>	The syntax element shown appears where a closing brace was expected to terminate a compound statement.
DPR8	this was found where a left parenthesis, "(" was expected: <syntax element>	A left parenthesis must follow the keyword IF and all function name keywords.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR9	this was found where a right parenthesis, “)” was expected: <syntax element>	The syntax element displayed could not be parsed in the current context, but a right parenthesis would be valid at this point.
DPR10	this was found where a relational operator such as “<=” was expected: <syntax element>	The expressions used in an IF test must contain relational tests possibly combined with logical operators to form compound tests.
DPR12	this was found where a comma was expected: <syntax element>	Commas must be used to separate items in lists and the arguments of functions.
DPR13	this was found where the keyword PROPERTY was expected: <syntax element>	With the exception of the DEBUG statement, the first statement in a property specification must be a PROPERTY statement specifying the properties to be computed.
DPR14	this was found where a property identifier was expected: <syntax element>	A property identifier must immediately follow the keyword PROPERTY and each comma in the PROPERTY statement.
DPR15	this property identifier is declared twice: <identifier>	Each property identifier may appear only once in the list of the PROPERTY statement.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR16	this was found where a property or variable name was expected: <syntax element>	In an assignment statement, the item just to the left of the assignment operator (=) must be a property name as declared in the PROPERTY statement, or an identifier representing a local variable.
DPR17	this was found where a layer or pin name was expected: <syntax element>	The syntax element shown is not the name of a pin or layer appearing in this Device statement, yet it appears as the argument of a function where a pin name or layer name is required.
DPR18	this was found where a right bracket, “]” was expected: <syntax element>	The property specification appears to end just prior to the syntax element shown, but no closing right bracket was found there.
DPR19	this was found where an expression representing a numeric value was expected: <syntax element>	The syntax element show was found where a constant, local variable, process variable, or numeric valued function was expected.
DPR20	this process variable does not have a numeric value: <variable>	The process variable referenced at this point must have a numeric value, but this one does not.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR21	this variable was used on the right before it was unconditionally initialized: <variable>	The right hand side of the statement contains the variable shown. However, this variable has either not been initialized in a prior statement, or has not been initialized in every IF/ELSE path leading to the current statement. For example, in the following, before reaching the statement A = X, the variable X would not be initialized if the IF condition were FALSE: if (b == 0) X = 1 A = X
DPR22	this was found where a pin name was expected: <syntax element>	The syntax element shown is not the name of a pin appearing in this Device statement, yet it appears as the argument of a function where a pin name is required.
DPR23	this variable was used on the right before it was given a value on the left: <variable>	The variable displayed is used on both the left and right sides of the current statement. It must have been given a value before reaching the current statement so that the value may be used in the right hand side of the current statement.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR24	assignment to this process variable is not allowed: <variable>	The variable shown is assigned a value by the current statement. However, this variable has been identified as a process variable. Changing the value of a process variable is not allowed from within a property computation.
DPR25	this was found where a net name was expected: <syntax element>	The syntax element shown appeared as an argument to a function where a net name was expected, but cannot be interpreted as a net name. It is best to surround the net name with double quotes (“ ”).
DPR35	this property was never assigned a value: <property name>	All properties declared in the PROPERTY statement must be assigned a value in the property computation. The property name shown was never assigned a value.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR36	this property was not assigned a value in all cases: <property name>	All properties declared in the PROPERTY statement must be assigned a value in the property computation. The property name shown was assigned a value in some of the IF/ELSE cases, but not in all. That is, it is possible to find a path through the program which never assigns a value to the variable. You must be sure it receives a value in all cases.
DPR37	this function cannot have identical arguments: <function name>	The function shown requires two arguments, but they cannot be identical.

Table 9-5. Property Specification Error Messages

Error Number	Message	Description
DPR38	both arguments of this function cannot be associated with the same layer: <function name>	The function shown requires two arguments, but they cannot represent shapes on the same layer. For example, using different pin names from the same layer is not allowed. Neither is using a pin name from a layer together with the layer itself. In all these disallowed cases, the function would return a trivial value which can be expressed in another way. For example if A and B are pins on layer L, then PERIMETER_COINCIDE(A, B) would always be 0 (zero) since A and B would have to be disjoint, and PERIMETER_COINCIDE(A, L) would be the same as PERIMETER(A) since pin A lies on layer L.

Chapter 10
LVS Circuit Comparison

This chapter discusses various LVS concepts pertaining to circuit comparison.

LVS Comparison

Calibre LVS applications compare electrical circuits from the specified source netlist and layout geometry. (You can also do netlist to netlist comparisons.)You do not need to supply initial correspondence between circuit elements, but if you do, the time of execution decreases. When the compared circuits are equivalent, Calibre LVS applications establish a one-to-one correspondence between elements of one circuit (instances, nets, ports, and instance pins) in the source netlist and elements of the layout circuit. When the compared circuits differ, Calibre LVS applications attempt to match elements of one circuit to elements of the other. This matching is completed when a one-to-one correspondence between the elements is established. The correspondence between elements can be used for cross-probing.

Calibre LVS matches as many elements as possible including elements that differ in the compared circuits. For example, nets that have different connections can be matched if most of their connections are equivalent.

To ensure that the matching of different elements does not generate misleading results, heuristics are applied. These heuristics are internal problem-solving techniques that select the best solution among those derived from alternative methods at different stages of the program.

Discrepancies reported in the LVS report show the differences between the two circuits. Calibre LVS algorithms treat the two circuits similarly. The LVS report presents discrepancies from the point of view of both the source *and* the layout. First, the LVS report presents the discrepancy as if the source data is correct, and

the layout data is incorrect, then it presents the discrepancy as if the layout data is correct, and the source data is incorrect. This form of presentation allows you to see and compare all possible views of the data.

LVS reports discrepancies in terms of incorrect circuit elements, along with additional information that helps classify and locate the errors. Calibre LVS attempts to suggest changes to the layout so that it matches the source circuit. For example, Calibre LVS may recommend that two nets be connected in the layout to match a single net in the source circuit.

Component Types

A LVS component type is a name that uniquely identifies the electrical or logical function of a layout or source instance. LVS uses component type values in the process of matching layout and source instances. Instances are required to have the same component type in order to be correctly matched. Instances with different component types are sometimes matched if they are identically connected, but discrepancies are reported in such cases.

The following sections describe the conventions LVS uses to determine component types of instances.

- **Mask Layout**. The component type of an extracted layout device is equivalent to the value of the element_name argument of the corresponding Device operation in the rule file. For example, the statement:

  ```
  device MP tran poly srcdrn srcdrn bulk
  ```

  specifies a MOS transistor device with component type MP.

- **Eddm**. The component type of a source Eddm instance is established as follows:

  a. LVS searches the list of property names specified in the LVS Component Type Property specification statement from left to right.

b. LVS uses, as the component type of the instance, the value of the first property from the list. The default property name list is:

```
["phy_comp", "element", "comp"]
```

If the specified list is null or if none of the specified properties are found on the instance, LVS then determines the component type according to the following:

c. LVS uses the Eddm component name if the instance has a default model called "schematic", or if there is no default model.

d. LVS constructs the component type, if the instance has a default model other than "schematic", by appending the default model name as shown:

eddm-component-name_default-model-name.

For more details on Eddm model selection in general, and default models in particular, refer to "Model Selection During Evaluation" in *A Guide to Design Process and Database Concepts*. This convention is used to match the behavior of the Place and Route command "LOAd LOgic."

e. LVS appends the first character of the model property to the component type if the component type as determined is M or LDD and the instance owns a model property whose first character is N, P, E, or D. This processing is case-insensitive.

For example, suppose that you have:

```
element = M
model = PMOS
```

and you use the default LVS setup. Then

```
LVS component type = MP
```

- **V7.0 erel file**. The component type of a source V7.0 erel instance is established as follows: The list of property names specified in the lvs_component_type_property application variable (which can be set with

the LVS Component Type Property specification statement in a rule file) is searched from left to right. The value of the first property from the list which is found on the instance is used as the component type of the instance. If the specified list is null or if none of the specified properties is found on the instance, then the Symed symbol name of the instance is used as its component type. The symbol name is the leaf name from the symbol's pathname. In the default property names list, "element" should be replaced with "spicemodel."

In addition, the following is done: If the component type as determined so far is "M" or "LDD", and the instance owns a "spicepar" property whose first character is one of "N", "P", "E" or "D", then the first character of the spicepar property is appended to the component type. Note that this processing is case-insensitive. For example, suppose that you have

spicemodel = M

spicepar = PMOS

and your lvs_component_type_property application variable is ["spicemodel"]. Then

LVS component type = MP

- **Spice netlist**. The component types of Spice elements are described in section "General Spice Syntax" in chapter 11, "Spice Format".

Component Subtypes

A LVS component subtype is an optional name that classifies, together with the component type, the electrical or logical function of a layout or source instance. Component subtypes do not affect the matching of source instances to layout instances. LVS reports differences in the subtypes of instances that are matched to each other, only if subtypes are specified for both instances.

The following sections describe the conventions LVS uses to determine component subtypes of instances.

- **Mask layout.** The component subtype of an extracted layout device is equivalent to the value of the optional model_name argument in the corresponding Device operation. For example, the operation:

  ```
  device C(PM) cap poly metal
  ```

  specifies a capacitor device with component type C and subtype PM. If a model_name is not specified in the Device operation, then the device does not have a subtype.

- **Eddm.** The component subtype of a source Eddm instance is equivalent to the value of the property whose name is specified in the LVS Component Subtype Property specification statement. If the statement does not specify a property name, or if the specified property is not found on an instance, then the instance does not have a subtype.

- **V7.0 erel file.** The component subtype of a source V7.0 erel instance is equivalent to the value of the property whose name is specified in the LVS Component Subtype Property specification statement. If the statement does not specify a property name, or if the specified property is not found on an instance, then the instance does not have a subtype.

- **Spice netlist**. The component subtypes of Spice elements are described in the section "General Spice Syntax" in chapter 11, "Spice Format".

An important thing to note is that during connectivity extraction, component subtypes as defined in the rule file appear in the extracted netlist as lower case. This has ramifications in the circuit comparison stage when case sensitivity may be an issue. See LVS Compare Case in the *SVRF Manual* for details.

Naming Conventions

Instance Pins and Pin Names

LVS uses instance pin names to match circuit elements in the connectivity comparison process. Pins and pin names are normally inherited by instances from rule file operations, Eddm parts, and Spice netlist elements.

- **Mask layout.** LVS specifies the pin names of extracted layout devices in the rule file, or by default convention.

  - **User-defined devices.** Device operations in the rule file specify user-defined device pin names. By user-defined here, we mean for the purpose of device recognition. Pin ordering in the extracted netlist will follow the order specified in the corresponding Device statement.

    Device types J, L, LDD, LDDD, LDDE, LDDN, LDDP, M and V are user-defined for the purpose of determining pin names and ordering.

  - **Built-in devices.** These have default pin naming and ordering conventions that cannot be overridden.

    Device types C, D, MD, ME, MN, MP, Q, and R are built-in for both recognition and LVS comparison.

- **Eddm.** LVS establishes pin names of source Eddm instances as follows:

  a. Searches from left to right, the list of property names specified in the LVS Pin Name Property specification statement.

  b. Uses, as the pin name, the value of the first property from the list that is found on the pin.

  c. If the specified list is null, or if none of the specified properties are found on the pin, then the value of the pin property is used. An error is reported if no pin property is found.

- V7.0 erel file. LVS establishes pin names of source V7.0 erel instances as follows:

  a. Searches from left to right, the list of property names specified in the LVS Pin Name Property specification statement.

  b. Uses the value of the first property from the list that is found on the pin as the pin name.

c. If the specified list is null, or if none of the specified properties are found on the pin, then the value of the pin property is used. An error is reported if no pin property is found.

- **Spice netlist**. Pin names of Spice elements are described in section "General Spice Syntax" in chapter 11.

Pin Filtering

LVS operates only on instance pins that have names. In a single design you can use instances with the same component type but different number of pins. For example, in a single design, you can have two-pin resistors and three-pin resistors.

For a given component type and a given number of pins, corresponding layout and source pins should have identical names. However, it is allowed for instances of layout components to have pins that are not present in the corresponding source components, and vice versa. The LVS comparison algorithm filters out these pins when it establishes correspondence between layout and source elements.

LVS filters out missing pins to allow higher-level layout components to have pins, such as power and ground, that do not appear on the corresponding schematic components.

After this pin filtering process, instances must have the same number of pins and the same pin names in order to be correctly matched to each other. Instances with different numbers of pins, or different pin names can be matched if they are similarly connected; discrepancies are reported in such cases.

For each component type, layout pins that are not present in the source, and source pins that are not present in the layout are listed in the LVS report. Missing power or ground pins are reported as warnings; other missing pins are reported as errors.

LVS classifies the names of power or ground pins if they are specified in the LVS Power Name or LVS Ground Name specification statements, respectively.

User-given Names

The nets, instances, and ports of layout and source databases can have user-given names, system-generated names, or both. User-given names are used by LVS to

establish Initial Correspondence Points. LVS reports differences between user-given names of layout and source elements.

LVS determines whether a name is user-given as follows:

- **Mask layout**. A name qualifies as user-given if it does not start with the characters n$, N$, i$, or I$ and does not contain any slash (/) characters (one leading slash is allowed). If a leading slash is present, the slash is ignored. For example, the layout name “/ABC” is user-given and forms an initial correspondence point with the source name “ABC”. The layout name “ABC” also is user-given and forms an initial correspondence point with the source name “ABC”.

- **Eddm**. A name qualifies as user-given if it does not start with the characters n$, N$, i$, or I$ and does not contain any slash (/) characters. Only top level or global names qualify as user-given. For example, “mynode” and “tran1” are user-given names, but “/cell1/mynode”, “i$1”, and “n$1” are not.

- **V7.0 erel file**. A name qualifies as user-given if it does not start with the characters n$, N$, i$, or I$ and does not contain any slash (/) characters. Only top level or global names qualify as user-given. For example, “mynode” and “tran1” are user-given names, but “/cell1/mynode”, “i$1”, and “n$1” are not.

- **Spice**. A *node* name qualifies as user-given if it contains at least one non-numeric character (letter), and does not contain any “/” characters, except that one leading slash is allowed. If a leading slash is present, it is ignored. An *element* name qualifies as user-given if, excluding the first character, the name contains at least one character which is not a digit or the “=” sign; also, the name must not contain any “/” characters. (The first character in Spice element names is always the spice element type). For example, “C2a”, “Xabc” and “M1==A” are user given element names, but “C123”, “X1”, “Xabc/X2” and “M1==2” are not.

  The prefixes “n$” and “i$” by convention denote system generated net and instance names in Mentor Graphics schematic databases. “/” is used as delimiter to form hierarchical pathnames. LVS allows a leading slash in layout names and in Spice node names for compatibility with the IC Station command Load Logic. Equal signs (“=”) are used by the Spice parser in

names that the parser generates for elements that are replicated by means of the M parameter.

Net and Instance Names

This section describes how net and instance names are specified in various sources of connectivity.

- **Mask Layout**. LVS obtains layout net names from the values of net properties on layout shapes and paths in the top level cell. These values are assigned as names to nets in the connectivity extraction process activated by LVS. Layout instances cannot be named in Mask LVS.
- **Eddm**. LVS obtains net names from the values of net properties of top-level nets. Instance names are obtained from the values of instance properties of top-level instances.
- **V7.0 erel file**. Net names in a V7.0 erel file (in both modes) are obtained from the values of net properties of top level nets. Instance names are obtained from the values of inst properties of top level instances.
- **Spice netlist**. LVS obtains net names from the node names in the netlist. Instance names are the element names in the netlist.

Ports and Port Names

This section describes how design ports and port names are specified in various sources of connectivity.

- For GDSII and CIF databases in Calibre LVS, specify ports by using the Port Layer Text and Port Layer Polygon specification statements.
- **Mask layout.** Calibre LVS specifies a layout port with the Port Layer Text specification statement in the rule file.
- **Eddm**. LVS specifies design ports in three ways.

  - Any top-level net labeled as external serves as a design port in LVS. For example, connected to a port instance, or leading to a symbol pin.

  - Any top-level net that has a net_comp property, with an arbitrary value, serves as a design port in LVS.

  - All global nets serve as design ports in LVS.

  In all cases, the port name is equal to the value of the net property of the net.

- **V7.0 erel file.** Design ports are specified in three ways.

  - Any top-level net labeled as external (such as connected to a port instance, or leading to a symbol pin) serves as a design port in LVS.

  - Any top-level net that owns a net_comp property (with arbitrary value) serves as a design port in LVS.

  - All global nets serve as design ports in LVS. In all cases, the port name is equal to the value of the net property of the net.

- **Spice netlist**. LVS specifies design ports are specified in two ways.

  - External nodes of a top-level subcircuit, if one is specified, serve as design ports in LVS.

  - All nodes with user-given names specified in .GLOBAL statements serve as design ports in LVS.

  In both cases, the node names serve as port names.

Power and Ground Nets

LVS uses power and ground nets in logic gate recognition, in filtering of unused MOS transistors and in other applications. Several power nets and several ground nets are allowed in a single design. A net is a power net (or a ground net, respectively) if

- the net name is listed in an LVS Power Name or LVS Ground Name specification statement; or,

- the net is connected to a port whose name is listed in the LVS Power Name or LVS Ground Name specification statement and there is no other net in the same design that has this name. If there are several ports with the same power (or ground) name that are connected to different nets, only one of them is used.

Built-in Device Types

Table 10-1 lists the built-in device types and their corresponding component type values. These devices are built-in from the standpoint of LVS comparison only, except as noted. The section "Component Types" shows how component types are determined in the source circuit and in the layout.

Table 10-1. Built-in Device Types

Device	Component Type
CMOS N transistor	MN[a]
CMOS P transistor	MP[a]
NMOS enhancement transistor	ME[a]
NMOS depletion transistor	MD[a]
MOS generic transistor	M
CMOS LDD N transistor	LDDN
CMOS LDD P transistor	LDDP
NMOS LDD enhancement transistor	LDDE
NMOS LDD depletion transistor	LDDD
MOS LDD generic transistor	LDD
Resistor	R[a]
Capacitor	C[a]
Diode	D[a]
Bipolar transistor	Q[a]
Jfet transistor	J

Table 10-1. Built-in Device Types [continued]

Device	Component Type
Inductor	L
Voltage source	V

Notes:
[a] built-in devices for both recognition and LVS comparison

MOS Transistors

MOS regular transistors receive special processing by LVS as follows: logic gate recognition, parallel transistor reduction, split-gate reduction, filtering of unused transistors, source/drain pin swapping by default, processing of soft substrate pins, and pin names are always case-insensitive. MOS LDD transistors are processed as follows (if they conform to Table 10-2 pin conventions): logic gate recognition, parallel transistor reduction, split-gate reduction, filtering of unused transistors, source/drain pin swapping by default, processing of soft substrate pins, and pin names are always case-insensitive.

MOS transistor devices (component types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD) must have at least three pins—gate, source and drain. In addition, they may have *any number* of additional pins with arbitrary names. The fourth pin typically represents bulk connection and by convention is called B, but this convention is neither required nor enforced by LVS. (However, the rule file Device definition syntax does enforce this convention). The optional pins may represent one or more bulk connections or they may be used for any other purpose. Table 10-2 lists the three required pin names:

Table 10-2. MOS Transistor Required Pin Names

Pin	Pin Name
MOS transistor gate	G or GATE
MOS transistor source	S or SOURCE
MOS transistor drain	D or DRAIN
optional pins	any names

LDD devices are MOS transistors with non-swappable source and drain pins. The five LDD transistor types LDDN, LDDP, LDDE, LDDD, and LDD correspond to the five regular transistor types MN, MP, ME, MD, and M respectively. The acronym LDD stands for Lightly Doped Drain. The LDD* and M devices are user-defined from the device recognition standpoint. Remember, pin ordering for user-defined devices is taken from corresponding Device statements in the rule file. Such MOS devices that do not have the required pin names shown above receive no special LVS processing.

The following specification statements control special processing functions used by Calibre LVS to recognize, reduce, and filter MOS transistors:

- LVS Recognize Gates
- LVS Reduce Parallel MOS
- LVS Reduce Series MOS
- LVS Reduce Split Gates
- LVS Filter Unused MOS
- LVS Filter
- LVS Reduce

Capacitors

LVS performs the following for capacitors: series capacitor reduction, parallel capacitor reduction, pin swapping if requested, processing of soft substrate pins, and pin names are always case-insensitive. Capacitor devices (component type C) must have at least two pins (positive and negative). In addition, they can have any number of additional pins with arbitrary names. The optional pins can represent

one or more substrate connections or they can be used for any other purpose. Table 10-3 lists the two required pin names:

Table 10-3. Capacitor Required Pin Names

Pin	Pin Name
capacitor positive pin	POS or P
capacitor negative pin	NEG or N
optional pins	any names

The following specification statements control special processing functions used by Calibre LVS to reduce capacitors:

- LVS Reduce Series Capacitors
- LVS Reduce Parallel Capacitors
- LVS Filter Unused Capacitors
- LVS Filter
- LVS Reduce

Resistors

LVS performs the following for resistors: series resistor reduction, parallel resistor reduction, pin swapping by default, processing of soft substrate pins, and pin names are always case-insensitive. Resistor devices (component type R) must have at least two pins (positive and negative). In addition, they can have any number of additional pins with arbitrary names. The optional pins can represent one or more substrate connections or they can be used for any other purpose. Table 10-4 lists the two required pin names:

Table 10-4. Resistor Required Pin Names

Pin	Pin Name
resistor positive pin	POS or P

Table 10-4. Resistor Required Pin Names

Pin	Pin Name
resistor negative pin	NEG or N
optional pins	any names

The following specification statements control special processing functions used by Calibre LVS to reduce resistors:

- LVS Reduce Series Resistors
- LVS Reduce Parallel Resistors
- LVS Filter Unused Resistors
- LVS Filter
- LVS Reduce

Diodes

LVS performs the following for diodes: parallel diode reduction, processing of soft substrate pins, and pin names are always case-insensitive. Diode devices (component type D) must have at least two pins (positive and negative). In addition, they can have any number of additional pins with arbitrary names. The optional pins can represent one or more substrate connections or they can be used for any other purpose. Table 10-5 lists the two required pin names:

Table 10-5. Diode Required Pin Names

Pin	Pin Name
diode positive pin	POS or P
diode negative pin	NEG or N
optional pins	any names

The following specification statements control special processing functions used by Calibre LVS to reduce diodes:

- LVS Reduce Parallel Diodes
- LVS Filter Unused Diodes
- LVS Filter
- LVS Reduce

Bipolar Transistors

LVS performs the following for bipolar devices: parallel bipolar transistor reduction, filtering of unused bipolar transistors, processing of soft substrate pins, and pin names are always case-insensitive. Bipolar devices (component type Q) must have at least three pins (collector, base, and emitter). In addition, they can have any number of additional pins with arbitrary names. The fourth pin typically represents substrate connection and by convention is called S, but this convention is neither required nor enforced by LVS. (However, the rule file Device definition syntax does enforce this convention.) The optional pins can represent one or more substrate connections or they can be used for any other purpose. Table 10-6 lists the three required pin names:

Table 10-6. Bipolar Transistor Required Pin Names

Pin	Pin Name
Q transistor collector	C
Q transistor base	B
Q transistor emitter	E
other pins	any names

The following specification statements control special processing functions used by Calibre LVS to reduce and filter bipolar transistors:

- LVS Reduce Parallel Bipolar

- LVS Filter Unused Bipolar
- LVS Filter
- LVS Reduce

Jfet Transistors

LVS performs the following for Jfets that conform to Table 10-7: processes soft substrate pins; pin names are always case-insensitive. Jfet transistor devices (component type J) must have at least three pins— gate, source and drain. In addition, they may have any number of additional pins with arbitrary names. The optional pins may represent one or more substrate connections or they may be used for any other purpose. The three required pin names must be as listed below:

Table 10-7. Jfet Transistor Required Pin Names

Pin	Pin Name
J transistor gate	G or GATE
J transistor source	S or SOURCE
J transistor drain	D or DRAIN
other pins	any names

Type J devices are considered user-defined from the standpoint of device recognition. If you specify them differently than what the table shows, these devices receive no special processing by LVS.

Inductors

LVS performs the following for inductors that conform to Table 10-8: processes soft substrate pins; pin names are always case-insensitive. Inductor devices (component type L) must have at least two pins— positive and negative. In addition, they may have any number of additional pins with arbitrary names. The

optional pins may represent one or more substrate connections or they may be used for any other purpose. The two required pin names must be as listed below:

Table 10-8. Inductor Required Pin Names

Pin	Pin Name
inductor positive pin	POS or P
inductor negative pin	NEG or N
optional pins	any names

Type L devices are considered user-defined from the standpoint of device recognition. If you specify them differently than what the table shows, these devices receive no special processing by LVS.

Voltage Sources

LVS performs the following for voltage sources that conform to Table 10-9: processes soft substrate pins; pin names are always case-insensitive. Voltage source devices (component type V) must have at least two pins—positive and negative. In addition, they may have any number of additional pins with arbitrary names. The optional pins may represent one or more substrate connections or they may be used for any other purpose. The two required pin names must be as listed below:

Table 10-9. Voltage Source Required Pin Names

Pin	Pin Name
voltage source positive pin	POS or P
voltage source negative pin	NEG or N
optional pins	any names

Type V devices are considered user-defined from the standpoint of device recognition. If you specify them differently than what the table shows, these devices receive no special processing by LVS.

MS and MF Schematic Devices

MS and MF are special device types that are supported by Mentor Graphics analog simulation tools and by LVS. They are 3-pin schematic symbols that represent 4-pin CMOS transistors. The fourth bulk pin is implied by the device type. MS devices have their bulk pin implicitly connected to the source, MF devices have their bulk pin floating.

LVS internally adds a virtual bulk pin to all MS instances in the schematic. The bulk pin is internally connected to the source net of the instance. The bulk pin name is “B”.

LVS internally adds a virtual bulk pin to all MF instances in the schematic. A virtual net is internally created for each instance to represent the floating bulk node. The bulk pin of the instance is internally connected to this virtual net. The bulk pin name is “B”.

To trigger this special processing, the LVS component type of the instance has to be either MN, MP, LDDN, or LDDP and the instance must have exactly three pins with standard pin names as specified in the section “Built-in Device Types”. The device type (MS or MF) is specified as the value of the element property on Eddm instances. This means that some property other then element or spice model, respectively, must be used to specify the LVS component type for these instances. The phy_comp property is a suggested choice. See the section “Component Types” for more details on specifying component types in the schematic.

Matching of Circuit Elements

LVS iterates between two methods of matching elements: a signature-based hashing method and a tracing method.

- **Signature-based method**. LVS assigns signatures to nets and instances in both circuits according to their type and connections. LVS then hashes circuit elements according to the:
  - Signature of the element
  - Signatures of elements in nearby environments

  - Presence of previously matched elements in their environments

  The environment size increases until at least one uniquely matching pair of elements is found. A correspondence is established between all uniquely matching elements found.

- **Tracing method**. LVS uses previously matched elements as initial correspondence points. Starting from these correspondence points, LVS traces both circuits one step at a time. LVS establishes a correspondence between elements that can be uniquely matched at each step. The tracing continues until LVS:

  - Matches all elements

  - Detects discrepancies that prevent further tracing

  - Detects interchangeable parts of the circuit that prevent further tracing

LVS repeats both methods until all elements are matched, or further elements can be matched. In the latter case, LVS tries to correct, internally, some of the errors. If errors can be corrected, then more elements are matched and the process is repeated.

Connectivity Comparison Results

LVS classifies elements of both compared circuits (nets, instances, and ports) into three categories:

- **Correct**. These elements belong to correctly implemented parts of the circuit. They are elements that uniquely match identical elements, and always have corresponding correct elements in the other circuit.

- **Incorrect.** These elements are certainly wrong. They are elements that have no identical elements in the other circuit, based on connectivity tracing and signature hashing. Incorrect elements can be matched to different elements in the other circuit, or be left unmatched (a suggested match). Although the source circuit is treated as a reference, its elements can also be classified as incorrect.

- **Unmatched.** These elements cannot be determined, by the algorithm, to be correct, or not. These elements cannot be uniquely matched to elements in the other circuit, nor can they be classified as incorrect. This can be caused by an incorrect element nearby.

Calibre LVS distinguishes between incorrect and unmatched elements to help you analyze errors. In most cases, it is enough to fix the elements classified as incorrect and ignore the list of unmatched elements. The unmatched elements are, in most cases, classified as correct when the incorrect elements are fixed.

Initial Correspondence Points

LVS uses pairs of nets, pairs of instances, and pairs of ports from both circuits, which have identical User-given Names as initial correspondence points. LVS trusts initial correspondence points and will match elements that form initial correspondence points, even if they differ from each other. LVS does not require that initial correspondence points exist. However, it is recommended that you specify initial correspondence points on ports of the top-level cell in the source and layout by adding text to the input pins.

Resolving Ambiguities

Ambiguities occur in highly parallel and symmetric circuits. These are circuits where parts can be interchanged without affecting the connectivity. In these cases, it is impossible to distinguish between the interchangeable parts.

LVS uses named nets, instances, and ports as initial correspondence points to resolve ambiguous situations. In addition, component subtypes are used to resolve ambiguities. LVS also resolves ambiguities by examining properties that are traced with the Trace Property specification statement. This last technique is only applied to groups of ambiguous elements that contain no more than a certain number of elements each. That number is specified in the rule file with the LVS Property Resolution Maximum specification statement and defaults to 8.

LVS sets a maximum limit to the size of environments used for hashing circuit elements. Various factors, including circuit size, determines the environment size. If the environment limit is exceeded and no unique match can be found, LVS proceeds to the ambiguity resolution stage. In the ambiguity resolution stage, LVS

arbitrarily matches some elements which cannot be resolved otherwise. LVS allows only a minimal amount of arbitrary matching to take place. The arbitrarily matched elements are listed in the LVS report. If the arbitrary match is incorrect, LVS produces discrepancies at a later stage. In this case, the user should assign names to arbitrarily matched elements, or other nearby elements.

To avoid arbitrary matching, it is recommended to name nets on interchangeable parts of the circuit. It is acceptable to name one element on every interchangeable part. It is always recommended to name the external ports of symmetric circuits.

Device Reduction

LVS internally reduces groups of devices in the layout and in the source; each group is represented by a single virtual device. After reduction, the circuits are compared in terms of the virtual devices. Device reduction handles situations where, for example, a single schematic device is implemented by a group of several parallel or series devices in the layout.

This section describes the semantics of device reduction, specific to each device grouping, how to specify tolerance levels for device reduction, and how to create programs that define how calculations are made during device reduction.

Here are some issues to be aware of:

- Initial correspondence points—For series device reduction in the top-level cell, nets that serve as initial correspondence points will break a series. Initial correspondence points do not break a series nor interfere with series reduction in lower-level cells for hierarchical LVS.

- Ports—For series device reduction, connection to a cell port breaks a series. This affects instances in the top-level cell and in hcells, provided that the port has not been removed, such as trivial ports.

Device Reduction Semantics

The following sections describe how various groups of devices are reduced by Calibre LVS.

Generic Device Reduction

Calibre LVS can reduce generic user-defined or built-in devices through the use of the generic LVS Reduce statement.

Parallel MOS Transistor Reduction

Calibre LVS can reduce a group of parallel MOS transistors to a single transistor. All transistors in the group must have the same component type, optional subtype, number of pins, and pin names. All gate, source, drain pins, and optional pins (if any), must be connected to the same nets. Device reduction can swap source and drain connections of MN, MP, ME, MD, and M devices. Optional pins can also be swapped if they are specified as logically equivalent. Section "Logically Equivalent Pins" describes how to specify logical equivalence. To be reduced, MOS transistors must have at least three pins with standard pin names, as specified in Table 10-2.

Figure 10-1 shows an example of parallel MOS transistor reduction.

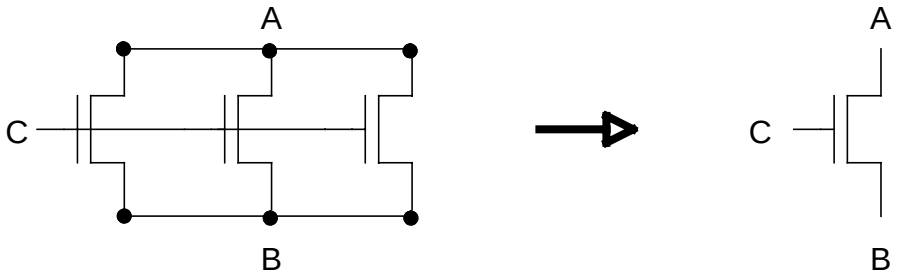


Figure 10-1. Parallel MOS Transistor Reduction

By default, the effective width and length values of the reduced transistor are computed as follows:

$$W = \mathrm{sqrt}(P * Q)$$
$$L = \mathrm{sqrt}(P / Q)$$

where sqrt is the square root function and

$$P = W1*L1 + W2*L2 + \ldots + Wn*Ln$$
$$Q = W1/L1 + W2/L2 + \ldots + Wn/Ln$$

where W*i* and L*i* are the width and length of the *i*th transistor, respectively.

By default, device reduction also computes the area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD), when these values are included in user-defined property calculations. The computation considers any possible swapping of source and drain pins. In the standard case, where all source pins are connected together and all drain pins are connected together, the formulas are as follows:

$$AS = AS1 + AS2 + \ldots + ASn$$
$$AD = AD1 + AD2 + \ldots + ADn$$
$$PS = PS1 + PS2 + \ldots + PSn$$
$$PD = PD1 + PD2 + \ldots + PDn$$

The values of AS and AD, and PS and PD are interchangeable if some of the transistors have source and drain pins swapped. Calibre LVS decides arbitrarily which pin of the resulting reduced device is called source and which pin is called drain; however, property computation is consistent with that decision. An example is shown in figure 10-2.

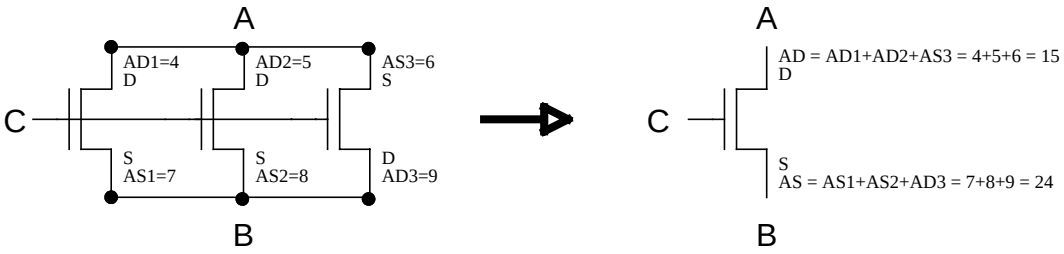


Figure 10-2. Effective AS/AD computation with pin swapping

The LVS Reduce Parallel MOS specification statement controls parallel MOS transistor reduction.

Effective property values are computed in both layout and source. If default effective property computation is used to compute width, length, area of source, area of drain, perimeter of source, and perimeter of drain values, the built-in property names “w”, “l”, “as”, “ad”, “ps”, or “pd” must be used, respectively, except when specified otherwise with Trace Property specification statements.

Recall that the L property is made available from the input database automatically in Calibre LVS if L is required for effective W calculations. The following rule file example shows how this may occur:

```
LVS REDUCE PARALLEL MOS YES
TRACE PROPERTY MP W W 0     //trace W only
```

In this example, only W will be traced for MP devices. If L is present in the input database, L will also be available automatically for property computations. When parallel MOS device reduction occurs, effective W and L values will be calculated for reduced MP devices and the Trace Property statement will have valid W values to trace.

The default effective property computation can be overridden, and other formulas can be specified as described in the section "User-defined Property Reduction".

Series MOS Transistor Reduction

Calibre LVS can reduce a group of series MOS transistors to a single transistor. All transistors in the group must have the same component type, optional subtype, number of pins, and pin names. All source and drain pins must be connected in series. The LVS Reduce Series MOS specification statement controls series MOS reduction.

Gate, bulk, and optional pins must be connected to the same nets (parallel), respectively. In MN, MP, ME, MD, and M devices, source and drain are equivalent and their polarity is immaterial. In LDDN, LDDP, LDDE, LDDD, and LDD devices, source and drain must alternate within the series; a source-to-source or drain-to-drain connection will break the chain at that point. Bulk and optional pins, for all transistor types, are interchangeable if they are specified as logically equivalent. Section "Logically Equivalent Pins" describes how to specify logical equivalence. To be reduced, MOS transistors must have at least three pins with standard pin names, as specified in Table 10-2.

Figure 10-3 shows an example of series MOS transistor reduction.

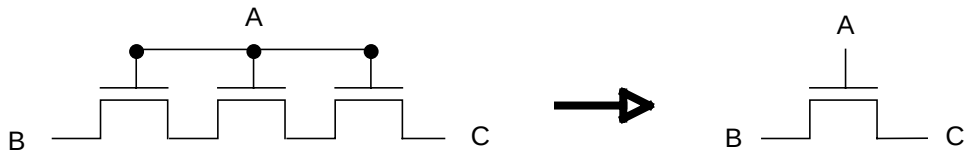


Figure 10-3. Series MOS Transistor Reduction

By default, the effective width and length values of the reduced transistor are computed as follows:

W = sqrt (P / Q)
L = sqrt (P * Q)

where sqrt is the square root function and

P = W1*L1 + W2*L2 + … + Wn*Ln
Q = L1/W1 + L2/W2 + … + Ln/Wn

where W*i*, L*i* are the width and length of the *i*th transistor, respectively.

By default, device reduction does not compute the values for area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD) in series MOS transistors.

The LVS Reduce Series MOS specification statement controls series MOS transistor reduction.

Effective property values are computed in both layout and source. If default effective property computation is used to compute width and length values, the built-in property names “w” and “l” must be used respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “User-defined Property Reduction”.

Semi-series MOS Transistor Reduction

Calibre LVS can reduce a group of semi-series MOS transistors to a single transistor.

All transistors in the group must have the same component type, optional subtype, number of pins, and pin names. All source and drain pins must be connected in series, with bypass nets as shown in figure 10-4. Gate, bulk, and optional pins must be connected to the same nets (parallel), respectively.

Gate, bulk, and optional pins must be connected in parallel (that is, to the same nets respectively). In MN, MP, ME, MD, and M devices, source and drain are equivalent and their polarity is immaterial.

In LDDN, LDDP, LDDE, LDDD, and LDD devices, source and drain must alternate within the series; a source-to-source or drain-to-drain connection will break the chain at that point. Bulk and optional pins, for all transistor types, are interchangeable if they are specified as logically equivalent. Section "Logically Equivalent Pins" describes how to specify logical equivalence. To be reduced, semi-series MOS transistors must have at least three pins with standard pin names, as specified in Table 10-2.

Figure 10-4 illustrates this functionality. The bypass net must not be connected to any devices outside of the respective "row" of the series-parallel structure.

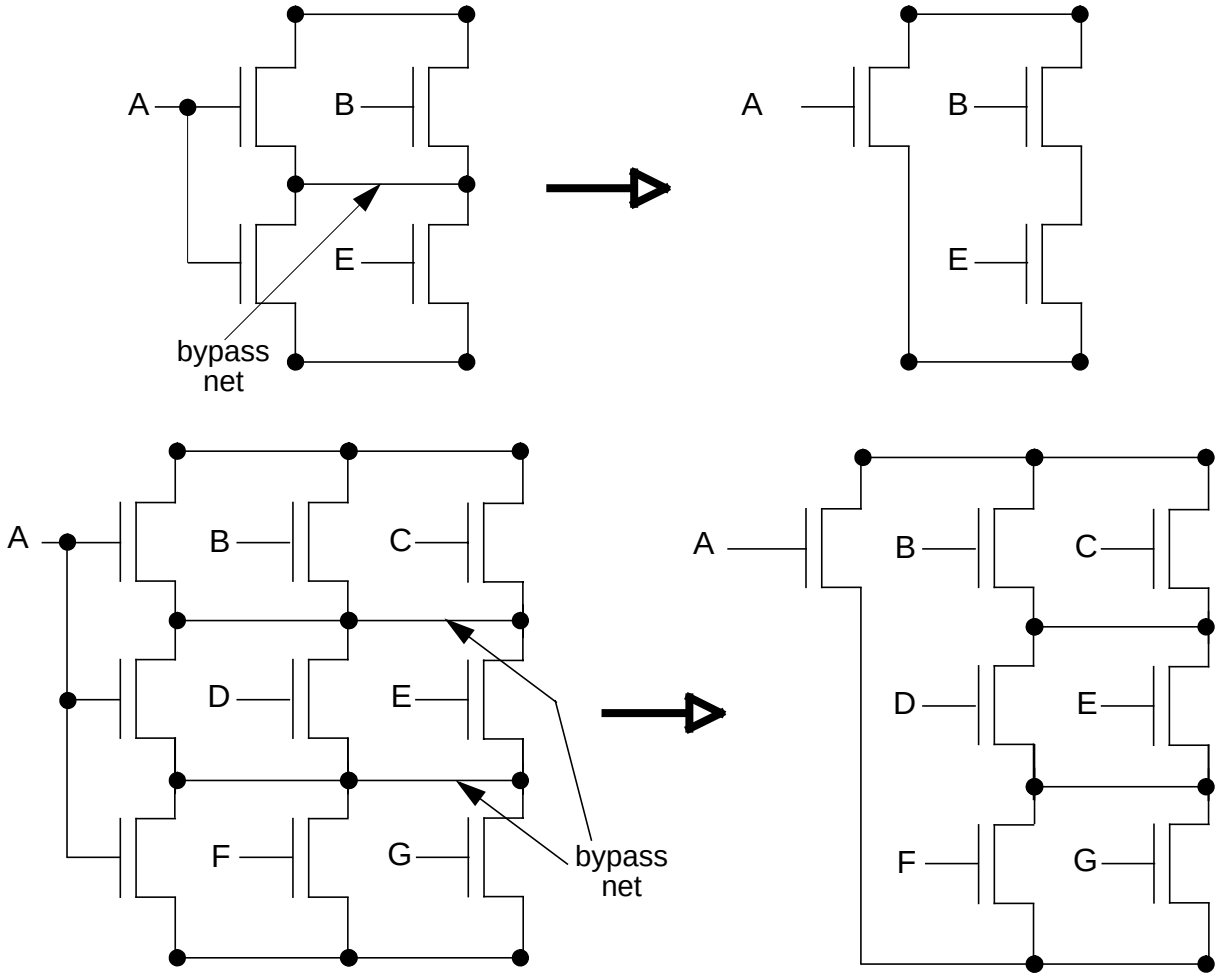


Figure 10-4. Reduce Semi-series MOS, Example

Semi-series MOS reduction is independent of regular series MOS reduction; either or both may be performed.

By default, the width and length values of the reduced transistor are computed as follows:

W = sqrt (P / Q)
L = sqrt (P * Q)

Where sqrt is the square root function and

$$P = W1*L1 + W2*L2 + \ldots + Wn*Ln$$
$$Q = L1/W1 + L2/W2 + \ldots + Ln/Wn$$

where W*i*, L*i* are the width and length of the *i*th transistor, respectively.

By default, device reduction does not compute the values for area of source (AS), area of drain (AD), perimeter of source (PS), and perimeter of drain (PD) in series MOS transistors.

The LVS Reduce Semi Series MOS specification statement controls semi-series MOS transistor reduction.

Effective property values are computed in both layout and source. If default effective property computation is used to compute width and length values, the built-in property names “w” and “l” must be used respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden and other formulas can be specified as described in the section “User-defined Property Reduction”.

Split Gate Reduction


Note

Split gate reduction implies parallel MOS transistor reduction even when Series MOS Transistor Reduction is not requested.

Calibre LVS can reduce a split-gate structure to a single gate structure.

If you request split gate reduction, then split-gate structures are reduced to single gate structures. A split-gate structure consists of two or more strings of MOS transistors (component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD or LDD). The transistors in each string are connected in series and the strings are tied to a common net at each end. The gate pins of respective transistors in each string are shared as shown in figure 10-5. Each group of respective transistors in the original structure is represented with a single transistor in the reduced structure.

When Logic Gate Recognition is enabled, then the order of respective transistors within each string can be different in different strings; when logic gate recognition is disabled then the order must be the same in all strings.

All the transistors must have the same the same component type, the same number of pins and the same pin names. Subtypes must be the same in each group of transistors that are collapsed together. For example, in Figure 10-5 subtypes must be the same in each row; they can be different in different rows. If there are more then three pins, then all those optional pins must be connected in parallel. That is, they must all be connected to the same nets respectively. In MN, MP, ME, MD and M devices, source and drain pins may be swapped. In LDD-type devices, the series connection must be between the source pin of one device and the drain pin of another. Optional pins may be swapped if they are specified as logically equivalent. See the section on Logically Equivalent Pins.

To participate in split gate reduction, MOS transistors must have at least three pins with the standard pin names as specified in the section Built-in Device Types. Initial correspondence points prevent split gate reduction; specifically, internal nets in a split gate structure are not collapsed with other nets if they form initial correspondence points.

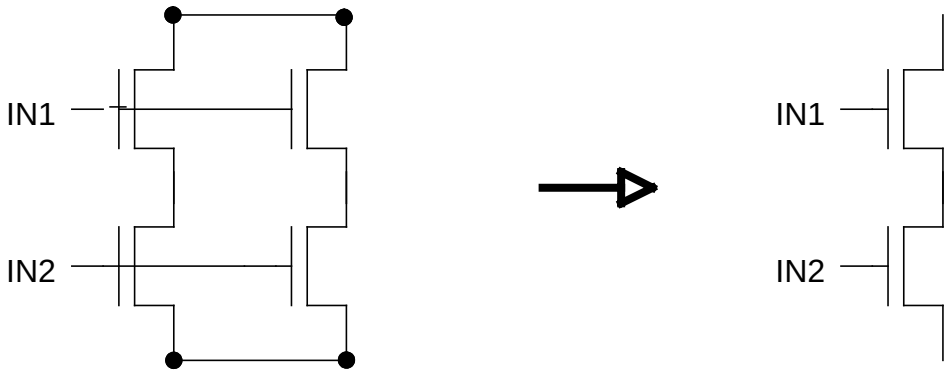


Figure 10-5. Split Gate Reduction

Individual transistors in a split-gate structure are matched based on their “gate” pin connections (transistor pin name G). Internal nets in a split-gate structure are matched to corresponding nets in the other design based solely on their relative distance from the “top” and “bottom” of the structure. All original internal nets are matched; as a result, several nets in one design will match a single net in the other

design. If there is a split-gate structure on both sides then a representative net is chosen from each group of nets that were collapsed together and respective nets in the other design are matched to that representative.

By default, for each group of transistors which is reduced to a single transistor, the width and length values of the reduced transistor are computed as follows:

$$L = \mathrm{sqrt}\ (P\ /\ Q)$$
$$W = \mathrm{sqrt}\ (P * Q)$$

where sqrt is the square root function and

$$P = W1*L1 + W2*L2 + \ldots + Wn*Ln$$
$$Q = W1/L1 + W2/L2 + \ldots + Wn/Ln$$

where W*i*, L*i* are the width and length of the *i*th transistor, respectively.

The LVS Reduce Split Gates specification statement controls split gate reduction.

Effective width and length values are computed in both layout and source. If default effective property computation is used to compute width and length values, the built-in property names “w” and “l” must be used, respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden and other formulas can be specified as described in the section “User-defined Property Reduction”.

Semi-split Gate Reduction

Semi-split gate reduction can be performed by using the SEMI ALSO option to the LVS Reduce Split Gates statement.

Calibre LVS can reduce a semi-split gate structure in addition full-split gates. Semi-split gate reduction is similar to full-split gate reduction, except that only some of the gate pins in the structure must be shared. Transistors with shared gate pins are collapsed; transistors with different gate pins are left separate. Reduction proceeds in horizontal rows from the top and bottom of the structure as long as the transistors in each row have shared gate pins. Reduction stops when a row is encountered where gate pins are not shared.

Figure 10-6 shows that transistors in row E are not reduced. Unlike full-split gates, the order of transistors in semi-split gates must be the same for all strings of MOS transistors; regardless of whether or not logic gate recognition is enabled. All strings must consist of the same number of transistors.

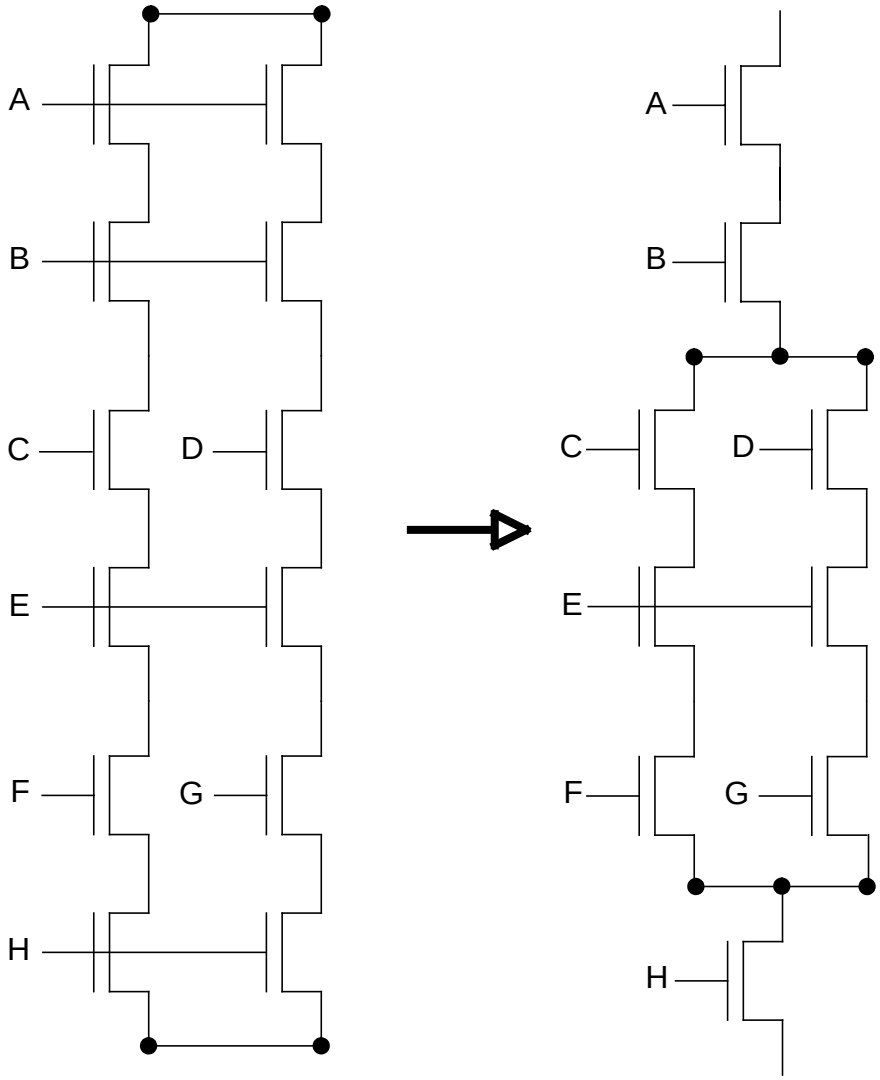


Figure 10-6. Reduce Split Gates, Example

The SAME ORDER option of the LVS Reduce Split Gates specification statement specifies that split gates should be collapsed only when the input order is the same for all strings of transistors that form the split gate.

Figure 10-7 shows the effect of the SAME ORDER option.

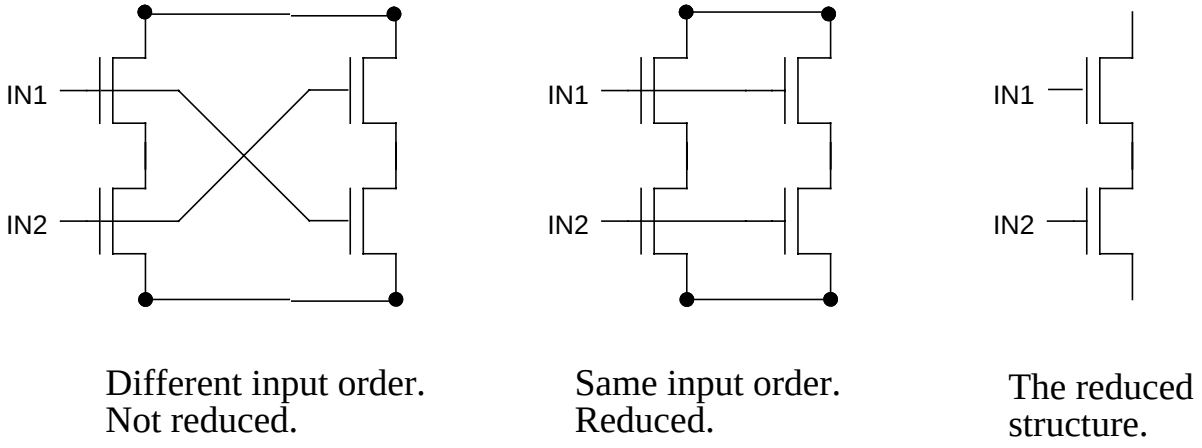


Figure 10-7. Reduce Split Gates SAME ORDER, Example

Parallel Bipolar Transistor Reduction

Calibre LVS can reduce a group of parallel bipolar transistors to a single transistor. All transistors in the group must have the same optional component subtype, number of pins, and pin names. All collector, base, emitter, and optional pins (if any), must be connected to the same nets. Device reduction can swap the optional pins if they are specified as logically equivalent. Section "Logically Equivalent Pins" describes how to specify logical equivalence. To be reduced, bipolar transistors must have at least three pins with the standard pin names as specified in table 10-6. Figure 10-8 shows an example of parallel bipolar transistor reduction.

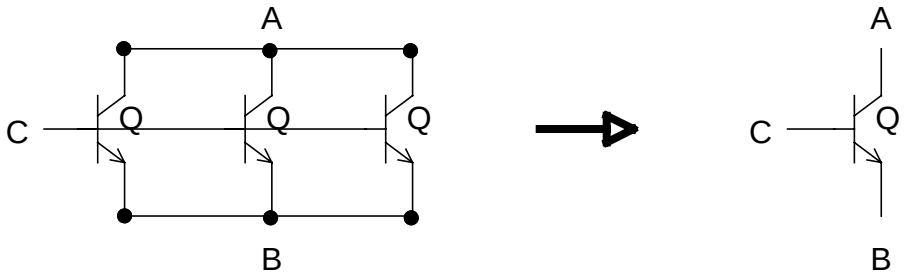


Figure 10-8. Parallel Bipolar Transistor Reduction

By default, the area of the reduced transistor is computed as follows:

$$A = A1 + A2 + \ldots + An$$

where A*i* is the area of the *i*th transistor.

By default, the width and length values of the reduced transistor are computed as follows:

```
W = sqrt ( P * Q )
L = sqrt ( P  /  Q )
```

where sqrt is the square root function and

```
P = W1 * L1 + W2 * L2 + … + Wn * Ln
Q = W1 / L1  + W2 / L2  + … + Wn / Ln
```

where W*i*, L*i* are the width and length of the *i*th transistor, respectively.

The LVS Reduce Parallel Bipolar specification statement controls parallel bipolar transistor reduction.

Effective property values are computed in both layout and source. If default effective property computation is used to compute area, width, and length values, the built-in property names “a”, “w”, and “l” must be used respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden and other formulas can be specified as described in the section “User-defined Property Reduction”.

Series Capacitor Reduction

Calibre LVS can reduce a group of serially connected capacitor devices to a single capacitor. All capacitors in the group must have the same optional component subtype, number of pins, and pin names. All positive and negative pins must be connected in series. The positive and negative pins must alternate within the series; a positive-to-positive or negative-to-negative connection will break the chain at that point, unless they are specified as logically equivalent. Section “Logically Equivalent Pins” describes how to specify logical equivalence. All optional pins must be connected to the same nets (parallel), respectively. The optional pins can be swapped if they are specified as logically equivalent.

To be reduced, capacitors must have at least two pins with the standard pin names as specified in table 10-3.

Figure 10-9 shows an example of series capacitor reduction.

Figure 10-9. Series Capacitor Reduction

By default, the capacitance of the resulting device is computed as follows:

$$C = 1 / (1/C1 + 1/C2 + \ldots + 1/Cn)$$

where C*i* is the capacitance of the *i*th capacitor, respectively.

The LVS Reduce Series Capacitors specification statement controls series capacitor reduction.

Effective capacitance values are computed in both layout and source. If default effective property computation is used to compute the capacitance value, the built-in property name “c” must be used, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “User-defined Property Reduction”.

Parallel Capacitor Reduction

Calibre LVS can reduce a group of parallel capacitors to a single capacitor. All capacitors in the group must have the same optional component subtype, number of pins, and pin names. All positive, negative, and optional pins (if any) must be connected to the same nets. The positive, negative, and optional pins can be swapped if they are specified as logically equivalent. Section “Logically Equivalent Pins” describes how to specify logical equivalence. All optional pins must be connected to the same nets (parallel), respectively. The optional pins can be swapped if they are specified as logically equivalent. To be reduced, capacitors must have at least two pins with the standard pin names as specified in table 10-3.

Figure 10-10 shows an example of series capacitor reduction.

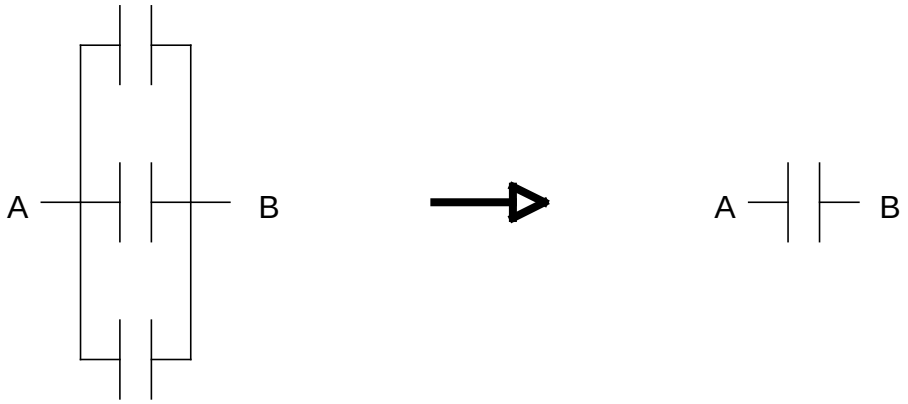


Figure 10-10. Parallel Capacitor Reduction

By default, the effective capacitance, area, and perimeter values of the resulting device are computed as follows:

$$C = C1 + C2 + \ldots + Cn$$
$$A = A1 + A2 + \ldots + An$$
$$P = P1 + P2 + \ldots + Pn$$

where C*i*, A*i*, and P*i* are the capacitance, area, and perimeter of the *i*th capacitor, respectively.

The LVS Reduce Parallel Capacitors specification statement controls parallel capacitor reduction.

Effective capacitance, area, and perimeter values are computed in both layout and source. If default effective property computation is used to compute capacitance, area, and perimeter values, the built-in property names “c”, “a”, and “p” must be used, respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden and other formulas can be specified as described in the section “User-defined Property Reduction”.

Series Resistor Reduction

Calibre LVS can reduce a group of serial resistors to a single resistor. All resistors in the group must have the same optional component subtype, number of pins, and

pin names. All positive and negative pins must be connected in series. The positive and negative pins must alternate within the series; a positive-to-positive or negative-to-negative connection will break the chain at that point, unless they are specified as logically equivalent. Section "Logically Equivalent Pins" describes how to specify logical equivalence. All optional pins must be connected to the same nets (parallel), respectively. The optional pins can be swapped if they are specified as logically equivalent.

To be reduced, resistors must have at least two pins with the standard pin names as specified in table 10-4.

Figure 10-11 shows an example of series resistor reduction.

Figure 10-11. Series Resistor Reduction

By default, the resistance value of the reduced transistor is computed as follows:

$$R = R1 + R2 + \ldots Rn$$

where R*i* is the resistance of the *i*th resistor, respectively.

By default, the width and length values of the resulting device are computed as follows:

$$W = \text{sqrt} (P / Q)$$
$$L = \text{sqrt} (P * Q)$$

where sqrt is the square root function and

$$P = W1*L1 + W2*L2 + ... + Wn*Ln$$
$$Q = L1/W1 + L2/W2 + ... + Ln/Wn$$

where W*i*, L*i* are the width and length of the *i*th resistor respectively.

The LVS Reduce Series Resistors specification statement variable controls series resistor reduction.

Effective resistance, width, and length values are computed in both layout and source. If default effective property computation is used to compute resistance, width, and length values, the built-in property names “r”, “w”, and “l” must be used, respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “User-defined Property Reduction”.

Parallel Resistor Reduction

Calibre LVS can reduce a group of parallel resistors to a single resistor. All resistors in the group must have the same optional component subtype, number of pins, and pin names. All positive, negative, and optional pins (if any) must be connected to the same nets. The positive, negative, and optional pins can be swapped if they are specified as logically equivalent. Section “Logically Equivalent Pins” describes how to specify logical equivalence. All optional pins must be connected to the same nets (parallel), respectively. The optional pins can be swapped if they are specified as logically equivalent. To be reduced, resistors must have at least two pins with the standard pin names as specified in table 10-4.

Figure 10-12 shows an example of parallel resistor reduction.

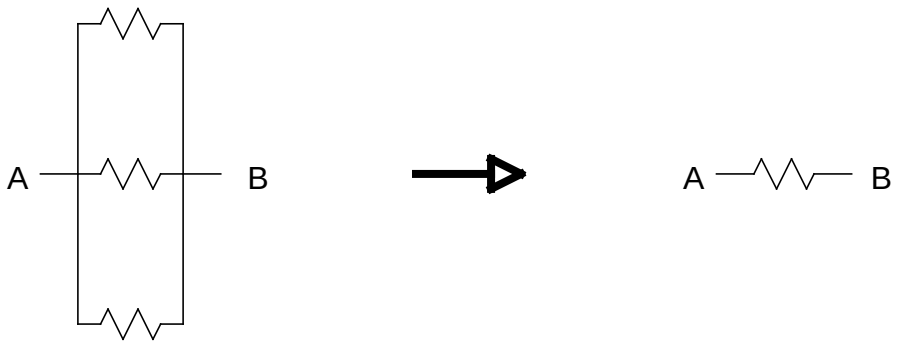


Figure 10-12. Parallel Resistor Reduction

By default, the resistance value of the reduced transistor is computed as follows:

$$R = 1 / (1/R1 + 1/R2 + \ldots + 1/Rn)$$

where R*i* is the resistance of the *i*th resistor.

By default, the width and length values of the resulting device are computed as follows:

W = sqrt (P * Q)
L = sqrt (P / Q)

where sqrt is the square root function and

P = W1*L1 + W2*L2 + … + Wn*Ln
Q = W1/L1 + W2/L2 + … + Wn/Ln

where W*i* and L*i* are the width and length of the *i*th resistor, respectively.

The LVS Reduce Parallel Resistors specification statement controls parallel resistor reduction.

Effective resistance, width, and length values are computed in both layout and source. If default effective property computation is used, then to compute resistance, width, and length values, the built-in property names “r”, “w”, and “l” must be used, respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “User-defined Property Reduction”.

Parallel Diode Reduction

Calibre LVS can reduce a group of parallel diodes to a single diode. All diodes in the group must have the same optional component subtype, number of pins, and pin names. All positive, negative pins, and optional pins, must be connected to the same nets. Device reduction can swap all pins if they are specified as logically equivalent. Section “Logically Equivalent Pins” describes how to specify logical equivalence. To be reduced, diodes must have at least two pins with the standard pin names as specified in table 10-5.

Figure 10-13 shows an example of parallel diode reduction.

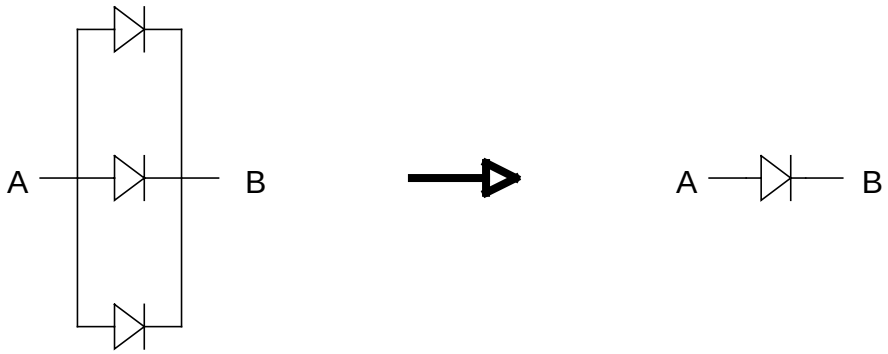


Figure 10-13. Parallel Diode Reduction

By default, the area and perimeter values of the reduced diode are computed as follows:

$$A = A1 + A2 + \ldots + An$$
$$P = P1 + P2 + \ldots + Pn$$

where A*i* and P*i* are the area and perimeter of the *i*th diode respectively.

The LVS Reduce Parallel Diodes specification statement controls parallel resistor reduction.

Effective area and perimeter values are computed in both layout and source. If default effective property computation is used to compute area and perimeter values, the built-in property names “a” and “p” must be used, respectively, except when specified otherwise with Trace Property specification statements.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “User-defined Property Reduction”.

Unequally Reduced Devices

Calibre LVS verifies that each group of parallel MOS transistors in the source corresponds to a group of parallel MOS transistors in the layout that has the same number of elements when parallel MOS transistor reduction is requested.

Warnings are issued in the LVS report if this is not the case. This check is only performed for properly formed MOS transistors; specifically, instances with

component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, or LDD that have at least three pins with the standard names as specified in table 10-2.

Placing groups of parallel MOS transistors in the source ensures that the layout will consist of a specified number of transistors. If this is not a requirement, then single transistors should be used in the source.

Missing and Unknown Property Values

Under certain conditions, LVS may assign “missing” or “unknown” values to properties during device reduction. This may occur during effective property calculation, using either built-in or user-defined effective property calculation formulas, and using either built-in or generic device reduction specification statements.

Consider a property X for which effective values are being computed, and consider a group of devices that are being reduced to a single device (the input group). The following rules apply:

- Original input devices have “missing” property values if the property values are not found in the input database.
- If property X is “missing” on all devices in the input group then the effective value for X is “missing”.
- Otherwise, if all input values are present and have valid numeric values then the effective value for X is calculated as specified by the built-in or user-specified formulas, whichever applies.

  Input values is defined as the set of all property values in the input group that participate in the calculation of effective value for X. This may include original values of X as well as values of other properties that participate in the calculation of X (so-called “partner” properties).

  If the effective value for X cannot be calculated because there is no formula for the calculation of X, or because of a run-time problem in the calculation (such as division by zero, overflow, and so on), or for another reason, then the effective value for X is “unknown”.

- Otherwise, the effective value for X is “unknown”.

For example, an “unknown” value is obtained when values of property X are present on some devices in the input group but missing on others; or if some devices in the input group already have “unknown” values for X; or if values of property X are present on all devices in the input group but values of another property that participates in the calculation of X are missing; and so on.

Unknown property values are reported as discrepancies in the `PROPERTY ERRORS` section of the LVS report if the property in question is traced. Unknown values are represented in the report with “?” characters. Missing property values in original input devices are reported as discrepancies in the `SOURCE ERRORS` or `LAYOUT ERRORS` sections of the report (unless disabled with LVS Report Option E). Missing property values on reduced devices are not reported at all, since they are necessarily caused by missing values on original input devices.

Device Reduction Program Structure

Device reduction programs may appear in various LVS Reduce... specification statements. For example, recall the syntax for parallel MOS reduction:

```
LVS Reduce Parallel MOS { YES [ reduction_program ] | NO }
```

A device reduction program is a list of instructions that control how device reduction is done and what is computed in the process. A device reduction program is always part of some LVS Reduce... specification statement and applies to devices on which that statement operates. The basic structure of a device reduction program consists of a reduction tolerance and/or an effective property computation. The entire program is enclosed by square brackets.

The reduction tolerance and effective property computation may appear in any order. At least one of one of these statements must be present in a device reduction program. At most one reduction tolerance section and at most one effective

property computation section may be specified per device reduction program. Here is a simple example:

```
LVS REDUCE PARALLEL RESISTORS YES [
     tolerance W 0
     effective W
     W = min( W )
]
```

Tolerance in Device Reduction

The device reduction statements, listed below, allow you to limit or prevent reduction of devices that have different property values.

LVS Reduce Parallel Bipolar
LVS Reduce Parallel Capacitors
LVS Reduce Parallel Diodes
LVS Reduce Parallel MOS
LVS Reduce Parallel Resistors
LVS Reduce Series Capacitors
LVS Reduce Series MOS
LVS Reduce Series Resistors
LVS Reduce Split Gates

You specify the desired limitations through the use of the TOLERANCE secondary keyword set.

```
[TOLERANCE property_name tolerance_number]
```

The parameters of this secondary keyword set is fully described on each dictionary page for the LVS Reduce statements.

For example:

```
LVS REDUCE PARALLEL MOS YES [ TOLERANCE L 0 W 0 ]
```

The TOLERANCE keyword is followed by one or more *property_name tolerance_number* pairs, indicating property names and respective tolerance values. Each *tolerance_number* parameter belongs to the *property_name* parameter that precedes it. Property names must be simple names; property-name/Spice-parameter combinations such as “instpar(w)” are not allowed. To handle such combinations use the LVS Property Map specification statement.

At most one TOLERANCE section may be specified per device reduction program.

Devices are not reduced together if they own the indicated property, the property has different values and the difference exceeds *tolerance_number*; *tolerance_number* is a floating point number indicating the tolerance in percentage points. The formula for calculating difference is abs((v1-v2)/v1)*100 where abs is the absolute value function and v1 and v2 are the property values being compared. It is, of course, arbitrary which property value gets to be called v1 and which one is v2. A property with zero value is considered to be infinitely different from any non-zero property value. Two properties with zero values are identical and the difference between them is zero.

When several *property_name tolerance_number* pairs are specified, the check is done for each property separately; devices are not reduced if the difference in at least one property exceeds the tolerance specified for that property.

In the process of device reduction, LVS iterates over series and parallel reduction steps; for example, series and parallel capacitor reduction steps are repeated. At each iteration, LVS computes effective property values for devices that have been reduced *so far* (like effective width, length, capacitance, resistance, and so on). In the first iteration, processing of TOLERANCE statements is based on original property values specified in the input database. In subsequent iterations, processing of TOLERANCE statements is based on effective property values computed so far. At each step, devices may have valid property values, or they may receive property values of type “missing” or “unknown”, as described in section Missing and Unknown Property Values. The treatment of the latter is described below.

If a property used in a reduction TOLERANCE statement is “missing” on a device, then the device participates in the reduction as if the TOLERANCE statement were not present. In other words, with respect to the reduction TOLERANCE statement, a device with a “missing” property value behaves as if the property value was identical to the values on all other devices. If the device is an original input device then a missing-property discrepancy is reported. (You can disable missing-property discrepancies with LVS Report Option E).

If a property used in a reduction TOLERANCE statement is “unknown” on a device then the device does not participate in any subsequent reduction iterations.

Devices with “unknown” property values that cause reduction to cease in this manner are reported under the headings “Source Instances With Undetermined Reduction TOLERANCE Properties” and “Layout Instances With Undetermined Reduction TOLERANCE Properties” in the Information And Warnings section of the LVS report. Here is an example:

```
o Layout Instances With Undetermined Reduction TOLERANCE
Properties:

   Listed below are layout instances which caused reduction to
   cease because: [a] LVS REDUCE ... [ TOLERANCE <property>
   <value> ] was specified, and [b] the <property> on that
   instance was not available. The instance in question was
   reduced from other instances, and the effective property
   value could not be computed.)

   instance                                    property

   M2001 (MN)  (reduced instance)              w: ?
   M2014 (MN)  (reduced instance)              w: ?
   M2021 (MN)  (reduced instance)              w: ?
```

Reduction Tolerance Examples

The following statement reduces parallel MOS devices only when the length values are equal.

```
LVS REDUCE PARALLEL MOS YES [TOLERANCE L 0]
```

The following statement reduces parallel MOS devices only when both width and length are equal.

```
LVS REDUCE PARALLEL MOS YES [TOLERANCE L 0 W 0]
```

The following statement reduces series resistors only when the resistance values are within 5% tolerance and the length values are equal.

```
LVS REDUCE SERIES RESISTORS YES [TOLERANCE R 5 L 0]
```

User-defined Property Reduction

You can specify an effective property computation section in most LVS Reduce statements. This optional section consists of a user-provided program that defines the calculations to take place during device reduction. These programs are written in a language similar to the one available for defining properties in association with the Device statement.

The presence of an effective property program cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property program, you must define formulas for all properties of interest, including built-in properties.

Some effective property programs can become lengthy and may extend over several lines in the rule file.

You can specify an effective property program in the following specification statements:

- LVS Reduce Parallel Bipolar
- LVS Reduce Parallel Capacitors
- LVS Reduce Parallel Diodes
- LVS Reduce Parallel MOS
- LVS Reduce Parallel Resistors
- LVS Reduce Series Capacitors
- LVS Reduce Series MOS
- LVS Reduce Series Resistors
- LVS Reduce Split Gates

For information on the syntax for the rule file statements above, refer to the *Standard Verification Rule Format (SVRF) Manual*.

You *cannot* specify an effective property program in the following statement:

- LVS Reduce Semi Series MOS

Effective Property Language Example

This section describes a sample usage of the effective property language by explaining the sample line by line.

This example illustrates many of the features of the language. The line numbers on the left are an aid for discussion and are not part of the example text and the language is case insensitive.

In the explanation that follows the example, the term “input group” refers to the set of devices being reduced to a single instance. Effective property values are calculated based upon this input group, and stored on the resulting (*effective*) instance.

```
 1  lvs reduce parallel resistors yes
 2    [
 3      effective r, l, w
 4      //
 5      p = sum (w*l)
 6      q = sum (w/l)
 7      w = sqrt ( p*q )
 8      l = sqrt ( p/q )
 9      //
10      checkForZero = prod ( r )
11      if ( checkForZero == 0 )
12        r = 0
13      else
14      { // demonstrate curly braces
15        // assign 1/sum(1/r) in two steps:
16        recipSum = sum(1/r)
17        r = 1 / recipSum
18      }
19    ]
```

- **Line 1**—A standard LVS Reduce specification statement.

  In this case, it enables parallel resistor reduction.

- **Line 2**—An opening square bracket.

  The opening square bracket ([ ) is necessary to indicate an effective property program. This square bracket must immediately follow the YES keyword. The square bracket may only be present if YES is present.

- **Line 3**—The EFFECTIVE property statement.

The EFFECTIVE property statement declares the names of all properties for which effective property values are computed throughout the program. Each property listed here must be assigned a value in the program. If an effective property is to be calculated, it must be listed in this statement.

In this example, the names represent both input values (properties found on the input group) and output values (the properties to be calculated and assigned to the resulting instance).

The properties are part of a comma (,) separated list.

- **Line 4**—A commented line.

  For information on the types of comments allowed in the effective property language, refer to section "Comments".

- **Lines 5 and 6**—Local variable declaration and value assignment.

  - Line 5—The local variable P is declared, and assigned the value:

    ```
    W1*L1 + W2*L2 + ... + Wn*Ln
    ```

    where W*i* and L*i* are the width and length of the *i*th device in the input group, respectively.

  - Line 6—The local variable Q is declared, and assigned the value:

    ```
    W1/L1 + W2/L2 + ... + Wn/Ln
    ```

    where W*i* and L*i* are the width and length of the *i*th device in the input group, respectively.

  The W and L variables in these equations are not the same variables in lines 7 and 8. The W and L variables referred to here, are retrieved from the input group, while the W and L variables in lines 7 and 8 are assigned to the resulting instance.

- **Lines 7 and 8**—Value assignment to variables declared in the EFFECTIVE property statement.

- Line 7—The value W, declared in the EFFECTIVE property statement, is assigned a value.

- Line 8—The value L, declared in the EFFECTIVE property statement, is assigned a value.

Since W and L are declared in the EFFECTIVE property statement, they must be assigned values somewhere in the program. These two lines satisfy that requirement. The resulting W and L values are assigned to the single instance which represents the reduced input group.

- **Line 9**—See line 4

- **Lines 10 through 18**—Property language usage showing how to avoid the use of zero (0) values from the input group.

  If you only provided the statement:

  ```
  r = 1/sum(1/r)
  ```

  and an R*i* in the input group had a zero value, the effective R value would be set to the unknown value. The unknown value is represented by a question mark (?) in the LVS Report file. The unknown value is assigned because division by zero is attempted and is undefined. The unknown value can cause property discrepancies to be reported when the Trace Property statement is in use. The unknown value can also cause input group reduction to be avoided when the you specify the TOLERANCE keyword in an LVS Reduce statement. Instead, lines 10-18 set the effective R value to zero if the input group contains any zero-valued R*i* values.

  - Line 10—The local, and temporary, variable checkForZero is declared and assigned the value:

    ```
    R1 * R2 * ... * Rn
    ```

    where R*i* represents the R value for the *i*th instance in the input group. Note that checkForZero will be zero if *any* of the input R values are zero.

  - Line 11—A conditional statement.

- Line 12—The effective R value is assigned a zero value if the conditional statement in line 11 is true. This avoids the use of the unknown value.

- Line 13—The ELSE statement.

- Line 14—Demonstrate the use of the opening curly brace ({) and how it can be used to group statements into a single logical statement. As in C or C++, this grouping is useful for indicating the extent of a dependent clause for a conditional statement.

- Line 15—A commented line that describes the goal of the statement group.

- Line 16—The local, and temporary, variable recipSum is declared and assigned the value:

  ```
  1/R1 + 1/R2 + ... 1/Rn
  ```

  where R*i* represents the R value for the *i*th instance in the input group

- Line 17—The effective R value is assigned the value:

  ```
  1 / recipSum
  ```

  which is equivalent to

  ```
  1 / ( 1/R1 + 1/R2 + ... 1/Rn )
  ```

  You could replace lines 14-18 with the single statement:

  ```
  R = 1/SUM(1/R)
  ```

  but, for this example, we wanted to show the usage of statement grouping within curly braces ({}).

- Line 18—A closing curly brace (}) ends the statement group.

- Line 19—A closing square bracket ( ]) ends the effective property program.

Note you could replace lines 14-18 by the following:

```
r = 1 / sum(1/r)
```

The built-in language is similar to the C programming language without the semicolons. Semicolons are left out for stylistic compatibility with the remainder of the rule file. The effective property language is derived from, and is quite similar to, the property specification language provided with the Device statement.

Effective Property Language Syntax

You can specify at most one effective property program per LVS Reduce specification statement. The program lists the names and types of the properties to be computed upon reduction, and specifies the method of computation from available data.

For the sake of the discussion below, “input group” means the set of devices which are being reduced to a single instance. Effective property values will be calculated based upon this input group, and stored on the resulting (*effective*) instance.

The effective property program is placed between a pair of square brackets at the end of an LVS Reduce specification statement. This placement is an extension of the use of square brackets to contain reduction TOLERANCE statements. Effective property programs cannot be shared by multiple LVS REDUCE statements.

- **Numeric Restriction**

  The effective property language handles only numeric valued properties; string valued properties are not supported. All literal values are specified in double precision floating point and effective calculations, specified by the user in the program, are also carried out in double precision. Note, however, that effective values are restricted to single precision during the LVS comparison phase. Effectively, values are restricted to single precision when they are assigned to the single instance representing a reduced input group.

- **Language Style**

  The built-in language is a blend of the expression and computational statement style of C, together with conventions common to the rest of the rule file, also added were necessary declarations and functions.

- **Structure**

  The effective property computation built-in language is structured as a sequence of statements. First there is an optional DEBUG statement, an optional WARN statement, a required EFFECTIVE statement, and finally a sequence of one or more property computation statements. Each property computation statement is either an assignment statement, an IF statement, or an IF … ELSE statement. There are no loop statements in the language.

- **Statement Placement and Continuation**

  In keeping with the previously established syntactic conventions of the rule file, semicolons or other separating devices are not used between statements. The DEBUG, WARN, EFFECTIVE, and IF statements are recognized by their keywords which must be the first item in the statement. Assignment statements are recognized by the equal sign (=) operator which is always the second item in the statement. Statements may be continued onto multiple lines by breaking them at any white space. Continuation characters are not employed. Common practice dictates that each statement begins on a new line, but this is not required.

- **Reserved Keywords**

  The keywords of the language such as DEBUG, PROPERTY, IF, and ELSE are reserved. Similar to general rule file syntax, if a word is surrounded by single quotes (‘) or double quotes (“), it is not taken to be a keyword but rather a normal identifier for a property name or a local (temporary) variable. Therefore, the reserved nature of keywords places no restrictions on property, or local variable naming.

- **Optional Keyword and Function Spellings**

  The longer keywords and function names often have a shortened form consisting of a set of letters from the complete name. In this section, the letters in the shortened form are shown in uppercase with the remaining letters in lower-case. Thus EFFective indicates that the spelling is either EFFECTIVE or EFF. If any of the optional letters are used, they must all be used; therefore, EFFECT is not a valid shortening of EFFective.

- **Case Sensitivity**

  The language is case insensitive. All upper case letters are converted to lower case for internal purposes. For example, a variable may be referred to as “maxL”, “MaxL”, or “MAXL” interchangeably within the same program. The use of mixed upper and lower case letters in this section is used only to show allowed abbreviations.

- **Data Sources**

  Data used in the computation may come from any of the following sources:

  - **Numeric constants**—You can use numeric floating-point constants directly within the program.

    Numeric constants are constructed according to the same rules as in the rest of the rule file, namely as valid C integer, float, or double constants. The following are some examples:

    3 3.0 -2.5 4.6e-10 5e8 5E9 0 1

  - **Instance data**—You can access property data associated with instances in the input group through the use of built-in vector functions. Vector functions include the SUM() and PROD() functions. For a complete list of vector functions, refer to section “Effective Property Language Vector Functions”.

    When you specify a property name “X” inside a vector function, it refers to individual X*i* values on the instances in the input group, treated

as a vector or array. In all other cases, “X” is a scalar and refers to the current property value on the reduced or “effective” instance.

- **Local variables**—You can declare local variables, also referred to as temporary variables, and assign them scalar values. Local variables cannot serve as arguments to vector functions because there are no such property values associated with the instances in the input group. In other words, there are no individual values for local variables associated per-instance in the input group.

  You can assign intermediate values to local variables within an effective property computation program. For compatibility with rule file style, it is neither necessary nor possible to declare such variables before use. Any name may be used which does not conflict with property names or process variables. However, it is best to avoid names beginning with “temp” or “init”, because these names are used in debugging output to identify temporary variables. Although the compiler and interpreter will not be confused by your use of these names, you may become so when reading the debugger output.

- **Operators**

  A subset of the operators of C are available. They have the same meaning and precedence as in C and are listed below in order of decreasing precedence.

```
-   !               (unary minus)      (logical negation)
*   /               (multiplication)   (division)
+   -               (addition)         (subtraction)
< <= == > = > !=    (relational operators: lt,le,eq,ge,gt,ne)
&&                  (logical and)
||                  (logical or)
=                   (assignment)
```

- **Comments**

  The comment conventions for the built-in language are the same as for the rest of the rule file. Any text beginning with a double slash (//) through the end of the line is ignored. Any text inside a pair of C-style markers (/* and */) is also ignored.

- **Commas**

  All commas shown in the language are required.

- **Parentheses**

  You can use parentheses in expressions to override normal operator precedence.

- **Numeric Expressions**

  The arithmetic operators - (unary), +, -, *, and / are used to build up numeric expressions from constants, variables, and function references.

- **Logical Expressions**

  Logical expressions may only appear within the parenthesis following the keyword IF. Basic logical expressions are formed by combining numeric expressions using the relational comparison operators. Logical expression may be further combined using parenthesis together with the logical AND (&&) and logical OR (||) operators.

- **Assignment Statements**

  The assignment statement has the following form:

  ```
  <local-variable-or-property-name> = <numeric-expression>
  ```

  Only one assignment operator (=) is allowed per assignment statement.

- **Flow Control**

  The only flow control is provided by the IF and IF ELSE statements, which have the same meaning and use as in C. These statements have the following forms:

  ```
  IF ( <logical-expression> ) <statement>

  IF ( <logical-expression> ) <statement> ELSE <statement>
  ```

where the parentheses are required.

- **Statement Grouping**

  You can use curly braces ({ }) to group one or more statements into a single statement, as in C. For a usage example, refer to the section "Effective Property Language Example".

Effective Property Language Statements

This section describes the effective property language statements, in the order they need to appear, if specified.

DEBUG Statement

This statement controls the tracing of the effective property computation. Tracing of can be useful in finding errors during the development of new effective property computation code.

This is an optional statement. If specified, it must be specified first in the effective property program. The DEBUG statement has the following syntax:

```
DEBUG  <range1> [ , <range2> , … <rangeK> ]
```

where each range is either an instance ID, or a pair of instance IDs separated by a hyphen (-). You must specify at least one range value, although it can contain a single instance ID.

An instance ID represents a single device to which the input group will be reduced. Instance IDs are unique across all devices in the design (layout or source) being transformed. Calibre LVS assigns instance IDs and are not necessarily sequential, merely unique. Instance IDs are most useful in debugging warnings produced when the WARN Statement is present, and an impossible mathematical operation is attempted.

The following is an example of a DEBUG statement:

```
DEBUG 0-2, 508
```

This statement prints a detailed report to the transcript of the property computation for each device instance with an ID of 0, 1, 2, and 508. The trace can produce an large amount of output, so the use of small ranges is recommended. You can obtain instance IDs from output produced by the WARN statement.

WARN Statement

This statement prints a warning to the transcript if the effective property program attempts to perform an impossible mathematical computation, such as dividing by zero or taking the square root of a negative number. Calibre LVS does not generate an error when performing an impossible mathematical computation.

This is an optional statement. If specified, it must appear after the optional DEBUG statement and before the required EFFective statement. The WARN statement has the following syntax:

```
WARN
```

This statement takes no arguments.

If the tool performs any *impossible* mathematical computations the effective property is set to the unknown value, which results in it being represented by a question mark (?) in the LVS Report. When a property is set to the unknown value, it can cause discrepancies when tracing properties. An unknown value also precludes devices in the input group from being reduced together if you specify TOLERANCE in the LVS Reduce statement.

The following is a sample transcript of the warning printed by the WARN statement:

```
**************************************************
EFFECTIVE property user-program WARNING:
Division by zero (result set to UNKNOWN).
while transforming layout with:
LVS REDUCE ... EFFECTIVE clause: (rule file line 58)
Use instance ID# 92 in DEBUG trace range.
**************************************************
```

You can use the instance ID# 92 in a DEBUG trace range to see exactly how the unknown value was determined for this input group and the effective (reduced) instance.

EFFective Statement

This statement is used to declare the names of the properties for which Calibre LVS computes effective values. Properties listed here are considered both input (should be present on instances in the input group), and output (calculated values are assigned to the effective/reduced instance).

This is a required statement. It must follow the optional DEBUG and WARN statements, if present. The EFFective statement has the following syntax:

```
EFFective <property_name_1>, <property_name_2>, ...
```

The following example declares four properties are declared with names W, L, AS, and AD:

```
EFFECTIVE W, L, AS, AD
```

Property names are treated as variables within computational statements. If you declare a property name in the EFFective statement, you must assign it a value somewhere in the program. If you do not, a compile error results.

All properties named in the EFFective statement are available to the associated effective property program. In the following example, the L and W properties are available to the effective property program and the LVS Reduce statement does not rely on L and W being specified in other property related statements.

```
lvs reduce parallel mos yes [
   effective w, l
   w = sum(w*l)    // w depends on l
   l = min(l)
]
```

When the program concludes, the value of each property is assigned to the single instance reduced from the input group.

For information on how Calibre LVS treats the values corresponding to the specified properties, refer to section “Data Sources”.

Effective Property Language Vector Functions

You use vector functions to evaluate an expression across all N instances in the input group. The resulting set of N expressions can be summed, multiplied, searched for a minimum, or searched for a maximum. You can count the number of instances in the input group with the COUNT() function.

The complete set of vector functions, and their syntax, is listed here:

```
SUM ( <vector_expression> )
PROD ( <vector_expression> )
MIN ( <vector_expression> )
MAX ( <vector_expression> )
COUNT()
```

where <vector_expression> is required by all vector functions, except COUNT(), and is the same as any other numeric expression except that it can only reference property names and simple numeric constants.

For example, if you declared the property name W in the EFFective statement, then the following assignment:

```
W = PROD ( W+1 )
```

shows a valid <vector_expression>. While the following assignment:

```
W = PROD ( W+sqrt(2) )
```

is not valid because “W+sqrt(2)” is not a valid <vector_expression>. It does not consist of only property names and simple numeric constants, but instead references another scalar function (sqrt()).

The following sections describe each vector function and give brief examples of their usage.

SUM (<vector_expression>)

Returns the sum of <vector_expression> applied to all N instances in the input group.

The following example:

```
P = SUM ( W*L )
```

returns:

```
W1*L1 + W2*L2 + ... + Wn*Ln
```

PROD (<vector_expression>)

Returns the product of <vector_expression> applied to all N instances in the input group.

The following example:

```
QQ = PROD ( W/L+1 )
```

returns:

```
(W1/L1+1) * (W2/L2+1) * ... * (Wn/Ln+1)
```

MIN (<vector_expression>)

Evaluates <vector_expression> across all N instances in the input group, and returns the minimum expression value found.

The following example:

```
L = MIN ( L )
```

returns the smallest L*i* found in the input group, where Li represents the value of L for the *i*th instance in the group).

MAX (<vector_expression>)

Evaluates <vector_expression> across all N instances in the input group, and returns the maximum expression value found.

The following example:

```
L = MAX ( L )
```

returns the largest L*i* found in the input group, where L*i* represents the value of L for the *i*th instance in the group.

COUNT()

Returns the number of instances present in the input group. This can be useful in calculating average values, as in the following example:

```
W = SUM(W) / COUNT()
```

Effective Property Language Numeric Functions

The following numeric built-in functions are provided:

ABS (<numeric_expression>)

Returns the absolute value of the specified numeric expression.

The following examples:

```
ABS(-1.5)
ABS(2.5)
```

return 1.5 and 2.5, respectively.

EXP (<numeric-expression>)

Returns the value of e (Napier’s constant, the base of natural logarithms) raised to the power of the numeric expression.

LOG (<numeric-expression>)

Returns the natural logarithm of the specified numeric expression. The value of the numeric expression must be greater than zero.

POW (<numeric-expression>, <numeric-expression>)

Returns the value of the first expression raised to the power of the second expression. If the first expression is zero, the second must be positive. If the first expression is negative, the second expression must be an integer.

SQRT (<numeric-expression>)

Returns the square root of the specified numeric expression. The value of the numeric expression must be zero or greater.

TRUNC (<numeric-expression>)

Returns the result of truncating the fractional part of the specified numeric expression. Truncation is toward zero. The absolute value of the numeric expression must not exceed 2,147,483,647 (2^32-1).

The following examples:

```
TRUNC(2.1)
TRUNC(2.9)
TRUNC(-2.1)
TRUNC(-2.9)
```

return 2, 2, -2, and -2, respectively, while:

```
TRUNC(9999999999)
```

returns UNKNOWN, because the argument exceeds 2147483647.

Further Effective Property Computation Examples

The following rule file examples illustrate effective property computations:

Example 1:

```
LVS REDUCE PARALLEL MOS YES [
                              // Reduce MOS devices in parallel
   effective W                // Calculate effective width value
   W = sum( W )               // … as sum of original widths,
]                             // … regardless of length
```

Example 2

```
LVS REDUCE PARALLEL MOS YES [
                              // Reduce MOS devices in parallel
   tolerance L 0              // … only if equal lengths
   effective W, L      // calculate effective width and length
   W = sum( W )           // Width is sum of original widths
   L = min( L )     //OK to use minimum since all are equal
]
```

Example 3

```
// This is equivalent to the default built-in effective
// property calculation for parallel MOS devices.
//
LVS REDUCE PARALLEL MOS YES [
                              // Reduce MOS devices in parallel.
   effective W, L, AS, AD, PS, PD
                           // Calculate these effective values.
   P = sum( W * L )                            // Sum of Wi * Li
   Q = sum( W / L )                            // Sum of Wi / Li
   W = sqrt( P * Q )                              // Effective W
   L = sqrt( P / Q )                              // Effective L
   AS = sum( AS )                  // Effective AS: sum of ASi
   AD = sum( AD )                  // Effective AD: sum of ADi
   PS = sum( PS )                  // Effective PS: sum of PSi
   PD = sum( PD )                  // Effective PD: sum of PDi
]
```

Example 4

```
// Reduce resistors in parallel only when all W values are
// equal. Effective W is equal to the original W (in a sense,
// W describes the type of resistor).
//
LVS REDUCE PARALLEL RESISTORS YES [
   tolerance W 0
   effective W
   W = min( W )      // OK to use minimum since all are equal
]
```

Example 5

```
LVS REDUCE PARALLEL RESISTORS YES [
                                    // Reduce parallel resistors
   tolerance W 0                       // … only if equal width;
   effective R, W
                    // calculate effective resistance and width
   R = 1 / sum( 1/R )     // same as the default calculation
   W = min( W ) //   //OK to use minimum since all are equal
]
```

Effective Property Computation Limitations

Property names used in the EFFECTIVE statement must be simple names; property-name/spice-parameter combinations such as “instpar(w)” are not accepted in effective property programs. Therefore, if a property string similar to instpar(w) is present on devices in the design, and an effective property program is provided for that device type, effective properties cannot be calculated during reduction.

Device Filtering

Calibre LVS allows you to filter out unused transistors during the LVS run. This could be done during a preliminary run, if you know that some devices should not be analyzed. The following sections describe how unused MOS and bipolar transistors are filtered.

You specify which unused devices are to be filtered with he LVS Filter Unused Option specification statement.

Filtering Unused MOS Transistors

Calibre LVS can internally filter unused MOS transistors from the source and layout circuits Filtering is performed only for properly formed MOS transistors; specifically, instances with component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, or LDD and have at least three pins with the standard names as specified in table 10-2.

The following configurations of transistors are filtered out:

1. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, and LDD transistors with floating source or drain pin.

2. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, and LDD transistors with source, drain and gate pins tied together.

3. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, and LDD transistors with floating gate pin and source and drain pins connected to a single power net.

4. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, and LDD transistors with floating gate pin and source and drain pins connected to a single ground net.

5. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, and LDD transistors with source shorted to drain, and gate pin connected to a power net.

6. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, and LDD transistors with source shorted to drain, and gate connected to a ground net.

7. MP and LDDP transistors with gate pin connected to a power net.

8. MN and LDDN transistors with gate pin connected to a ground net.

Figure 10-14 shows examples of the above filters, where “x” denotes a floating pin and “o” denotes a pin connected to other instances and/or ports.

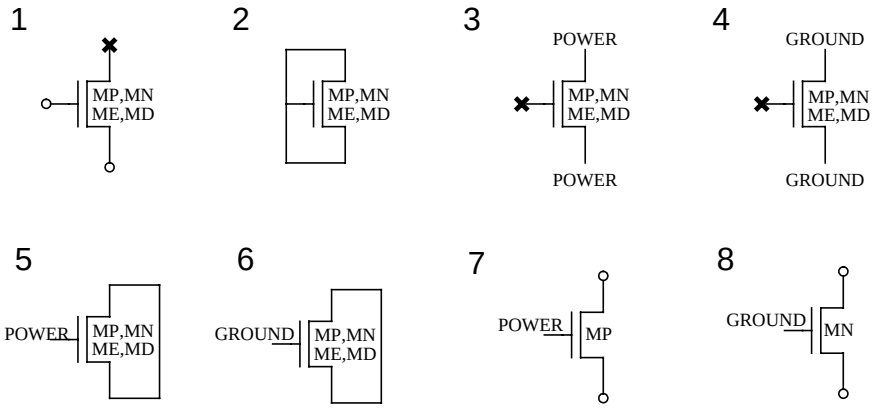


Figure 10-14. Unused MOS Transistors

The LVS Filter Unused MOS specification statement controls unused MOS transistor filtering.

Filtering Unused Bipolar Transistors

Calibre LVS can internally filter unused MOS transistors from the source and layout circuits. Filtering is performed only for properly formed MOS transistors; specifically, instances with component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, or LDD, and have at least three pins with the standard names as specified in table 10-2.

Calibre LVS can internally filter unused bipolar transistors from the source and layout circuits. Bipolar filtering is useful for verifying gate-array layouts. Filtering is performed only for properly formed bipolar transistors; specifically, instances with component type Q, and have at least three pins with the standard names as specified in table 10-6.

The following configurations of transistors are filtered out:

- Q transistors with base and emitter tied together.

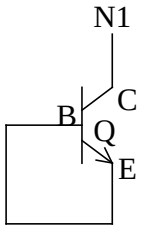


Figure 10-15. Unused Bipolar Transistor

The LVS Filter Unused Bipolar specification statement controls unused bipolar transistor filtering.

The LVS Filter specification statement is another useful rule file statement that enables instance filtering in the source or layout that meet certain criteria. Another useful statement is LVS Box, which specifies cells to be treated as “black boxes” during comparison.

Nets

This section discusses how various nets are handled during the LVS comparison phase.

Global Schematic Bulk Nets

This feature is implemented only for Eddm format, where it is possible to specify implied global bulk nodes for 3-pin MN and MP transistors.

This is done by adding the following parameters to the viewpoint:

- **nmos_bulk_node**: Specifies the name of the net to serve as the implied bulk node for 3-pin MN and LDDN transistors.

  LVS adds virtual bulk pins to all 3-pin MN transistors, and connects them to the specified net when **nmos_bulk_node** is specified.

- **pmos_bulk_node**: Specifies the name of the net to serve as the implied bulk node for 3-pin MP and LDDP transistors.

LVS adds virtual bulk pins to all 3-pin MP transistors, and connects them to the specified net when **pmos_bulk_node** is specified.

LVS names the bulk pin “B”. To participate in this process, transistors must have exactly three pins with standard pin names as specified in the section “Built-in Device Types”. LVS terminates with an error, if the specified net does not exist in the viewpoint.

Usage of Power and Ground Nets

LVS uses power and ground nets for Logic Gate Recognition, Filtering Unused MOS Transistors, and other applications. Several power nets and several ground nets are allowed in a single design. A net is a power net (or a ground net, respectively) if:

- The net name is listed in the LVS Power Name (or LVS Ground Name) specification statement.
- The net is connected to a port whose name is listed in the LVS Power Name (or LVS Ground Name) specification statement, and there is no other net in the same design that has this name. If there are several ports with the same power (or ground) name that are connected to different nets, only one of them is used.

Isolated Nets

Isolated nets are not connected to any instances, nor to any ports of the top-level cell on which LVS operates. Isolated layout nets and isolated source nets are ignored during comparison, unless they have user-given names that are present in both circuits. The LVS report lists isolated layout nets.

Pass-through Nets

Pass-through nets are connected to ports of the top-level cell on which LVS operates, but are not connected to any instance pins. Pass-through layout nets and pass-through source nets are ignored during comparison, unless the nets or their ports have user-given names that are present in both circuits. This filtering is done

because pass-through nets are often present in the layout, but are rarely present in a schematic design. The LVS report lists pass-through layout nets and their ports.

Logic Gate Recognition

LVS recognizes logic gates and semi-gates (pullup and pulldown structures and other series-parallel logic gates) in transistor-level circuits. LVS internally represents groups of transistors that form gates and semi-gates in each circuit by virtual gate instances, and comparison is done at this level. Results are reported either on the transistor or the gate level or both, whichever is appropriate.

- LVS forms regular CMOS gates from transistors with component type MP (pullup) and MN (pulldown).
- LVS forms LDD CMOS gates from transistors with component type LDDP (pullup) and LDDN (pulldown).
- LVS forms regular NMOS gates from transistors with component type MD (pullup) and ME (pulldown).
- LVS forms LDD NMOS gates from transistors with component type LDDD (pullup) and LDDE (pulldown).

Other series-parallel gates are formed from the above types as well as component types M and LDD. To see how component types are determined refer to the section “Component Types” above.

The LVS Recognize Gates specification statement specifies whether logic gate recognition should be performed. The secondary keyword SIMPLE prevents the formation of complete AOI and OAI gates and higher level series-parallel structures. In the case of AOI and OAI gates, LVS instead forms structures of type SUP, SDW, SPUP, and SPDW.

Logic gate recognition allows you to swap the order of logically equivalent pins and devices in transistor level implementations of logic circuits. For example, you can swap the two input pins of a NAND gate. The swappability is described with each gate type.

Only properly configured MOS transistors can form logic gates. Namely, they must have at least three pins with the standard pin names as specified in the section "Built-in Device Types" above. Other than this, there is no restriction on component subtype, number of optional pins, or optional pin names of the participating transistors. However, the number of pins and pin names must be identical for all transistors in a gate. By default, all transistors in a half-gate must have the same component subtype. The secondary keyword MIX SUBTYPES of the LVS Recognize Gates specification statement allows mixing of different transistor subtypes in the same half-gate. Half-gates are pullup or pulldown sections of logic gates, serial up and serial down structures, series-parallel up and series-parallel down structures, Sm* and SPm* structures, and so on. LVS verifies that subtypes correspond in layout and source, and that connections of substrate pins and other optional pins are the same in the layout and source.

Recognition Processes

- **Internal device and net matching**: LVS matches individual transistors in logic gates based on their "gate" pin connections (transistor pin name G), and all gate types are matched.

  LVS matches internal nets in logic gates based on their relative distance from the output pin or pins of the gate, and all gate types, except for complete AOI and OAI gates and higher-level series-parallel structures. Note that internal nets are matched in all gate types that are formed when the secondary keyword SIMPLE is specified.

- **Overriding pin and device swapping**: The physical order of connections to logic gate inputs, or of parallel device groups normally considered logically equivalent, can be checked in by specifying that logic gate recognition is not to be performed (set the LVS Recognize Gates to NONE).

- **Exceptions**: Listed below are various exceptions to the logic gate recognition process

  - MOS transistors with a subtype beginning with “X” or “x” and followed by at least one other character do not participate in the formation of logic gates, for example:

```
  DEVICE M(XP) gate poly sd sd well     // Does not form
                                          //logic gates.
DEVICE M(XABC) gate poly sd sd well     // Does not form
                                          //logic gates.
DEVICE M(X) gate poly sd sd well   // Forms logic gates.
```

    By using the XALSO secondary keyword in the LVS Recognize Gates specification statement, you can override this behavior and allow these devices to participate in the formation of logic gates.

  - A net connected to any pin other than a transistor’s source or drain, such as a substrate pin, is never made internal to a logic gate.

  - A net with a user-given name that appears in both layout and source, and a net connected to a substrate pin (or any other pin) is never made internal to a logic gate.

  - LVS does not form logic gates in cases where a choice must be made between two or more transistors that are equally qualified to participate in the gate. LVS forms only half-gates in these cases to prevent false discrepancies involving subtypes and property values.

    In Figure 10-16, LVS must make a choice between the mp(x) and mp(y) transistors. LVS will not form a complete NAND2 gate, but will form a SDW2 structure.

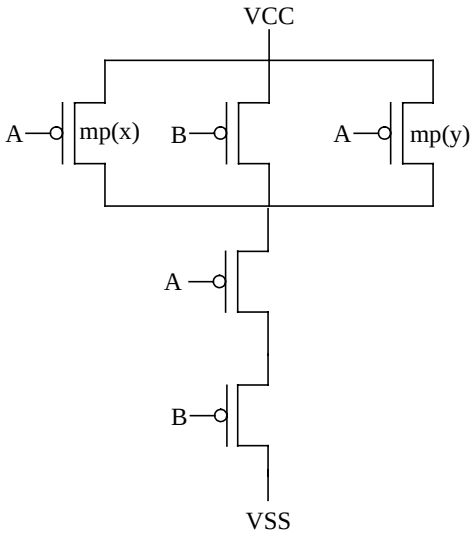


Figure 10-16. LVS Logic Gate Selection, Example

Regular CMOS Gates

- **INV**. CMOS inverter.

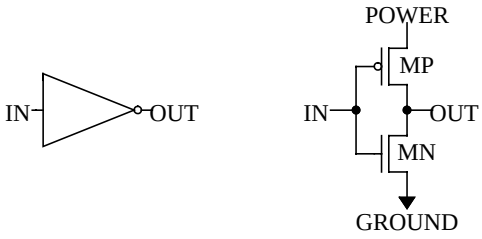


Figure 10-17. INV - CMOS inverter

- **NANDn**. *n*-input CMOS NAND.

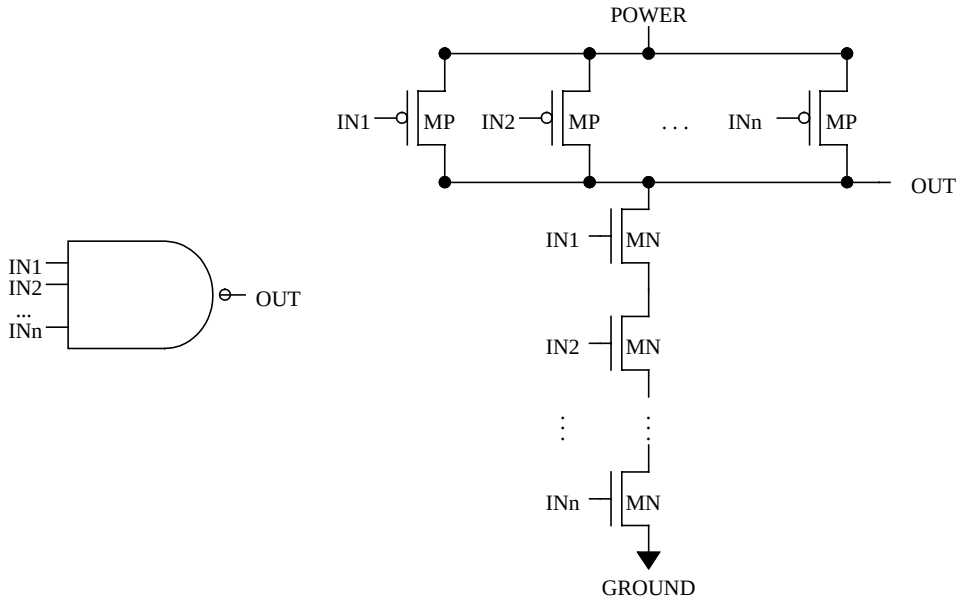


Figure 10-18. NANDn - n-input CMOS NAND

LVS considers all input pins of a NANDn gate logically equivalent. In figure 10-18, signals IN1, IN2, …, INn can all be interchanged.

- **NORn**. *n*-input CMOS NOR.

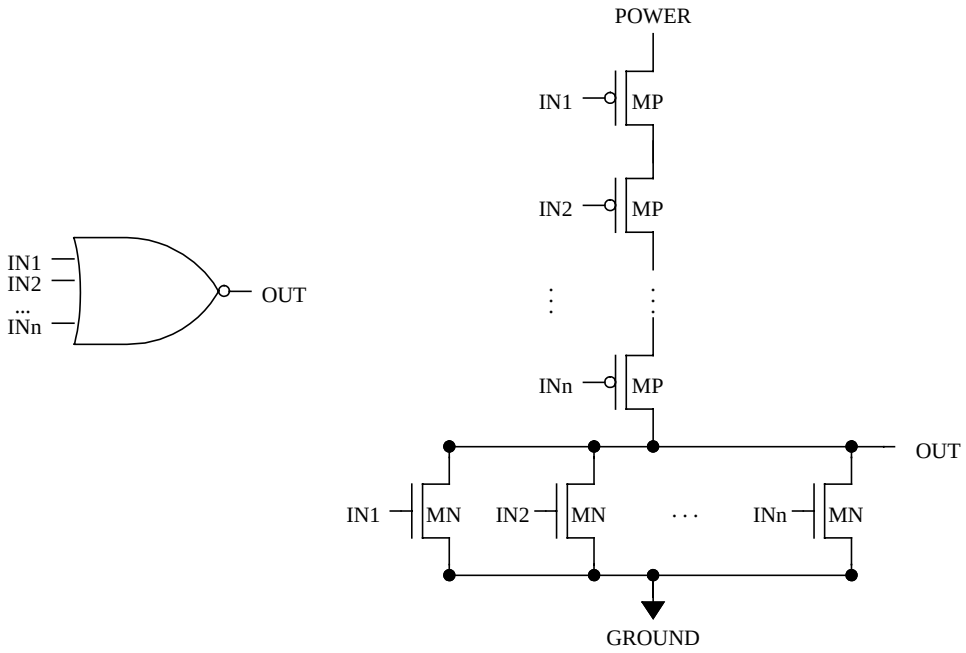


Figure 10-19. NORn - n-input CMOS NOR

LVS considers all input pins of a NORn gate logically equivalent. In figure 10-19, signals IN1, IN2, …, INn can all be interchanged.

- **AOI_n1_n2_…_nm**. CMOS and-or-invert consisting of *m* AND structures with *n1*, *n2*, …, *nm* inputs each, respectively, leading to an OR-invert structure.

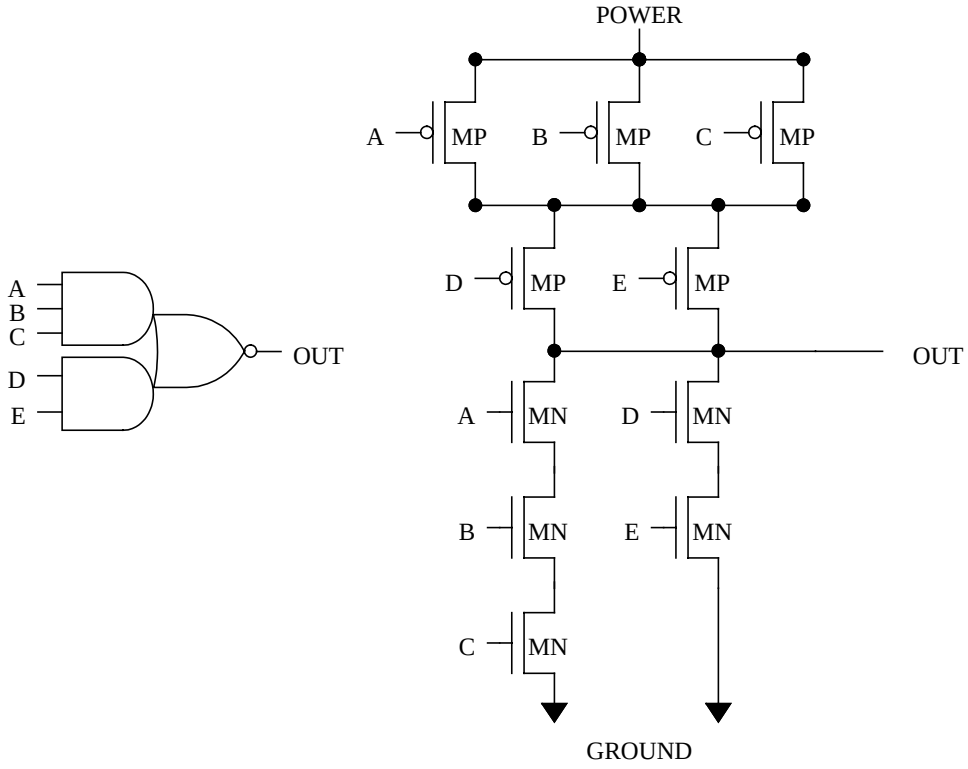


Figure 10-20. AOI_3_2 - CMOS and-or-invert

LVS considers the input pins of each one of the AND structures in an AOI gate logically equivalent. In figure 10-20, signals A, B and C may be interchanged. Signals D and E may also be interchanged.

The order of the parallel pullup groups in an AOI gate is interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name AOI_n1_n2_…_nm always has n1 >= n2 >= … >= nm. In the figure, the group of three parallel MP transistors connected to A, B, and C, respectively, could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

You can prevent the formation of AOI gates by including the LVS Recognize Gates SIMPLE statement in the rule file. In which case, LVS instead forms separate structures of type SPUP and SDW.

- **OAI_n1_n2_ … _nm**. CMOS or-and-invert consisting of *m* OR structures with *n1*, *n2*, … , *nm* inputs each, respectively, leading to an AND-invert structure.

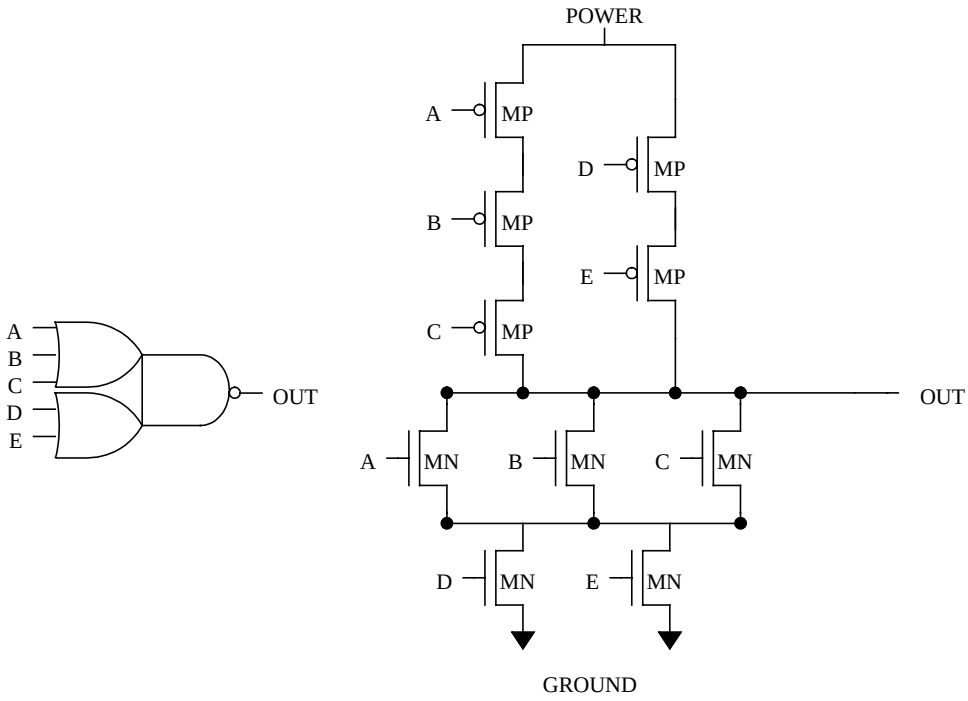


Figure 10-21. OAI_3_2 - CMOS or-and-invert

LVS considers the input pins of each one of the OR structures in an OAI gate logically equivalent. In figure 10-21, signals A, B, and C can be interchanged. Signals D and E can also be interchanged.

The order of the parallel pulldown groups in an AOI gate is interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name OAI_n1_n2_ … _nm always has n1 >= n2 >= … >= nm. In the figure, the group of three parallel MP transistors connected to A, B, and C, respectively, could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

You can prevent the formation of OAI gates by including the LVS Recognize Gates SIMPLE statement in the rule file. In which case, LVS instead forms separate structures of type SUP and SPDW.

- **SUPn** (serial up). *n*-input CMOS serial pullup.

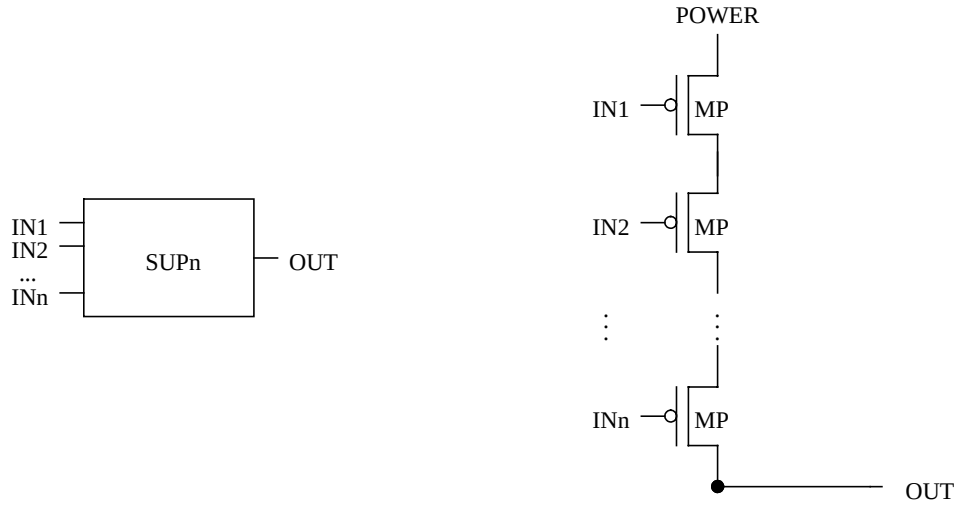


Figure 10-22. SUPn - n-input CMOS serial up

LVS considers all input pins of a SUPn gate logically equivalent. In Figure 10-22, signals IN1, IN2, …, INn can all be interchanged.

LVS represents pullups as stand-alone gates when they are not contained in complete gates, or when they are contained in OAI gates but complex gate recognition is turned off by including the LVS Recognize Gates SIMPLE statement in the rule file.

- **SDWn** (serial down). *n*-input CMOS serial pulldown.

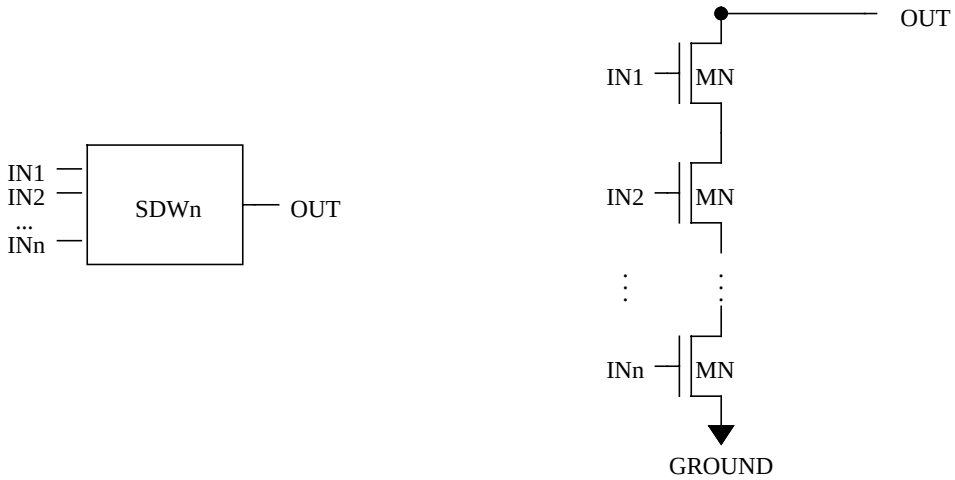


Figure 10-23. SDWn - n-input CMOS serial down

LVS considers all input pins of a SDWn gate logically equivalent. In Figure 10-23, signals IN1, IN2, …, INn can all be interchanged.

LVS represents serial pulldowns as stand-alone gates when they are not contained in complete gates, or when they are contained in AOI gates but complex gate recognition is turned off by including the LVS Recognize Gates SIMPLE statement in the rule file.

- **SPUP_n1_n2_…_nm** (serial-parallel up). CMOS serial-parallel pullup. This is a series of *m* parallel groups consisting of *n1*, *n2*, …, *nm* transistors, respectively, leading to a power net.

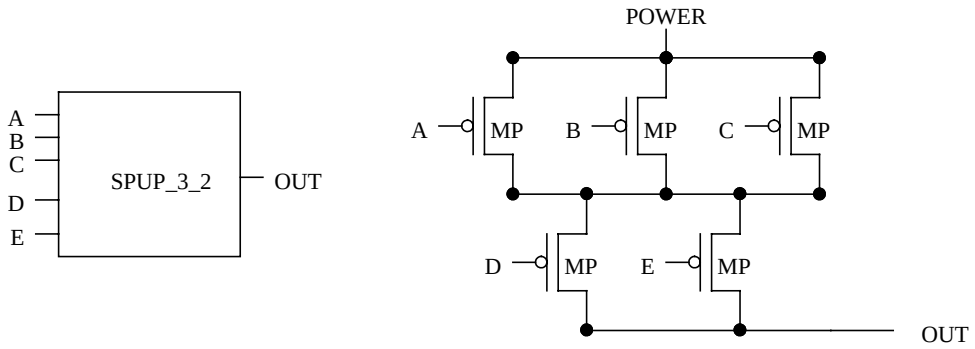


Figure 10-24. SPUP_3_2 - CMOS serial-parallel up

LVS considers the inputs to the transistors in each parallel group logically equivalent. In figure 10-24, signals A, B, and C can be interchanged, and signals D and E can be interchanged.

LVS represents SPUP structures as stand-alone gates when they are not contained in complete AOI gates, or when they are contained in AOI gates but complex gate recognition is turned off by including the LVS Recognize Gates SIMPLE statement in the rule file.

The order of the parallel groups in a SPUP gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name SPUP_n1_n2_ …_nm always has n1 >= n2 >= … >= nm. The group of three parallel MP transistors connected to A, B, and C, respectively could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

- **SPDW_n1_n2_…_nm** (serial-parallel down). CMOS serial-parallel pulldown. This is a series of *m* parallel groups consisting of *n1, n2, …, nm* transistors, respectively, leading to a ground net.

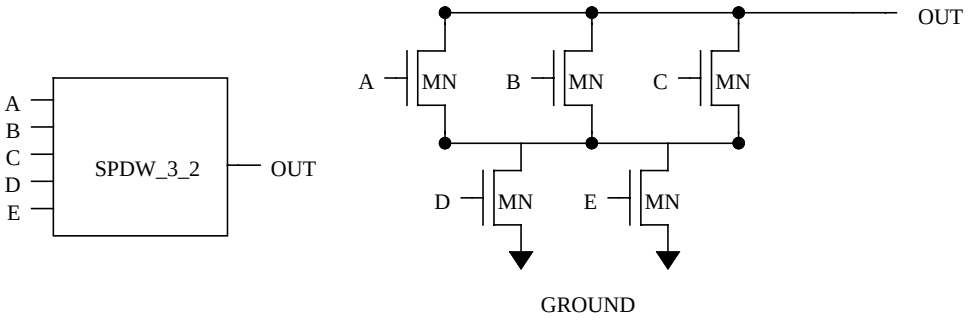


Figure 10-25. SPDW_3_2 - CMOS serial-parallel down

LVS considers the inputs to the transistors in each parallel group logically equivalent. In Figure 10-25, signals A, B, and C can be interchanged, and signals D and E can be interchanged.

LVS represents SPDW structures as stand-alone gates when they are not contained in complete OAI gates, or when they are contained in OAI gates but complex gate recognition is turned off by including the LVS Recognize Gates SIMPLE statement in the rule file.

The order of the parallel groups in a SPDW gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name SPDW_n1_n2_ …_nm always has n1 >= n2 >= … >= nm. In Figure 10-25, the could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

- **SMPn, SMNn, SMn.** Series of *n* MP, MN, or M devices.

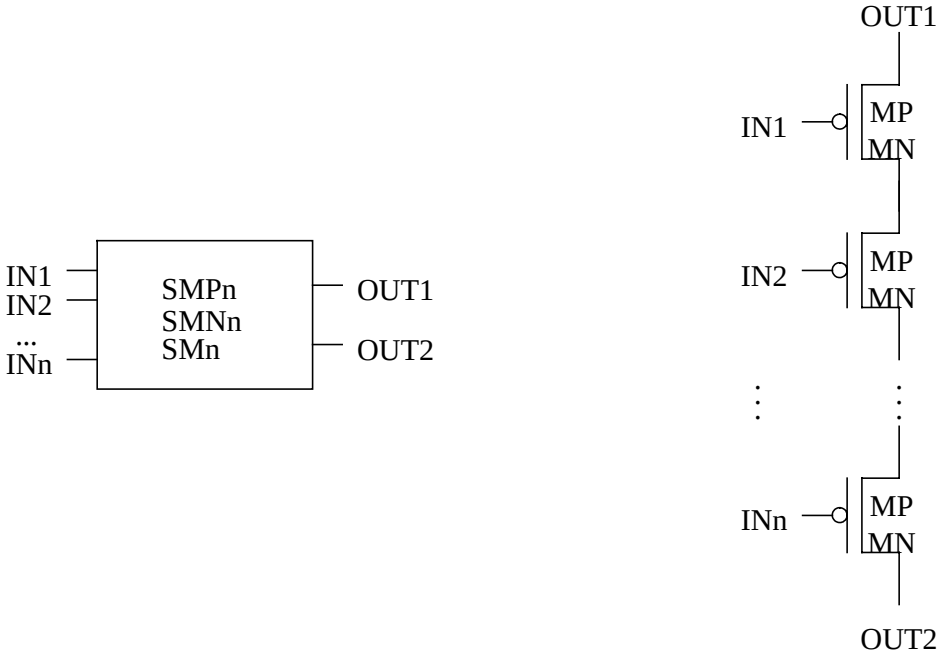


Figure 10-26. SMPn, SMNn, SMn - series of n MP, MN, or M devices

LVS considers all input pins of a SMPn, SMNn, or SMn gate logically equivalent, and the two output pins logically equivalent. In Figure 10-26, signals IN1, IN2, …, INn can all be interchanged, and signals OUT1 and OUT2 can be interchanged.

- **SPMP_n1_n2_…_nm, SPMN_n1_n2_…_nm, SPM_n1_n2_…_nm.** Serial-parallel structures of MP, MN, or M devices respectively. Each structure is a series of *m* parallel groups consisting of *n1, n2, …, nm* transistors, respectively, connected between any two nets (other then power in the case of MP, or ground in the case of MN).

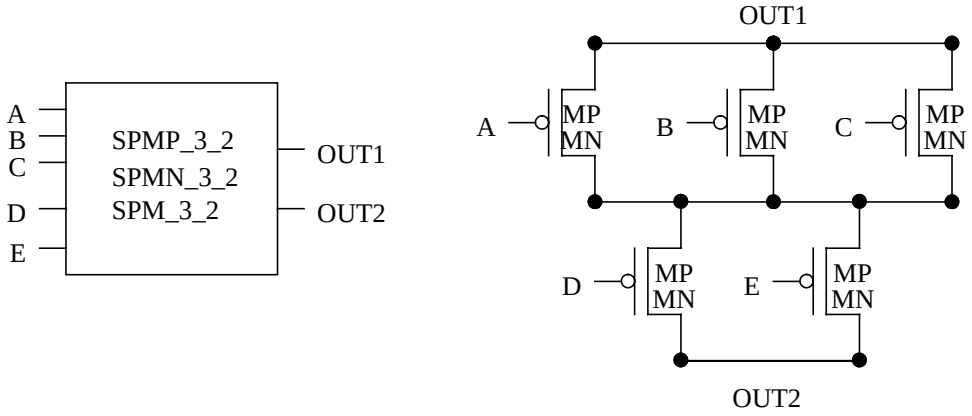


Figure 10-27. SPMP_3_2, SPMN_3_2, SPM_3_2 - CMOS series-parallel structure

LVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In Figure 10-27, signals A, B and C can be interchanged, signals D and E can be interchanged, and signals OUT1 and OUT2 can be interchanged.

The order of the parallel groups in a SPMP, SPMN, or SPM gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name suffixes "_n1_n2_ …_nm" always have n1 >= n2 >= … >= nm. The group of three parallel transistors connected to A, B, and C, respectively, could have been placed below the group of two parallel transistors connected to D and E, respectively.

- **SPMP(*expression*), SPMN(*expression*), SPM(*expression*).** High level serial-parallel structures consisting of MP, MN, or M devices, respectively. The series and parallel groups can be nested to an unlimited number of levels. The structure can be connected between any two nets.

The ***expression*** in parentheses describes the structure, and consists of a list of numbers, + and * operators, and parentheses. It has the following syntax:

- **+** Denotes connection in parallel.
- ***** Denotes connection is series.
- **(***expression***)** Denotes a substructure.
- *expression* ***** *n* Denotes a parallel group of *n* transistors, connected in series with the structure described by *expression.*
- *expression* **+** *n* Denotes a series of *n* transistors, connected in parallel with the structure described by *expression.*

The transistor gate pins form the input pins of the structure. The output pins of the structure are the two nets at the “top” and “bottom” of the structure. There can be any number of input pins but there are always exactly two output pins.

Inputs to transistors connected in parallel are logically equivalent, as are transistors connected in series. In general, the order of any group of substructures connected in parallel is interchangeable, and the order of any group of substructures connected in series is also interchangeable. The two outputs are also logically equivalent.

You can prevent the formation of high level serial-parallel structures by including the LVS Recognize Gates SIMPLE statement in the rule file.

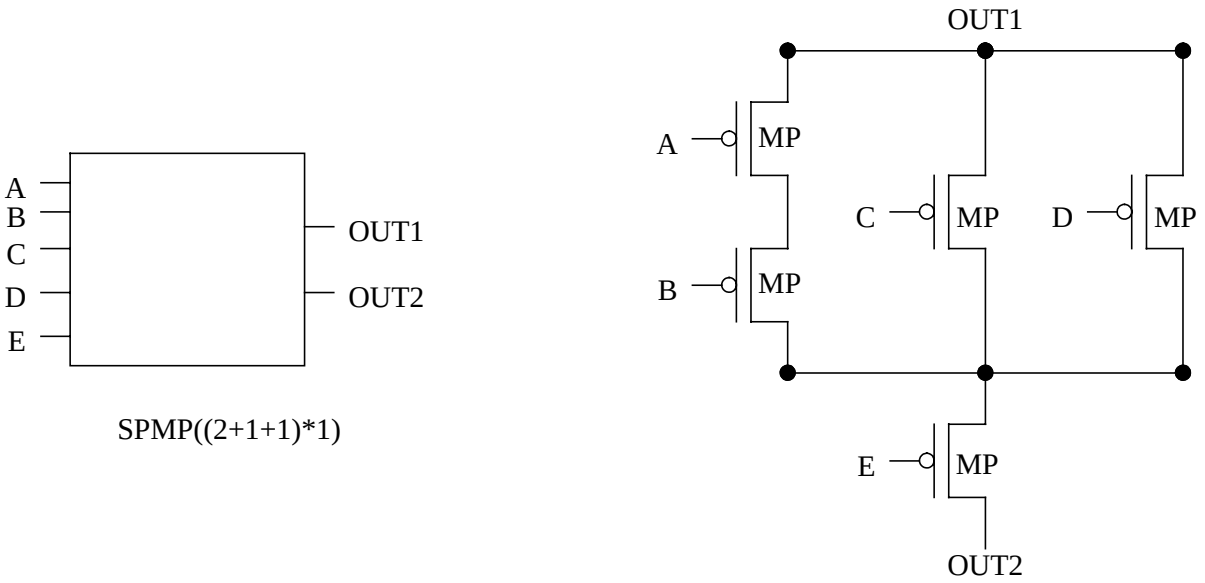


Figure 10-28. SPMP((2+1+1)*1) - CMOS High Level Series-Parallel Structure

In Figure 10-28, signals A and B can be interchanged, signals C and D can be interchanged, and signals OUT1 and OUT2 can be interchanged. The single transistor connected to E could be moved from the bottom to the top of the structure.

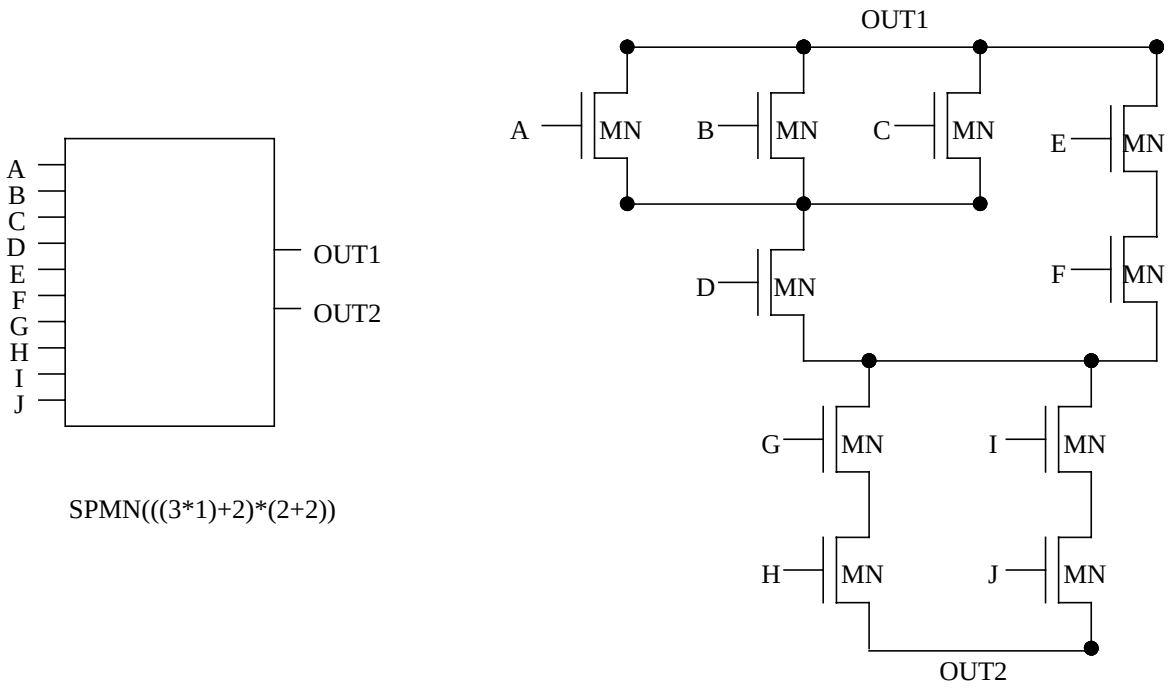


Figure 10-29. SPMN(((3*1)+2)*(2+2)) - CMOS High Level Series-Parallel Structure

In Figure 10-29, the following signals are interchangeable:

- A, B and C
- E and F
- G and H
- I and J
- The pair (G, H) and the pair (I, J)
- OUT1 and OUT2

In addition, transistor D could be placed above A, B and C, and the four transistors labeled G, H, I, and J could be moved from the bottom to the top of the structure.

Regular NMOS Gates

- **INV**. NMOS INVerter.

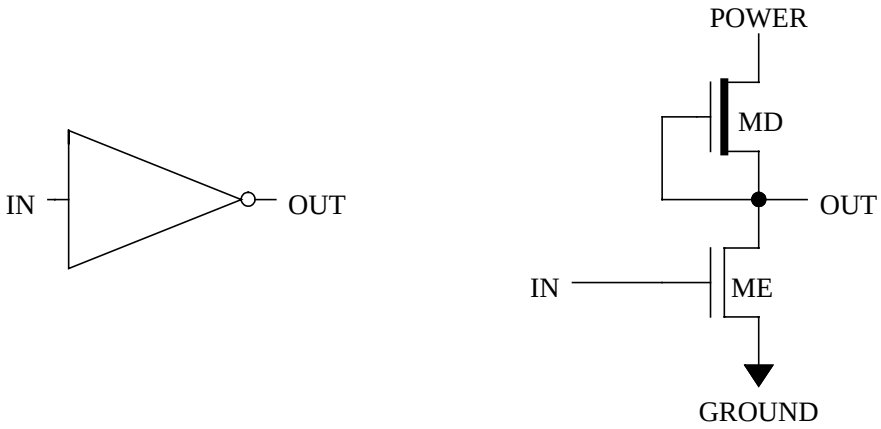


Figure 10-30. INV - NMOS inverter

- **NANDn**. *n*-input NMOS NAND.

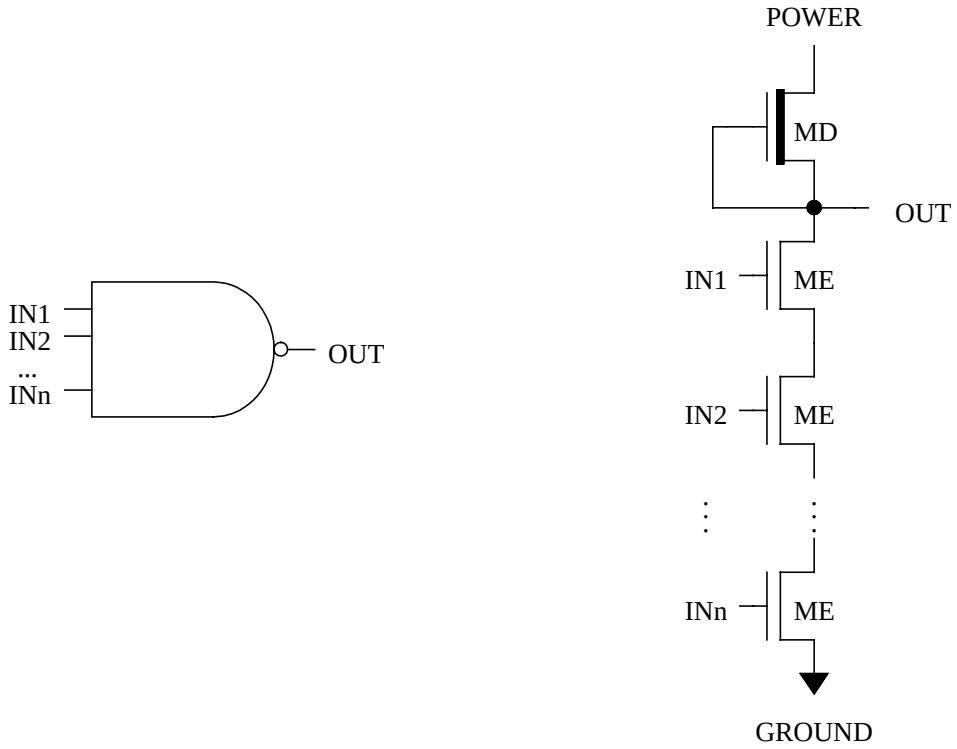


Figure 10-31. NANDn - n-input NMOS NAND

LVS considers all input pins of a NANDn gate logically equivalent. In Figure 10-31, signals IN1, IN2, …, INn can all be interchanged.

- **NORn**. *n*-input NMOS NOR.

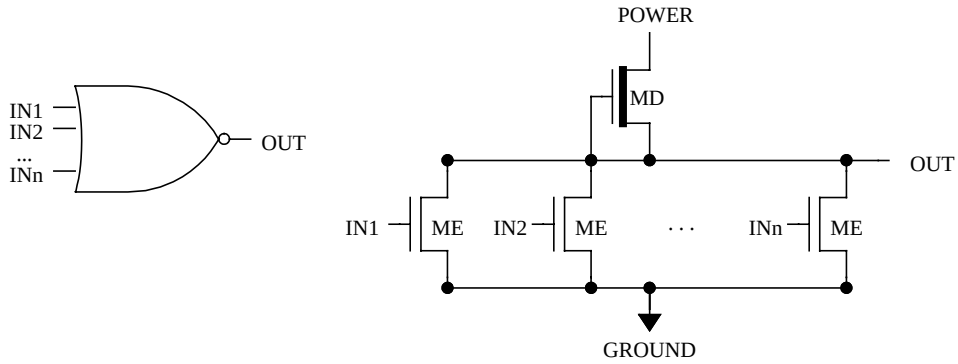


Figure 10-32. NORn - n-input NMOS NOR

LVS considers all input pins of a NORn gate logically equivalent. In Figure 10-32, signals IN1, IN2, …, INn can all be interchanged.

- **OAI_n1_n2_…_nm**. NMOS or-and-invert consisting of *m* OR structures with *n1*, *n2*, …, *nm* inputs each, respectively, leading to an AND-Invert structure.

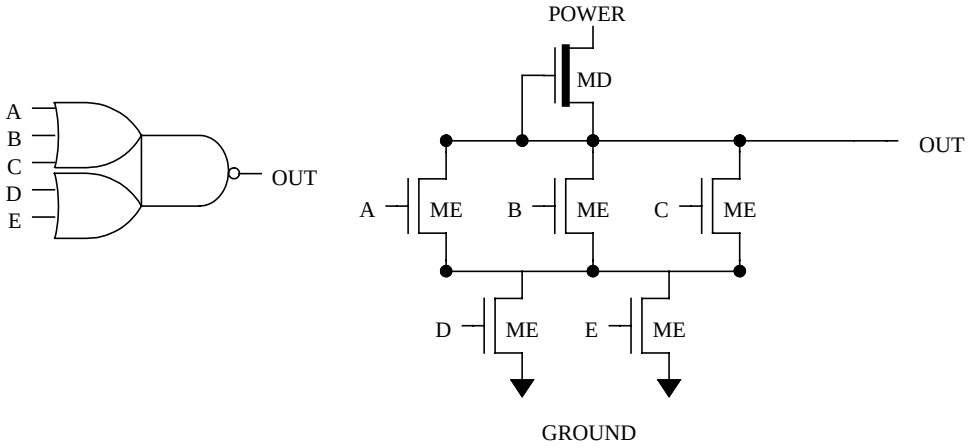


Figure 10-33. OAI_3_2 - NMOS OAI

LVS considers the input pins of each one of the OR structures in an OAI gate logically equivalent. In Figure 10-33, signals A, B, and C can be interchanged, and signals D and E can be interchanged.

The order of the parallel pulldown groups in an AOI gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name OAI_n1_n2_ …_nm always has n1 >= n2 >= … >= nm. In Figure 10-33, the group of three parallel ME transistors connected to A, B, and C, respectively, could have been placed below the group of two parallel ME transistors connected to D and E, respectively.

You can prevent the formation of OAI gates by including the LVS Recognize Gates SIMPLE statement in the rule file. In which case, LVS instead forms structures of type SPDW from the pulldown part of the gate.

- **SDWn** (serial down). *n*-input NMOS serial pulldown.

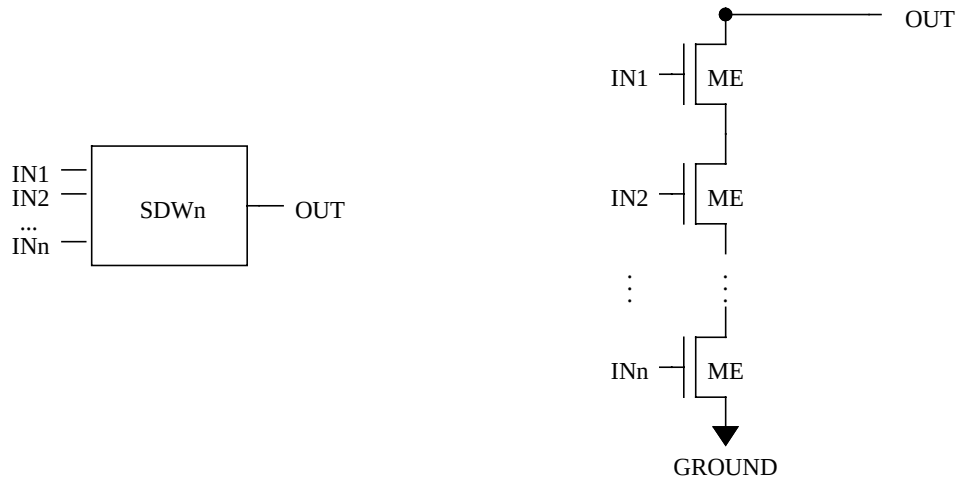


Figure 10-34. SDWn - n-input NMOS serial-down

LVS considers all input pins of a SDWn gate logically equivalent. In figure 10-34, signals IN1, IN2, …, INn may all be interchanged.

LVS represents NMOS serial pulldowns as standalone gates when they are not contained in complete gates.

SPDW_n1_n2_…_nm (serial-parallel down). NMOS serial-parallel pulldown structure. This is a series of *m* parallel groups consisting of *n1, n2, …, nm* transistors, respectively, leading to a ground net.

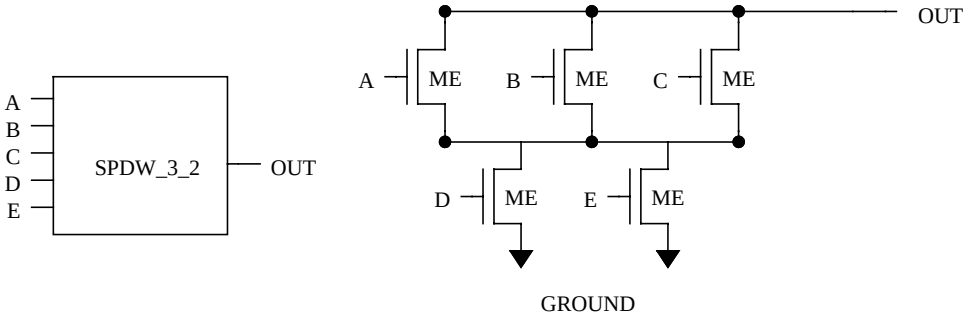


Figure 10-35. SPDW_3_2 - NMOS serial-parallel down

LVS considers the inputs to the transistors in each parallel group logically equivalent. In Figure 10-35, signals A, B, and C can be interchanged, and signals D and E can be interchanged.

LVS represents SPDW structures as standalone gates when they are not contained in complete OAI gates, or when they are contained in OAI gates but complex gate recognition is turned off by including the LVS Recognize Gates Simple statement in the rule file.

The order of the parallel groups in a SPDW gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name SPDW_n1_n2_ …_nm always has n1 >= n2 >= … >= nm. In Figure 10-35, the group of three parallel ME transistors connected to A, B, and C, respectively, could have been placed below the group of two parallel ME transistors connected to D and E, respectively.

- **SMDn**, **SMEn** - series of n MD or ME devices.

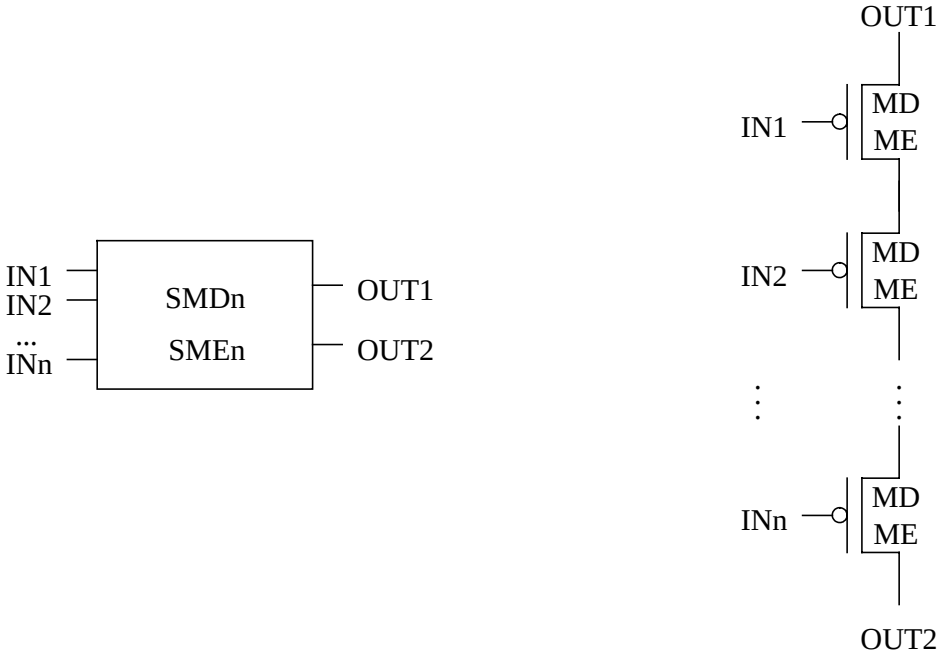


Figure 10-36. SMDn, SMEn - series of n MD or ME devices

LVS considers all input pins of a SMDn or SMEn gate logically equivalent, and the two output pins logically equivalent. In Figure 10-36, signals IN1, IN2, …, INn can all be interchanged, and signals OUT1 and OUT2 can be interchanged.

- **SPMD_n1_n2_…_nm**, **SPME_n1_n2_…_nm** - Serial-parallel structures of MD or ME devices, respectively. Each structure is a series of *m* parallel groups consisting of *n1, n2, …, nm* transistors, respectively, connected between any two nets (other then power in the case of MD, or ground in the case of ME).

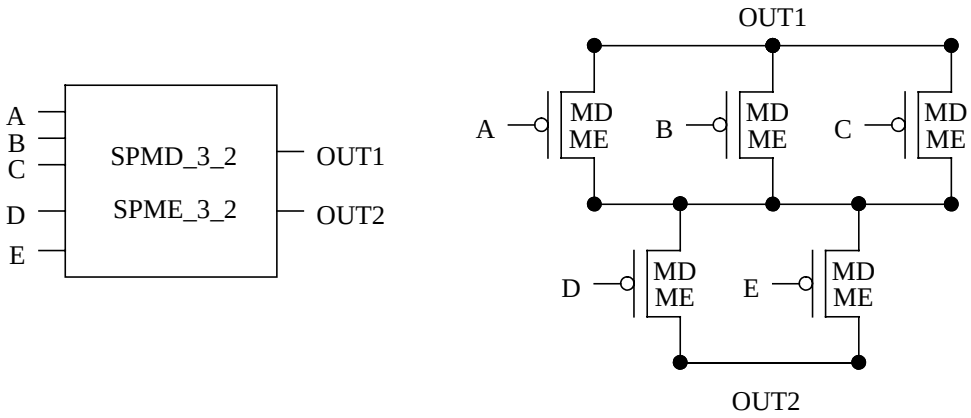


Figure 10-37. SPMD_3_2, SPME_3_2 - NMOS series-parallel structure

LVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In Figure 10-37, signals A, B, and C can be interchanged, signals D and E can be interchanged, and signals OUT1 and OUT2 can be interchanged.

The order of the parallel groups in a SPMD or SPME gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The names SPMD_n1_n2_ …_nm and SPME_n1_n2_ …_nm always have n1 >= n2 >= … >= nm. In Figure 10-37, the group of three parallel transistors connected to A, B and C respectively may have been placed below the group of two parallel transistors connected to D and E, respectively.

- **SPME(*expression*), SPMD(*expression*)-** High level serial-parallel structure consisting of ME or MD devices respectively. The series and parallel groups can be nested to an unlimited number of levels. For more details, refer to the description of SPMP(*expression*) and SPMN(*expression*) on page 10-81.

LDD Gates

LDD devices are MOS transistors with non-swappable source and drain pins. Recall that there are five types of LDD devices: LDDN, LDDP, LDDE, LDDD,

and LDD corresponding to the four regular MOS transistor types: MN, MP, ME, MD, and M.

LDD transistors form logic gates in the way the corresponding regular MOS transistors form gates with some extra conditions listed below. CMOS style gates are formed with LDDN and LDDP, which replace MN and MP in regular gates. NMOS style gates are formed with LDDE and LDDD, which replace ME and MD in regular gates. Other series-parallel gates are formed from the above mentioned types as well as component type LDD. Preceding sections describe regular gates in detail.

LDD Voltage Gates

Voltage gates are gates that require power and/or ground nets. These gates include:

INV	NANDn
NORn	AOI_n1_n2_…_nm
OAI_n1_n2_…nm	SUPn
SDWn	SPUP_n1_n2_
SPDW_n1_n2_	

The following conditions apply when forming LDD voltage gates:

1. Each parallel group of LDD-type transistors must have their source pins connected to the same net, and their drain pins connected to the same net.
2. The net connecting one parallel group to the next must be connected to the source pins of one group, and the drain pins of the other.
3. The power net must be connected to the source pins of the LDDP or LDDD transistors.
4. The ground net must be connected to the source pins of the LDDN or LDDE transistors.

5. The output of an LDD voltage gate is called *output(d)* while the output of a regular MOS gate is called *output*. The (*d)* means that the output net is connected to drain pins of LDD transistors.

Figure 10-38 shows a CMOS style AOI_3_2 gate made with LDD-type transistors.

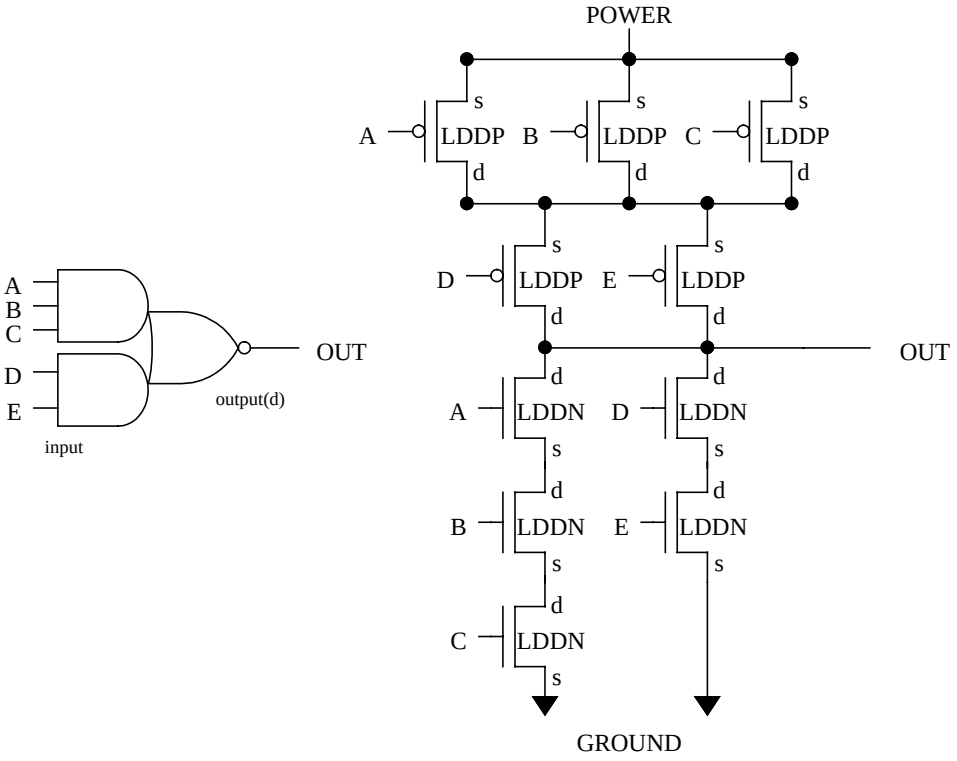


Figure 10-38. LDD AOI_3_2 gate

LDD Non-Voltage Gates

Non-voltage gates are gates that do not require power or ground nets. These gates include:

SLDDPn	SLDDNn
SLDDn	SPLDDP_n1_n2_…_nm
SPLDDN_n1_n2_…_nm	SPLDD_n1_n2_…_nm

SPLDDP(expression)	SPLDDN(expression)
SPLDD(expression)	SLDDDn
SLDDEn	SPLDDD_n1_n2_…_nm
SPLDDE_n1_n2_…_nm	SPLDDD(expression
SPLDDE(expression)	

The following conditions apply when forming LDD non-voltage gates.

1. Each parallel group of LDD-type transistors must have their source pins connected to the same net, and their drain pins connected to the same net.

2. The net connecting one parallel group to the next must be connected to the source pins of one group, and the drain pins of the other.

3. The letters LDD replace the letter M in the gate name. For example, SLDDPn corresponds to SMPn, and SPLDDE(*expression*) corresponds to SPME(*expression*).

4. A LDD-type non-voltage gate has two non-swappable outputs. One is connected only to source pins and is called *output(s)*. The other is connected only to drain pins and is called *output(d)*. Recall that the outputs of a regular MOS gate are both called *output*.

Figure 10-39 shows a SLDDP3 gate.

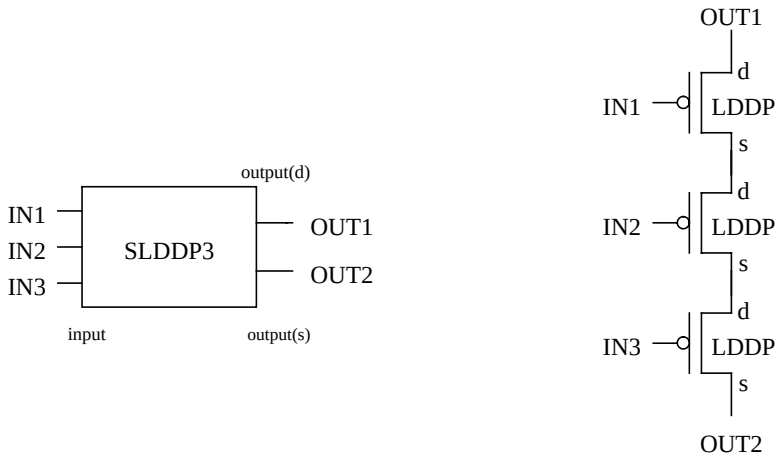


Figure 10-39. SLDDP3 gate

Mixed Gates

LDD-type and regular MOS transistors form mixed non-voltage gates, such as gates that do not require power or ground nets. Mixed gates are composed of a S or SP gate of one type in series with a unique single transistor of the other type. The mixed gate is formed if and only if there is a *unique* available single transistor of the other type.

One regular and one LDD-type transistor in series also form a mixed gate.

The LDD-type transistors and the regular MOS transistors must be of the same kind (D, E, N, P, or generic) to form a mixed gate.

A mixed gate has two outputs, one connected to regular MOS transistors is called *output*, and the other is called *output(d)* if it is connected to drain pins of LDD-type transistors or *output(s)* if it is connected to source pins of LDD-type transistors.

A mixed gate contains the name of both transistor types that compose it, the single transistor first, for example: SPMN-LDDN(D)_3_1. The (D) means that the output net connected to the LDD transistors is connected to drain pins. This gate is shown in figure 10-40.

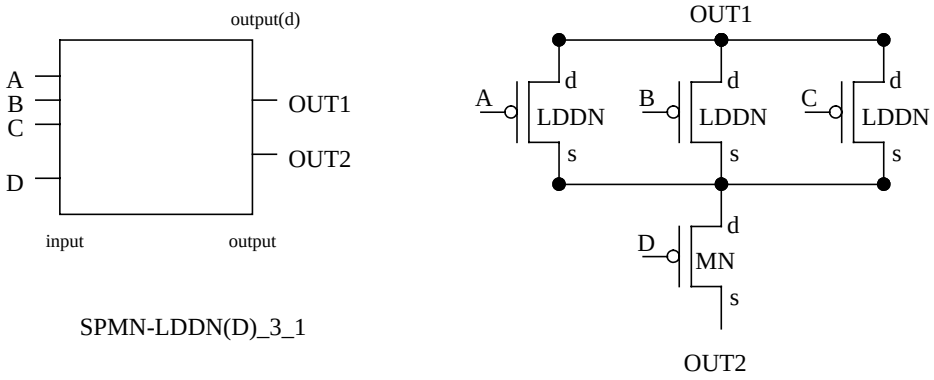


Figure 10-40. SPMN-LDDN(D)_3_1 mixed gate

Excluding Transistors

MOS transistors do not form logic gates if their component subtype begins with the characters X or x followed by at least one other character (the shorthand for this situation is X+). Note that if the subtype consists of only one character, such as X, then the transistor *will* form logic gates.

You can use these transistors to prevent pin swapping of logic gate inputs or to verify the order of individual transistors in series/parallel structures. In other respects, these transistors behave as usual; for example, they are subject to LVS Reduce Parallel MOS, LVS Reduce Series MOS, LVS Reduce Split Gates, and similar statements.

Rule file example:

```
DEVICE M(XP) gate poly sd sd well
                    // Does not form logic gates.
DEVICE M(XABC) gate poly sd sd well
                    // Does not form logic gates.
DEVICE M(X) gate poly sd sd well
                    // Forms logic gates.
```

Spice netlist example:

```
M1 1 2 3 4 XP       $ Does not form logic gates.
M2 1 2 3 4 XABC     $ Does not form logic gates.
M3 1 2 3 4 X        $ Forms logic gates.
```

The optional keyword XALSO in the LVS Recognize Gates specification statement disables the special treatment of X+ subtypes in logic gate recognition. It instructs LVS to form logic gates from MOS devices even if their subtype begins with the characters X or x (followed by at least one character).

Overriding Of Pin Swapping In Logic Gates

The physical order of connections to logic gate inputs which are normally considered logically equivalent can be checked in either of the following ways.

- Specify that logic gate recognition is not to be performed by specifying LVS Recognize Gates NO in your rule file.
- Use component subtypes X+ as described in the "Excluding Transistors" section.

Overriding Of Device Swapping In Logic Gates

The physical order of parallel device groups which are normally interchangeable in logic gates can be checked in any of the following ways.

- Specify that logic gate recognition is not to be performed by specifying LVS Recognize Gates NO in your rule file.
- Use component subtypes X+ as described in the "Excluding Transistors" section.

Pin Swapping

Logically Equivalent Pins

LVS allows the order of connections to logically equivalent pins of layout instances to differ from the order of connections to the corresponding pins of the corresponding source instances. This is sometimes referred to as pin swapping.

LVS forms equivalence of input pins in logic gates. Refer to section "Logic Gate Recognition" below for a special case of this capability. For other component types, there are several methods for designating logically equivalent pins.

Pins that have identical names are always considered logically equivalent by LVS.

Default Pin Swapping for Devices

- LVS considers source (S) and drain (D) pins of regular MOS transistors (component types MN, MP, ME, MD, and M) logically equivalent.
- LVS considers POS and NEG pins of all capacitor devices (component type C) logically equivalent if the value of the LVS All Capacitor Pins Swappable specification statement is YES. Otherwise, LVS applies the rules described in this section.
- LVS considers pins of all resistor devices (component type R) logically equivalent.

Rule File Pin Swap Lists

LVS designates pins of extracted layout devices as logically equivalent by placing the pin names in a single *pin_swap* list in a Device operation. For example:

```
DEV C cap1 poly m1 (POS NEG) //swappable POS, NEG
```

In addition, pins on a single layer in the Device statement are always swappable.

```
DEV FOO seed m1(p1) m1(p2) poly(p3) //swappable p1, p2
```

Affected Database Systems. LVS utilizes pin swappability information from rule file Device operations during circuit comparison, for most input database systems (both source and layout). This is described below.

The following database systems all take pin swappability from Device operations in the current rule file. (The current rule file is the rule file used for the particular LVS execution):

- Calibre geometrical database systems (for example, GDSII, CIF, ACSII and BINARY).
- SPICE
- EDDM
- CNET databases derived from SPICE or EDDM.

CNET databases derived from Calibre geometrical database systems do not take pin swappability from Device operations in the current rule file. Instead, those CNET databases preserve pin swappability from Device operations in the original rule file that was used to create them.

There is an exception. If the layout is a database system with inherent pin swappability information (specifically: Calibre geometrical database systems), and the source is a database system with no pin swappability information (specifically: SPICE, EDDM, or CNET derived from SPICE or EDDM), then source components take pin swappability from corresponding layout components. This is done based on component type, component subtype, pin number and pin names.

Application. Pin swappability from rule file Device operations proceeds as follows.

For every primitive instance during circuit comparison:

1. LVS looks in the rule file for a Device operation with the same component type (element name), the same component subtype (model name), the same number of pins and the same pin names as in the instance. (If the instance has no component subtype (model name), then LVS looks for a Device

operation with no model name). If such a Device operation exists then pin swappability information from the Device operation is applied to the instance.

2. Otherwise, LVS looks in the rule file for a Device operation with the same component type (element name), the same number of pins and the same pin names as in the instance, and no model name. If such a Device operation exists then pin swappability information from the Device operation is applied to the instance.

3. Otherwise, if the instance has no component subtype (model name), then LVS examines all Device operations in the rule file that have the same component type (element name), the same number of pins and the same pin names as in the instance, regardless of model name. If there is at least one such Device operation, and all such Device operations have the same pin swappability, then that pin swappability is applied to the instance.

4. Otherwise, swappability from rule file Device operations is not applied to the instance.

As mentioned, this process is performed for primitive instances. It is not performed for non-primitive instances. In flat circuit comparison, all instances are primitive. In hierarchical circuit comparison, instances are primitive if they are represented in the input netlist with standard Spice device element statements (such as M, C, R, and so forth), or if they are represented with primitive subcircuit calls (see “Subcircuits” on page 11-33 for more information), or if they are instances of LVS Box cells. Instances of regular (non-empty) hcells in hierarchical circuit comparison are not primitive and thus do not inherit pin swappability from Device operations.

Pin swappability in Device operations can be indicated explicitly by listing swappable pins in parentheses, as in (s d), or implicitly (pins on the same layer are swappable).

Notes:

- The above steps are performed *after* any pins have been discarded using the LVS Discard Pins By Device specification statement, if present.

- In the above steps, all name comparisons are case insensitive. In all examples below, assume that the rule file contains no other Device operations.

Example 1

Assume that in the rule file you specify 5-pin MOS devices as user-defined devices as follows:

```
DEVICE mos5pin gate gate(G) psd(S) psd(D) well1(B)
  well2(B2)
```

You compare Spice to Spice and you enter 5-pin MOS devices in the Spice netlist with primitive subcircuit calls like this:

```
.subckt mos5pin D G S B B2
.ends
x1 1 2 3 4 5 mos5pin
```

The rule file Device mos5pin definition will be applied to instances of mos5pin in both layout and source. LVS will determine that pins S and D of mos5pin are swappable (because they are on the same layer) and will use this information in circuit comparison.

Example 2

Rule file:

```
DEVICE C(A) cap1 poly m1 (POS NEG) // C with model A
DEVICE C(B) cap2 m1 m2 // C with model B
```

Spice:

```
C1 1 2 $[A] $$ C with model A
C2 3 4 $[B] $$ C with model B
```

C1 receives pin swappability from the DEVICE C(A) operation according to step (1) above. Its pins are swappable. C2 receives pin swappability from the Device C(B) operation according to step (1) above. Its pins are not swappable (unless indicated swappable by other means).

Example 3

Rule file:
```
DEVICE C cap poly m1 (POS NEG) // C with no model
```

Spice:
```
C1 1 2 $$ C with no model
```

C1 receives pin swappability from the Device operation according to step (1) above. Its pins are swappable.

Example 4

Rule file:
```
DEVICE C cap poly m1 (POS NEG) // C with no model
```

Spice:
```
C1 1 2 $[A] $$ C with model A
```

C1 receives pin swappability from the Device operation according to step (2) above. Its pins are swappable.

Example 5

Rule file:
```
DEVICE C(A) cap1 poly m1 // C with model A
DEVICE C    cap2 m1 m2 (POS NEG) // C with no model
```

Spice:
```
C1 1 2 $[A] $$ C with model A
C2 3 4 $[B] $$ C with model B
C3 5 6 $$ C with no model
```

C1 receives pin swappability from the DEVICE C(A) operation according to step (1) above. Its pins are not swappable (unless indicated swappable by other means). C2 receives pin swappability from the DEVICE C operation according to step (2) above. Its pins are swappable. C3 receives pin swappability from the DEVICE C operation according to step (1) above. Its pins are swappable.

Example 6

Rule file:

```
DEVICE C(A) cap1 poly m1 (POS NEG) // C with model A
DEVICE C(B) cap2 m1 m2 (POS NEG) // C with model B
```

Spice:

```
C1 1 2 $$ C with no model
```

Since both DEVICE C(A) and DEVICE C(B) operations indicate the same pin swappability, C1 receives pin swappability from those Device operations according to step (3) above. Its pins are swappable.

Example 7

Rule file:

```
DEVICE C(A) cap1 poly m1 (POS NEG) // C with model A
DEVICE C(B) cap2 m1 m2 // C with model B
```

Spice:

```
C1 1 2 $$ C with no model
```

Since the DEVICE C(A) and DEVICE C(B) operations indicate different pin swappability, C1 does not receive pin swappability from either Device operation. Its pins are not swappable (unless indicated swappable by other means).

Spice as Layout System

When a Spice netlist represents the layout, LVS utilizes pin swappability information from a Device operation during circuit comparison. This occurs when the specification statement Layout System SPICE is indicated, or when you execute:

```
calibre -spice … -lvs …
```

This command line entry uses Spice as intermediate representation for the layout, and the source can be of any type. This is useful when user-defined devices with swappable pins are entered as primitive subcircuits in Spice netlists.

Hcell Pins

Generally, there is no logical equivalence between hcell pins. (Hcells are hierarchically corresponding cells in LVS-H). There are two exceptions to this rule, namely, trivial pin swappability (see “Trivial Pin Swappability” on page 13-10) and pin swapping in memory cells and containing blocks (see “SRAM Bit-Cell Recognition” on page 13-11). Refer to those sections for more information.

Tracing Properties

LVS compares (traces*)* the values of selected properties on layout instances to the values of corresponding properties on corresponding source instances. Discrepancies are reported when these values are different. The rules that control which properties are traced and how the comparison is done are specified in the rule file with the Trace Property specification statement.These rules are sometimes referred to as “trace property rules.”

Built-in Property Classification

The LVS circuit comparison module recognizes certain property names as built-in properties, for the purpose of device reduction and for other processing. For example, LVS computes effective values for built-in properties when it reduces devices in series and parallel. See “Device Reduction”.

Properties are classified based on their names; specifically, W, L, AS, AD, PS, PD, C, R, A, P denote width, length, area of source, area of drain, perimeter of source, perimeter of drain, capacitance, resistance, area and perimeter respectively. This convention is used in the layout as well as in the source. However, Trace Property specification statements functions may override this built-in naming convention and may specify different property names in the source.For example, the statement:

```
TRACE PROPERTY MP(X) WIDTH W 0
```

implies that for elements of type MP(X), the width property in the source is “WIDTH”.

Reading Built-in W/L Partner Properties

LVS automatically reads from the input database L properties if they are required for the calculation of effective W values during device reduction. L properties are read if required, even when they are not traced themselves. Specifically, for a particular device type and optional subtype, L properties are automatically read from the input database if the following conditions are all satisfied:

- relevant device reduction is requested for that device type and subtype
- L is required to calculate effective W for the device type and subtype
- W appears in Trace Property, reduction TOLERANCE, or similar statements for the device type and subtype

If these conditions are satisfied then L properties are read from the input database even if L itself does not appear in any Trace Property or similar statements. Similarly, LVS automatically reads W properties if they are required for the calculation of effective L values.

W and L are sometimes called “partner” properties because one is often required to calculate effective values for the other during device reduction.

Automatic reading of partner L/W properties is triggered only when one property is in fact required to calculate effective values for the other. For example, if the user specifies their own formula for calculation of effective W, which does not require L, then L is not automatically read.

If any of the requested properties or their required partners are not present in the input database, then missing property discrepancies are reported in the “Source Errors” or “Layout Errors” sections of the LVS report (unless disabled with LVS Report Option E.)

Example:

```
LVS REDUCE PARALLEL MOS YES
TRACE PROPERTY MP W W 0   // trace W only; L is not mentioned
```

In this example, W is the only property traced for MP devices. However, since MP devices are being reduced in parallel and property L is required to calculate effective values for W, both W and L will be read from the input database for MP devices.

Comparing Device Counts After Reduction

The “M” property represents a multiplier factor and can be traced in Spice netlists. It is available for tracing in all built-in Spice elements (R, C, L, D, Q, J, M, V) as well as in primitive subcircuit calls (X calls referencing empty subcircuits) and LVS Box subcircuit calls (X calls referencing subcircuits that are designated as LVS Box elements). For built-in Spice elements, the M property is equal to the value of the M parameter if one is specified in the Spice element; otherwise, the M property defaults to 1. For primitive and LVS Box subcircuit calls, the M property is always equal to 1. (When you specify an M parameter in a subcircuit call, you get M individual subcircuit calls connected in parallel, each with property M equal to 1). For other subcircuit calls the M property not available for tracing.

With this factor, you can keep track of device counts during device reduction and you can compare those counts in layout and source. For each pair of post-reduction device instances, you can compare the number of original devices in the layout versus the number of original devices in the source that participated in the formation of the particular pair. This is shown below:

```
DEVICE MP PGATE PGATE PSD PSD NWELL [
    PROPERTY M         // Define a M property for the device.
    M=1                // Set M to constant 1.
]
LVS REDUCE MP PARALLEL [
    EFFECTIVE M
    M=SUM(M)   // Add up M values during parallel reduction.
]
TRACE PROPERTY MP M M 0 // Compare M values layout to source.
```

Now assume that we compare GDSII to Spice using the above set of statements. Consider the following cases:

Layout GDSII:	3 MP devices in parallel
Source netlist:	M1 1 2 3 4 P M=3
Result:	Correct

Layout GDSII:	3 MP devices in parallel
Source netlist:	M1 1 2 3 4 P M2 1 2 3 4 P M3 1 2 3 4 P
Result:	Correct (M values in Spice default to 1)

Layout GDSII:	3 MP devices in parallel
Source netlist:	M1 1 2 3 4 P M=2 M2 1 2 3 4 P
Result:	Correct (M=2 value in M1 is added to the default value M=1 in M2)

Layout GDSII:	3 MP devices in parallel
Source netlist:	M1 1 2 3 4 P
Result:	Property Error (M=3 in layout, M=1 in source)

Layout GDSII:	3 MP devices in parallel
Source netlist:	M1 1 2 3 4 P M=2
Result:	Property Error (M=3 in layout, M=2 in source)

Note of course that the Device operation is not necessary if you compare Spice to Spice.

See also LVS Spice Replicate Devices in the *SVRF Manual*.

Chapter 11
Spice Format

Introduction

A Spice netlist may serve as a source of connectivity for LVS. LVS will parse the netlist and compare it to the layout. The Spice variety accepted by LVS is described in this section. It is compatible with most common varieties of Spice, such as Spice 2 and Hspice, as well as the Dracula CDL format. The name of the netlist file and an optional name of the top level subcircuit are specified as part of the LVS invocation.

If a top-level subcircuit name is specified to LVS, then this subcircuit serves as the top-level network. The pins of this subcircuit serve as design ports in LVS. If a top-level subcircuit name is not specified, then LVS looks for any element statements and subcircuit calls in the netlist that are not part of any subcircuit definition. These statements then serve as the top-level network.

Some special features:

- Subcircuits that contain no devices are treated as primitive components (black boxes).
- Parameters are passed between levels of hierarchy. Primitive subcircuits can own arbitrary parameters.
- Alphanumeric names are allowed for nets and devices.

In flat LVS, the LVS Spice parser creates temporary files in directory $MGC_HOME/tmp. The environment variable $MGC_TMPDIR overrides $MGC_HOME/tmp; if $MGC_TMPDIR is set then temporary files are written to that directory instead. If neither environment variable is set then temporary files

are written to the current working directory. In hierarchical LVS, the spice parser does not generate temporary files.

Spice-like Property Syntax

Property names in Trace Property, LVS Filter, LVS Property Map, and similar specification statements in a rule file can be followed by a parameter name in parenthesis. This format instructs LVS to interpret the values of these properties as strings in Spice-like syntax, and to parse those strings to obtain the parameter indicated in parentheses. The parameter name should be one of the following (either upper or lower case):

- **w**: MOS width
- **l**: MOS length
- **r**: Resistor resistance
- **c**: Capacitor capacitance
- **a**: Diode area
- **p**: Diode perimeter

In EDDM designs, and V7.0 ".erel" designs, LVS uses the values of properties whose names start with the string "ms_". All ms_ property values should be character strings. For every instance, LVS appends the values of the ms_ properties found on the instance to the value of the property indicated before the parentheses. The ms_ property values are appended in alphabetical order of the ms_ property names. The resulting string is then parsed. In EDDM designs, LVS uses only ms_ properties that are specified as visible in the EDDM viewpoint.

The "( )" naming convention allows you to specify the property name and the actual parameter name to be parsed out of the string. You need to use the "( )" naming convention only to refer to property name-value pairs in Mentor Graphics databases (EDDM or IC Station), and only for properties that have Spice-like string values. You can also use this convention to refer to parameters in a real

Spice netlist; in this case, only the letter in parenthesis is used, and the property name is ignored. For a Spice netlist, you can simply specify the parameter name, such as “w” or “l”, with no additional property name, or parentheses.

For example,

```
instpar(w)
```

indicates that a MOS transistor width value should be extracted from the value of property instpar. Any ms_ properties are appended, if applicable.

```
foo(r)
```

indicates that a resistance value should be extracted from the value of property foo. Any ms_ properties are appended, if applicable.

The property value, followed by the values of optional ms_ properties, should form a parameter line in Spice-like syntax. The line can contain any Spice arguments which are valid for the particular device type (not including the device name and node numbers).

General Spice Syntax

- **MOS Transistors**

  The width and length values can appear in any order anywhere in the device parameter line, as follows:

  ```
  …  <L=VAL> <W=VAL> …
  ```

  The L and W values may also appear without a prefixing “L=” and “W=”; in this case they should appear as the first two tokens in the string. “L=” and “W=” may be specified in upper or lower case. The length and width values are specified in meters, and are numeric values with optional scaling factors.

It is not required to start the device parameter line with a model name. The following are some examples:

```
L=5U W=2U
5U 2U
L=10U W=5U AD=100P AS=100P PD=40U PS=40U
10U 5U 2P 2P
MODM L=5U W=2U
MOD1 L=10U W=5U AD=100P AS=100P PD=40U PS=40U
```

- **Capacitors**

  The device parameter line should contain the capacitance value as the first token. The capacitance is specified in Farads. It is a number with an optional scaling factor. The scaling factor may be followed by an optional unit name as in 1PF. This optional unit name is ignored.

  ```
  VALUE …
  ```

  The following are some examples:

  ```
  1PF
  10P IC=3V
  ```

- **Resistors**

  The device parameter line should contain the resistance value as its first token. The resistance is specified in Ohms. It is a number with an optional scaling factor.

  ```
  VALUE …
  ```

  The following are some examples:

  ```
  100
  1K  TC=0.001,0.015
  ```

- **Diodes**

  The first token of the device parameter line can contain an optional area value, and the second token can contain an optional perimeter value. The

area and perimeter are numeric values with optional scaling factors. If a non-numeric value is found in the first or second position, the corresponding area and/or perimeter values are assumed to be missing. The area is specified in square meters. The perimeter is specified in meters.

```
<AREA <PERIM>> …
```

The device parameter line must not contain a model name. The following are some examples:

```
3.0P IC=0.2
3.0P 5.0U IC=0.2
```

Spice Notational Conventions

Table 11-1 shows the notational conventions used to describe the Spice syntax in the following sections.

Table 11-1. LVS Spice Netlist Notational Conventions

Syntax	Description
< >	Indicates an optional argument.
|	Indicates choice.
…	Indicates repetition (zero or more times).
UPPER	Upper case letters indicate literal keywords.
lower	Lower case letters indicate arguments to be substituted by other values.
+	Spice continuation character used when the syntax description spans across several lines.
bold	LVS uses bold arguments.
italic	LVS does not use italicized arguments but does check their syntax.
<ARG=val>	Unless indicated otherwise, optional arguments preceded by literal names can appear in any order.

Table 11-1. LVS Spice Netlist Notational Conventions [continued]

Syntax	Description
<val>	Arguments not preceded by literal names must appear exactly in the order shown.

Case Sensitivity

Unless indicated otherwise, all names, identifiers and keywords are case insensitive. This includes node names, element names, subcircuit names, parameter names, scaling factors, and so forth.

Continuation Character

The continuation character (+) must be the first non-white-space character of every line of a statement, other than the first line.

General Spice Syntax Summary

The following is a summary of the general spice syntax.

- White space characters: space, tab, cr, lf, vt, ff, “,”
- Numeric values:

  integer 12
  floating point 3.14
  integer or floating + integer exponent 1E-14, 2.65E3
  number + scale factor 12P, 3.14E-2U
  number + scale factor + comment unit 3.14FFarad

- Scale factors:

  T=1E12 G=1E9 MEG=1E6 K=1E3 MIL=25.4E-6
  M=1E-3 U=1E-6 N=1E-9 P=1E-12 F=1E-15

Arithmetic Expression

Generally, arithmetic expressions can appear anywhere a number can appear. Valid operators are (+, -, *, and /). Parenthesis ( ) can be used to specify precedence. Standard high to low precedence applies, in the following order: ( ), *, /, +, then -. Parameter names can be used within arithmetic expressions. Arithmetic expressions can be enclosed in single quotes, but this is not mandatory. Examples:

```
m1 1 2 3 4 p  w=2+3    l=k*(3+2*(a+5))
m1 1 2 3 4 p  w='2+3'  l='k*(3+2*(a+5))'
```

Comments

Lines that start with an asterisk (*) or a dollar sign ($) are treated as comments. Any text on a Spice line that is preceded by a “$” is treated as a comment. The following are some examples:

```
* This is a comment.
$ This is a comment, too.
C1 a 2 100p          $ This is also a comment?
```

There are exceptions to this rule. Element statements, including R, C, D, Q, J, M, and V have comment-coded parameters, such as $W=<value> and $L=<value>. These comment-coded parameters do not force everything following them to be ignored.

LVS allows you to mix the order of comment-coded parameters with regular Spice parameters in R, C, D, Q, J, M, and V elements in Spice netlists. For example, the following entries are valid:

```
R0 A B 2.2 $[RN] $W=1 M=2 $L=2
C0 A B 2.2 $[RN] $A=1 W=1 $P=2 M=2
```

In the first line, the regular Spice parameter M=2 appears after the comment-coded parameters $[RN] and $W=1. In the second line, the regular Spice parameters W=1 and M=2 are intermixed with the comment-coded parameters $[RN] $A=1 and $P=2.

When a comment character ($) is followed by text that is not a valid comment-coded construct, then the ($) and the rest of the line is treated as comment and ignored. For example, in the following line, everything after $mycomment is ignored:

```
R0 A B 2.2 $mycomment $[RN] $W=1 M=2 $L=2
```

This relaxed syntax may not be valid for Spice simulation because anything following a ($) comment is ignored by Spice simulators; thus, intermixing of comment-coded and regular parameters is not recommended.

Comment-coded Extensions

LVS uses two types of comment-coded extensions to extend the basic Spice syntax: “*.” extensions and “$” extensions. “*.” extensions are used to add entire statements to the language. “$” extensions are used to add new fields to existing Spice statements. For example:

```
*.CONNECT 10 20
C1 1 2 100 $A=100 $P=40
```

Since both * and $designate comments in Spice, a Spice simulator would ignore these extensions and any fields that follow them on the same Spice line.

You can mix the order of comment-coded parameters with regular Spice parameters in R, C, D, Q, J, M, and V elements in Spice netlists. For example, these statements are valid:

```
R0 A B 2.2 $[RN] $W=1 M=2 $L=2
C0 A B 2.2 $[RN] $A=1 W=1 $P=2 M=2
```

In the first line, the regular Spice parameter M=2 appears after the comment-coded parameters $[RN] and $W=1. In the second line, the regular Spice parameters W=1 and M=2 are intermixed with the comment-coded parameters $[RN], $A=1, and $P=2.

Such mixing is valid in LVS, but note that it may not be valid for Spice simulation because anything following a “$” comment is ignored by Spice simulators;

therefore, intermixing of comment-coded and regular parameters is not recommended.

When a “$” comment character is followed by text that is not a valid LVS comment-coded construct, then the “$” and the rest of the line is treated as comment and ignored. For example, in the following line, everything after $mycomment is ignored:

```
R0 A B 2.2 $mycomment $[RN] $W=1 M=2 $L=2
```

Control Statements

.END

.END <comment>

where comment is any text, normally the name of the data file being terminated. For example:

```
.END chip
```

LVS applications ignore this statement and issue a warning, for example:

```
Warning: .END ignored in file “z.net” at line 5
```

Any statements after a .END are ignored.

.INCLUDE or .INC

.INCLUDE pathname
-or-
.INC pathname

where pathname specifies a file. You can enclose pathname in single or double quotes. This control statement causes the named file to be included “in place” in the netlist. Multiple levels of inclusion are allowed. For example:

```
.INCLUDE  params
C1 1 2 10P
.INC  /net/user1/circuitfile
```

.OPTION SCALE

.OPTIONS <option> …
-or-
.OPTION <option> …
-or-
.PC <option> …
-or-
.CONTROL <option> …

Any option can be specified, but the tool ignores all options except for SCALE.

Options can have these formats:

```
opt
opt=x
```

Where opt is the option name and x is the value assigned to that option. Expressions are allowed.

You can redefine an option in a netlist. At any point in the netlist, the last definition seen will be used. Options default to 1 when not assigned a value, and can be reset by specifying x to be zero (0). Options specified within subcircuit definitions are used locally in those subcircuits only; those specified outside of subcircuit definitions apply globally.

The option:

```
SCALE=X
```

sets the size multiplier for the following parameters:

- Element R: W, L
- Element C: A, P
- Element D: A, P
- Element Q: A, W, L (includes $EA, if used)

- Element J: A, W, L
- Element M: W, L, AD, AS, PD, PS

LVS multiplies one-dimensional parameters by the value of SCALE, and areas by the value of SCALE squared. When SCALE=1, element parameters are entered with units of meters.

For example, set scale to 1E-6 to enter parameters in microns; areas are then in square microns.

In this example, M1 and M2 have equal sizes:

```
.OPTIONS SCALE=1E-6                    $ Sets scale to 1E-6.

.SUBCKT AAA
M1 1 2 3 4 PMOS W=4 L=1 AS=4 AD=4          $ No scale factors
.ENDS

.OPTIONS SCALE=1                       $ Sets scale to 1.

.SUBCKT BBB
M2 1 2 3 4 PMOS W=4U L=1U AS=4E-12 AD=4E-12  $ scale factors
.ENDS

X1 AAA
X2 BBB
```

*.CAPA

*.CAPA

This statement instructs LVS to ignore capacitor (C) elements in the netlist. It is coded as a comment and has no arguments.

*.CONNECT

*.CONNECT
-or-
*.J

This statement takes two or more node names as arguments and is coded as a comment. LVS shorts the specified nodes together into one net. The resulting net inherits all user-given names (if any) from the original nodes. Any one of the original names can serve as an initial correspondence point in LVS.

This control statement can appear at any level of hierarchy; shorts propagate up the hierarchy through subcircuit pins as necessary (these are called “deep shorts”). Shorts propagate in both hierarchical and flat execution. When LVS reports information about a shorted net, as in a discrepancy, it uses the name of one of the original nets. The choice is arbitrary, but there is preference, in the following order, for power and ground names, global nets, user-given names, and subcircuit pin names.

*.CONNECT and *.J statements that appear in a Spice netlist outside of subcircuit definitions apply to global nets anywhere in the netlist and to non-global nets in the top level subcircuit. The top level subcircuit is specified with the Source Primary or Layout Primary specification statements in the rule file. If you do not specify a top level subcircuit the *.CONNECT and *.J statements apply to global nets anywhere in the netlist, and to non-global nets in the top level network.

In the following example, global nets VCC1 and VCC2 will be connected together; local nets A and B in SUB1 also will be connected together, but local nets A and B in SUB2 will remain separate.

```
*.CONNECT VCC1 VCC2
*.CONNECT A B
*.GLOBAL VCC1 VCC2

.SUBCKT SUB2
C1 VCC1 VCC2
C2 A B
.ENDS

.SUBCKT SUB1    $ top level subcircuit
C3 A B
X1 SUB2
.ENDS
```

The *.CONNECT or *.J statements must be used to join together different nets in a netlist. The *.EQUIV statement only specifies correlation between different source and layout names.

*.DIODE

*.DIODE

This statement instructs LVS to ignore diode (D) elements in the netlist. It is coded as a comment and has no arguments.

*.EQUIV

*.EQUIV < new_name = old_name > …

This statement is coded as a comment. It typically specifies equivalence between different source and layout names. It is placed in a Spice netlist and renames the models and nodes in the netlist as follows:

- Model names equal to old_name are replaced with new_name. LVS performs this translation for model names that are part of the standard Spice syntax, and model names coded as comments in the form $[mname] or $.MODEL=mname. Model names are translated at all levels of hierarchy.

  In the following example, LVS replaces model names TP and TN by P and N, respectively. This allows LVS to recognize these devices as component type MP and MN, respectively.

  ```
  *.EQUIV P=TP N=TN
  M1 1 2 3 4 TP
  M2 1 2 3 4 TN
  ```

- Node names equal to old_name are replaced with new_name. LVS performs this translation on global node names, and all node names in the top level subcircuit, or top-level network (if a top-level subcircuit is not specified). It is not performed on non-global node names at lower levels of hierarchy.

In the following example, node 1 is renamed VCC and global node 0 is renamed VSS:

```
*.GLOBAL 1 0
*.EQUIV VCC=1 VSS=0
```

In the following example, TOP is the top level subcircuit as specified in the rule file. Node names SA, SB, and SC in TOP are renamed LA, LB, and LC, respectively. Node names SA and SB in BOTTOM remain unchanged because they are not top level nodes and are not global.

```
*.EQUIV LA=SA LB=SB LC=SC

.SUBCKT BOTTOM P1 P2
C1 SA P1 100
C2 SB P2 100
.ENDS

.SUBCKT TOP SA SB
C1 SA SB 100
C2 10 SC 100
X1 20 30 Bottom
.ENDS
```

*.EQUIV does not connect different nodes together, it merely changes the node names. If the same new_name is assigned to two nodes, they remain distinct but are assigned the same name. LVS issues a warning, and does not use the name as an initial correspondence point.

*.LDD

*.LDD

This statement is coded as a comment and takes no arguments. This statement controls how the LVS Spice reader processes $LDD designators in M elements in Spice. When you specify the LVS Spice Conditional LDD specification statement with the YES secondary keyword in the rule file, the Spice reader processes $LDD designators in M elements only if a *.LDD statement is present somewhere in the netlist and ignores $LDD designators otherwise. When you specify the secondary keyword NO with the LVS Spice Conditional LDD specification statement in the rule file, or when the LVS Spice Conditional LDD statement does

not appear in the rule file, the *.LDD statement has no effect on LVS execution. The following example shows how the *.LDD statement can be used in Spice syntax:

```
*.LDD
M2 4 5 6 7 P $LDD[PPP]
```

*.XPINS

*.XPINS

This statement supports the E2LVS EDIF converter and similar programs. It is coded as a comment and takes no arguments. The keyword XPINS stands for “explicit pin connections”. The statement must be contained within a subcircuit definition, and specifies that pins of the enclosing subcircuit should be treated as standalone objects, distinct from identically named nets within the subcircuit. Connections between pins and nets must be indicated explicitly with *.J or *.CONNECT statements even when the respective pins and nets have identical names. Pins of the subcircuit are referred to by preceding the pin name with the string “==”. Pin references are allowed only within *.J or *.CONNECT statements. The “==” is not part of the pin name and merely indicates the type of reference.

Example:

```
.SUBCKT SS1 A B   $ Subcircuit SS1 with pins A and B.
*.XPINS           $ Subcircuit has explicit pin connections.
*.J A ==A         $ Join net A with pin A of SS1.
*.J B ==B         $ Join net B with pin B of SS1.
*.J 1 ==B         $ Join net 1 with pin B of SS1.
*.J ==A ==B       $ Join pins A and B of SS1.
C1 A B            $ A capacitor hooked to nets A and B.
.ENDS
```

In subcircuits with explicit pin connections, nets and pins remain separate unless they are connected explicitly with *.J or *.CONNECT statements. Therefore, you can have pins and nets with identical names even though they are not connected

together. In the following example, capacitor C1 is hooked up to nets A and B in SS2 but not to the respective pins A and B:

```
.SUBCKT SS2 A B
*.XPINS
C1 A B
.ENDS
```

When LVS reads a Spice netlist, it creates a net for each pin in the .SUBCKT pin list, even if the pin is not connected to any elements within the subcircuit. Therefore, in the example above, there will be two different nets both called A in SS2; one hooked up to the subcircuit pin and one hooked up to capacitor C1. Similarly, there will be two nets called B in SS2.

Contrast this with a regular Spice subcircuit (without *.XPINS), where nets and pins are one and the same thing and pin-to-net connections are implicit. In the following example, capacitor C1 is connected to nets A and B in SS3 and also to the respective pins A and B of SS3.

```
.SUBCKT SS3 A B
C1 A B
.ENDS
```

Other Control Statements

The following list identifies the control statements ignored by the LVS Spice parser, where warning messages are not transcripted.

.AC	.IC	.NOISE	.PROTECT	.TF
.DC	.MEASURE	.OP	.PZ	.TITLE
.DCVOLT	.MODEL	.PLOT	.SAMPLE	.TRAN
.DISTO	.NET	.PRINT	.SENS	.UNPROTECT
.FOUR	.NODESET	.PROBE	.TEMP	.WIDTH
.GRAPH				

The following list identifies the control statements ignored by the LVS Spice parser, where warning messages are transcripted:

.ALTER	.DATA	.DEL	.LIB	.OPTIONS

Element Statements

Resistor Element

Rxxx n1 n2 <mname> <r *<tc1 <tc2* **<scale <m <ac>>>>>>**
+ <L=l> <W=w> <parnam=pval> … <**$SUB=ns**>
+ <$[mname] | $.MODEL=mname>** <**$W=w> <**$L=l**> **<$X=x> <$Y=y>**

Rxxx n1 n2 <mname> <r <TC=*tc1 <tc2* **<scale>>>>** <**M=m**> **<AC=ac>**
+ <L=l> <W=w> <parnam=pval> … <**$SUB=ns**>
+ <$[mname] | $.MODEL=mname>** <**$W=w> <**$L=l**> **<$X=x> <$Y=y>**

Rxxx n1 n2 <mname> <R=r> *<TC1=tc1> <TC2=tc2>* **<SCALE=scale>**
+ <**M=m**> **<AC=ac> <L=l> <W=w> <parnam=pval> …** <**$SUB=ns**>
+ <$[mname] | $.MODEL=mname>** <**$W=w> <**$L=l**> **<$X=x> <$Y=y>**

where the arguments are defined in Table 11-2. The following are some examples:

```
R1 1 2 4
R2 n5 n6 4 TC=2 3 4 M=3 W=10U L=20U AAA=5 BBB=7 $[x] $comment
R3 na nb 4 1 1 4 M=3 $SUB=3 $.MODEL=x $W=10U $L=20U
```

To enter resistor devices with more than one substrate pin, define a primitive subcircuit as described in the section ".SUBCKT or .SUB or .MACRO".

LVS first tries to interpret the fourth token as a resistance value. If that is not possible, for example if it is not a numeric value or previously defined parameter, then LVS interprets that token as model name.

Table 11-2. Resistor Element

Argument Name	Description	Trace Property Name
Rxxx	Resistor element name. Must begin with a "R" followed by any number of alphanumeric characters.	
n1	Positive terminal node name. String of any number of alphanumeric characters.	

Table 11-2. Resistor Element [continued]

Argument Name	Description	Trace Property Name
n2	Negative terminal node name. String of any number of alphanumeric characters.	
mname	Optional model name. String of any number of alphanumeric characters.	
r	Resistance in ohms.	r
tc1	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies resistance (r).	
m	Optional multiplier factor used to simulate multiple parallel resistors. Normally, divides resistance (r), multiplies width (w). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the resistor are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
ac	Optional AC resistance for AC analysis.	ac
l	Optional length.	l
w	Optional width.	w
parnam=pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.	
$[mname] or $.MODEL=mname	Optional model name, coded as a comment. String of any number of alphanumeric characters. Overrides the regular optional mname parameter.	

Table 11-2. Resistor Element [continued]

Argument Name	Description	Trace Property Name
$W=w	Optional width, coded as a comment. Overrides the regular optional w parameter	w
$L=l	Optional length, coded as a comment. Overrides the regular optional l parameter	l
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: "R"
LVS component subtype: mname
LVS pin names: "pos" (positive), "neg" (negative), "sub" (optional substrate)

Capacitor Element

Cxxx n1 n2 <mname> <c *<tc1 <tc2* **<scale <ic <m>>>>>>>**
\+ **<L=l> <W=w> <parnam=pval> …** <**$SUB=ns**>
\+ <**$[mname] | $.MODEL=mname**> **<$A=a> <$P=p> <$X=x> <$Y=y>**

Cxxx n1 n2 <mname> <c <TC=*tc1 <tc2* **<scale>>>> <IC=ic>** <**M=m**>
\+ **<L=l> <W=w> <parnam=pval> …** <**$SUB=ns**>
\+ <**$[mname] | $.MODEL=mname**> **<$A=a> <$P=p> <$X=x> <$Y=y>**

Cxxx n1 n2 <mname> <C=c> *<TC1 =tc1> <TC2=tc2>* **<SCALE =scale>**
\+ **<IC=ic>** <**M=m**> **<L=l> <W=w> <parnam=pval> …** <**$SUB=ns**>
\+ <**$[mname] | $.MODEL=mname**> **<$A=a> <$P=p> <$X=x> <$Y=y>**

where the arguments are described in Table 11-3. The following are some examples:

```
C1 1 2 10P
C3 n1 n2 10P M=4 W=10U L=20U AAA=5 BBB=7 $[mc] $comment
C4 n1 n2 10P 1 1 4 $SUB=3 $.MODEL=mc $A=10P $P=40U
```

To enter capacitor devices with more than one substrate pin, define a primitive subcircuit as described in section ".SUBCKT or .SUB or .MACRO".

LVS first tries to interpret the fourth token as a resistance value. If that is not possible, for example if it is not a numeric value or previously defined parameter, then LVS interprets that token as model name.

Table 11-3. Capacitor Element

Argument Name	Description	Trace Property Name
Cxxx	Capacitor element name. Must begin with a “C” followed by any number of alphanumeric characters.	
n1	Positive terminal node name. String of any number of alphanumeric characters.	
n2	Negative terminal node name. String of any number of alphanumeric characters.	
mname	Optional model name. String of any number of alphanumeric characters	
c	Capacitance in farads.	c
tc1	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies capacitance (c).	
ic	Optional initial voltage across the capacitor in volts.	ic
m	Optional multiplier factor used to simulate multiple parallel capacitors. Normally, divides capacitance (c), multiplies width (w). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the capacitor are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
l	Optional length.	l
w	Optional width.	w
parnam= pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam

Table 11-3. Capacitor Element [continued]

Argument Name	Description	Trace Property Name
ns	Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.	
$[mname] or $.MODEL =mname	Optional model name, coded as a comment. String of any number of alphanumeric characters. Overrides the regular optional mname parameter	
$A=a	Optional area, coded as comment.	a
$P=p	Optional perimeter, coded as comment.	p
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: "C"
LVS component subtype: mname
LVS pin names: "pos" (positive), "neg" (negative), "sub" (optional substrate)

The Dracula CDL statement *.CAPA is supported. It instructs LVS to ignore capacitor elements in the netlist.

Inductor Element

Lxxx n1 n2 <l <tc1 <tc2>>> **<SCALE=scale> <M=m> <R=r>**
+ <parnam=pval> … <$SUB=ns> <$[mname] | $.MODEL=mname>
+ <$X=x> <$Y=y>
-or-
Lxxx n1 n2 <L=l> <TC1=tc1> <TC2=tc2> **<SCALE=scale> <M=m> <R=r>**
+ <parnam=pval> … <$SUB=ns> <$[mname] | $.MODEL=mname>
+ <$X=x> <$Y=y>

where the arguments are defined in Table 11-4. The following are some examples:

```
L1 1 2 100
```

```
l2 n1 n2 100 10 20 SCALE=2 IC=30 M=4 DTEMP=40 R=200 FOO=300
+ $SUB=n3 $[lmod]
L3 na nb L=100 TC1=10 TC2=20 SCALE=2 IC=30 M=4 DTEMP=40 R=200
+ FOO=300 $SUB=nc $.MODEL=lmod $X=1000 $Y=2000
```

To enter diode devices with more than one substrate pin, define a primitive subcircuit as described in the section ".SUBCKT or .SUB or .MACRO".

Table 11-4. Inductor Element

Argument Name	Description	Trace Property Name
Lxxx	Inductor element name. Must begin with a "L" followed by any number of alphanumeric characters.	
n1	Positive terminal node name. String of any number of alphanumeric characters.	
n2	Negative terminal node name. String of any number of alphanumeric characters.	
l	Optional inductance in Henries.	l
tc1	Ignored.	
tc2	Ignored.	
scale	Optional scale factor. Multiplies inductance (l) and resistance (r).	
m	Optional multiplier factor used to simulate multiple parallel inductors. Normally, divides inductance (l) and resistance (r). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the inductor are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
r	Optional resistance in Ohms.	r
w	Optional width.	w

Table 11-4. Inductor Element [continued]

Argument Name	Description	Trace Property Name
parnam= pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.	
$[mname] or $.MODEL =mname	Optional model name, coded as a comment. String of any number of alphanumeric characters.	
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: “L”
LVS component subtype: mname
LVS pin names: “pos” (positive), “neg” (negative), “sub” (optional substrate)

By default, positive and negative pins of inductor elements in Spice netlists are not swappable in LVS. However, like other devices, inductor elements entered in Spice netlists inherit pin swappability from DEVICE operations in the rule file. For example:

```
DEVICE L ind m1(pos) m1(neg)          // implicit swappability
DEVICE L ind m1(pos) m2(neg) well(sub) (pos neg)
                                      // explicit swappability
```

In the first example, pos-neg pin swappability for inductor devices is implied by the fact that these pins have the same layer, m1. In the second example, pos-neg pin swappability for inductor devices is specified explicitly with a swap list.

Junction Diode Element

Dxxx nplus nminus mname <**AREA=a**> <**PJ=pj**> <**M=m**> <*OFF*>
+ **<parnam=pval> …** <**$SUB=ns**> **<$X=x> <$Y=y>**
-or-
Dxxx nplus nminus mname <**a <pj**>> <**M=m**> <*OFF*>
+ **<parnam=pval> …** <**$SUB=ns**> **<$X=x> <$Y=y>**

where the arguments are defined in Table 11-5. The following are some examples:

```
D1 1 2 mdio
D2 a b mdio 2P 3U M=3 AAA=5 BBB=7
D3 a b mdio AREA=2P PJ=3U M=3 $SUB=c
```

To enter diode devices with more than one substrate pin, define a primitive subcircuit as described in the section ".SUBCKT or .SUB or .MACRO".

Table 11-5. Junction Diode Element

Argument Name	Description	Trace Property Name
Dxxx	Diode element name. Must begin with a "D" followed by any number of alphanumeric characters.	
nplus	Positive (anode) terminal node name. String of any number of alphanumeric characters.	
nminus	Negative (cathode) terminal node name. String of any number of alphanumeric characters.	
mname	Model name. String of any number of alphanumeric characters.	
a	Optional diode area.	a
pj	Optional periphery of junction.	p
m	Optional multiplier factor to simulate multiple diodes. Normally, multiplies area (a) and perimeter (p). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the diode are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m

Table 11-5. Junction Diode Element [continued]

Argument Name	Description	Trace Property Name
OFF	Ignored.	
parnam=pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
ns	Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.	
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: “D”
LVS component subtype: mname
LVS pin names: “pos” (positive), “neg” (negative), “sub” (optional substrate).

The Dracula CDL statement *.DIODE is supported. It instructs LVS to ignore diode elements in the netlist.

BJT Element

Qxxx nc nb ne *<[ns]>* <**ns**> **mname** <**AREA=a**> <**M=m**> *<OFF>*
+ **<parnam=pval> … <$SUB=ns>** <**$EA=a**> <**$W=w**> <**$L=l**>
+ <$X=x> <$Y=y>
-or-
Qxxx nc nb ne *<[ns]>* <**ns**> **mname** <**a**> <**M=m**> *<OFF>*
+ *<IC=vbe, vce>* **<parnam=pval> …<$SUB=ns>** <**$EA=a**>
+ <**$W=w**> <**$L=l**> **<$X=x> <$Y=y>**

where the arguments are defined in Table 11-6. The following are some examples:

```
Q23 10 24 13 QMOD AREA=5P
Q23 10 24 13 QMOD 5P
```

```
Q50A neta netb netc netsub modq4 M=3 AAA=5 BBB=7
+ $L=2U $EA=4P $W=6U $comment
```

To enter Q devices with more than one substrate pin, define a primitive subcircuit as described in the section ".SUBCKT or .SUB or .MACRO".

Table 11-6. BJT Element

Argument Name	Description	Trace Property Name
Qxxx	BJT element name. Must begin with a "Q" followed by any number of alphanumeric characters.	
nc	Collector terminal node name. String of any number of alphanumeric characters.	
nb	Base terminal node name. String of any number of alphanumeric characters.	
ne	Emitter terminal node name. String of any number of alphanumeric characters.	
[ns]	Optional substrate terminal node name enclosed in square brackets. String of any number of alphanumeric characters. If present, must be enclosed in []. *Ignored by LVS.*	
ns	Optional substrate terminal node name. String of any number of alphanumeric characters.	
mname	Model name. String of any number of alphanumeric characters.	
a	Optional emitter area.	a
areab	Used (Optional).	
areac	Used (Optional).	
m	Optional multiplier factor to simulate multiple BJTs. Normally, multiplies area (a) and width (w). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the BJT are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
OFF	Ignored.	

Table 11-6. BJT Element [continued]

Argument Name	Description	Trace Property Name
vbe	Used (Optional).	
vce	Used (Optional).	
parnam=pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
$SUB=ns	Optional substrate terminal node name, coded as a comment. Overrides regular Spice substrate terminal field, if present.	
$EA=a	Optional emitter area, coded as a comment. Overrides <AREA=a> or <a>. A practical use would be to use $EA in the source, calculate the "A" property in the layout, then use TRACE PROPERTY Q A A 1 for LVS comparison.	a
$W=w	Optional width, coded as a comment.	w
$L=l	Optional length, coded as a comment.	l
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: “Q”
LVS component subtype: mname
LVS pin names: “c” (collector), “b” (base), “e” (emitter), “s” (optional substrate).

When the syntax is ambiguous and can be interpreted as either a 3-pin or a 4-pin device, LVS will interpret it as a 3-pin device.

JFET Element

Jxxx nd ng ns <nb> mname <AREA=a> <W=w> <L=l> <M=m> *<OFF>* + *<IC=vds, vgs>* **<parnam=pval> …** <**$SUB=nb**> **<$X=x> <$Y=y>**

-or-

Jxxx nd ng ns <nb> mname <a> <w> <l> <M=m> *<OFF> <IC=vds, vgs>* + **<parnam=pval> …** <**$SUB=nb**> **<$X=x> <$Y=y>**

where the arguments are defined in Table 11-7. The following are some examples:

```
J1 10 24 13 JM1
Jabc netd netg nets jmod AREA=8P W=3U L=7U M=2 AAA=5 BBB=7
J234 netd netg nets jmod 8P 3U 7U M=2 $SUB=netb
```

To enter JFET devices with more than one substrate pin, define a primitive subcircuit as described in the section ".SUBCKT or .SUB or .MACRO".

Table 11-7. JFET Element

Argument Name	Description	Trace Property Name
Jxxx	JFET element name. Must begin with a "J" followed by any number of alphanumeric characters.	
nd	Drain terminal node name. String of any number of alphanumeric characters.	
ng	Gate terminal node name. String of any number of alphanumeric characters.	
ns	Source terminal node name. String of any number of alphanumeric characters.	
<nb>	Optional bulk connection node name. String of any number of alphanumeric characters.	
mname	Model name. String of any number of alphanumeric characters.	
a	Optional area.	a
w	Optional gate width.	w
l	Optional gate length.	l

Table 11-7. JFET Element [continued]

Argument Name	Description	Trace Property Name
m	Optional multiplier factor to simulate multiple JFETs. Normally, multiplies area (a) and width (w). If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the JFET are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
OFF	Ignored.	
vds	Ignored.	
vgs	Ignored.	
parnam= pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
$SUB=nb	Optional bulk terminal node name coded as a comment. String of any number of alphanumeric characters. Overrides the regular fourth terminal field if both are present.	
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: “J”
LVS component subtype: mname
LVS pin names: “d” (drain), “g” (gate), “s” (source), “b” (optional bulk).

MOSFET Element

Mxxx nd ng ns <nb> mname <L=l> <W=w> <AD=ad> <AS=as>
+ <PD=pd> <PS=ps> <NRD=nrd> <NRS=nrs> <RDC=rdc>
+ **<RSC=rsc> ,OFF>** *<IC=vds, vgs, vbs>* **<M=m> <parnam=pval> …**
+ **<$LDD<[type]>> <$X=x> <$Y=y>**
-or-

Mxxx nd ng ns <nb> mname <l> <w> <ad> <as> <pd> <ps>
+ <nrd> <nrs> <rdc> <rsc> *<OFF> <IC=vds, vgs, vbs>* **<M=m>**
+ <parnam=pval> … <$LDD<[type]>> <$X=x> <$Y=y>

where the arguments are defined in Table 11-8.

LVS will interpret this as a 4-pin device when the syntax is ambiguous and can be interpreted as either a 3-pin or a 4-pin device.

LVS does not recognize the Spice statement ".OPTION wl"; specify LVS Reverse WL YES in the rule file instead. Normally, if the prefixes "L=" and "W=" are not used, then "l" must precede "w".

Other characters can follow the parameter names W and L. LVS interprets any parameter name that starts with W as width, and with L as length.

Table 11-8. MOSFET Element

Argument Name	Description	Trace Property Name
Mxxx	MOSFET element name. Must begin with an "M" followed by any number of alphanumeric characters.	
nd	Drain terminal node name. String of any number of alphanumeric characters.	
ng	Gate terminal node name. String of any number of alphanumeric characters.	
ns	Source terminal node name. String of any number of alphanumeric characters.	
nb	Optional bulk terminal node name. String of any number of alphanumeric characters.	
mname	Model name. String of any number of alphanumeric characters.	
l	Optional channel length.	l
w	Optional channel width.	w
ad	Optional drain area.	ad

Table 11-8. MOSFET Element [continued]

Argument Name	Description	Trace Property Name
as	Optional source area.	as
pd	Optional drain perimeter.	pd
ps	Optional source perimeter.	ps
nrd	Optional number of squares of drain diffusion.	nrd
nrs	Optional number of squares of source diffusion.	nrs
rdc	Optional additional drain resistance due to contact resistance.	rdc
rsc	Optional additional source resistance due to contact resistance.	rsc
OFF	Ignored.	
vds	Ignored.	
vgs	Ignored.	
vbs	Ignored.	
m	Optional multiplier factor to simulate multiple MOSFETs. Normally, multiplies w, ad, as, pd and ps. If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the MOSFET are created instead, and *m* for each copy is set to 1. Defaults to 1 if not specified.	m
parnam= pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
$LDD	Optional LDD device designator, coded as a comment.	
type	Optional LDD device type. String of any number of alphanumeric characters. Allowed (but not required) only in conjunction with $LDD. If specified, replaces mname.	
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type:

Without $LDD:

If mname starts with 'N' or 'n':	“MN”
If mname starts with 'P' or 'p':	“MP”
If mname starts with 'E' or 'e':	“ME”
If mname starts with 'D' or 'd':	“MD”
Otherwise:	“M”

With $LDD:

If mname (or type) starts with 'N' or 'n':	“LDDN”
If mname (or type) starts with 'P' or 'p':	“LDDP”
If mname (or type) starts with 'E' or 'e':	“LDDE”
If mname (or type) starts with 'D' or 'd':	“LDDD”
Otherwise:	“LDD”

LVS component subtype: Normally mname; if [type] is specified then type.
LVS pin names: “d” (drain), “g” (gate), “s” (source), “b” (optional bulk).

The following are some examples:

```
M2 10 24 13 14 TYPE1
M3 10 24 13 TYPE2
Ma1 netd netg nets netb pmos L=3U W=2U AD=4P AS=5P PD=6U
+ PS=7U NRD=3 NRS=4 M=2 AAA=5 BBB=7
Ma2 netd netg nets netb pmos 3U 2U 4P 5P 6U 7U M=2
```

To enter MOS devices with more than one substrate pin, define a primitive subcircuit as described in the section “.SUBCKT or .SUB or .MACRO”.

Voltage Source Element

Vxxx nplus nminus <<DC <=>> dc> <M=m> <parnam=pval> …
+ <$X=x> <$Y=y>

where the arguments are defined in Table 11-9. The following are some examples:

```
V1 n1 n2 5
V2 n1 n2 DC 5
V3 n1 n2 DC=5 AAA=5 BBB=7
```

Table 11-9. Voltage Source Element

Argument Name	Description	Trace Property Name
Vxxx	Voltage source element name. Must begin with a “V” followed by any number of alphanumeric characters.	
nplus	Positive terminal node name. String of any number of alphanumeric characters.	
nminus	Negative terminal node name. String of any number of alphanumeric characters.	
dc	Optional DC and transient analysis value of the source.	dc
parnam= pval	Optional parameter name set to a value. Arbitrary parameter names are allowed. “parnam” must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.	parnam
m=	Optional multiplier factor to simulate multiple voltage sources. Defaults to 1 if not specified. Normally, its only effect is to set the *m* value of the voltage source. If LVS Spice Replicate Devices YES is specified in the rule file, then *m* parallel copies of the voltage source are created instead, and *m* for each copy is set to 1.	m
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.	

LVS component type: “V”
LVS component subtype: none
LVS pin names: “pos” (positive), “neg” (negative).

Subcircuits

.SUBCKT or .SUB or .MACRO

.SUBCKT subname < n1 n2 … < / m1 m2 … > > < parnam = pval > …
-or-

.SUB subname < n1 n2 … < / m1 m2 … > > < parnam = pval > …
-or-
.MACRO subname < n1 n2 … < / m1 m2 … > > < parnam = pval > …

where the arguments are defined in Table 11-10.

Table 11-10. Subckt Statement

Argument Name	Description
subname	Subcircuit name. String of any number of alphanumeric characters.
n1 n2 …	Node names for external reference. Strings of any number of alphanumeric characters. Any element nodes appearing in the subcircuit but not included in this list or in a .GLOBAL or *.GLOBAL statement are strictly local.
/ m1 m2 …	Optional additional node names for external reference. The "/" characters in the node name list are treated as whitespace, unless you specified the LVS Spice Slash Is Space NO specification statement in your rule file.
parnam = pval	Optional parameter name set to a value for use only in the subcircuit. The value applies to this subcircuit and to any subcircuits called by this subcircuit. It is overridden by an assignment in the subcircuit call or by a value set in a .PARAM statement. "parnam" must begin with a letter followed by any number of alphanumeric characters.

".SUBCKT", ".SUB" and ".MACRO" are all equivalent.

The statements following the subcircuit statement (and preceding the ENDS or EOM statement) define the subcircuit. Subcircuit definitions may contain subcircuit calls. Subcircuit definitions may contain other (nested) subcircuit definitions.

When resolving node names inside subcircuit definitions, .GLOBAL nodes normally have precedence over subcircuit pins with the same name. To indicate the opposite, specify LVS Spice Prefer Pins Yes in the rule file.

Here is an example:

```
.SUBCKT  res2  in1 in2  res=5
R1  in1   3   res
R2   3   in2  res
.ENDS  res2
```

Primitive Subcircuits:

A subcircuit that does not contain any element statements or subcircuit calls is considered a primitive component in LVS:

LVS component type: subname
LVS component subtype: none
LVS pin names: n1 n2 … (and optional m1 m2 …)

The parameter names “parnam” can be entered as property names in the $trace_property_numeric() function. This function can be used to trace parameter values specified in primitive subcircuit definitions or primitive subcircuit calls. Here is an example:

```
.SUBCKT  primitive  a b c  par1=5U  par2=10U
.ENDS  primitive
```

Design Ports:

When the top-level network is enclosed in a subcircuit definition, LVS uses the external node names of the top-level subcircuit (“n1 n2 …” and optional “m1 m2 …“) as design ports.

Duplicate Pins:

The LVS Spice parser allows duplicate pin names in subcircuit definitions, but issues warnings about them. Only the first pin in each group of duplicate pin names actually connects to elements within the subcircuit; other pins in the group are unused. For example:

```
.SUBCKT  bar a a
C1 a b 100
.ENDS
X1 1 2 bar
```

In the subcircuit call X1, node 1 will connect to C1 in “bar” through pin “a” of “bar”. Node 2 will not connect to any devices in “bar”. A warning will be issued about duplicate definition of pin “a” in “bar”.

Duplicate .SUBCKT Definitions:

The LVS Spice parser classifies subcircuits by subcircuit name and number of pins. Subcircuit definitions with the same name but different number of pins are allowed and are not considered duplicate. Subcircuit definitions with the same name and the same number of pins are duplicate. When duplicate subcircuit definitions are present in a netlist, one of those definitions is used and all others are discarded with warnings. The user should not rely on any specific subcircuit definition (first, last or other) to be chosen.

.ENDS or .EOM

.ENDS <subname>
-or-
.EOM <subname>

where “subname” is the subcircuit name. This statement must be the last one for any subcircuit definition. The following are some examples:

```
.ENDS opamp
.EOM
```

Subcircuit Calls

Xyyy < n1 n2 … > < / > <subname> < parnam = pval > … <M=m>
+ <$[mname] | $.MODEL=mname> <$T=tx ty r a>
+ <$X=x> <$Y=y> <$PINS <pin=node> … >

where the arguments are defined in Table 11-11. Here is an example:

```
X1  2  4  17  opamp  wn=100  ln=5
```

Table 11-11. Subcircuit Call

Argument Name	Description
Xyyy	Subcircuit call name. Must begin with an “X” followed by any number of alphanumeric characters.
n1 n2 …	Node names for external reference. Strings of any number of alphanumeric characters. The program references the names in the order they are specified in the subcircuit definition.
/	Optional. The “/” characters in the node name list and in the subcircuit reference name are treated as whitespace, unless you specify the LVS Spice Slash Is Space NO specification statement in your rule file. See $PINS also.
subname	Subcircuit reference name. Must appear in a corresponding .SUBCKT or .MACRO definition in the netlist. A subcircuit call can appear before or after the corresponding subcircuit definition.
parnam = pval	Optional parameter name set to a value for use only in this subcircuit call. The value applies to the referenced subcircuit, and to any subcircuits called indirectly by the referenced subcircuit. Overrides a parameter value assigned in the subcircuit definition, but overridden by a value set in a .PARAM statement. The parameter does not have to be specified in the subcircuit definition. “parnam” must begin with a letter followed by any number of alphanumeric characters.
m	Optional multiplier factor. Generates M subcircuit calls connected in parallel. The name of the first subcircuit call is Xyyy. The names of subsequent calls receive “==n” suffixes, where n are serial numbers starting with 2. For example the line: `X1 1 2 3 AAA M=3` generates 3 subcircuit calls of AAA connected in parallel. The subcircuit names are X1, X1==2, X1==3.

Table 11-11. Subcircuit Call [continued]

Argument Name	Description
mname	Optional model name, coded as a comment. String of any number of alphanumeric characters. Valid only on calls to primitive subcircuits. Used as LVS component subtype for the call.
$T=tx ty r a	Optional layout transform consisting of x translation, y translation, reflection and rotation-angle. Used in LVS-H only. The translation components tx and ty are integer numbers in database units. The reflection *r* is along the X axis and is either 0 or 1 which denote the absence or presence of reflection respectively. The rotation angle *a* is an integer number in degrees and the only valid values are 0, 90, 180 or 270; rotation is counter-clockwise.
$X=x $Y=y	Optional X,Y coordinates coded as comments. Integer numbers in database units.

Table 11-11. Subcircuit Call [continued]

Argument Name	Description

$PINS <pin=node> …	Optional argument that specifies pin connections by name (as opposed to by order). Coded as a comment. Each “pin” is a pin name in the .SUBCKT definition, and each “node” is a node name in the subcircuit call. The specified node connects to the specified pin in the subcircuit. Pin and node names are strings of any number of alphanumeric characters. Imbedded “/” characters are treated as part of the names. For example:
	`.SUBCKT FOO A B`
	`$ subcircuit contents…`
	`.ENDS`
	`X1 FOO $PINS B=1 A=2`
	In subcircuit call X1, pin B connects to node 1 and pin A connects to node 2.
	A subcircuit call may reference fewer pins than specified in the respective .SUBCKT definition; any pins that are not referenced in the subcircuit call are floating. For example:
	`.SUBCKT SSS A B C D`
	`$ subcircuit contents…`
	`.ENDS`
	`X2 1 2 SSS`
	`X3 SSS $PINS B=1 D=2`
	In subcircuit call X2, pins C and D are floating. In subcircuit call X3, pins A and C are floating. See notes below.

Notes:

1. Duplicate pins are not supported. That is, you cannot use the $PINS specification when the respective .SUBCKT definition has multiple pins with the same name.
2. The $PINS argument overrides the standard Spice node specification <n1 n2 …> if one is present.
3. The LVS Spice parser performs consistency checking regarding number of pins and pin names in the subcircuit call and the respective .SUBCKT definition. Mismatches are reported as errors. The number of <pin=node> pairs in a $PINS argument determines number of pins. Note that floating pins in the subcircuit call are usually allowed; see discussion of "Floating Pins" below.
4. Subcircuit definitions and subcircuit calls are classified by subcircuit name and number of pins. For example, you can have 2-pin and 3-pin versions with the same subcircuit name. The number of pins specified in a $PINS argument in a subcircuit call will classify the subcircuit call.

Here is another example that shows how a subcircuit call redefines a defined subcircuit parameter:

```
.SUBCKT  yyy  a  b  res=5   $ 'res' defaulted to 5 ohm…
R1  a  b  res            $ …and used to specify resistance.
.ENDS  yyy
X1  5  6  yyy  res=7    $ 'res' redefined on subcircuit call.
```

Primitive Subcircuit Calls. A primitive subcircuit call is a subcircuit call that references a primitive subcircuit. A primitive subcircuit is a subcircuit that does not contain any element statements or subcircuit calls, or a subcircuit that appears in an LVS Box specification statement. For example:

```
.SUBCKT primitive a b c
.ENDS
X1 1 2 3 primitive par1=5U  par2=10U $[model1]
```

The parameter names, “parnam”, may be entered as property names in the Trace Property specification statement. As always, they override parameter values assigned in the subcircuit definition. In the example above, you could trace properties “par1” and “par2”.

All primitive subcircuit calls own a special property called M, which is available for tracing. The “trace property” name is M. The value is always 1. Note that when you specify the optional M multiplier factor in a primitive subcircuit call, you get M individual subcircuit calls connected in parallel, each with property M equal to 1. The M property is *not* available for tracing in non-primitive subcircuit calls.

Floating pins. LVS allows floating pins in subcircuit calls. A subcircuit call may reference fewer pins than specified in the respective .SUBCKT definition; any pins that are not referenced in the subcircuit call are floating. For example:

```
.GLOBAL VCC VSS

.SUBCKT SSS A B C D
…
.ENDS

.SUBCKT XXX E VCC VSS
…
```

```
.ENDS

.SUBCKT ZZZ
X1 1 2 SSS                  $ Pins C and D are floating.
X2 SSS $PINS B=1 D=2        $ Pins A and C are floating.
X3 XXX                $ Pins E, VCC and VSS are floating.
.ENDS
```

As shown in the example, in regular subcircuit calls, pin reference is positional, starting with the first pin, and floating pins are at the end of the pin list. When the $PINS syntax is used, pin reference is by name and any other pins are floating.

In hierarchical operation, for each floating pin in a subcircuit call, if the pin name is not global, then the Spice reader creates a respective floating net in the subcircuit that contains the call. The floating net is connected to the floating pin. The floating net name consists of the instance name (including the prefix X) followed by a slash (/) followed by the pin name. In the example above, pin `C` of `X1` is connected to a net called X1/C in ZZZ and pin D of X1 is connected to a net called X1/D in ZZZ.

Floating pins with global names are connected to the respective global nets. In the example above, pin E of X3 is connected to a net called X3/E in ZZZ but pins VCC and VSS of X3 are connected to the global nets VCC and VSS respectively.

Note that the generated net name, for example X1/C, is in fact a hierarchical pathname.

If this hierarchical pathname also appears literally as a net name in the containing subcircuit then they will both refer to the same net. For this reason you must be careful when using hierarchical pathnames as literal net names in a Spice netlist.

In flat operation, floating pins are represented simply by the respective nets within the particular subcircuit calls, so no additional nets are created.

Floating pins are not allowed if there is more than one respective .SUBCKT definition. For example, the following is not allowed:

```
.SUBCKT YYY A B C
…
.ENDS
```

```
.SUBCKT YYY A B C D
…
.ENDS
X1 1 2 YYY          $ Error: Ambiguous pin count.
```

You can instruct LVS applications to forbid floating pins in subcircuit calls by specifying the LVS Spice Allow Floating Pins NO specification statement in your rule file.

Nested Subcircuits

You can nest subcircuit definitions. Nested subcircuit definitions have local scope. In other words, an embedded subcircuit is visible only from its immediate parent in the nesting hierarchy and from other subcircuits nested under the parent. It is not visible from places above the parent in the nesting hierarchy or from siblings of the parent.

For example, the following design has two capacitors (from local bbb definition #1) and 4 resistors (from local bbb definition #2).

```
.subckt aaa d e f     $$ Beginning of subcircuit aaa.
$
.subckt bbb a b c     $$ Local definition of bbb #1;
c1 a b                $$   scope is aaa.
c2 b c
.ends bbb
$
x1 d e f bbb          $$ Call to bbb #1.
.ends aaa             $$ End of subcircuit aaa.

.subckt ccc d e f     $$ Beginning of subcircuit ccc.
$
.subckt bbb a b c     $$ Local definition of bbb #2;
r1 a b                $$   scope is ccc.
r2 b c
.ends bbb
$
.subckt ddd a b c
x1 a b c bbb          $$ Call to bbb #2.
.ends ddd
$
x1 d e f bbb          $$ Call to bbb #2.
```

```
x2 1 2 3 ddd
.ends ccc              $$ End of subcircuit ccc.

.subckt top
x1 g h l aaa
x2 i j k ccc
.ends
```

.PARAM

.PARAM < **parnam = pval** > **…**

where “<parnam = pval>…” are parameter names assigned to parameter values.

When you use the parameter name in a subcircuit description, the specified value is automatically substituted. This type of value is referred to as a global parameter. Each “parnam” must begin with a letter, followed by any number of alphanumeric characters.

Parameter values set in a .PARAM statement override those set in a .SUBCKT or .MACRO statements, or in subcircuit calls. Here is an example:

```
.PARAM  width=1U
X1  9  10  mos2 width=5U length=6U
*
.SUBCKT  mos2 in1 in2 width = 10U length=20U aread=30P
+  areas=40P
M1  in1  in2  3  4   pmos w=width l=length ad=aread  as=areas
.ENDS  mos2
```

In the above example, M1 has the following values:

w = 1U	(from the .PARAM statement)
l = 6U	(from the subcircuit call)
ad = 30P	(from the .SUBCKT statement)
as = 40P	(from the .SUBCKT statement)

See LVS Spice Redefine Param in the *SVRF Manual* as a related specification statement.

.GLOBAL

.GLOBAL < **node1 node2 …** >
-or-
***.GLOBAL** < **node1 node2 …** >

where “node1 node2 …” are node names defined as external references for all subcircuits in the netlist.

The specified nodes are globally shared by all subcircuits. The .GLOBAL statement provides a convenient means of communicating power supplies, clocks, and so on through the netlist. The form *.GLOBAL is coded as a comment and is entirely equivalent to .GLOBAL. Here is an example:

```
.GLOBAL  4  7  VDD  VSS
```

Nodes with user-given names specified in a .GLOBAL statement are treated as design ports in LVS, unless you specify the LVS Globals Are Ports NO specification statement in your rule file.

When resolving node names inside subcircuit definitions, .GLOBAL nodes normally have precedence over subcircuit pins with the same name. To indicate the opposite, specify LVS Spice Prefer Pins YES in the rule file.

You can discard the post-colon “:” suffixes from node names with the Virtual Connect Colon specification statement.

The Dracula CDL statement *.GLOBAL is supported and is equivalent to .GLOBAL as described above.

Subcircuit pin preferences—The preference of .SUBCKT pins over global signals when resolving Spice netlists is handled by the LVS Spice Prefer Pins statement.

Example:

```
.GLOBAL VCC VSS
.SUBCKT MYCELL VCC VSS
C1 VCC VSS
.ENDS
X1 A B MYCELL
```

Normally, capacitor C1 is connected to the global nets VCC and VSS respectively. However, if LVS Spice Prefer Pins YES is specified then C1 is connected to the VCC and VSS pins of MYCELL, which are in turn connected to top level nets A and B.

You can specify that subcircuit pin assignments override global signals throughout subcircuits and their sub-hierarchies by using the LVS Spice Override Globals statement.

Chapter 12
Utilities

This chapter describes the input requirements, invocation usage, and procedures for the following utilities:

- EDIF-to-LVS (E2LVS) – a converter that translates a EDIF structural netlist into a Spice-like netlist for use as input to Calibre LVS/LVS-H.
- Verilog-to-LVS (V2LVS) – a converter that translates a Verilog structural netlist into a Spice-like netlist for use as input to Calibre LVS/LVS-H.
- Dracula: File Conversion and User Notes
- Compare Two GDSII Databases
- Rules Syntax Checker

EDIF-to-LVS

The EDIF-to-LVS (E2LVS) program translates an EDIF (Electronic Design Interchange Format) netlist into a LVS Spice netlist suitable for Calibre LVS/LVS-H comparison against a layout. The following sections describe invocation, usage, and translation considerations and issues. The last section provides a sample netlist translation.

Usage

```
e2lvs   { -e  edif_input_file | -l input_list_file }
        -o  output_file [ -s  spice_input_file ]
        [ { -sb cell_file  | -ss cell_file } ]
        [ -a  char1 [ char2 ] ] [ -b  char ]
        [ -c  char1 [ char2 ] ] [ -r  char1 [ char2 ] ]
        [ -i ] [ -n  cell_name_file ] [ -p ]
        [ -w  warning_level ]
        [-cb] [-ictrace]
```

Description

E2LVS translates a structural EDIF 2 0 0 design into an equivalent Spice netlist for use as input to Calibre LVS/LVS-H. This netlist is an extended form of traditional Spice. The section "General Spice Syntax" in chapter 11, "Spice Format" describes the netlist format and extensions used in a LVS Spice netlist.

This tool allows the following inputs:

- An EDIF netlist file, or a file containing an ordered list of EDIF netlist files, one name per line.

  The correct EDIF netlist file order is necessary because a cell definition must be parsed before it is instantiated in another cell. Thus, if a cell in EDIF fileA uses a cell defined in EDIF fileB, then fileB must be listed before fileA in the list of EDIF files. If the files are unordered, E2LVS issues an error message indicating that a cell is undefined.

- A Spice file (optional) with one or more .INCLUDE statements, one statement per line.

  E2LVS includes the optionally specified Spice file at the beginning of the output Spice netlist file using a .INCLUDE statement. The input Spice file can contain multiple .INCLUDE statements. The optional Spice file generally contains leaf-level Spice cell definitions.

E2LVS translates the specified EDIF netlist file(s) into a single output Spice netlist file. This file is overwritten if it already exists. Figure 12-1 illustrates the E2LVS flow:

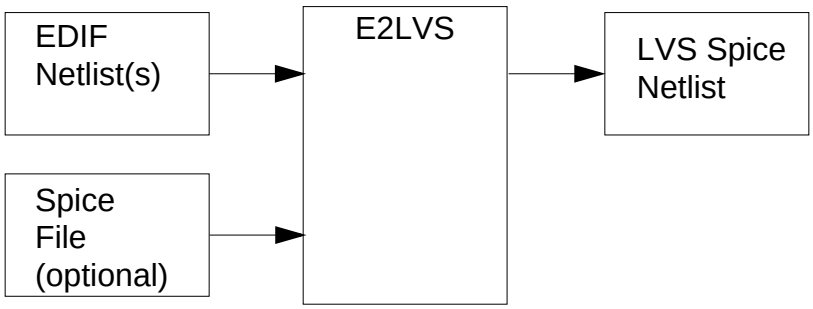


Figure 12-1. E2LVS Flow

E2LVS creates a file named “e2lvs_names” when cells with the same name in different libraries are found, since these names will collide in the translated Spice netlist. The “e2lvs_names” file lists the generated cell names and their associated original cell names. The command line switch -n allows you to specify a filename other than “e2lvs_names”. Refer to section “EDIF Cell Names Versus Spice Subcircuit Names” for more information.

During translation, non-structural syntax in the EDIF design is ignored by E2LVS. The EDIF design is assumed to be syntactically correct with only limited syntax checking to ensure that undesired states are not reached within the software.

To use the output Spice file as input into Calibre LVS, certain specification statements must be included in the rule file. These statements are identified in the section “Rule File” on page 2-1.

E2LVS searches initially for a calibrelvs license, then for a caldrclvseve license, then for an ictrace license. Command line switches may be used to control the license that gets used for a given run.

Arguments

Entering e2lvs -h prints a help line.

- **{-e *edif_input_file* | -l *input_list_file*}**

  Specifies the location of the EDIF input file(s). Possible choices are:

  -e Specifies the location of a single EDIF input file.

  -l Specifies the location of a file containing a list of EDIF input files, one per line. The files must be specified in the order cells are instantiated. Refer to section "Cell Statement" for more information.

- **-o *output_file***

  Specifies the location of the translated output Spice netlist file.

- **-s *spice_input_file***

  Specifies the location of an optional input Spice file.

- **{-sb *cell_file* | -ss *cell_file*}**

  Specifies how to translate EDIF cells. Possible choices are:

  -sb Specifies to translate EDIF cell names to Spice black boxes.

  ⇒ **-ss** Specifies to translate EDIF cell names to Spice subcircuit calls.

  If the optional *cell_file* parameter is not specified with the -sb command line switch, then all empty EDIF cells are translated into Spice black boxes; that is, empty subcircuit calls. Otherwise, the specified file contains the names of EDIF cells that are to be treated as black boxes. Empty EDIF cells that are not listed are assumed to be specified in an input Spice file. EDIF cells are considered "empty" when the contents parameter in the view statement is undefined.

  If the optional *cell_file* parameter is not specified with the -ss command line switch, then definitions of all empty EDIF cells are found in the Spice subcircuits specified by the -ss command line switch. If the parameter is specified, then only specified EDIF cells exist as Spice subcircuits and any empty EDIF cells that are left are translated to Spice black boxes.

  The default behavior (-ss) assumes that cell descriptions not found in the EDIF file(s) are specified in the optional input Spice file.

- **{-a *char1* [*char2*] }**

  Specifies one or two array delimiters to use when expanding port, net, and instance array EDIF names to Spice names. The second array delimiter is optional. The default is “[ ]”. The character “_” is permitted.

- **-b *char***

  Specifies a bundle delimiter to use when expanding portBundle and netBundle EDIF names to Spice names. The default is “_”.

- **{-c *char1* [*char2*] }**

  Specifies one or two name delimiters to use as substitution characters for illegal Spice names generated due to EDIF rename statements. It is also used to generate unique Spice subcircuit names when cell names are duplicated in multiple libraries, or when EDIF array and bundle expansion causes name collisions. Unique names are created by prepending the specified *char* in front of a cell name.

  The default delimiter is “#”. It is substituted in place of the illegal Spice characters “,”, “=” and “$”.The second character delimiter is optional. When specified, it substitutes in place of the “/” character, which can be an illegal Spice character in certain cases. Otherwise, the “/” character is left alone.

- **{-r *char1* [*char2*] }**

  Specifies two bus delimiters to use when expanding a renamed array string. Valid delimiters are “[]”, “< >”, “{}”, and “( )”. Refer to section “Arrays and Bundles” for an example.

- **-i**

  Specifies to ignore names specified in EDIF rename statements. The original EDIF names are used in generating the Spice netlist. Refer to section “Rename and Name Conflict” for an example.

- **-n *cell_name_file***

  Specifies the location of a file containing a list of generated cell names used when duplicated cell names were found during translation.

- **-p**

  Specifies to pass EDIF properties, on instances, as Spice properties. Only integer, number, and string properties are passed through. The converter

ignores other properties and issues a warning. The converter also ignores property options, such as unit, owner, or subproperties. You should be cautious of using this option in hierarchical LVS, because parameters on subcircuit calls in Spice cause flattening of the respective subcircuits.

- **-w *warning_level***

  Controls the amount of warning message output. Possible choices are:

  -w *0* Selects to output no warning messages.

  ⇒ **-w *1*** Selects to output all warning messages.

- **-cb**

  Specifies to use a Calibre CB (caldrclvseve) license.

- **-ictrace**

  Specifies to use an ictrace license.

Examples

```
e2lvs -e design.edif -o design.spi

e2lvs -e reg.edif -s cells.spi -o reg.spi

e2lvs -l edif_files -a '()' -s prim.spi -c '@'
     -o spice_file.spi

e2lvs -e mem.edif -s prim.spi -r '<>' -n new_cell_names
     -o mem.spi
```

Untranslated EDIF Syntax

Many aspects of EDIF syntax are not translated into Spice because they are irrelevant in Spice and to Calibre LVS. In general, only constructs that are applicable to a structural EDIF netlist are translated. Thus, the following EDIF constructs are ignored in a given input file:

- View statements specified with a view type other than NETLIST.
- Cell statements specified with cell type RIPPER or TIE.
- Multiple NETLIST views of a cell.

- *designator*, *property*, *comment*, *userData*, *parameter*, *parameterAssign*, *technology* statements.
- *portInstance* statement.
- Multidimensional arrays.
- Keyword mappings.

EDIF vs. Spice Syntax Considerations

This section describes the following EDIF vs. Spice syntax considerations:

- Identifiers.
- Name scope.
- Arrays and bundles.

Identifiers

EDIF identifiers contain alphanumeric and underscore characters. An identifier must start with an ampersand (“&”) if the first character of the identifier is not a letter. A maximum 255 characters is allowed, exclusive of the ampersand character.

Spice identifiers are made up of any number of alphanumeric characters and have fewer restrictions. Any printable character is valid except leading “$”, “=”, “,” and “/” characters. The “/” restriction is relaxed for $PINS strings as described below. E2LVS supports the following identifiers:

- Embedded and trailing “$”: B$2 or B$
- Embedded and trailing “/” in the $PINS construct: $PINS x=a/a y=a//

The following identifiers are not supported:

- Leading “$”: $B

- "/" in .SUBCKT, pin, instance, and node names, unless explicitly specified with the -c command line switch.

Name Scope

The scope of a name in EDIF is always defined by the statement it is defined in, such as library, cell, and view statements. The name extends from just after its description to just before the end of the smallest enclosing name scope. It is illegal for objects to have the same name if they are in the same scope. For example, no two cells in the same library may be given the same name. However, if they are different scopes, identical names can be used.

The scope of a subcircuit name in Spice is global. However, pin names and instance names within a subcircuit are local to that subcircuit. Because EDIF allows identical names across different scopes, name collision can occur during translation due to Spice's global name scope. Refer to section "EDIF Cell Names Versus Spice Subcircuit Names" for more information.

Arrays and Bundles

Arrays describe a number of objects (net, port, or instance) of the same type with the same name. For example, the statement "(port (array inbus 32) (direction input))" creates 32 input ports named "inbus" that are indexed from 0 through 31. The member statement allows access to each inbus object. For example, the statement "(member inbus 31)" accesses the last port in "inbus".

A *bundle* is a collection of objects that can be referred to by a name. A portBundle is used to collect ports, arrayed ports, and other port bundles together into a group. Ports are collected with the listOfPorts statement. Similarly, a netBundle is used to collect nets together with a listOfNets construct. A netBundle cannot contain other net bundles.

Arrays and bundles are flattened during translation. That is, given an array, the equivalent number of Spice objects are created and named to represent the individual members of the array. This is applicable to all EDIF net, port, and instance arrays.

In the following example, translating the EDIF array “inbus” to Spice involves appending “[*i*]”, where *i* is the array index and “[]” are the default array delimiters:

```
EDIF : (port (array inbus 32) (direction input))
SPICE: inbus[0], inbus[1], … inbus[31]
```

By default, an underscore (“_”) is inserted between the bundle name and the listed items when bundled names are flattened.

The following example shows how a portBundle is translated into Spice:

```
EDIF : (portBundle pbExample
(listOfPorts (port a)
(port b)
(port (array c 3))))

SPICE: pbExample_a, pbExample_b, pbExample_c[0],
pbExample_c[1], pbExample_c[2]
```

Default array and bundle delimiters can be changed by invoking E2LVS with the -a and -b command line switches, respectively. In addition, you can recognize special bus delimiters in the case of “renamed” arrays with the -r command line switch. For example, the statement “(port (array (rename A_8_TO_5 “A<8:5>”) 4))”, would expand to A[8], A[7], A[6], A[5] by invoking E2LVS with the -r command line switch.

If a name collision occurs during array or bundle expansion, the name is made unique by prepending as many “#” characters as necessary. The default collision character, “#”, can be changed with the -c command line switch.

EDIF-to-Spice Translation Issues

This section describes the following EDIF vs. Spice translation issues:

- EDIF cell names versus Spice subcircuit names.
- Rename and name conflict.
- EDIF versus Spice connectivity.

EDIF Cell Names Versus Spice Subcircuit Names

A principal difference between EDIF and Spice is that EDIF utilizes different name scope levels while Spice names are global. In EDIF, a cell is uniquely identified by its library, cell, and view names. In Spice, a subcircuit is uniquely identified by its subcircuit name and the number of pins. Two issues arise because of this difference:

- How to translate EDIF cells to Spice subcircuits and still preserve each individual EDIF cell.
- How to correlate the Spice subcircuit names when invoking E2LVS with the -ss or -sb command line switch.

The first issue can be resolved by mapping the EDIF cell name to a Spice subcircuit name. In the example below the Spice subcircuit name becomes “fcell”:

```
(library cmoslib
   (cell fcell
      (view my_netlistview (viewType NETLIST)
      …
)))

.SUBCKT fcell …
```

The following conditions are necessary for successful translation:

- Cells must be a single view of type NETLIST. If there are multiple NETLIST views, a warning is issued and all views after the first NETLIST view are ignored.
- Most EDIF cell names are unique across different EDIF libraries.

E2LVS produces a Spice subcircuit name *#cellname* if an EDIF cell name appears in multiple libraries. E2LVS also issues warnings to indicate the name conflict and subsequent translation to a new name. The translator writes this information out to a file named “e2lvs_names”. You can specify a filename other than “e2lvs_names” with the -n command line switch.

The second issue, how to correlate Spice subcircuit names when invoking E2LVS with the -ss or -sb switch, is addressed by translating the Spice subcircuit names directly to EDIF cell names. The following criteria apply:

- EDIF cells either reside inside the EDIF external statement or in an EDIF library where the cell's content in its netlist view is empty.
- Multiple EDIF cells must refer to the same Spice subcircuit, where multiple EDIF cells are defined as those having the same cell names located in different libraries.

Refer to section "Cell Statement" for more information about EDIF cell definitions and implementations.

Rename and Name Conflict

The rename statement extends an EDIF name by enabling you to associate a given string with it. For example, in the statement "(rename busA_0 "busA(0)")", the string busA(0) refers to EDIF name busA_0, which is a legal name in Spice.

When a renamed name includes an illegal Spice character, such as "$", E2LVS generates its equivalent Spice name by substituting a "#" in place of the illegal characters. For example:

```
(rename dollarbusB_31 "$busB_31")
```

The translated Spice name for $busB_31 is #busB_31.

In some cases, such as in Spice pin names, the character "/" is valid and may be present to show hierarchy. Thus, the translator leaves the "/" characters alone unless you specify otherwise with the -c command line switch.

All EDIF rename statements are used in the generated Spice netlist unless the name would be illegal in Spice. The command line -i specifies to ignore the rename statement and the original EDIF name is used in the generated Spice netlist.

For example, the statement “(port (rename bus22 “bus(22)”))” generates a Spice pin name “bus22” when -i is specified. Otherwise, the renamed names (such as “bus(22)” above) are used in the generated Spice netlist.

EDIF Versus Spice Connectivity

EDIF ports and nets within a cell can be connected to one another or to ports of other cells with the joined statement. These connection are represented as follows in the Spice netlist:

```
*.J <net> ==<port>
*.J ==<port1> ==<port2> ==<port3>
```

Ports are preceded by “==” to distinguish then from nets.

EDIF-to-Spice Translations

This section describes how various EDIF statements are translated directly into Spice. The following statements are covered:

edif	cell
status	instance
port and portBundle	joined
net and netBundle	rename

Edif Statement

The edif statement contains all the hierarchy and design information transmitted within a file. The edif statement syntax is as follows:

```
(edif edifFileNameDef edifVersion edifLevel keywordMap
  {<status>|external|library|design|comment|userData}
```

E2LVS ignores the *design*, *comment*, and *userData* parameters. The substructures *edifFileNameDef*, *edifVersion*, *edifLevel*, and *keywordMap* are required for reading an EDIF file. They are translated into comments in the generated Spice netlist.

Valid *edifVersion*, *edifLevel*, and *keywordMap* parameters that provide version information are:

```
(edifVersion 2 0 0)
(edifLevel 0)
(keywordMap (keywordLevel 0))
```

Unsupported specifications cause E2LVS to terminate with an error message.

The *status*, *external*, *library*, *design*, and *comment* parameters embody the actual design data and can be specified in any order. Not all of them need be present in a given edif statement. The following example shows how version, level, keyword and status information is translated to Spice comments:

```
(edif Prototype
   (edifVersion 2 0 0)
   (edifLevel 0)
   (keywordMap (keywordLevel 0))
   (status (written (timeStamp 1987 6 1 12 00 00)
      (program “EDIF_NETLIST_OUT”)))
   (external Standard_Cells …)
   (library VHSIC_CMOS …)
   (design barrel_shifter
      (cellRef b_shift (libraryRef VHSIC_CMOS)))
)
Spice Translation:
* edif Prototype
* edifVersion 2 0 0
* WRITTEN:
* timestamp 1987 6 1 12 00 00
* program EDIF_NETLIST_OUT
…
```

Status Statement

The status statement can appear in a variety of EDIF sections. It provides historical information about the edif file. The status statement syntax is as follows:

```
(status {written|comment|userData})
```

E2LVS ignores the *comment* and *userData* parameters. The *written* parameter is translated into a comment in the generated Spice netlist. There can be multiple written statements. The syntax is as follows:

```
(written timeStamp

{<author>|<program>|<dataOrigin>|property|comment|userData}
```

E2LVS ignores the *property*, *comment*, and *userData* parameters. The *timeStamp* parameter is required. The *author*, *program*, and *dataOrigin* parameters are translated into comments in the generated Spice netlist.

The example below shows how a written statement is translated into Spice:

```
(written
   (timeStamp 1986 11 30 22 7 29)
   (author “J.Smith”)
   (dataOrigin “Liverpool”)
   (program “EdifWriter”))

Spice Translation:
*WRITTEN:
* timestamp 1986 11 30 22 7 29
* author J.Smith
* dataOrigin Liverpool
* program EdifWriter
```

Port and PortBundle Statements

Ports are the basic means of communicating signal information between cells. A port array is declared by prefixing the port name with the reserved word “array”. The port statement syntax is as follows:

```
(port portNameDef
   {<direction>|<unused>|<designator>|<dcFaninLoad>
<dcFanoutLoad>|

<dcMaxFanin>|<dcMaxFanout>|portDelay|property|comment|userDat
a}]
```

E2LVS ignores all parameters except *portNameDef*. The *portNameDef* parameter is translated to .SUBCKT pin names.

Port arrays are expanded into their individual elements and translated to pins in a subcircuit. E2LVS generates pin names with indices from 0 to *n*-1, where *n* is the size of the array, as shown below:

```
(port (array c 3))

Spice Translation:
c[0] c[1] c[2]
```

A portBundle is used to collect ports, arrayed ports, and other bundles of ports into a group. The syntax is as follows:

```
(portBundle portNameDef listOfPorts
  {property|comment|userData}]
```

E2LVS ignores the *property*, *comment*, and *userData* parameters. portBundle statements are expanded into individual pins when translated to Spice, as shown below:

```
(portBundle pbExample
  (listOfPorts (port a) (port b) (port (array c 3))))
  …
  (portRef (member c 1) (portRef pbExample))

Spice Translation:
pbExample_a pbExample_b pbExample_c[0] pbExample_c[1]
pbExample_c[2]
```

Net and NetBundle Statements

The net statement syntax is as follows:

```
(net netNameDef joined

{<criticality>|netDelay|figure|net|instance|commentGraphics|
     property|comment|userData}
```

E2LVS ignores all parameters except *netNameDef*, *joined*, *net*, and *instance*. They are translated to node names in subcircuit calls. The *joined* and *instance* statements are translated by E2LVS. Refer to the sections "Joined Statement" and "Instance Statement" for more information.

In Spice, connections are established by translating pin names to node or net names. E2LVS translates net names to node names in a subcircuit call or to *.J statements only when nets include joined statements. If *netNameDef* is an array, then the net is expanded into its bits.

netBundles are expanded and individual net names are translated to Spice node names. Please refer to the section "Arrays and Bundles" for more information. The syntax is as follows:

```
(netBundle netNameDef listOfNets
  {figure|commentGraphics|property|comment|userData}
```

E2LVS ignores all the netBundle parameters except *netNameDef* and *listOfNets*.

Cell Statement

The cell statement syntax is as follows:

```
(cell cellNameDef cellType
  {<status>|view|<viewMap>|property|comment|userData}
```

E2LVS ignores all parameters except for *cellNameDef* and *cellType*. The cell is translated into a subcircuit call

```
.SUBCKT cellNameDef pin1, pin2 …
*.XPINS
```

where the pins are defined by the ports specified in the interface statement. The *.XPINS call is used to indicate that pin connections for this Spice cell are specified explicitly through Spice *.J statements. Refer to section "Joined Statement" for more information.

The *cellNameDef* parameter names the cell and translates it to a subcircuit name. The *cellType* parameter defines the cell's use. Only GENERIC cells are translated.

A cell name in EDIF must be unique within each library or external definition. If two cells with the same name are completely defined with a content statement, but they appear in different libraries, the language implies no relationship between the cells. These cells are unique because they exist in different libraries. Because subcircuit names are global in Spice, EDIF cells with the same name produce a

warning message from the translator. During E2LVS translation the first cell name is used, and any cells bearing the same name are prepended with as many "#" characters as necessary to generate unique names.

If an EDIF cell's netlist view had no contents, just an interface, then E2LVS assumes that the cell is implemented external to the input EDIF files as a primitive cell definition in Spice. In this case, E2LVS translates cells with the same names and does not issue warning messages.

A cell can be instantiated in other cells to build a design hierarchy. However, an instantiated cell must have been defined earlier in the file or declared earlier in an external library. Hence, the necessity for EDIF files to be in the correct order for input into E2LVS with the -l command line switch.

Instance Statement

The EDIF instance statement specifies to include a copy of a previously defined cell view. The instance statement syntax is as follows:

```
(instance instanceNameDef viewRef | viewList
  {<transform>|parameterAssign|<designator>|portInstance
    timing|property|comment|userData}]
```

E2LVS ignores all parameters except for *instanceNameDef* and *viewRef*. The instance is translated to a Spice subcircuit call "XinstanceNameDef *cellnameDef*", where *cellnameDef* is the name of the Spice subcircuit.

An instance statement in EDIF translates to a subcircuit call in Spice. The *instanceNameDef* parameter can be an array structure, in which case the instance arrays are expanded to individual subcircuit calls using the array naming mechanism mentioned in section "Arrays and Bundles". The subcircuit call uses the $PINS *pin=node* construct. Pin names come from the port names in the interface of the cell's netlist view. The nodes are picked up through the net and joined statements.

The *viewRef* parameter references a defined cell view (as defined by the view statement) thus explicitly pinpointing the cell that is being instantiated.

Joined Statement

The joined statement is used to specify ports that are connected together. The joined statement syntax is as follows:

```
(joined {portRef|portList|globalPortRef}
```

E2LVS translates all parameters. Joined statements are translated to *.J statements.

When used in an interface statement, joined refers to ports defined in the interface. That is, pins in the .SUBCKT definition. When used in a net statement, the net name is treated as a node name in Spice and seen either as nodes in a subcircuit or in a *.J statement.

When port arrays, lists, or bundles are specified in a joined statement, the individual members are joined in a parallel manner. The first members of each reference are joined, then the second members, and so on. In all cases, the sizes of the expanded references must be equal.

The *portRef* parameter references a port that has been defined earlier. The *portList* parameter specifies an ordered list of ports that have been defined earlier. The *globalPortRef* parameter is treated as a portRef. That is, as a reference to a previously defined port.

Rename Statement

The rename statement allows user-defined names to refer to EDIF names. This construct is useful because EDIF identifiers are restricted to alphanumeric characters. The syntax is as follows:

```
(rename identifier|name|stringToken|stringDisplay
```

E2LVS ignores the *name* and *stringDisplay* parameters. By default, the *stringToken* parameter is used in the Spice netlist. When *stringToken* includes illegal Spice characters, a warning message is issued and “#” is substituted in place of the illegal characters. Sometime the original EDIF name can cause a conflict with a rename construct. In this case, as mentioned earlier for name

collisions during array and bundle expansion, the collided name becomes unique by prepending as many “#” characters as necessary.

The *stringToken* parameter has a special value when the rename statement refers to an array and the -r command line switch is specified. Refer to section “Arrays and Bundles” for more information.

If the -i command line switch is specified, then the rename statement is ignored and the original EDIF name or the *identifier* is used in the generated Spice netlist. The *identifier* is made up of alphanumeric or underscore characters. An identifier must be preceded with an “&” if the first character is not a letter.

The example below defines a port named “portA(0)” instead of the EDIF name “portA0”. By default, E2LVS creates a Spice netlist using the renamed name of **“**portA(0)”. If the -i command line switch is specified, the Spice netlist will contain the original name, “portA0”.

```
(port
   (rename portA0 “portA(0)”)
   (direction Input))
```

Netlist Example

This section provides a sample EDIF netlist and its Spice equivalent, as translated by E2LVS.

Assume the following EDIF netlist:

```
(edif DicTracy
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap (keywordLevel 0))
  (status(written
     (timestamp 97 12 09 10 49 26)
     (program “XLDF2EDIF  X8830”(Version “V00.03”))))
  (external CellLib
   (edifLevel 0)
   (technology(numberDefinition))
   (cell GCG001
     (cellType GENERIC)
     (view LOGIC(viewType NETLIST)
       (interface
         (port P2(direction INOUT))
         (port P1(direction INOUT)))
     )
   )
   (cell GCG002
     (cellType GENERIC)
     (view LOGIC(viewType NETLIST)
       (interface
         (port P3(direction INOUT))
         (port P1(direction INOUT))
         (port P2(direction INOUT)))
     )
   )
  )
  (library Logic
   (edifLevel 0)
   (technology(numberDefinition))
   (cell (rename b31BLK_h0 “b31BLK-0”)
     (cellType GENERIC)
     (comment
      “A FILE:b31BLK-0 CDATE:970714 UDATE:970714 USEQ:000
ATRB:IBLK JISSOU:JSK1”
```

```
       “B MODEL:MDL1”
       “PE LOC:VAS”
       “FILE:0 CELL:2 NET:5 PORT:6”
      )
      (view LOGIC(viewType NETLIST)
        (interface
          (port (rename P_q1 “P?1”)(direction INOUT))
          (port (rename P_q2 “P?2”)(direction INOUT))
          (port (rename P_q3 “P?3”)(direction INOUT))
          (port (rename P_q4 “P?4”)(direction INOUT))
          (port (rename P_q5 “P?5”)(direction INOUT))
          (port (rename P_q6 “P?6”)(direction INOUT))
        )
        (contents
          (instance In1
            (viewRef LOGIC (cellRef GCG002(libraryRef
CellLib))))
          (instance In2
            (viewRef LOGIC (cellRef GCG001(libraryRef
CellLib))))
          (net i1
            (joined
              (portRef P1(instanceRef In1))
              (portRef P_q1)
              (portRef P_q4)))
          (net i2
            (joined
              (portRef P2(instanceRef In1))
              (portRef P_q2)))
          (net i3
            (joined
              (portRef P1(instanceRef In2))
              (portRef P_q3)))
          (net o2
            (joined
              (portRef P3(instanceRef In1))
              (portRef P_q5)))
          (net o3
            (joined
              (portRef P2(instanceRef In2))
              (portRef P_q6)))
        )
      )
```

```
   )
(cell (rename b31REG_h0 “b31REG-0”)
     (cellType GENERIC)
     (comment
      “A FILE:b31REG-0 CDATE:970714 UDATE:970714 USEQ:000
ATRB:IREG JISSOU:JSK1”
      “B MODEL:MDL1”
      “PE LOC:VAS”
      “FILE:1 CELL:0 NET:7 PORT:6”
     )
     (view LOGIC(viewType NETLIST)
       (interface
         (port P1001(direction INOUT))
         (port P1002(direction INOUT))
         (port P1003(direction INOUT))
         (port P2001(direction INOUT))
         (port P2002(direction INOUT))
         (port P2003(direction INOUT))
       )
       (contents
         (instance B1
           (viewRef LOGIC (cellRef b31BLK_h0)))
         (net ib1
           (joined
             (portRef P1001)
             (portRef P_q1(instanceRef B1))))
         (net ib2
           (joined
             (portRef P1002)
             (portRef P_q2(instanceRef B1))))
         (net ob1
           (joined
             (portRef P2001)
             (portRef P_q4(instanceRef B1))))
         (net ob2
           (joined
             (portRef P2002)
             (portRef P_q5(instanceRef B1))))
         (net ob3
           (joined
             (portRef P2003)
             (portRef P_q6(instanceRef B1))))
         (net GND
```

```
            (joined
              (portRef P_q3(instanceRef B1))))
        )
      )
    )
(design TOP
(cellRef b31REG_h0 (libraryRef Logic))
)
)
```

Note that CellLib is external, implying the descriptions for those cells must come from a Spice file. Assume E2LVS was invoked as follows:

```
e2lvs -e reg.edif -s cells.spi -o reg.spi
```

Rename statements are processed because the -i command line switch was not specified. The generated Spice netlist located in file reg.spi is:

```
*
* Translated by e2lvs(version 0.0) on April 22, 1998
*
.INCLUDE cells.spi
*
* edif DicTracy
* edifVersion 2 0 0
* edifLevel 0
* keyWordLevel 0
* status
* written
* timestamp 97 12 09 10 49 26
* program “XLDF2EDIF  X8830”
* Version “V00.03”


.SUBCKT b31BLK-0 P?1 P?2 P?3 P?4 P?5 P?6
*.XPINS
XIn1 GCG002 $PINS P1=i1 P2=i2 P3=o2
XIn2 GCG001 $PINS P1=i3 P2=o3
*.J i1 P?1
*.J i1 P?4
*.J i2 P?2
*.J i3 P?3
*.J o2 P?5
```

```
*.J o3 P?6
.ENDS b31BLK-0


.SUBCKT b31REG-0 P1001 P1002 P1003 P2001 P2002 P2003
*.XPINS
XB1 b31BLK-0 $PINS P?1=ib1 P?2=ib2 P?3=GND
P?4=ob1 P?5=ob2 P?6=ob3
*.J ib1 P1001
*.J ib2 P1002
*.J ob1 P2001
*.J ob2 P2002
*.J ob3 P2003
.ENDS b31REG-0
```

Verilog-to-LVS

The V2LVS (Verilog-to-LVS) converter translates a Verilog structural netlist into as LVS Spice netlist suitable for Calibre LVS/LVS-H comparison against a layout. The following sections describe its usage in detail. The complete V2LVS BNF is located in Appendix C.

Description

The V2LVS (Verilog-to-LVS) converter translates a Verilog structural netlist into a LVS Spice netlist suitable for Calibre LVS/LVS-H comparison against a layout. This netlist is an extended form of traditional Spice. The section Spice-like Property Syntax describes the netlist format and extensions used in a LVS Spice netlist.

This tool takes three inputs in any order: a Verilog design file (the structural netlist), a Verilog primitive library file, and an optional Spice library file. The Verilog library file generally contains the module definitions that are already implemented in a Spice library file. The Spice library file generally contains transistor-level details of the primitive modules. Multiple Verilog design files or Verilog library files must be concatenated prior to running V2LVS.

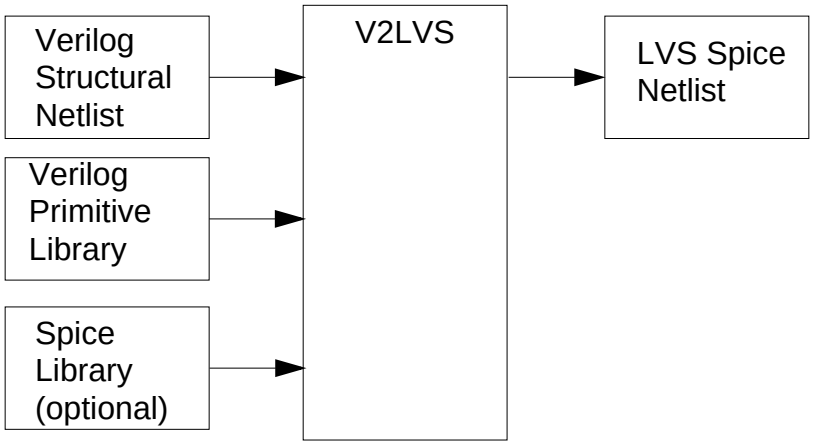


Figure 12-2. V2LVS Flow

The converter translates the Verilog netlist. The Verilog primitive library file is accessed to find the pins associated to nets shown in positional cell instances. If a Spice library file is specified, a .INCLUDE statement referencing the file appears at the beginning of the output LVS Spice netlist.

The optional Spice library read by LVS and the input Verilog primitive library must have identical cell names and the cells in both libraries must have matching terminal names. Only one Verilog primitive library file can be specified. V2LVS supports the ‘include compiler directive, so this library file can contain a set of ‘include directives.

Warning messages for behavioral syntax found in the Verilog netlist and modules containing behavioral syntax are not translated. The Verilog design is assumed to be syntactically correct.

To use the output Spice file as input into Calibre LVS, certain specification statements must be included in the rule file. These statements are identified in the section “Rule File” on page 2-1.

V2LVS searches initially for a calibrelvs license, then for a caldrclvseve license, then for an ictrace license. Command line switches may be used to control the license that gets used for a given run.

Usage

```
v2lvs -v verilog_design_file -o output_spice_file
      [-l verilog_lib_file ] [ -s spice_library_file ]
      [ -s0 groundnet ] [-s1 powernet ] [ -sk ]
      [ -p prefix ] [ -w warning_level ]
      [ -a array_delimiters ] [ -c char1[char2] ]
      [ -u unnamed_pin_prefix ] [ -t svdb_dir ]
      [ -b ] [ -n ] [ -i ] [ -e ] [ -h ]
      [ -cb ][ -ictrace ]
```

Arguments

- **-v *verilog_design_file***

  Specifies the filename of the Verilog structural netlist.

- **-o *output_spice_file***

  Specifies where to place the output LVS Spice netlist. Defaults to standard out.

- **-l *verilog_lib_file***

  Specifies the location of the Verilog primitive library file.

- **-s *spice_library_file***

  Specifies the location of the Spice library file to be included in the output.

- **-s0 *groundnet***

  Specifies the default net name for mapping to pin connections with a value of zero (0). Outputs the specified names in place of Verilog supply0 nets and generates .GLOBAL declarations in the output netlist.

- **-s1 *powernet***

  Specifies the default net name for mapping to pin connections with a value of one (1). Outputs the specified names in place of Verilog supply1 nets and generates .GLOBAL declarations in the output netlist.

- **-sk**

  Specifies that Verilog supply0 and supply1 nets are not connected to the global power and ground nets.

- **-p *prefix***

  Adds *prefix* to Verilog gate level primitive cells.

- **-w *warning_level***

  Controls the amount of warning message output. Possible level choices are:

  0 Selects to output no warning messages.

  1 Selects to output warning messages for skipped blocks and modules only.

  ⇒ ***2*** Selects to output level 1 and calls to undeclared modules and pin arrays with widths wider than ports.

  3 Selects to output level 2 and called port array mismatches and unsupported compiler directives.

  4 Selects output level 3 plus all ignored constructs.

- **-a *array_delimiters***

  Changes the array delimiter characters. The default is "[ ]".

- **-c *char1***[***char2***]

  Sets the substitution characters for escaped identifier characters illegal in Spice. ***char1*** replaces "$", ",", and "=" and ***char2*** replaces "/". No space is needed between the two user-supplied arguments.

- **-u *unnamed_pin_prefix***

  Specifies a prefix to add to unnamed pin connections in module instantiations.

- **-t *svdb_dir***

  Specifies the SVDB database directory to add source netlist pin direction information. This argument is used in xCalibre also.

- **-b**

  Retains the leading backslash for escaped identifiers that are not also legal Verilog identifiers.

- **-n**

  Specifies unconnected pins to receive numbered connections starting with 1000.

- **-i**

  Specifies that calls to subcircuits with pins be done in order according to traditional Spice rather than with $PINS. Refer to the section Using –i to Generate Simulation Output later in this chapter for further information.

- **-e**

  Specifies that empty .SUBCKT definitions are generated for all modules (no instances are translated). This is useful for generating “black box” subcircuits from library files. See Using the –e Switch to Create LVS Box Subcircuits later in this chapter.

- **-h**

  Prints a help message.

- **-cb**

  Specifies to use a Calibre CB (caldrclvseve) license.

- **-ictrace**

  Specifies to use an ictrace license.

V2LVS compiles a structural Verilog design into an equivalent netlist in extended Spice form. The Spice netlist can be then used as input to Calibre LVS by specifying this in the rules file:

```
SOURCE PATH <OutputSpiceFile>
SOURCE PRIMARY <TopCellName>
SOURCE SYSTEM SPICE
```

If a module contains any behavioral syntax, it is assumed that it will be available as a Spice library and it is not translated into Spice (a warning is issued). V2LVS requires only the -v (Verilog design file) switch. The Spice output (-o) is printed to standard out by default. An optional Verilog library file (-l) normally contains primitive Verilog module declarations that are already implemented in a Spice library file (–s). The (-s) spice library file is included (by .INCLUDE statement) at the top of the output Spice file.

Library Files

V2LVS can use a Verilog library file to specify declarations to represent leaf modules that are actually defined in Spice. V2LVS reads the Verilog library file to gather the port interface names that it can then use during instantiation. The Verilog library file may contain full Verilog modules including user-defined primitives and behavioral syntax.

It is also possible to use V2LVS without a Verilog library file. The information contained in the Verilog library file is not strictly necessary for mapping Verilog to Spice. This is discussed later in Using V2LVS Without a Verilog Library File.

Supported Verilog Syntax

This section covers each section of the formal BNF specification for the Verilog language as specified in the IEEE standard (IEEE Computer Society, *IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*, IEEE-STD 1364-1995). It is best read in conjunction with the standard document itself or with another Verilog text (for example, D.E Thomas and P.R. Moorby, *The Verilog Hardware Description Language*, 3rd ed., Kluwer Academic Publishers, Boston, 1996.) The following section details specific translations made for each applicable section of Verilog syntax. A summary BNF derived from the V2LVS Verilog grammar file is included.

Modules

Section F.1 of the formal BNF specification in the IEEE standard describes the top-level syntax for modules, port declaration and the types of declarations that can be made within modules.

Modules and macromodules are supported by V2LVS. Modules are mapped directly into Spice subcircuits. Macromodules are treated identically as modules. If a particular module name is declared more than once, the first declaration is used and subsequent declarations are ignored after a warning is issued.

The list of ports to a module may be represented as simple named identifiers (portnameA, portnameB). When declared this way, they may be called via named connections in a module instance.

Example 1—simple Verilog module to Spice subcircuit

Top level module B creates an instantiation of module A called inst1 passing in the arguments w1, w2, and w3:

```
module A ( in1, in2, out3 );
input in1, in2;
output out3;
endmodule

module B ();
wire w1, w2, w3;
A inst1( .in1(w1), .in2(w2), .out3(w3) );
endmodule
```

is translated into the Spice subcircuit:

```
.SUBCKT A in1 in2 out3
.ENDS

.SUBCKT B
Xinst1 A $PINS in1=w1 in2=w2 out3=w3
.ENDS
```

The list of ports to a module may also be represented using explicit external names.

Example 2—named port declarations in a Verilog module

The module A declares ports INA, INB and OUTC which are connected to in1, in2 and out3 respectively:

```
module A ( .INA(in1), .INB(in2), .OUTC(out3));
input in1, in2;
output out3;
endmodule

module B ();
wire w1, w2, w3;
A inst1( .INA(w1), .INB(w2), .OUTC(w3) );
endmodule
```

is translated to the Spice subcircuit:

```
.SUBCKT A INA INB OUTC
*.CONNECT INA   in1
*.CONNECT INB   in2
*.CONNECT OUTC   out3
.ENDS

.SUBCKT B
Xinst1 A $PINS INA=w1 INB=w2 OUTC=w3
.ENDS
```

Port and bit selections in port references are also supported.

Example 3—port selection, bit selection and array mapping

Module AA uses a Port Selection [2:3] on the input array TH, a bit selection [4] on the input array UH. SH is a simple array that is used intact as an output array.

```
module AA (TH[2:3], UH[4], SH) ;
input [3:0] TH;
input [0:7] UH;
output [3:4] SH;
endmodule

module BB ( );
wire[0:1] w1;
wire w2;
wire [1:2] w3;

AA inst1 ( .TH(w1), .UH(w2), .SH(w3));
endmodule
```

will translate to the Spice netlist:

```
.SUBCKT AA TH[2] TH[3] UH[4] SH[3] SH[4]
.ENDS

.SUBCKT BB
Xinst1 AA $PINS TH[2]=w1[0] TH[3]=w1[1] UH[4]=w2 SH[3]=w3[1]
SH[4]=w3[2]
.ENDS
```

User-defined Primitives

UDP declarations are treated the same way as behavioral module declarations. They are ignored in translation, but they can be called from other modules. UDP declarations are used to specify the calling interface of the underlying module. UDP declarations, like modules containing behavioral statements, are assumed to be available as Spice subcircuits directly and are ignored (after warning) during translation. This is discussed in Section F.5 of the IEEE standard.

Default Parameters

The defparam (default parameter) declarations are ignored.

Declarations

Section F.2 of the IEEE standard describes the various declarations that can be made within a module.The following declaration types are meaningful to V2LVS: input, output, inout, all net types, and parameter (see below).

The following strictly behavioral declaration types are not supported by V2LVS and cause the module they are in not to be translated: reg, time, integer, real, realtime, event, function, and task.

Port declarations—input, output, and inout declarations are used to determine direction information and port array width in module port interfaces. A parameter declaration can be used to specify range width information. Simple arithmetic and logical expressions may be used in specifying range width information.

Net types—V2LVS uses the supply0 and supply1 net types to specify net connections to power and ground nets when numeric values are used for connections. For example, 1’b1 becomes a power connection using the supply1 net name. Multiple nets may be declared for each of the supply0 and supply1 net types. Ranges may also be declared on these nets.

Interaction Between Net Types and –s0, -s1, and –sk Switches

The –s0 and –s1 command line switches cause all numeric values to be overridden by their respective arguments.

Connecting different supply nets together globally—Certain design flows have situations where different power and ground net names are used in different places in the design and the user wishes all of these power and ground nets to be connected together. This typically happens when different naming conventions are used in different places in a design and the user wishes to create one logical net from several names used in different places in the design. The –s0 and –s1 switches can be used to accomplish this.

For example, when the –s1 PWR switch is used, 1’b1 will always be translated to PWR. Use these switches to translate numeric signals into the same net name throughout the design regardless of supply0 and supply1 declarations. Also note that these switches will cause all power nets to be connected to one another and all ground nets to be connected to one another.

Keeping different supply nets separated—Other design flows, most often mixed digital/analog circuits, have the need for separate power and ground nets which are not connected. The –sk switch causes local supply0 and supply1 declarations to continue to take precedence over –s0 and –s1 arguments. It also causes these power and ground nets not to be connected to one another. Use the -sk option in conjunction with –s0, -s1 and local supply0 and supply1 nets to support circuits with multiple power and ground supplies that are to be kept separate. The following tables illustrate a power signal pin translation under various combinations of conditions:

Table 12-1. Power Signal Pin Translation

Power signal pin	Default translation	-s1PWR translation	-s1PWR -sk translation

//no local
//supply1
1’b1
	VDD
	.GLOBAL VDD
		PWR
		.GLOBAL PWR
			PWR
			.GLOBAL PWR

supply1 VDD1; 1’b1	VDD1 .GLOBAL VDD1 .GLOBAL VDD	PWR *.CONNECT VDD1 PWR .GLOBAL PWR	VDD1 .GLOBAL VDD1 .GLOBAL PWR

Table 12-1. Power Signal Pin Translation

Power signal pin	Default translation	-s1PWR translation	-s1PWR -sk translation
Comment	Keep nets separate Use VDD naming	Connect all nets Change default power naming from VDD to PWR	Keep nets separate Change default power naming from VDD to PWR

supply0 and supply1 nets may be used similarly to other wire nets. In addition, the first supply0 net may be used to connect bit values of 0 and the first supply1 net may be used to connect bit values of 1. (See Calling Conventions).

Example 4—use of supply0 and supply1 net types

The following example uses supply0 and supply1 net types within a module:

```
module A ( in1, out1 );
input [0:1] in1;
output [0:1]out1;
endmodule

module B ();
supply1 [0:1] PWR;
supply0 [0:1] GND;
wire [0:1] w1;
A inst1( 2'b01, w1 );
endmodule
```

with no optional switches, it is translated into the following Spice circuit:

```
.SUBCKT A in1[0]  in1[1]  out1[0]  out1[1]
.ENDS

.SUBCKT B
Xinst1 A $PINS in1[0]=GND[0] in1[1]=PWR[0] out1[0]=w1[0]
out1[1]=w1[1]
.ENDS
.GLOBAL PWR[0]
.GLOBAL PWR[1]
```

```
.GLOBAL GND[0]
.GLOBAL GND[1]
```

When the –s1 VDD –s0 VSS switches are used, it translates as follows:

```
.SUBCKT A in1[0]  in1[1]  out1[0]  out1[1]
.ENDS

.SUBCKT B
*.CONNECT VDD PWR[0]
*.CONNECT VDD PWR[1]
*.CONNECT VSS GND[0]
*.CONNECT VSS GND[1]
Xinst1 A $PINS in1[0]=VSS in1[1]=VDD out1[0]=w1[0]
out1[1]=w1[1]
.ENDS
.GLOBAL VDD
.GLOBAL VSS
```

Notice that the *.CONNECT statements connect all power nets to one another and all ground nets to one another. If the –sk switch is used in addition to the –s1 and –s0 switches, the translation is the same except the *.CONNECT statements are omitted and all power and ground nets are made .GLOBAL:

```
.SUBCKT A in1[0]  in1[1]  out1[0]  out1[1]
.ENDS

.SUBCKT B
Xinst1 A $PINS in1[0]=GND[0] in1[1]=PWR[0] out1[0]=w1[0]
out1[1]=w1[1]
.ENDS
.GLOBAL VDD
.GLOBAL VSS
.GLOBAL PWR[0]
.GLOBAL PWR[1]
.GLOBAL GND[0]
.GLOBAL GND[1]
```

vectored and scalared connections are all broken apart into individual pins for translation to Spice, so these keywords are ignored.

Other Net Types

All other net types are handled as wires for the purposes of LVS. Drive strengths, charge strengths and delays are all ignored.

Net Assignment

Net assignment initialization creates a connection between two nets using a *.CONNECT statement in Spice.

Example 5—use of continuous assignment with nets

```
module AA ( OUT1, OUT2 );
output OUT1, OUT2;
assign OUT1=OUT2;
endmodule
```

is converted to the following Spice netlist:

```
.SUBCKT AA OUT1 OUT2
*.CONNECT  OUT1   OUT2
.ENDS
```

Primitive Instances

Section F.3 of the IEEE standard describes the use of primitive instances. Gates are handled similarly to modules.

Ranges of gates are not supported (that is, xor an_xor[0:1] (a, b, c);). If a range is encountered on a gate instance, a warning is issued and the instance is skipped in translating the module.

Primitive gates that are called must be present in the Spice library for each pin combination called. They must have identical pin ordering between the Verilog definition and the Spice definition. They cannot be present in the Verilog library file as modules since the use of a gate type keyword in Verilog will cause a syntax error:

```
module and ( in, out, control ); // causes a syntax error.
endmodule
```

V2LVS will emit calls to gate level modules using the same name and pin ordering as the Verilog gate. Gate instances that do not have names are sequentially numbered by V2LVS (note xor in Example 6 below). Warnings are always issued for instantiation of gate level primitives since these are not usually part of a purely structural netlist.

Example 6—use of Verilog primitive gates

```
module A () ;
wire a, b, c, d, e, f;
and i1 ( a, b, c );
xor (d, e, f );
endmodule
```

is converted into the following Spice circuit:

```
.SUBCKT A
Xi1 a b c and
X0 d e f xor
.ENDS
```

Use of the –p switch to avoid collisions with gate level primitives—The –p switch on the command line may be used to add a prefix to all gate level primitive calls. For example, the above Verilog module will be converted into the following Spice subckt if the command line switch "–p v2lvs_" is used:

```
.SUBCKT A
Xi1 a b c v2lvs_and
X0 d e f v2lvs_xor
.ENDS
```

This can be used to avoid name collisions between Verilog gate level primitives and incompatible Spice library subcircuits that have the same name.

Supported Gate Level Primitives

V2LVS recognizes the following gate level primitives in Verilog: and, nand, or, nor, xor, xnor, buf, bufif0, bufif1, not, notif0, notif1, pulldown, pullup, nmos, pmos, rnmos, rpmos, cmos, rcmos, tran, rtran, tranif0, rtranif0, tranif1, and rtranif1.

Module Instantiations

Section F.4 of the IEEE standard describes the instantiation of other modules within a module. Ranges of modules are not supported (that is, A an_A[0:1] (a, b, c);). If a range is encountered on a module instance, a warning is issued and the instance is skipped in translating the module.

Generally a module translates directly into a subcircuit. See Example 1 for a straightforward example of a module instantiation. Verilog also supports array and concatenation constructs.

Calling Conventions

V2LVS uses the following calling conventions for modules that have arrays in their port interface.

Module port connections can be made with:

- simple arrays or ranges (for example a[0:3])
- binary, hex, octal or decimal bit expressions (for example 2’b01)
- concatenations of the above—these join together ranges or bit expressions into a single array (for example, { a, 2’b01 } represents a concatenation of wire a[0:3] and the bit expression 2’b01)
- multiple concatenations—these cause an integral number of repetitions of the contents of a concatenation (for example { 2 { a, 2’b01 }})

Example 3 shows a simple case of instantiations using ranges. Example 4 shows a simple case of instantiation using bit expressions.

Bit Expressions

Bit expressions take the form n’<t><val> where:

- n is the width of the bit expression
- <t> is one of b, h, o, or d (case does not matter) where b is binary, h is hexadecimal, o is octal, and d is decimal.

- <val> is the value represented by the bit expression.

Bit expressions are supported by V2LVS. Special bit values of x and z are not supported and will cause termination. Bit expressions are converted into Spice pins using the following conventions: by default, pin connections of value 1 are assigned to global net VDD and pin connections of value 0 are assigned to global net VSS.

The V2LVS command line options "–s0" and "–s1" cause a new default value for net connections to 1 or 0. Modules that declare at least 1 supply0 or supply1 net cause connections of value 0 or 1 to be connected to the first declared supply0 or supply1 net respectively. (Note, the –s0 and –s1 options override this assignment and cause the values of 0 and 1 to be connected to global ground and power nets.)

Example 7—module conversion using multiple concatenation

In this example, module B creates an instantiation of module A called inst1. It passes in a multiple concatenation of a simple array and a bit expression:

```
module A ( in1, out1 );
input[0:7] in1;
output out1;
endmodule

module B ();
wire [0:1]a;
wire b;
A inst1( .in1( { 2 { a, 2'b01 } } ), .out1(b) );
endmodule
```

It is translated into the following Spice netlist. The pins that connect to the bit expression are tied to VSS and VDD. The concatenation joins together wire a and the bit expression. The multiple concatenation causes two repetitions of the contents of the concatenation:

```
.SUBCKT A in1[0] in1[1] in1[2] in1[3] in1[4] in1[5] in1[6]
in1[7] out1
.ENDS

.SUBCKT B
```

```
Xinst1 A $PINS in1[0]=a[0] in1[1]=a[1] in1[2]=VSS in1[3]=VDD
in1[4]=a[0]
+ in1[5]=a[1] in1[6]=VSS in1[7]=VDD out1=b
.ENDS
```

Unnamed Concatenation Expressions in Declarations

Example:

```
module A ( {B,C} )
```

The port is unnamed. This module can be instantiated with positional calling only. Attempts to call via named connections will always leave the unnamed port unconnected.

Example 8—unnamed concatenation in module declaration

In the following example module B creates an instance of module A called inst1 in which module A is declared with an unnamed concatenation.

```
module A( {B, C} );
input B, C;
endmodule

module B();
wire [0:1] a;
A inst1( a );
endmodule
```

It is converted to the following Spice subcircuit:

```
.SUBCKT A ##1000[0]  ##1000[1]
*.CONNECT ##1000[0]    B
*.CONNECT ##1000[1]    C
.ENDS

.SUBCKT B
Xinst1 A $PINS ##1000[0]=a[0] ##1000[1]=a[1]
.ENDS
```

Calling Conventions for Mismatched Arrays

In Verilog, it is possible to call instances of modules with array pin arguments that are either wider or narrower than those declared in the interface of the module. This section documents the conventions used to map calling pins to module ports under these circumstances.

When the calling pin expression is wider than the port expression in the module, calling pins are mapped to called ports from the right-most calling pin:

Example 9—calling connection is wider than called ports

For example, in the following Verilog circuit, module B an instance of module A using a bus wire (IN_ARR) that is wider than the declared interface (P) that it connects to.

```
module A ( P,Q );
input [3:0]P;
output Q;
endmodule

module B ();
wire [0:7] IN_ARR;
wire X;
A inst1 (IN_ARR, X );
endmodule
```

In Spice, the module call generated maps the right-most pins of IN_ARR to the instance pins P:

```
.SUBCKT A P[3] P[2] P[1] P[0] Q
.ENDS

.SUBCKT B
Xinst1 A $PINS P[3]=IN_ARR[4] P[2]=IN_ARR[5] P[1]=IN_ARR[6]
P[0]=IN_ARR[7] Q=X
.ENDS
```

A warning is issued when calling pins are wider than the port they are passed through to pins narrower than module interface ports. When the calling pin expression is narrower than the port expression in the module, calling pins are

mapped to called ports from the right-most called port. Remaining ports are mapped to random undeclared nets.

Example 10—calling connection is narrower than called ports

In the following Verilog circuit, module B creates an instance of module A using a bus wire (IN_ARR) which is narrower than the declared interface (P) that it connects to:

```
module A ( P,Q );
input [3:0]P;
output Q;
endmodule

module B ();
wire [0:1] IN_ARR;
wire X;
A inst1 ( .P(IN_ARR), .Q(X) );
endmodule
```

The generated Spice instance right-justifies the IN_ARR pins with the P pins and connects the leftmost P pins to sequential unconnected nets.

```
.SUBCKT A P[3] P[2] P[1] P[0] Q
.ENDS

.SUBCKT B
Xinst1 A $PINS P[1]=IN_ARR[0] P[0]=IN_ARR[1] Q=X
.ENDS
```

Unconnected Pins

Instantiations that leave pins uncalled in Verilog will not appear in the output. Calibre LVS will take care of these unconnected pins. If the –n switch is used, it causes unconnected pins to be connected to sequential undeclared numbered nets. The –i switch also causes V2LVS to generate numbered unconnected pins.

Example 11—unconnected pins

The following Verilog circuit:

```
module A ( B, C, D);
input B, C;
output [1:0] D;
endmodule

module B ();
wire bb,cc;
A anA (.B(bb), .C(cc));
A anA1 ( x, b );
endmodule
```

converts into the following Spice subcircuit:

```
.SUBCKT A B C D[1] D[0]
.ENDS

.SUBCKT B
XanA A $PINS B=bb C=cc [1] D[0]=1001
.ENDS
```

Using the –n switch, this will translate as:

```
.SUBCKT A B C D[1] D[0]
.ENDS

.SUBCKT B
XanA A $PINS B=bb C=cc D[1]=1000 D[0]=1001
.ENDS
```

Behavioral Statements

Section F.6 of the IEEE standard describes the behavioral statements that can be used within modules. All behavioral statements cause the module they are in to be skipped in translation except for the assign statement (continuous assignment). The assign statement can only be used to permanently connect one net to another. In the translated Spice, the assign statement is mapped to a *.CONNECT statement to connect one net to another. Net initialization has the same effect as continuous assignment.

Example 12—assignment and net initialization

The module:

```
module B ( a );
input [0:1] a;
wire [0:1] a;
wire [0:1] w;
wire [0:1] b;
wire [0:1] c;
wire d,e;
wire [0:1]x=b;
assign w=a;
assign { d,e } = c;
A anA ( w[0:1], a[0:1] );
endmodule
```

will translate to the Spice circuit:

```
.SUBCKT B a[0] a[1]
*.CONNECT  x[0]   b[0]
*.CONNECT  x[1]   b[1]
*.CONNECT  w[0]   a[0]
*.CONNECT  w[1]   a[1]
*.CONNECT d c[0]
*.CONNECT e c[1]
XanA w[0] w[1] a[0] a[1] A
XanA1 x[0] x[1] b[0] b[1] A
.ENDS
```

Other behavioral statements including initial, always, deassign, force, release, repeat, posedge, negedge, if .. else .. ifnone, case ... endcase, casez, casex, default, forever, repeat, while, for, wait, disable, begin .. end, and fork … join cause the module they are in to be skipped during translation.

Specify Section

Specify blocks are parsed in V2LVS, but result in no output to a Spice netlist. (The Specify block is used to specify timing information for paths across a Verilog module.)

Expressions

Expressions are discussed in section F.8 of the IEEE standard. Expressions are used throughout the Verilog language. V2LVS supports expressions used for structural description. These include things like identifiers, binary, hex, octal and decimal numbers, port ranges, port selections, bit selections concatenations, multiple concatenations, and so on.

Identifiers—Regular and escaped identifiers are supported. V2LVS supports identifiers of up to 2048 characters in length. Verilog supports mixed case identifiers. By default, Spice identifiers are not case sensitive. If case sensitivity is required during LVS, the Source Case YES specification statement is used in your rule file.

Escaped identifiers—Verilog's escaped identifiers allow any non-white space ASCII character to be used in an identifier. Escaped identifiers are not recommended in the V2LVS flow. They can present problems throughout the flow as each tool tries to deal with the incompatible identifiers in its own way. Debugging is also hindered as various mapped names are used in different places in the flow.

In addition, Spice has a flat namespace that does not include busses. Verilog port and bit selections are mapped to Spice identifiers that contain brackets by default (see –a switch). Use of brackets in escaped identifiers can lead to unpredictable name pairings that are not specified by the Verilog standard. For example, bit 3 of an array A in Verilog is mapped to the Spice identifier A[3]. Most Verilog simulators will not map an escaped identifier \A[3] onto bit 3 of array A. They are required by the Verilog standard to match a regular identifier onto the same escaped identifier (for example, A is the same as \A).

To help provide alternatives for handling identifiers in V2LVS conversion, the –b switch is provided. This will cause the leading "\" character to be retained on escaped identifiers that are not also legal Verilog identifiers. This will have the effect of making sure that bit 3 of A is not the same as \A[3].

Example 13—escaped identifiers and use of the –b switch

```
module DDD ( b, sum, ci );
input  [2:0] b;
```

```
input ci;
output [2:0] sum;
    wire  \b[2] , \b[1] ;
    wire \ci ;
    assign \b[2]  = b[1];
    assign \b[1]  = b[0];
    assign sum[2] = \b[2] ;
    assign sum[0] = \b[1] ;
    A A1 ( .i(\b[2] ), .zn(sum[1]), .ci(\ci ) );
endmodule

module A ( i, zn, ci );
input i, ci;
output zn;
endmodule
```

will translate as follows when the –b switch is not used:

```
.SUBCKT DDD b[2]  b[1]  b[0]  sum[2]  sum[1]  sum[0]  ci
*.CONNECT b[2] b[1]
*.CONNECT b[1] b[0]
*.CONNECT sum[2] b[2]
*.CONNECT sum[0] b[1]
XA1 A $PINS i=b[2] zn=sum[1] ci=ci
.ENDS

.SUBCKT A i zn ci
.ENDS
```

It will translate as follows when the –b switch is used:

```
.SUBCKT DDD b[2]  b[1]  b[0]  sum[2]  sum[1]  sum[0]  ci
*.CONNECT \b[2] b[1]
*.CONNECT \b[1] b[0]
*.CONNECT sum[2] \b[2]
*.CONNECT sum[0] \b[1]
XA1 A $PINS i=\b[2] zn=sum[1] ci=ci
.ENDS

.SUBCKT A i zn ci
.ENDS
```

In V2LVS, most escaped identifiers are mapped directly to Spice identifiers since Spice supports most ASCII characters in its identifiers. Some characters cannot be translated directly to Spice. Leading “$”, “,”, “=”, and “/” are escaped identifier characters which, under certain circumstances, are illegal in Spice.

Leading “$”, “,”, and “=” are mapped to “#”. For example aa$ is mapped to aa#. If multiple identifiers would map to the same name, the second identifier encountered is mapped to the identifier with a “#” prepended. Multiple collisions result in additional “#” characters prepended.

“/” characters are left as is in V2LVS. In certain circumstances, “/” characters will produce undesirable results, however. Any undesired effects from using “/” in escaped Verilog identifier names can be overcome by using the –c switch.

V2LVS also provides the –c switch to change the characters used for substitution. The first character in the string supplied to the switch will replace the characters leading “$”, “,”, and “=” while the second character (if present) will replace “/” characters. For example, if you desire to have leading “$”, “,”, and “=” replaced with “_”, use “–c _”. If you desire to have “/” characters replaced with “#” characters, use “–c ##” on the command line. (The first # character will keep the substitution for leading “$”, “,”, and “=” as “#” while the second “#” will cause “/” characters to be changed to “#” as well).

Integer numbers—Decimal, Hex, Octal, and Binary representations of numbers are supported. V2LVS does not support “x” or “z” in numbers. Translation is terminated if they are encountered.

Concatenated expressions and multiple concatenations—Concatenated expressions and multiple concatenations are supported.

Parameters in expressions—Parameters may be referenced in expressions as long as they return a constant value that can be evaluated in a structural context.

Example 14—use of a parameter value in a structural netlist

```
module A ( inA );
parameter width=8;
input [width-1:0] inA;
endmodule
```

is translated into the following subcircuit:

```
.SUBCKT A inA[7]  inA[6]  inA[5]  inA[4]  inA[3]  inA[2]
inA[1]  inA[0]
.ENDS
```

Unsupported expression handling—Function calls and real numbers are not supported. Ternary expressions (?:) are not supported in any structural application.

Unary and binary expressions—These expressions are often used for behavioral syntax and cause the module they are contained in to be skipped in translation (a warning is issued).

Some binary expressions may be used in port declarations to specify range values. The supported binary expressions include +, -, *, /, %, <, =<, >, >=, ==, !=, &&, ||.

Some binary expressions may be used with bit expressions. The supported binary expressions for use with bit expressions include +, -, <, =<, >, >=, ==, !=, &&, ||.

Other Language Features

V2LVS supports the following compiler directives: `include, `define, `ifdef, `else, `endif, `undef. All other compiler directives are ignored to the end of the line where they appear.

V2LVS does not support the argument form of `define ( `define a(b,c) …).

Using V2LVS Without a Verilog Library File

The information contained in the Verilog library file is not strictly necessary for mapping Verilog to Spice. V2LVS implements a number of heuristic algorithms for mapping undeclared module instances to Spice subcircuits.

Instances of Undeclared Spice Primitive Modules with Named Ports

When a module is instantiated with named ports, connections are made using the $PINS construct supported by Calibre Spice.

Example 15—call to named port of undeclared primitive module

```
spice_module instance_aa ( .A(in1), .B(in2), .C(.in3) );
```

is mapped to the Spice instance:

```
Xinstance_aa spice_module $PINS A=in1 B=in2 C=in3
```

When the instantiated module is called with signal pins that contain arrays, the called port is assumed to be of the same width as the set of signal pins passed in. Furthermore, it is assumed to have pin names starting at n-1 and going to 0.

Example 16—call to pin array in undeclared primitive module

```
wire [3:0] in1;
wire in2, in3;
spice_module instance_aa ( .A(in1), .B(in2), .C(.in3) );
```

will map to the Spice instance:

```
Xinstance_aa spice_module $PINS A[3]=in1[3] A[2]=in[2]
A[1]=in[1] A[0]=in[0] B=in2 C=in3
```

Instances of Undeclared Spice Primitive Modules with Ordered Ports

When a module is instantiated with ordered ports, the order in the Verilog instance is assumed to be the same order as the Spice module definition.

Example 17—positional call to undeclared primitive module

```
spice_module instance_aa ( in1, in2, in3 );
```

is mapped to the Spice instance:

```
Xinstance_aa in1 in2 in3 spice_module
```

In order for these instances to work properly, the Spice module ports must have the same order as the Verilog instance calls. Also, pins must be supplied for each port specified in the Spice subcircuit. LVS will support calls to circuits that have missing pins as long as the call is not ambiguous (that is, more than one subcircuit with the same name and different numbers of pins).

Correcting Errors

When using V2LVS without a Verilog library, there are two situations which can caused undesired results:

- When positional instantiations are used in Verilog and the Spice pin order is not the same as the Verilog pin order, V2LVS has no information with which to make the correct pin connections if a Verilog library is not used.

- When pin arrays are connected to undeclared Verilog modules, V2LVS assumes that the called array's pins are named array_name[n-1] through array_name[0]. If this is not the case, a Verilog library is necessary to show the correct array boundaries on the called pin.

Using the –e Switch to Create LVS Box Subcircuits

The –e switch can be used to generate empty subcircuits from a Verilog library file. This can be useful in conjunction with the LVS Box rule file statement to perform partial comparison of a structural netlist without comparing the low-level circuit descriptions. To make use of this switch, the following steps are generally used:

1. Generate a structural netlist using V2LVS in the usual way:
   v2lvs –v model.v –l lib.vlib –o model.spi

2. Generate a Spice netlist from the Verilog library file or files using the –e switch:
   v2lvs –v lib.vlib –o lib.spi –e

3. Use the LVS Box specification statement for the layout cells that correspond to the subcircuits in the Verilog library file (lib.vlib and lib.spi above).

Using –i to Generate Simulation Output

By default, V2LVS utilizes a Calibre LVS extension ($PINS) to make pin connections. The –i switch can be used to generate standard Spice output that is acceptable to many spice simulators. These may be used in conjunction with

xCalibre generated netlists to perform detailed simulations of critical nets using Spice-based simulators.

Since named connections are not possible in Spice without use of an extension like $PINS, in most situations, a Verilog library (-l) will be necessary when the –i switch is used.

Note that V2LVS is intended to translate Verilog netlists for use with Calibre LVS. The –i switch is provided as a convenience only. V2LVS is not guaranteed to produce Spice output that is suitable for simulation or compatible with any particular simulator.

Also note that V2LVS does not read the Spice library file specified with the –s switch. The –s switch merely instructs V2LVS to issue a .INCLUDE statement at the start of its Spice output. Inconsistencies in pin configuration between the Verilog and Spice libraries are not detected by V2LVS. This type of inconsistency can be detected by LVS, however.

Example 18—using the -i switch for standard Spice output

File src.v:

```
// Verilog source
module top ();
wire w1, w2, w3;
A inst1( .in1(w1), .in2(w2), .out3(w3) );
endmodule
```

File lib.v:

```
// Verilog library
module A ( in1, in2, out3 );
input in1, in2;
output out3;
endmodule
```

File lib.spi:

Spice library with pin order which does not match Verilog library above

```
.SUBCKT A out3 in2 in1
```

```
.ENDS
```

Run v2lvs with default ($PINS) and -i:

```
v2lvs -v src.v -o src_without_pins.spi -l lib.v -s lib.spi -i
v2lvs -v src.v -o src_with_pins.spi -l lib.v -s lib.spi
```

Rule file:

```
// This LVS Rule file Compares $PINS output with -i output
SOURCE PATH src_with_pins.spi
SOURCE PRIMARY top
SOURCE SYSTEM SPICE

LAYOUT PATH src_without_pins.spi
LAYOUT PRIMARY top
LAYOUT SYSTEM SPICE

LVS REPORT lvs.rep
```

Execute calibre -lvs rules

The run comes up INCORRECT showing the mismatched connections on pins out3 and in1.

Generating an xCalibre Source Template File

The –t *svdb* switch may be used to generate a file which xCalibre can use to determine directions of ports declared in the Verilog netlist.

When the –t switch is used, V2LVS will generate a file named *svdb*/template/%source.stl.

This file will contain a single starting line which consists of:

```
%%SOURCE
```

followed by a .stl format template for each module in the Verilog netlist. The module template consists of a module name declaration line:

```
%  <module_name> <module_name>
```

followed by one line per interface pin:

```
<pin_name> <pin_name> 0 <io_spec>
```

Where <pin_name> is the expanded pin name used in the spice netlist output (that is, busses are expanded and <io_spec> is one of i (input), o (output), io (inout).

For example, the Verilog netlist

```
module A ( X, Y, Z);
input X;
output Y;
inout [0:1] Z;
endmodule

module B(V,W );
output V;
input W;
endmodule
```

produces a source template file:

```
%%SOURCE
% A A
X X 0 i
Y Y 0 o
Z[0] Z[0] 0 io
Z[1] Z[1] 0 io
% B B
V V 0 o
W W 0 i
```

xCalibre can read the %source.stl file in the same way that it reads individual layout .stl files when they are available. See xCalibre documentation for additional information regarding the use of –t and the svdb database.

Dracula: File Conversion and User Notes

Converting Dracula Command Files

This section describes how to convert a Dracula command file into a rule file from the command line.

From the Command Line

To convert a Dracula command file into a rule file from the command line, use the following procedure:

1. Determine where your Dracula command file is and where you want to put the resulting rule file.

2. Enter the following on the command line:

   ```
   $MGC_HOME/bin/drac_cvt sourcefile destpath
   [-nodrac] [-text text_include_file]
   ```

   where *sourcefile* is the pathname of the Dracula command file you want converted and *destpath* is the pathname of the rule file that you want created.

   Invocation of drac_cvt with no arguments displays a usage statement and the date of the binary build. Use this date to identify the version you are using.

3. Use an ASCII text editor to view the rule file.

   If you find incorrect operations, use the ASCII text editor to edit the rule file. The type of editing depends on the Dracula commands that were converted. Note that the Dracula commands are added as comments next to their rule file counterparts.

Dracula User Notes

This section describes differences between Dracula and Mentor Graphics DRC applications that you should be aware of when transitioning to these applications.

Acute Angles

Acute angles between adjacent edges in the same polygon are flagged automatically by Dracula in one-layer DRC commands unless a range-type measurement is being used. Two-layer acute angle overlaps are also flagged automatically by Dracula in two-layer DRC commands unless, again, a range-type measurement is being used. In DRC, measurement and output at acute angles must be *explicitly requested* in a dimensional check operation using the appropriate form of the ABUT keyword. (Acute angles within single layers are normally flagged by the Flag Acute statement or by the Drawn Acute operation.

Connects

DRC does not have a counterpart to the Dracula CONNECT-LAYER command. Any shielding is implicit in the order of the input layers in a Connect ... BY layer operation.

DRC allows any number of layers in a Connect operation. Dracula allows only three. Two or more Dracula CONNECT commands can be combined into one DRC Connect operation. For example, the Dracula CONNECT commands:

```
CONNECT  met1  poly BY  contact
CONNECT  met1  psd  BY  contact
CONNECT  met1  nsd  BY  contact
```

can be combined into the DRC Connect operation:

```
CONNECT  met1  poly  nsd  psd  BY  contact
```

Dracula only has the connect by contact form. Many times this requires that “pseudo-contact” layers be generated. Since DRC has a “direct connection” operation, these “pseudo-contacts” are normally not required. For example, the Dracula commands:

```
AND  nsd  nwell  nwcont
AND  psd  psub  pscont
CONNECT  nsd  nwell  BY  nwcont
CONNECT  psd  psub  BY  pscont
```

are equivalent to the DRC operations:

```
CONNECT  nsd  nwell
CONNECT  psd  pwell
```

Dracula does not establish connectivity in the contact layer in a Connect command. A Stamp command is necessary to do this. DRC will establish connectivity in any layer parameter to a Connect operation.

Description Statements

- CNAMES-CSEN

  This command is ignored since there is no accurate translation to SVRF.

- ABORT-P-G-SHORT

  The following commands:

  ```
  ABORT-P-G-SHORT = YES
  ABORT-P-G-SHORT = ALL
  ABORT-P-G-SHORT = SHORT
  ABORT-P-G-SHORT = OPEN
  ```

  are translated to:

  ```
  LVS ABORT ON SUPPLY ERROR YES
  LVS POWER NAME VCC VDD
  LVS GROUND NAME VSS GND GROUND
  ```

  Note that these are not necessarily the power/ground names in the design, but these are the names that Dracula LVS uses to identify nets that may be involved in a power-ground short. You may need to edit the LVS Power Name and LVS Ground Name statements to find shorts in a design.

  The following Dracula command is ignored:

  ```
  ABORT-P-G-SHORT = NO
  ```

- POWER-NODE and GROUND-NODE

  POWER-NODE and GROUND-NODE are translated to LVS Power Name and LVS Ground Name operations, respectively.

  The following commands:

  ```
  POWER-NODE  = VDD, DVDD, AVDD, PVDD
  GROUND-NODE = GND, DGND, AVSS, PVSS
  ```

  are translated to:

  ```
  LVS POWER NAME “VDD” “DVDD” “AVDD” “PVDD”
  LVS GROUND NAME “GND” “DGND” “AVSS” “PVSS”
  ```

  Unlike Dracula, SVRF does not allow the use of a wildcard character for power and ground names used in the analogous commands. Hence, if a wildcard character is used in the Dracula POWER-NODE or GROUND-NODE statements, a warning is issued. For example, the Dracula command:

  ```
  POWER-NODE  = VDD, VCC, VAB*
  ```

  is translated to:

  ```
  LVS POWER NAME VDD VCC
  // Warning! Unused because Wildcard not accepted: VAB*
       (near line N in file X)
  LVS POWER NAME “VDD” “VCC”
  ```

  Dracula allows you to use suffixes to designate power (:P) and ground (:G) nodes. SVRF has no such convention, therefore all suffixes are removed in translation. For example, the Dracula command:

  ```
  POWER-NODE  = VCC, VDD:P
  ```

  is translated to:

  ```
  LVS POWER NAME “VCC” “VDD”
  ```

Device Filtering

Dracula filters out devices depending on the configuration of a given device, by way of the LVSCHK statement. This is a complex statement with many options. The analogous rule in SVRF is LVS Filter Option.

The default filtering statement:

LVS FILTER OPTION AB RC RE RG

mimics the default Dracula filtering. It is added to the output of all translated Dracula LVS files, *triggered* by the presence of either a CONNECT-LAYER or a TEXTSEQUENCE statement in a Dracula command file.

HEDTEXT files

The contents of Dracula EDTEXT and HEDTEXT files are translated into Text and Layout Text specification statements, respectively.

By default, the converted EDTEXT and HEDTEXT files go directly into the translated rule file.

When you specify -text *text_include_file* on the command line, the converted EDTEXT and HEDTEXT files go into the specified *text_include_file*. The translated rule file will contain the statement:

INCLUDE "*text_include_file*"

Input Statements

- CONNECT-LAYER and TEXTSEQUENCE

  CONNECT-LAYER and TEXTSEQUENCE commands specify texting layer order and are commonly used in Dracula LVS. They list the chip layers from bottom to top. For example:

  CONNECT-LAYER NSD PSD POLY METAL
  TEXTSEQUENCE NSD PSD POLY METAL

  One of the two is required to order texting layers in Dracula. If a TEXTSEQUENCE statement is not present in a rule file, Dracula uses the

CONNECT-LAYER statement, which is always present in Dracula LVS files. This statement establishes the order of text attachment when several layers exist under free-floating (unattached) text. For the above examples, the translation to SVRF is:

```
LABEL ORDER METAL POLY PSD NSD
```

Note that the layer order is reversed in this translation, as required for the Label Order specification statement in SVRF.

Operation Statements

- ANDNOT

  The ANDNOT command is translated as in the following example:

  ANDNOT A B C D

  is translated to:

  C = A AND B
  D= A NOT B

- CORNER

  The Dracula file converter translates the CORNER operation to an equivalent sequence of SVRF commands.

  The CORNER statement:

  ```
  CORNER{[A|B|C]} input_layer rel_a rel_b {CORNER_SIZE n}
   output_section
  ```

  where:

  - **rel_a**: is translated as INSIDE | OUTSIDE
  - **rel_b**: is translated as INNER | OUTER
  - **n**: when greater than 0.0 is translated as the corner size (optional)

- **output_section**: is translated as output_layer | output_clause (or both)
- **output_clause**: is OUTPUT c_name l_num {d_num}

- ELEMENT MOS

  ELEMENT MOS {[<type>]} is translated to DEVICE M, MN, MP, MD, or ME depending on the optional code <type>. In the table below, * means a string of 0 or more characters (not whitespace)

  ```
  <type>DEVICE

  N*   MN(N*)
  P*   MP(P*)
  D*   MD(D*)
  E*   ME(E*)
  other M(<type>)
  no type M
  ```

  ELEMENT LDD {[<type>]} is translated to DEVICE LDD, LDDN, LDDP, LDDD, or LDDE depending on the optional code <type>.

  ```
  <type> DEVICE

  N*   LDDN(N*)
  P*   LDDP(P*)
  D*   LDDD(D*)
  E*   LDDE(E*)
  other LDD(<type>)
  no type LDD
  ```

- COVERAGE

  COVERAGE is translated to the Density operation in SVRF. For example, the following Dracula commands:

  ```
  COVERAGE POLY LT 0.6 100 10 POLYERR
  COVERAGE POLY LT 0.6 100 10 LAYER ACTIVE POLYERR1
  ```

  are translated, respectively, to the SVRF commands:

  ```
  POLYERR = DENSITY POLY < 0.6 WINDOW 100 STEP 10
  POLYERR1 = DENSITY POLY < 0.6 INSIDE OF LAYER
                                ACTIVE WINDOW 100 STEP 10
  ```

- ECONNECT and NDCOUNT

  ECONNECT and NDCOUNT commands are translated to the Density operation in SVRF. For example, the following Dracula commands:

```
ECONNECT MOS[N] NDIFF CONN ?:P OUTPUT ENCV 49

NDCOUNT MOS[P] PSRCDRN GT 2 OUTPUT PSDX 42
```

  are translated, with warnings, to the following SVRF commands, respectively:

```
// ***** Warning! Use ERC SELECT CHECK to select the desired ERC
// related check

ENCV49 {
    ( DEVICE LAYER MN(N) ) INTERACT ( NET NDIFF "?:P" )
}

// ***** Warning! Use ERC SELECT CHECK to select the desired ERC
// related check

PSDX42 {
    ( DEVICE LAYER MP(P) ) INTERACT PSRCDRN > 2 BY NET
}
```

  Another example shows that the following Dracula commands:

```
ECONNECT MOS[N] nsd CONN vdd &
ECONNECT MOS[N] nsd CONN vss OUTPUT vdvsn 48
```

  are translated to the warning and commands:

```
// ***** Warning! Use ERC SELECT CHECK to select the desired ERC
// related check

vdvsn48 {
    ( ( DEVICE LAYER MN(N) ) INTERACT ( NET nsd "vdd" ) )
                                        INTERACT ( NET nsd "vss" )
}
```

  Note that the following limitations cause ECONNECT or NDCOUNT commands to be ignored:

  - An ECONNECT or NDCOUNT command with layer ALL is not translated.

- An elem parameter PAD is not translated.

- LCONNECT

  LCONNECT is translated to Net or Not Net for SVRF. The syntax for LCONNECT is:

  LCONNECT *layer* CONN/DISC *label trapfile* OUTPUT *c-name l-name d-name*

  where *layer*, CONN or DISC, *label*, *c-name*, and *l-name* are used in the translation. The optional parameter *d-name* is used in translation, if present, and *trapfile* is ignored.

  LCONNECT is used to select geometry on a given layer depending on whether that geometry is labelled (CONN for “connected”) or not labelled (DISC for “disconnected”) with a specified text string. The Dracula output layer designators *c-name*, *l-name*, and *d-name* are used to form the SVRF rule name, as shown below:

  The following command:

  LCONNECT diff DISC vcc OUTPUT difvcc 40

  is translated to:

  difvcc40 { NOT NET diff vcc }

  The following command:

  LCONNECT diff CONN vcc layer_x OUTPUT difvcc 40

  is translated to:

  difvcc40 { NET diff vcc }
  layer_x = NET diff vcc

  The following command:

  LCONNECT diff DISC vcc layer_x

  is translated to:

  layer_x = NOT NET diff vcc

- LINK

  This is a dangerous Dracula statement. It is used to attach a label to ALL pieces of geometry on a specified layer. It is commonly used to label wells or the chip bulk. This is why it is dangerous. Stamping should be used to label substrates, so that multiple stamped substrates —which may be the path for an electrical short — can be found. LINK bypasses this important substrate integrity-checking mechanism.

  It is important to create a rule file that verifies substrate connectivity to find shorts.

  The syntax for LINK is:

  LINK *layer* TO *label*

  For example, the following command:

  LINK pwell TO VSS

  is translated to:

  CONNECT 901
  SCONNECT 901 pwell LINK “VSS”

  A second LINK statement in the same Dracula rule file, such as:

  LINK nwell TO VDD

  is translated to:

  CONNECT 902
  SCONNECT 902 nwell LINK “VDD”

- LVSCHK

  If there are Element statements that translate to capacitor devices, and CAPVAL is specified, then Trace Property statements are produced for each Device C(X).

If there are Element statements that translate to diode devices, and DIOAREA or DIOPERI are specified, then Trace Property statements are produced for each Device D(Y).

If there are Element statements that translate to resistor devices and RESVAL is specified, then Trace Property statements are produced for each Device R(Z).

If there are Element statements that translate to MOS/LDD devices (MN, MP, MD, ME, M LDDN, LDDP, LDDD, LDDE, and LDD), and LPERCENT or WPERCENT is specified, then Trace Property statements are produced for each Device MN / MP /MD / ME / M / LDDN / LDDP / LDDD / LDDE / LDD(X). If WEFFECT is specified in the LVSCHK command, its value is used in MOS/LDD DEVICE statements; otherwise, the default 0 is used.

For example, the following commands:

```
ELEMENTMOS[NNGATE CPOLY NDIFF PSUB
ELEMENTMOS[P]PGATE CPOLY PDIFF NXWELL
ELEMENTDIO[DP]FUSE PDIO NDIO
ELEMENTRES[MR]MTRES MT1
PARAMETERRES[MR]1.00
ELEMENTRES[WR]RWELL NXWELL
PARAMETERRES[WR]1200.00
ELEMENTCAP[PC]CAPPL CPOLY POLY2
PARAMETERCAP[PC]4.6E-16
LVSCHK[..] LPERCENT=2 WPERCENT=3 CAPVAL=4 RESVAL=5
    DIOAREA=6 DIOPERI=7
```

are translated to:

```
DEVICE MN(N) NGATE CPOLY NDIFF NDIFF PSUB [0]
TRACE PROPERTY MN(N) L L 2
TRACE PROPERTY MN(N) W W 3
DEVICE MP(P) PGATE CPOLY PDIFF PDIFF NXWELL [0]
TRACE PROPERTY MP(P) L L 2
TRACE PROPERTY MP(P) W W 3
DEVICE D(DP)  FUSE PDIO NDIO
TRACE PROPERTY D A A 6
TRACE PROPERTY D P P 7
DEVICE R(MR) MTRES MT1 MT1 [1]
TRACE PROPERTY R(MR) R R 5
DEVICE R(WR) RWELL NXWELL NXWELL [1200]
TRACE PROPERTY R(WR)  R R 5
//PARAMETER CAP [PC] 4.6E-16
DEVICE C(PC) CAPPL CPOLY POLY2 [4.6E-16 0]
TRACE PROPERTY C(PC) C C 4
```

The following example shows how the use of WEFFECT is implemented. The Dracula commands:

```
ELEMENT  MOS[N]   NDEVICE   P1NR   NSDALL  PSUB
ELEMENT  MOS[LN]  LNDEVIC   P1NR   NSDALL  PSUB
LVSCHK[..] WPERCENT=3  LPERCENT=1.5  WEFFECT=0.3
```

are translated to the SVRF statements:

```
DEVICE MN(N) NDEVICE P1NR NSDALL NSDALL PSUB [0.3]
TRACE PROPERTY MN(N) L L 1.5
TRACE PROPERTY MN(N) W W 3
DEVICE M(LN) LNDEVIC P1NR (G) NSDALL (S) NSDALL (D) PSUB (B) [
     property W, L
     weffect = 0.3
    // Replace with effective width factor if desired (eg. 0.5).
    W = 0.5 * ( perim_co(S, LNDEVIC) + perim_in(S, LNDEVIC) +
        perim_co(D, LNDEVIC) + perim_in(D, LNDEVIC) )
    L = area(LNDEVIC) / W
    if ( weffect != 0 ) {
```

```
        if ( bends(LNDEVIC) != 0 ) {
            if ( W > L )
              W = W - weffect * bends(LNDEVIC) * L
            else
              L = L - weffect * bends(LNDEVIC) * W
        }
    }
]
TRACE PROPERTY M(LN) L L 1.5
TRACE PROPERTY M(LN) W W 3
```

- PATHCHK

  The Dracula file converter translates the Pathchk statement to ERC Pathchk. SVRF does not support, and ignores, the Dracula option X. SVRF does not use optional trapfiles, and issues a warning message if a trapfile is present in the Dracula PATHCHK statement. The following examples show how Pathchk statements are translated:

  The Dracula command:

  ```
  PATHCHK LEVEL 1 OUTPUT NOPW 19
  ```

  is translated to the command:

  ```
  // ***** Warning! Use ERC SELECT CHECK to select the desired ERC
                                                    // related check
  NOPW19 {
      PATHCHK GROUND && ! POWER
  }
  ```

  The Dracula command:

  ```
  PATHCHK[F] LEVEL 2 OUTPUT NOGR 20
  ```

  is translated, with warnings, to the command:

  ```
  // ***** Warning! Options F, X allowed only for LEVEL 4
  // ***** Warning! Use ERC SELECT CHECK to select the desired ERC
  // related check
  NOGR20 {
      PATHCHK POWER && !GROUND
  }
  ```

The Dracula command:

```
PATHCHK LEVEL 3 OUTPUT NOPWGR 21
```

is translated to the command:

```
// ***** Warning! Use ERC SELECT CHECK to select the desired ERC
// related check
NOPWGR21 {
    PATHCHK !POWER && !GROUND
}
```

The Dracula command:

```
PATHCHK[FX] LEVEL 4 ALLF
```

is translated, with warnings, to the command:

```
// ***** Warning! Ignoring option X not supported
// ***** Warning! Use ERC SELECT CHECK to select the desired ERC
// related check
allf = PATHCHK !LABELED EXCLUDE UNUSED
```

Please refer to ERC Select Check for information on selecting desired rule checks.

- SCONNECT

  SCONNECT is converted to the SVRF Sconnect operation statement as shown below.

  The following commands:

```
CONNECT-LAYER A B C D E
SCONNECT D B by CONT
SCONNECT D A by CONT
SCONNECT E C by PCONT
```

  are translated to:

```
SCONNECT D B A by CONT
SCONNECT E C by PCONT
```

The following commands:

```
CONNECT-LAYER A B
SCONNECT A B by C
```

are translated to:

```
SCONNECT B A by C
```

and a warning is issued.

The following commands:

```
CONNECT-LAYER B A
SCONNECT A B by C
```

are translated to:

```
SCONNECT A B by C
```

- SOFTCHK

  SOFTCHK is converted to the SVRF LVS Softchk operation statement as shown below.

  The following command:

  ```
  SOFTCHK lower-layer {trapfile} OUTPUT c-name l-num {d-num}
  ```

  is translated to:

  ```
  LVS SOFTCHK lower-layer LOWER
  ```

  The following command:

  ```
  SOFTCHK lower-layer {trapfile} OUTPUT[U] c-name l-num {d-num}
  ```

  is translated to:

  ```
  LVS SOFTCHK lower-layer CONTACT
  ```

  The following command:

  ```
  SOFTCHK lower-layer {trapfile} OUTPUT[A] c-name l-num {d-num}
  ```

is translated to:

LVS SOFTCHK lower-layer CONTACT ALL

The following command:

ABORT-SOFTCHK = YES/NO

is translated to:

LVS ABORT ON SOFTCHK YES | NO

Polygon Topologicals

Dracula lacks many of the Not... polygon topological operations. For example, in a Dracula deck the following commands:

```
SELECT  layA  OUTSIDE  layB  layC
NOT  layA  layC  layD
```

can be written:

```
layD =  layA  NOT OUTSIDE  layB
```

if layC was only being used to generate layD, thus saving one operation. Even if layC is used elsewhere, the following should be used in DRC:

```
layC  =  layA  OUTSIDE  layB
layD  =  layA  NOT OUTSIDE  layB
```

Because of concurrency, this is still considered one operation.

Region Option

In Dracula, the [R] option automatically turns on a type of opposite metric. In DRC, the REGION keyword is completely orthogonal to the metric being used. Refer to the dimensional check operations Enclosure, External, and Internal for a description of the REGION keyword.

Select command

Not all options for the Dracula Select command are translated. The following is a list of those options that are translated:

Group I: INSIDE, OUTSIDE, HOLE
Group II: CUT, TOUCH, ENCLOSE, OVERLAP
Group III: LABEL
Group IV: VERTEX
Group V: ANGLE
Group VI: None
Group VII: NOT for the Group I and Group II options
Group VIII: CONN

Singularities

Singularities are flagged automatically by Dracula in Width commands and two-layer DRC commands. In DRC, measurement and output at singularities must be *explicitly requested* in a dimensional check operation using the SINGULAR keyword or an appropriate form of the ABUT keyword. Refer to the dimensional check operations Enclosure, External, and Internal for a description of the SINGULAR and ABUT keywords.

In Dracula, when the [C] option is specified with ENC, EXT, INT, or WIDTH, drac_cvt no longer generates the SINGULAR keyword.

Specification Statements

The following specification statements are required to compile a rule file and are inserted into the output file.

- SVRF DRC and ERC specification statements:

  DRC RESULTS DATABASE "drc.db" ASCII
  ERC RESULTS DATABASE “erc.db” ASCII
  DRC SUMMARY REPORT "drc.sum"

- SVRF LVS specification statements:

  ```
  SOURCE PRIMARY "top"
  SOURCE PATH "lvs.spice"
  LVS REPORT "lvs.rep"
  MASK RESULTS DATABASE none
  ```

- Optional SVRF specification statements: The following specification statement is inserted, but commented out. Uncommented, this statement ensures compatibility with (outmoded) GDSII readers and viewers that have a polygon vertex limit.

  ```
  // DRC MAXIMUM VERTEX 199
  ```

- Capacitance and resistance unit statements:

  ```
  UNIT CAPACITANCE F
  UNIT RESISTANCE OHM
  ```

  The Dracula PARAMETER CAP statement specifies values with units of farads/unit area and farads/unit length. The PARAMETER RES statement specifies a value with units of ohms/square.

The following Dracula statements are translated to SVRF specification statements.

- LVSCHK[S] and LPECHK[S]

  If Dracula uses either of these statements:

  ```
  LVSCHK[S]
  LPECHK[S]
  ```

  the following is the default translation:

  ```
  LVS REDUCE SPLIT GATES YES
  ```

- CARE-SPLIT ORDER

If Dracula has any of the following statements:

```
CARE-SPLIT-ORDER = YES
FIX-INPUT-ORDER  = INSTANCE
FIX-INPUT-ORDER  = EXCEPT-LOGIC
FIX-INPUT-ORDER  = YES
```

the SVRF translation is:

```
LVS REDUCE SPLIT GATES YES SAME ORDER
```

- LISTERROR

  LISTERROR is converted to the corresponding DRC Maximum Results and ERC Maximum Results specification statements.

- SOURCE SYSTEM SPICE

  The SVRF specification statement Source System Spice is always added to the converted rule file.

Stamp Command

In Dracula, the command STAMP A BY B stamps layer A *in place*. Throughout the remainder of the Dracula rule deck, A represents the stamped layer A. In DRC, Stamp A BY B is a layer operation and, as such, *creates* a derived layer representing the stamped layer A. No input layers are affected. Therefore, a different layer needs to be defined to represent the stamped layer A.

Substrate Pins

Substrate pins for RES, CAP, and DIO Elements are translated as in the following example.

These commands:

```
ELEMENT DIO[SDP]FUSE PDIO NDIO PSUB
ELEMENT RES[SND]RESD NDIFF PSUB
PARAMETER RES[SND]60
ELEMENT CAP[SPC]CAPPL CPOLY POLY2 PSUB
PARAMETER CAP[SPC]4.7E-16
```

are translated as:

```
DEVICE D(SDP) FUSE PDIO NDIO PSUB
DEVICE R(SND) RESD NDIFF PSUB [60]
//PARAMETER CAP[SPC] 5.7E-16
DEVICE C(SPC) CAPPL CPOLY POLY2 PSUB [4.7E-16 0]
```

Virtual Connections

Colons in label names direct Dracula to treat the nets attached to those labels as connected. For example, geometry labelled as VCC:1 and geometry labelled as VCC:2 are treated as pieces of the same net (VCC). To mimic this default Dracula behavior, the SVRF statement Virtual Connect Colon is added to the output of all translated Dracula LVS files. The insertion of Virtual Connect Colon is *triggered* by the presence of either a CONNECT-LAYER or a TEXTSEQUENCE statement in a Dracula command file.

Miscellaneous

The Dracula file converter:

- Translates Dracula ELEMENT statements describing BJTs with four pins.
- Translates operations that create or manipulate a layer named PAD.
- Quotes text strings in converted EDTEXT and HEDTEXT files, as well as Dracula LCONNECT, GROUND-NODE, POWER-NODE, and LINK rules.
- Ignores, and does not issue a warning for the following Dracula commands: PROGRAM-DIR, PRINTFILE, OUTDISK, KEEPDATA, MULTILAB, SAMELAB, EXCEPTION-ON, and STATUS-COMMAND.
- Does not generate the SVRF LVS Filter Unused Option statement with the AB option when the Dracula command KEEP-SHORT-MOS = YES is specified.
- Checks any Attach layer to ensure it is a Connect layer for each input layer declaration.

Compare Two GDSII Databases

The compare_gds utility is used to compare two GDSII databases. It does a flat layer-by-layer XOR (or NOT) and writes an ASCII DRC results database. It allows layers in one database to be compared to different layers in the other database, sets of layers to be compared to different sets, particular datatypes to be compared, sets of datatypes to be compared to different sets of datatypes, and so on. The compare_gds utility (32-bit) is located in $MGC_HOME/bin.

File comparison is done by allowing an optional SVRF parameter to follow each input database as follows:

```
compare_gds database1 top_cell1 [ -RULES rule-file1 ]
            database2 top_cell2 [ -RULES rule-file2 ]
            output-database [ -NOT | -XOR ] [ -NOKEEPEMPTY ]
```

Each rule file is queried for its Layer Map statements and the target layers guide the comparison. You need to know how layer maps work to understand how to use this capability.

This utility compares two GDSII databases *database1* and *database2* with TOP-cells *top_cell1* and *top_cell2*. The comparison is between layers (from 0 to 8191) that have geometry in at least one of the databases. For each layer L with geometries in at least one of the input databases, the geometries are flattened and a Boolean XOR is done between the resulting two layers. Results of the XOR are written to the output DRC results database with the rule check name “diff_L” where L is the layer number. If the XOR is empty, diff_L will be an empty rule check unless the -NOKEEPEMPTY switch is specified; in that event, diff_L will not exist.

The program does not consider datatype nor does it compare text.

By default, an XOR is performed on the layers. The -NOT switch changes this to do a Boolean NOT of *database1* and *database2* in that order.

The TOP cell name in the ASCII DRC results database is *top_cell1^top_cell2* if doing XOR, otherwise *top_cell1-top_cell2*.

Note, of course, that the data must have the same origin or it will be considered different data also.

Rules Syntax Checker

The rules_syntax_checker is a utility you can run against a rule file to validate the syntax of its statements. The full path to this executable is $MGC_HOME/shared/pkgs/icv.${VCO}/tools/misc/rules_syntax_checker. As shown in the following example, this utility prompts you for the name of the rule file and then runs the check. It creates a zero length file named “compiled” if you answer “y(es)”.

```
$ rules_syntax_checker

Please input the file name (CTRL-D to abort): rule_file

Successful compilation; compilation time = 0.02
```

You can also specify the pathname to the rule file upon invocation, as follows:

```
$ rules_syntax_checker rule_file
```

You may want to do this if you are writing a shell script to automate the process.

Encountering an error terminates the check. The rules file compiler will generate an error message for the first such mistake it finds where the offending word sequence appears in the error message, for example:

```
Error INP1 on line 15 of rule_file - superfluous or invalid
input object: SORCE.
```

After correcting the first such error, recheck the rule file to find the next error. Repeat this process until the rules file loads successfully.

Chapter 13
Hierarchical LVS

Calibre LVS-H is a *fully-hierarchical* LVS application. Unlike flat verification applications, which completely flatten the input database and operate on the resulting flat geometries, Calibre LVS-H maintains the database hierarchy and exploits this hierarchy to reduce processing time, memory usage, and LVS discrepancy counts.

Calibre LVS-H may be invoked from the shell command line with the -spice and -hier command line options. It may also be invoked from the Calibre LVS GUI. It includes the hierarchical circuit extractor (which involves hierarchical connectivity extraction and hierarchical device recognition), the hierarchical Spice netlister and the hierarchical LVS comparison module. It uses the same rule file as its flat counterparts Generally there are no statements which need to be added or removed. (Some limitations do exist, however, regarding the use of certain operations; those are described below. Also, for hierarchical LVS comparison you may need to specify the names of cells that correspond in layout and source). Calibre LVS-H imposes no design restrictions concerning geometry overlapping cell placements or overlaps of cell placements.

Hierarchical Circuit Extraction

Specifying hcells with the -spice option can slow down circuit extraction in cases where it would be beneficial to expand the cells (such as during dense overlap removal). For example, standard cell designs with routing in blocks and one block per channel can fall into this category. You can circumvent this by specifying the Layout Top Layer specification statement in the rule file, which expands cells by one layer.

Hierarchical LVS Comparison

This section describes the differences between flat LVS circuit comparison and its hierarchical variation.

Pin Swappability

Pin swap information, from rule file Device statements, is not used in hierarchical LVS comparison. It is not possible to specify swappable pins for user-defined devices in LVS-H.

Model Names

Device element names MP, MN, ME, MD, Q, and D must be accompanied by a model name in the rule file that is identical to the model name used in the source netlist. This is required for the netlist to netlist comparison flow.

There are two exceptions to this rule:

- Device element names MP, MN, ME, and MD can be specified without model names in the rule file if the corresponding model name used in the source netlist is P, N, E, and D, respectively.
- Device element names Q and D can be specified without model names in the rule file if the corresponding model name used in the source netlist is Q and D, respectively.

Connectivity Dependent Transformation

Consider a hierarchically-corresponding cell specified by -hcell or -automatch that contains two or more nets shorted together at a higher level of hierarchy. These nets will be recognized as one net, if they are shorted together in all instances of the cell. However, they are treated separately, if the nets are shorted together in some, but not all, instances of the cell. This can affect LVS transformation operations such as unused device filtering.

Isolated Layout Nets

Hierarchical LVS does not report isolated layout nets.

Hierarchical Device Recognition

The Device operation includes two secondary keywords that apply to hierarchical applications; BY NET and BY SHAPE. The following sections describe the processes of these keywords, as well as the computation of two hierarchical properties.

BY NET device recognition

Consider a layout cell with two or more nets that are shorted together at a higher level of hierarchy in some but not all placements of the cell. The BY NET mode of the hierarchical Device operation (which is the default) currently treats those nets separately when classifying the device.

BY SHAPE device recognition

The BY SHAPE option of the hierarchical Device operation promotes each device up the hierarchy until all pin geometries are fully merged, which may be excessive.

Property computation: pin_net(), named_net()

Hierarchical pin_net() currently returns node numbers local to the cell, which is not generally correct. named_net() is not implemented hierarchically.

Hierarchical Layer Operations

Some hierarchical layer operations preserve the original layout hierarchy. Others can cause some amount of hierarchical degradation. When degradation occurs, geometries on derived layers that would otherwise be part of a lower level cell can be promoted to higher levels of hierarchy. If those geometries serve as seed or pin

geometries in device recognition, or if they, in turn, derive seed or pin geometries, then the respective devices can be recognized at higher levels of hierarchy than would be expected. This promotion is not generally a problem; however, if a device is promoted up across the boundary of a hierarchical LVS correspondence cell (as specified with the -automatch or -hcell option), then that cell may no longer match its schematic description, which results in discrepancies in hierarchical LVS.

The following operations accurately preserve the original layout hierarchy:

One-layer booleans:Or
Two-layer booleans: And, Or, Not
Polygon topologicals: (Not) Inside, (Not) Outside, (Not) Cut, (Not) Enclose (Not) Touch, (Not) Interact
Polygon measurement: (Not) Area
Sizing: Size with no OVERLAP ONLY
Extent: Extent
Connectivity related: (Not) Net, Net Area, Net Area Ratio, Stamp
Text based selection: Text
Copying: Copy
Connectivity extraction: Connect, Attach, Label Order
Device recognition: Device

The following operations can cause degradation of the layout hierarchy. They are not commonly used in LVS; if they must be used, it is recommended that the LVS comparison step be done flat. That is, use the -hier option but specify no correspondence cells except for the top level cell. Circuit extraction will be done hierarchically and comparison will be done flat.

Two-layer booleans: Xor
Holes: Holes
Polygon measurement: Perimeter, Vertex, (Not) Donut
Sizing: Size with OVERLAP ONLY
ShiftingShift
Extent: Extents (as opposed to Extent)
Expand edge: Expand Edge
Edge based selection: (Not) With Edge
Edge topologicals: (Not) Inside Edge, (Not) Outside Edge,

(Not) Coincident Edge, (Not) Coincident Inside Edge,
(Not) Coincident Outside Edge
(Not) Touch Edge, (Not) Touch Inside Edge,
(Not) Touch Outside Edge
Edge measurement: (Not) Angle (note following section for exception),
(Not) Length, Path Length
Merging: Merge
Dimensional check: External, Internal, Enclosure

The following operations produce flat or empty layers by definition and can not be used in hierarchical LVS.

One-layer booleans: And, Xor
Magnification:Magnify
Connectivity related:Polynet, Ornet
Flattening:Flatten
Edge measurement:(Not) Angle operations that would select a 0 degree edge but not a 90 degree edge, or a 90 degree edge but not a 0 degree edge.
Density measurement:Density
Methodology:Pins, Ports, Topex

The following operations can generate only DRC error layers and hence are not applicable to LVS.

Error-directed:Drawn Acute, Drawn Offgrid, Drawn Skew

Most types of Angle and Not Angle operations produce hierarchical layers and are allowed in hierarchical LVS. Only two types produce flat layers and are not allowed:

- Angle OR Not Angle that would select a 0 degree edge but would not select a 90 degree edge.

- Angle AND Not Angle that would select a 90 degree edge but would not select a 0 degree edge.

When an illegal type of Angle and Not Angle operation is used, a message is printed and the application terminates.

Cell Pushdown

Cell pushdown is an optimization in the hierarchical database constructor that under certain conditions pushes cell placements down into underlying cells. This often provides significant performance benefits. However, in Calibre LVS-H it is sometimes desirable to prevent the pushdown of certain cells, such as cells that contain devices. The following rules govern which cells are candidates for pushdown in Calibre LVS-H:

- If Layout Top Layer is specified in the rule file, then only top-layer cell placements are candidates for pushdown, and non-top-layer cell placements are never pushed down. If Layout Top Layer is not specified then the usual criteria are used to select candidates for pushdown (specifically, very small cell placements are candidates for pushdown). As a result, when Layout Top Layer is properly specified, the hierarchical database constructor in Calibre LVS-H will not push cells that contain devices down into other cells. This prevents undesired pushdown of devices down into hcells.

- Layout LVS Box cells in Calibre LVS-H are never considered to be very small cells or top-layer cells even if they otherwise fit the criteria for such cells. As a result, the hierarchical database constructor in Calibre LVS-H will never push layout LVS Box cells down into other cells. This prevents undesired pushdown of LVS Box cells down into hcells.

Top-layer cell placements are placements of cells that do not contain any layers other then Layout Top Layer layers.

These rules apply to Calibre LVS-H as well as to certain other hierarchical Calibre applications, including xCalibre (but they do not apply to Calibre DRC-H).

Hcells

Hcells in Calibre LVS-H are cells that exist in both the layout and source designs and that correspond between layout and source (or are expected to exist and correspond). The term *hcell* stands for *hierarchically corresponding cell*. Hcells come in pairs, consisting of a layout cell and a corresponding source cell. In

Calibre applications other then LVS, hcells may have a more general meaning, and may simply designate cells that need to be preserved for some purpose.

Hcells in Calibre LVS-H may be specified in a number of ways: in the rule file with the Hcell specification statement; in an external cell correspondence file with the -hcell command line option; or implicitly with the -automatch command line option. Each of these methods may establish its own list of cell name pairs, each pair consisting of a layout cell name and a corresponding source cell name. Those lists are combined together and used as a single hcell list. All cell name pairs are treated equally, regardless of how they were established. Note that the Calibre LVS-H circuit extraction stage (calibre -spice) is not aware of hcells established with the -automatch command line option (since that option operates only later, in the circuit comparison stage). Hcells are optional. If you do not specify hcells with any of the methods described above then the hcell list is empty.

In a pair of hcells, the layout cell name and corresponding source cell name may be the same or they may be different. You may specify a 1-to-many relation by placing a layout cell name in several hcell pairs with different source cell names. Similarly, you may specify a many-to-1 relation by placing a source cell name in several hcell pairs with different layout cell names. However, many-to-many relations are not allowed. Note that the latter restriction is enforced by the Calibre LVS-H circuit comparison stage (calibre -lvs -hier) but is not enforced by the rule file compiler or by the Calibre LVS-H circuit extraction stage (calibre -spice).

Case sensitivity of hcell names in Calibre LVS is controlled by the LVS Compare Case specification statement. Specifically, hcell names are case-insensitive by default; hcell names are case sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES. This applies in Calibre LVS to circuit extraction as well as circuit comparison. (Other Calibre applications, such as DRC, may behave differently with respect to case sensitivity of hcells; for more information refer to “DRC Use of Hcells” on page 6-9).

Hcells in Calibre LVS-H are used mainly in the circuit comparison stage (calibre -lvs -hier). To some extent they also affect hierarchical circuit extraction (calibre -spice) as described below. In any event, hcell specification is optional.

Circuit comparison: Hcells are compared as hierarchical entities. For each pair of hcells, the layout cell is compared to the indicated source cell. All other cells are expanded in the circuit comparison stage down to the next level of hcells (or

down to primitive devices when there are no lower level hcells). When no hcells are present, circuit comparison operates at primitive device level. When a layout cell corresponds to several different source cells (a 1-to-n relation), the layout cell is compared against each of the indicated source cells. When several different layout cells correspond to a single source cell (a n-to-1 relation), each of the layout cells is compared to the single indicated source cell. The top level cells (Layout Primary and Source Primary) always correspond and do not need to appear in the cell correspondence file. By default, primitive devices correspond by component type as in flat LVS. In primitive devices with non-built-in types you can override this by including their names in the hcell list as well. The hcell list then exclusively determines their correspondence. Warnings are issued for cell names that do not exist in the input data.

Hcell instances may be expanded by the circuit comparison algorithm if it decides that expansion is necessary in order to match the layout and source hierarchies. For more information refer to LVS Expand Unbalanced Cells.

Circuit extraction: Hierarchical circuit extraction (calibre -spice) does not require an hcell list. Extraction is performed hierarchically based on the original database hierarchy. Generally, cells in the original database hierarchy are preserved; however, some cells may be expanded to improve performance. Calibre utilizes a variety of heuristics to determine when cell expansion is desired. A common example is dense overlap removal, which expands dense cells that overlap each other.

The hcell list prevents cell expansion in the circuit extraction stage. The hierarchical circuit extractor preserves all layout hcells and will never expand them as part of dense overlap removal or similar heuristics. Specifying hcells in the circuit extraction phase ensures that all hcells are preserved as subcircuits in the extracted layout netlist and are available for use as hcells in the comparison phase. Note that hcell specification may slow down circuit extraction in cases where it would be beneficial to expand those cells (for example due to dense overlap).

Many-Many Cell Correspondence

Many-many cell correspondence exists when there is at least one pair of corresponding layout and source cells, where the layout cell also corresponds to

other cells in the source (outside of the pair) and the source cell also corresponds to other cells in the layout (outside of the pair). Cell correspondence is specified, for example, with hcell lists and/or with the `-automatch` command line argument. Here is an example of an hcell list with many-many correspondence:

```
aa aa
aa bb
bb bb
```

Many-many cell correspondence is a global error that causes hierarchical LVS to abort without comparing individual cells. A pair of cell names that leads to the many-many correspondence is indicated in the Calibre LVS-H transcript. In the example above, the following message may appear in the transcript:

```
ERROR:  Correspondence "bb" "bb" leads to a many-many
        correspondence.
```

This means that adding the pair “bb bb” to the cell correspondences established so far leads to a many-many correspondence.

Many-many cell correspondence is also indicated with a respective secondary comparison status in the LVS report. The primary comparison status is usually `NOT COMPARED` but may be `INCORRECT` if input errors also exist.

Hierarchical Pins

Calibre LVS-H treats pins of hcells differently than Calibre LVS. The following sections describe a couple of differences between the two applications.

Matching hcell Pins

Pins of hierarchically corresponding cells (hcells) do not have to be texted. Untexted hcell pins not uniquely matched within the cell are matched by context. In other words, nets, and instances at higher levels of hierarchy can be matched first and this may, in turn, induce specific matching of pins, nets, and instances within cells at lower levels of hierarchy. Nevertheless, texting of hcell pins is

recommended; it often improves run time in the hierarchical LVS circuit comparison module, and it improves discrepancy reporting.

Trivial Pin Swappability

In certain cases, information about the logical equivalence of pins can be carried up from a device or logic gate level to respective pins of a containing hcell. The respective hcell pins are then logically equivalent or “swappable” as well. This is called trivial pin swappability. Specifically, hcell pins connected directly to swappable pins of a primitive device or logic gate within the hcell are swappable, provided that they are not connected to anything else within the hcell. For complex gates, only first level pin swapping is allowed at the hcell level; that is, you can interchange pins within swappable groups but you cannot interchange groups at the hcell level. For logic gates, you must use a LVS Recognize Gates statement to enable logic gate recognition and it is recommended that you use LVS Power Name and LVS Ground Name statements to allow formation of complete gates.

Trivial pin swappability is not carried up from logic gate pins where the properties or subtypes of respective transistors within the logic gate are not symmetric. Only “traced” properties are considered; values are considered equal if they fall within the specified tolerances. If a difference exists then the respective hcell pins are not swappable.

For example, pins A through C of CELLX in Figure 13-1 are swappable because they are connected directly to input pins of a NAND gate and nothing else. Pins E and F are swappable because they are connected directly to pins of a resistor device and nothing else.

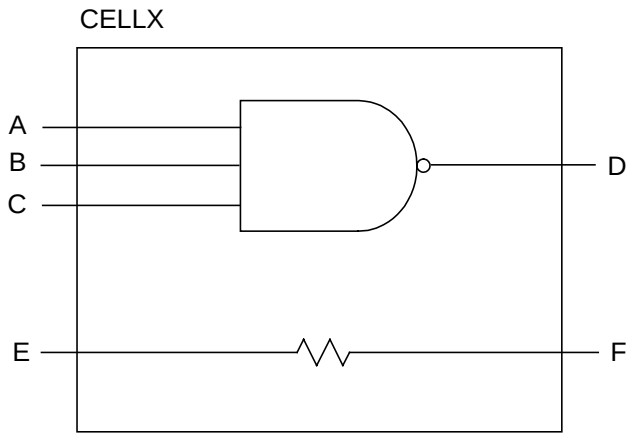


Figure 13-1. Trivial Pin Swappability

SRAM Bit-Cell Recognition

Calibre LVS-H recognizes the common SRAM bit-cell structure. As a result, hierarchical LVS allows you to swap certain hcell pins in memory designs, as described below (hcells are corresponding cells used in hierarchical circuit comparison).

In Figure 13-2, the names POWER and GROUND were replaced with SUP1 and SUP2 respectively.

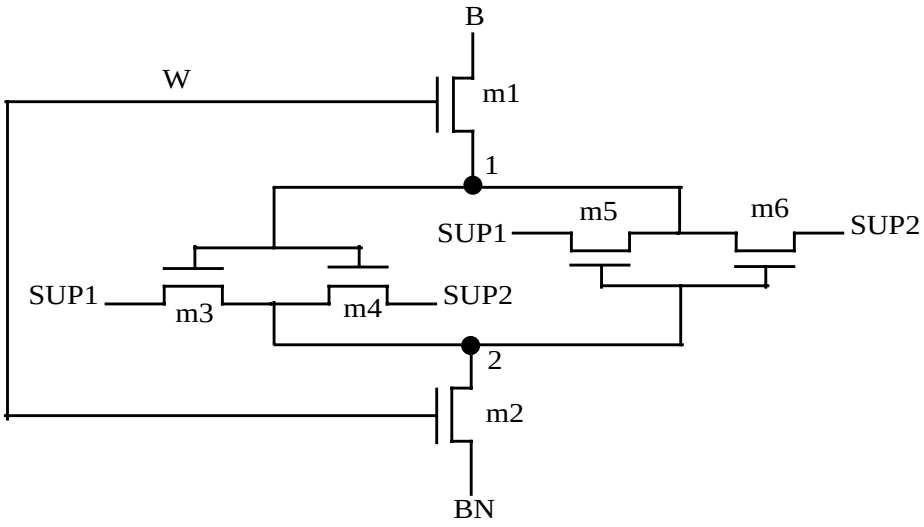


Figure 13-2. SRAM Bit-cell

We use the names B, BN, W, SUP1 and SUP2 for reference only; no text is actually required by LVS. B and BN must serve as pins of the cell that contains the structure and must not be connected to any other devices in the cell. W may be connected to other devices in the cell and may or may not serve as a pin of the cell. The internal nets designated 1 and 2 must not be connected to any other devices in the cell. The nets designated SUP1 and SUP2 must be the same nets, respectively, in the top and bottom structure; other then that, SUP1 and SUP2 may be any nets and do not have to be designated as power or ground supplies. SUP1 and SUP2 may be connected to other devices in the cell and may or may not serve as pins of the cell. Transistors may be of any MOS type (M, MN, MP, ME, MD, LDD, LDDN, LDDP, LDDE, LDDD) as long as symmetry is observed, as described below.

Note that this processing is performed after expansion of non-corresponding cells. Thus, the term “cell” in this context refers to the design hierarchy as it is seen by the circuit comparison module *after* expansion of non-corresponding cells.

LVS checks component types, subtypes, properties and substrate pin connections to ensure that the structure is indeed symmetric. As a result, there is no loss of information relative to flat comparison. Specifically, component types, subtypes, properties and substrate pin connections must be equal, respectively, on 1) the transistors designated m1 and m2, 2) the transistors designated m3 and m5; and 3) the transistors designated m4 and m6. If LDD-type devices are used then polarity is checked and must observe the symmetry of the structure. Only properties traced with Trace Property specification statements are checked; values are considered equal if they fall within the specified tolerances. Substrate pins are any pins other then D, G or S.

Given this structure, LVS-H recognizes that pins B and BN of the cell are swappable. This information is used when processing placements of the cell. A cell may contain several such transistor level structures, and respective B/BN pins are swappable pairwise.

In addition, LVS-H carries swappability information as far as possible up the hierarchy. Specifically, swappability information is carried up along nets that are connected only to a respective placement pin and to a external port of the cell, and are not connected to any other objects in the cell. In figure 13-3, given that pins B

and BN of cell X are swappable, then in cell Y pin B1 will be swappable with BN1, B2 will be swappable with BN2, and B3 will be swappable with BN3.

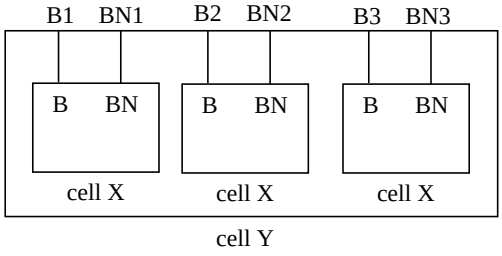


Figure 13-3. Carrying pin swappability up the hierarchy

High-short Resolution

High-short resolution is a process used in hierarchical Calibre when processing connectivity information. It is applied to increase performance and capacity and to allow correlation between layout and source databases. In high-short resolution, Calibre looks for net segments that are separate at the cell level but are consistently connected together at higher levels of hierarchy in all placements of the particular cell (for example, split power rails). Calibre joins such net segments together to form a single net at the cell level. High-short resolution is performed both during hierarchical connectivity extraction and during hierarchical circuit comparison, with some minor differences between the two as described below.

- **Connectivity Extraction** — A high-shorted group of pins is a group of pins in a cell that are consistently connected together higher up in the hierarchy in all placements of the cell in the design. Such pins are also called “globally connected” pins.

  - If a high-shorted group of pins contains pins with different names, then pins are grouped by name; pins with the same name are merged together, but pins with different names remain separate.

  - If a high-shorted group of pins contains pins with different names as well as pins with no name at all, then the pins that do not have names are in a group of their own; they are merged within that group, but are not merged with pins that do have names.

- If a high-shorted group of pins contains pins with one name only and other pins with no name at all, then all pins in the group are merged together.

This processing is identical in all cells, including LVS Box cells. Table 13-1 overviews some possibilities of how high-shorted pins are resolved.

Table 13-1. High-shorted Pin Resolution Examples

Original High-shorted Pins	Pins After High-short resolution	Notes
A B	A B	Different names remain separate …
A B 1	A B 1	… even if unnamed pins are present.
A A A B B	A B	Pins are merged by name.
A A A B B 3 4 5	A B 3	Pins are merged by name, and unnamed pins are merged.
A 1	A	Only one name, therefore all pins are merged.
A A 1 2 3	A	Only one name, therefore all pins are merged.

Note that it is possible for pins (and respective nets) in a cell to remain separate within the cell even though they are globally connected and if they have identical names within the cell. For example, this may happen if the global connection traverses multiple levels of hierarchy and the respective nets at a higher level have different names or if one of them is unnamed. In such cases, the pins (and respective nets) remain separate within the original cell; one of them receives the (common) name and the others are left unnamed. However, the connectivity extractor recognizes the global connection and does not report false open circuit warnings.

Note also that the LVS hierarchical circuit comparison module performs high-short resolution of its own in order to bring the layout and source to a common representation. There is no change in the algorithm there, and that

algorithm may in fact merge pins with different names if there is a difference between pin names in layout and source.

- **Circuit Comparison** — In hierarchical circuit comparison, high short resolution is performed equally in both layout and source. Circuit comparison may combine together high-shorted pins (and respective nets) even when they have different user-given names, except when this can be safely avoided. More precisely, circuit comparison will not combine together high-shorted pins with different user-given names if:

  1) all names involved in the high-short are user-given
  2) all names involved in the corresponding high-short in the other design are also user-given
  3) all names involved in the high-short appear in both layout and source. (In cases of many-to-one or one-to-many hcell relations, these conditions must hold in all variants of the hcell).

  When pins with different names are combined together, circuit comparison remembers all original names and stores them on the combined pin. These names can be used later, for example, to establish initial correspondence points. The same is true for high-shorted nets. However, this is *not* done in LVS Box cells; in those cells, one of the original names is chosen to represent the combined pin and the other names are rejected. Name conflicts in such cases are resolved in favor of, in order of precedence:

  1) names which appear in both layout and source
  2) power names, in rule file order
  3) ground names, in rule file order
  4) names that begin with the letter 'V' or 'v' ('V' is commonly used in power supplies)
  5) alphabetically lesser names.

  Circuit comparison resolves high shorts in LVS Box cells as well as regular cells, even when the LVS Box cells are entered as empty subcircuits in a Spice netlist. But circuit comparison does not resolve high shorts in built-in devices such as R, C, MP, and so on, even when they appear in LVS Box statements. Furthermore, circuit comparison does not resolve high shorts in user-defined primitive devices unless they appear in LVS Box statements.

Parameterized Cells

By default, the Spice netlist reader in hierarchical LVS flattens all parameterized subcircuits. A subcircuit is parameterized if it has at least one subcircuit call that references it and that specifies parameter values. Note that the subcircuit is parameterized regardless of whether or not those parameters are actually used within the subcircuit. In the following example, both subcircuits AAA and BBB are parameterized:

```
.SUBCKT AAA 1 2
M1 1 2 VCC VCC P W=WIDTH L=LENGTH
.ENDS

.SUBCKT BBB 1 2
M1 1 2 VCC VCC P
.ENDS

X1 N1 N2 AAA WIDTH=5 LENGTH=6
X2 N1 N2 BBB WIDTH=5 LENGTH=6
```

Parameterized subcircuits are flattened down to the bottom of their sub-hierarchy, that is, down to primitive devices. Empty subcircuits are treated as primitive devices and are not flattened. Parameter passing is handled and (X,Y) locations, if present in the netlist, are transformed to the top level of the subcircuit being flattened.

Flattening of parameterized subcircuits can be disabled in the rules file with the LVS Preserve Parameterized Cells specification statement.

Hierarchical Cell Cycles

Cycles in the cell hierarchy are global errors that cause hierarchical LVS to abort prior to the comparison stage. This error appears only in hierarchical LVS reports. See the section “Hierarchical Cells Forming a Cycle” on page 14-57 of the “Results” chapter.

Hierarchical Spice

When you use a Spice netlist with Calibre LVS-H, you can control how the Spice netlist reader treats the Spice netlist. The following sections describe some of the variations you can control.

Dollar Signs in Cell Names

The LVS hierarchical Spice netlister (calibre -spice) netlists cell names that begin with a dollar sign "$" with leading underscore characters. Usually, two leading underscores are added. Additional underscores can be added as necessary to avoid conflicts with user cell names. Specifically, the number of underscores added is one larger then the number of leading underscores in any user cell name that begins with a series of underscores followed by a "$" character; and, it is not smaller than two.

For example, the cell name "$xyz" will appear as "__$xyz" in the extracted layout netlist. This convention allows the extracted netlist to be used by downstream tools.

Net Names

Usually, if a net is named (texted) in the layout, then that name appears in the hierarchical Spice netlist. However, if a net name is not valid for netlisting, then the internal net number appears instead. To be valid for netlisting, a name must be non-empty and obey the following semantics:

- It must not contain embedded whitespace or control characters; leading and trailing whitespace and control characters are allowed and are stripped off from the output);
- It must not contain Spice special characters and must not consist solely of digits.

The latter restriction prevents collisions with internal net numbers. Recall that every layout net receives a internal net number; in addition, some nets may have names.

Ports and Port Names

Port objects created with Port Layer Text and Port Layer Polygon specification statements are output in the hierarchical Spice netlist in the top level cell only. Port objects appear as pins in the top level .SUBCKT line. Of course, the netlister ensures that the identifier used for the top level subcircuit pin is identical to that used for the respective net within the subcircuit. The naming rules are as follows:

- Calibre does not consider ports and nets are texted if their names are not valid for netlisting, which follows the same rules as net names described in the above section.
- Calibre determines the net names in the top cell, if port objects are present, from net or port names, whichever is texted. The net name has precedence if both a net and its port are texted.
- Calibre issues a warning and ignores a port name if the name appears on a different net.
- Calibre arbitrarily chooses a port name if two port names appear on the same net; Calibre silently discards the unselected port name.

To be valid for netlisting, a port name must obey the same rules as described for nets in section 11.12.2.

“M” Device Representation

Devices with element name M in the rule file are considered to be user-defined for the purpose of device recognition (they are built-in for LVS comparison, though). Nevertheless, in the hierarchical layout netlist they are represented, when possible, with Spice MOS elements, not primitive subcircuit calls. For example, the following in your rule file:

```
DEVICE m(h) gate gate(g) sd(s) sd(d) bulk(b)
[
     property w, l  ....
]
```

generates something like this in the layout netlist:

```
M1 1 2 3 4 h w=2e-6 l=1e-6 $X=210000 $Y=200000
```

To qualify for the M representation, the Device operation must meet the following conditions: the element name must be “M” or “m”; the device must have 3 or 4 pins named G, S, D and optional B (upper or lower case, in any order); pins D and S must be swappable; pins G and B may not be swappable with any other pins; a model name must be indicated. If these conditions are not met then the device is represented with primitive subcircuit calls.

Cell Statistics

`.SUBCKT` statements in the hierarchical Spice layout netlist are each followed with a comment line containing statistics about the respective layout cell. For example:

```
.SUBCKT PVDD2 1 2
** N=452 EP=2 IP=516 FDC=249
```

N is the number of nets in the cell. EP is the number of external pins in the cell (pins of the cell). IP is the number of internal pins in the cell (placement pins in the cell). Note that N and IP may not be identical to what is actually present in the netlist, because not all layout nets and placement pins are represented in the netlist. For example, the netlist normally does not contain floating nets or placements of cells that have no devices.

FDC is the flat device count in the cell. This is the number of all primitive devices in the cell, including the sub-hierarchy of the cell, counted flat. In this context, primitive devices are objects formed with rule file Device operations. LVS Box cells are treated as normal cells. The FDC number is a good measure of cell size.

Note that this data is provided for information only. It is not used or interpreted by the LVS circuit comparison module and it is not an integral part of the netlist.

Hierarchical Netlister Warnings

The hierarchical Spice netlister, used in the Calibre -spice functionality, may issue the warnings described below. Unless indicated otherwise, these warnings appear in the Calibre transcript and in the circuit extraction report file.

- Port naming conflict

  ```
  WARNING: Top level port name "<name>" on net <net-number>
  at location (<x>,<y>) already used on net <net-number>;
  ignored.
  ```

  This warning indicates a top-level port naming conflict in the hierarchical Spice netlister. This occurs when two or more nets are connected to top-level-cell ports with identical names. The warning appears only when the port names would have been otherwise used for netlisting; specifically, the two nets in question must be unnamed or must have names that are not valid for netlisting, and the port name must be valid for netlisting. The hierarchical Spice netlister uses the port name to identify one of the nets and its port, and uses the internal number of the other net to identify that net and its port. The net that receives the port name is chosen arbitrarily. The warning message indicates the port name, the net number and port location where the port name was rejected, and the net number where the port name was used. For example:

  ```
  WARNING: Top level port name "aaa" on net 1 at location
  (5,5) already used on net 2; ignored.
  ```

- Bad device

  ```
  WARNING: BAD DEVICE on layer <layer> at location
  (<x>,<y>) in cell <name>
  ```

  This warning indicates a bad device. The device seed layer, location, and cell name are reported. This warning appears in the circuit extraction report file and also in the respective subcircuit in the hierarchical Spice netlist,

where it is preceded with “**”. This warning does not appear in the Calibre transcript. For example:

```
WARNING: BAD DEVICE on layer pgate at location (20,20) in
cell zcel
```

Chapter 14
Results

This chapter discusses the various files and reports generated by Calibre applications. The various files and reports include:

- Session Transcript
- DRC Results Database
- DRC Summary Report
- LVS Report
- Circuit Extraction Report
- Mask Results Database
- Cross-Reference Files

Session Transcript

Calibre applications produce a transcript showing statistics relative to the major program functions. These functions are:

- Rule file compilation
- Layout data input
- Initialization section
- Executive processes

Rule File Compilation

The transcript section “Standard Verification Rule File Compilation Module” shows the pathname of the rule file, the contents of the rule file, and the amount of CPU and real time required for execution.

Errors, when encountered, terminate the execution, and are reported below the line identifying the pathname of the rule file.

```
--- RULE FILE = drc.db/gds/brules_drc
```

Refer to the *Standard Verification Rule Format (SVRF) Manual* for a description of the compilation error messages.

Layout Data Input

The section “Calibre Layout Data Input Module”, shows cell, layer, and text information, and a summary of the layout data. This information is reported in the following subsections:

- GDSII Stream Summary Information
- GDSII Stream Data for Individual Cells
- Text Objects for Connectivity Extraction
- Text Objects for With Text Operations
- Ports (Calibre LVS / LVS-H / MGC only)
- Layer Read Summary (Geometries)
- Layer Read Summary (Text for Connectivity Extraction)
- Layer Read Summary (Text for With Text Operations)
- Cell and Placement Summary (Calibre DRC-H only)
- Layout Data Input Module Summary

The Calibre layout data input module will report as notes in the transcript all GDSII (layer,datatype) pairs that contain geometric data which will not be required in the run.

Limiting Transcript Output

In the transcript, Calibre applications print all connectivity extraction text objects in the run. Since the number of these objects can be excessive, you can use the Text Print Maximum specification statement to limit the number of objects printed in each block.

Hierarchical and Flat Counts

In Calibre DRC-H, both the transcript and the DRC summary report file provide many statistics independent of the derived layer statistics. These statistics appear as a pair of numbers, with the second in parenthesis. For example:

```
--- TOTAL GEOMETRIES WRITTEN TO ORIGINAL LAYERS = 866804
(6508994)
```

or

```
DRC RuleCheck 5.2.1.1 COMPLETED. Number of Results = 20
(134567)
```

The first number is the hierarchical count and the second is the (estimated) flat count.

Initialization Section

This section is included for LVS applications, and reports the following processes:

- Global initialization
- Connectivity extraction
- Device recognition
- LVS initialization

- Database creation.

Executive Process

This section, titled “Calibre:: Executive Module”, reports event logs, warning messages, and summary information. It also reports operating parameters, such as maximum results per check, maximum vertices per result polygon (DRC), and connect node number placement and device extraction (LVS). The executive module performs gate reduction, recognition, and comparison and reads the layout and source databases.

The following sections discuss specifics of the executive module section.

Layer Statistics

When you generate a layer in the executive module, the operation generating it prints statistics about the generated layer. The sample statistics for a layer are different among Calibre applications.

Layer Statistics in Calibre DRC

The following example shows layer statistics:

```
nplus = diff NOT plus
---------------------
nplus (TYP=1 CFG=1 ECT=504382 OCT=381317 SRT=1 CMP=T
                                    MPN=14231 AREA=166961.875)
CPU TIME = 140  REAL TIME = 142  LVHEAP = 40/55/56
```

The following list describes what each line specifies.

- The first line shows the layer operation as it appears in the rule file. This line is followed by a dashed line.

  A rule check name replaces the derived layer name when the layer operation is within a rule check. The rule check name is followed by double colons (::) and the value *n*, which indicates the *n*th layer operation of the rule check.

- If the layer operation is within a rule check

It is important to note that multiple operations may appear in a single block. This indicates concurrency. The following example shows three concurrent operations in the same block:

```
5.4.2.2::<1> = EXT poly < 1.2 SINGULAR
5.8.3::pg24 = EXT poly < 2.4 REGION OPPOSITE
5.4.1::<1> = INT poly < 0.98
-------------------------------------------------------
5.4.2.2::<1> (TYP=3 ECT=0 CCT=0)
5.8.3::pg24 (TYP=1 CFG=0 ECT=55264 OCT=32611 SRT=1 CMP=T
                                    MPN=0 AREA=1941002.595)
5.4.1::<1> (TYP=3 ECT=0 CCT=0)
CPU TIME = 383  REAL TIME = 388  LVHEAP = 58/61/63
```

- The second line shows the name of the generated layer, or a rule check identifier, followed by several statistics. The following sections describe these statistics:
  - **TYP**: This specifies the type of derived layer.

    1: Derived polygon layer

    2: Derived edge layer

    3: Derived error layer

  - **CFG**: This specifies the numbering configuration of the derived layer.

    0 Neither polygon nor node numbers

    1 Polygon numbers

    2 Node numbers

    3 Both polygon and node numbers

    This configuration is a space-saving device, and ensures that a layer receives the minimal configuration required.

  - **ECT**: This specifies the number of edges on the derived layer. It does not count vertical edges for type 1 layers.

- **CCT:** This specifies the number of edge clusters on the derived layer. It is only generated for TYP=3.
- **OCT**: This specifies the number of objects on the derived layer and is an internal statistic.
- **SRT**: This specifies whether the derived layer is sorted.

  1 Sorted
- **CMP**: This specifies whether the derived layer is compressed.

  T Compressed
- **MPN**: This specifies the maximum number of polygons for the derived layer. This is only reported for TYP=1 and CFG=1 or CFG=3.

  For layer selectors, this number may not correspond to the actual number of polygons on the layer, but it generally will for layer constructors.
- **AREA**: This specifies the total area of the derived layer. This is only reported for TYP=1. The value is reported in square user units.

- The third line shows the time required to generate the layer, and the LVHEAP statistics. Refer to the section "LVHEAP Statistics" for further information.

Layer Statistics in Calibre DRC-H

The following shows an example of hierarchical instantiation:

```
nplus = sdm NOT ppm
--------------------------------
nplus (HIER TYP=1 CFG=1 HGC=4578 FGC=316325 VHC=F VPC=F)
CPU TIME = 4  REAL TIME = 4  LVHEAP = 27/31/31
```

The following list describes what each line specifies.

- The first line shows the layer operation as it appears in the rule file. This line is followed by a dashed line.
- The second line shows the name of the generated layer followed by several statistics. The following sections describe these statistics:
  - **HIER**: This specifies that the layer has a hierarchical instantiation. If nplus had an exclusively flat instantiation, the statistics would be the same as in Calibre DRC. If the layer had a dual instantiation, both flat and hierarchical statistics would be reported.

    There are four variations of the HIER statement:

    - **HIER**: Indicates a “natural” hierarchical instantiation.
    - **HIER-FMF**: Indicates a layer in fully-merged form.
    - **HIER-PMF**: Indicates a layer in partially-merged form.
    - **HIER-LSL**: Indicates a “large-shape” layer (such as the database extent), and initiates special internal optimizations.
  - **TYP**: This specifies the type of derived layer.

    1: Derived polygon layer

    2: Derived edge layer

    3: Derived error layer
  - **CFG**: This specifies the numbering configuration of the derived layer.

    0 Neither polygon nor node numbers

    1 Polygon numbers

    2 Node numbers

    3 Both polygon and node numbers

This configuration is a space-saving device and ensures that a layer receives the minimal configuration required.

- **HGC**: This specifies the number of objects on the layer, which are counted hierarchically.
- **FGC**: This specifies an estimated number of flat objects on the layer. The operation computes this value by multiplying the number of objects in the cell by the total number of flat placements of the cell in the hierarchy. The operation adds up this total for each cell. An “object” is a polygon for type 0 and 1 layers, an edge for type 2 layers, and an edge cluster for type 3 layers.
- **VHC** and **VPC**: Internal statistics that identify the connectivity status of the layer.

The flat statistics, MPN and AREA, do not appear in the Calibre DRC-H report. However, if the layer appears in a DRC Print Area specification statement, then the total flat area of the layer does appear in the report. Reporting of flat area is optional for Calibre DRC-H because it requires significant calculation time.

- The third line shows the time required to generate the layer, and the LVHEAP statistics.

LVHEAP Statistics

The LVHEAP numbers, which appear with all derived layer statistics in the transcript, report approximate current memory usage for Calibre DRC applications. These applications run completely in memory when layers are memory-based.

The report of memory usage consists of three numbers, represented in megabytes (2^{20} bytes). For example:

```
LVHEAP = 28/47/49
```

- The first number is the amount of memory being used at the time of the report.

- The second number is the total amount of memory allocated by the application.
- The third number is the maximum amount of memory that has ever been allocated up to the time of the report.

Therefore, the maximum memory requirement of the application is the largest number reported, which is generally the third LVHEAP number.

DRC Results Database

This section describes the DRC results database, which you can generate in ASCII, binary, or GDSII outputs.

The DRC results database is simply a collection of geometric objects grouped by rule check. Each rule check in the DRC results database contains a list of objects which comprise the DRC execution output from the unassigned layer operation(s) associated with the rule check statement in the rule file; these objects are referred to as DRC results. There are two types of DRC results in the DRC results database: polygons and edge clusters (see the "DRC Concepts" chapter for details). Derived polygon layer data becomes polygons within the DRC results database, derived error layer data becomes edge clusters (of 1, 2, 3, or 4 edges) within the DRC results database, and derived edge layer data becomes edge clusters (of 1 edge) within the DRC results database.

Each DRC result has an associated number which is unique within the set of DRC results for each rule check statement. When the rule check statement is executed, this numbering is consecutive, beginning with 1.

In addition to DRC results, each rule check in the DRC results database may also contain text which has been mapped from the rule file during DRC execution. This is called check text. Check text may consist of the rule check comment(s), or the complete text of the rule check statement from the rule file, or neither. It may also contain the rule file pathname and title, if present, of the rule file over which the rule check was (last) executed. The presence and composition of check text is controlled by the DRC Check Text specification statement.

A DRC results database created by Calibre DRC is always created new for each invocation of the enabling command. A file (specified using a DRC Results Database statement in the rule file) is produced to represent a DRC results database. This file can be an ASCII, binary, or GDSII DRC results database. The former is in human-readable form and is the most common DRC results database format produced by Calibre DRC. An ASCII DRC results database produced by Calibre DRC can be loaded by ICrules into ICgraph and hence, as an ICrules DRC results database, be scanned by the ICrules DRC results presentation commands. An ICrules DRC results database can also be written out as an ASCII DRC results database.

Calibre DRC/DRC-H allows you to output as many DRC results databases as needed during one DRC/DRC-H execution. This allows you to easily view a subset of rule checks or more easily integrate with third-party tools. Refer to the DRC Check Map specification statement for more details.

Calibre DRC will complete the write (including file close) to a DRC results database as soon as all of the DRC rule checks which contribute to that database have completed. A message is transcribed when output to a specific DRC results database is completed. This allows users to begin processing DRC results as soon as they are available without having to wait for LVS, ERC or PEX modules to complete.

The DRC results database has an intrinsic ordering on its constituent rule check statements. In addition, all DRC applications execute a check set in the order in which the rule check statements in the check set appear in the rule file. Therefore, assuming that the DRC results database is created by Calibre DRC, the ordering of rule check statements in the DRC results database will correspond to the order in the rule file.

The list of DRC results associated with a rule check statement in the DRC results database may be empty. In this case, we say that the rule check itself is empty. Empty rule checks in the DRC results database may be created by DRC execution itself. Hence, there is a difference between an empty DRC results database (one containing no rule check statements) and a DRC results database which contains rule check statements but no results.

ASCII and Binary DRC Results Databases

ASCII and binary DRC results databases are very similar structurally. This section describes their basic structures, then presents information specific to each.

The DRC results database is a collection of geometric objects grouped by rule check statements from the rule file. Each section in the DRC results database contains the DRC output from the unassigned layer operation(s) associated with the rule check statement in the rule file; these objects are referred to as DRC results. There are two types of DRC results in the DRC results database: polygons and edge clusters.

- Derived polygon layer data become polygons.
- Derived error layer data become edge clusters (of 1, 2, 3, or 4 edges).
- Derived edge layer data become edge clusters (of 1 edge).

Each DRC result has an associated *number* that is unique within the set of DRC results for each rule check statement. When the rule check statement is executed, this numbering is consecutive, beginning with 1 (one).

Each rule check in the DRC results database can also contain text that has been mapped from the rule file during DRC execution; this is called check text. Check text can consist of rule check comments, the complete text of the rule check, or neither. It can also contain the rule file pathname and title, if present. The presence and composition of check text is controlled by the DRC Check Text specification statement.

The DRC results database has an intrinsic ordering on its constituent rule check statements. In addition, all DRC applications execute a check set in the order in which the rule check statements in the check set appear in the rule file. Therefore, assuming that the DRC results databases created by Calibre DRC, the ordering of rule check statements in the DRC results database will correspond to the order in the rule file.

The list of DRC results associated with a rule check statement in the DRC results database can be empty. In this case, we say that the rule check itself is empty. Empty rule checks in the DRC results database can be created by DRC execution

itself, as described in the next chapter. Therefore, there is a difference between an empty DRC results database (one containing no rule check statements) and a DRC results database which contains rule check statements but no results.

ASCII DRC Results Database Format

Calibre DRC can generate an ASCII DRC results database if the ASCII keyword is specified in the DRC Results Database specification statement.

No blank lines appear in the ASCII DRC results database, and data always start at the beginning of the line. Figure 14-1 shows an example ASCII DRC results database.

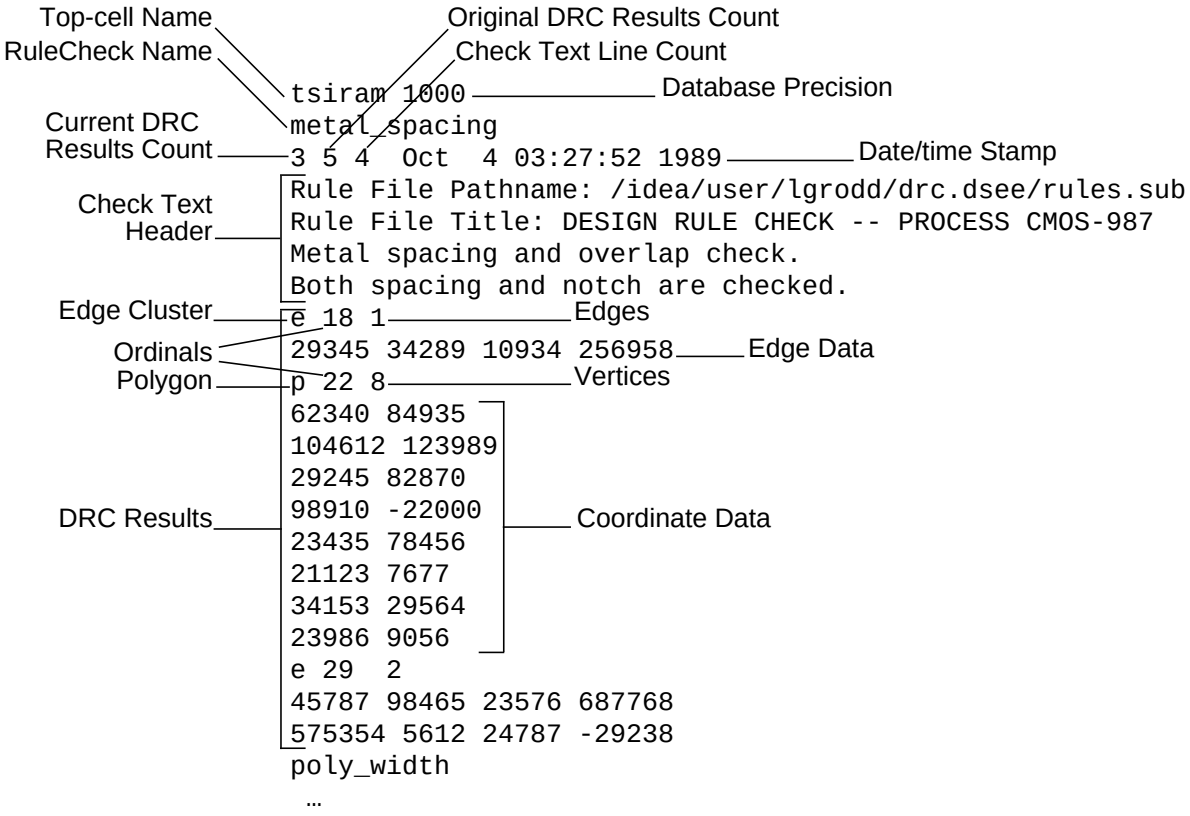


Figure 14-1. ASCII DRC results database (sample)

Cell Name and Database Precision

The first line shows the top-cell name. The top-cell name is the value of the Layout Primary specification statement. The string “drc” is shown if no cell name is specified in the statement.

An integer specifying the database precision follows the cell name. The rest of the ASCII DRC results database is organized by rule check statement, with the information for each rule check statement beginning on a new line. Blank lines are permitted only before and after rule check statement blocks and as check text, but leading and trailing spaces are otherwise always permitted.

Rule Check Name, Result Count, and Execution Time

The first line for each rule check group contains the name of the rule check. Rule check statement names are assumed to be unique. The next line contains three numbers followed by a date/time stamp, separated by one or more spaces.

- The first number is the current count of DRC results.
- The second number is the original count of DRC results.
- The third number is the number of check text lines.
- The date/time stamp shows when the rule check was executed. The date/time format is as follows (blanks are significant):

  mmm dd hh:mm:ss yyyy

Check Text Report

After the rule check name, result counts, and date/time stamp, the default check text is shown as header information. The default header information includes:

- The pathname of the rule file.
- The title of the rule file.
- Any rule check comments.

This information can be removed from the header, or more information can be added with the DRC Check Text specification statement.

DRC Result Listing

Following the header information is a list of DRC results. Each DRC result listing begins on a new line. The DRC results can be one of two types: a polygon or an edge cluster; distinguished by the respective signatures “p” and “e”. These signatures begin the listing for each DRC result.

Following the signature are one or more spaces and then a number that specifies the ordinal of the DRC result within the rule check statement. For polygons, the ordinal is followed one or more spaces, then by the number of vertices within the polygon. For edge clusters, the ordinal is followed by one or more spaces, then the number of edges in the cluster.

The DRC result coordinate data begin on the line following the signature for each result, and consist of integers in database units.

- For polygons, the coordinate data include a list of coordinates; each coordinate occupies one line showing the x-coordinate then the y-coordinate, separated by one or more spaces. The coordinates are listed in counterclockwise order; the number of coordinates corresponds to the vertex count on the signature line and will not exceed 4096.

- For edge clusters, the coordinate data are a list of the edges; each edge occupies one line showing the x-coordinate and the y-coordinate of one endpoint, separated by spaces, followed by the x-coordinate and the y-coordinate of the other endpoint, separated by one or more spaces.

Optionally, you can append additional data to the signature lines for each DRC result by using the DRC Cell Name specification statement. You can append the cell name of an individual DRC result and the transformation matrix data that represents a cell’s position in top-level or cell-level coordinate space.

Binary DRC Results Database Format

Calibre DRC generates a binary DRC results database if the BINARY keyword is specified in the DRC Results Database specification statement. The binary format

is primarily intended as an intermediate step to translation to external database formats where file size is an issue—the binary format is approximately 2X smaller than ASCII format.

The BNF for a binary DRC results database is similar to that of its ASCII counterpart and is as follows:

```
// '' delimits a literal byte.

<binary DRC results database>
   -> <signature> <version>
      <top cell name> <precision>
      <rule check> [ … <rule check> ] EOF//EOF is just that.

<rule check>
   -> <check name>
      <current result count> <original result count>
                                <text line count> <date string>
      [ <text line> …  ]          // Number = <text line count>
      [ <result> …  ]         // Number = <current result count>

<result> -> <edge cluster result> | <polygon result>

<edge cluster result> -> 'e' <result number> <edge count>
             <edge> [ … <edge> ]      // Number = <edge count>
<edge> -> <x1> <y1> <x2> <y2>

<polygon result> -> 'p' <result number> <vertex count>
      <vertex> [ … <vertex> ]        // Number = <vertex count>

<vertex> -> <x> <y>

<top cell name> -> <string>
<check name> -> <string>
<date string> -> <string>
<text line> -> <string>

<precision> -> <long>
<current result count> -> <long>
<original result count> -> <long>
<text line count> -> <long>
<result number> -> <long>
```

```
<edge count> -> <short>
<vertex count> -> <short>

<x1> -> <long>
<y1> -> <long>
<x2> -> <long>
<y2> -> <long>
<x> -> <long>
<y> -> <long>

<string> -> [ <character> [ … <character> ] ] '0x00'

<signature> -> 'C' 'A' 'L' 'I' 'B' 'R' 'E' 'B' 'I' 'N' 'A' 'R'
               'Y' 'D' 'R' 'C' 'R' 'E' 'S' 'U' 'L' 'T' 'S' 'D'
               'A' 'T' 'A' 'B' 'A' 'S' 'E' '0x00'
<version> -> byte                              // Currently 2.

<character> -> byte except '0x00'
<short> -> 2-byte integer, MSB first, LSB last
<long> -> 4-byte integer, MSB first, LSB last
```

GDSII DRC Results Database Format

Calibre DRC generates a GDSII DRC results database if the GDSII keyword is specified in the DRC Results Database specification statement.

Note

This section applies only to flat Calibre applications. Please refer to section “Hierarchical DRC Results Database” for potentially different semantics in the hierarchical case.

The GDSII format is a standard GDSII stream representation of the DRC results database. A GDSII DRC results database has the following BNF (please refer to GDSII documentation for more information):

```
HEADER BGNLIB LIBNAME UNITS <structure> ENDLIB
<structure> -> BGNSTR STRNAME { <boundary> | <path> }* ENDSTR
<boundary> -> BOUNDARY LAYER DATATYPE XY ENDEL
<path> -> PATH LAYER DATATYPE XY ENDEL
```

Where

- The GDSII version number in the header record will be 3.0.
- The modification and last access times in the BGNLIB and BGNSTR records will be the date/time of database creation. Years are relative to 0 BC and January is month 1.
- The library name in the LIBNAME structure will be “drc.db”.
- The UNITS will be drawn from the rule file Precision and Unit Length specification statements, or their defaults.

There is only one cell record. The name of the cell is the value of the Layout Primary specification statement with an optional string appended, which is the string following the GDSII keyword in the DRC Results Database specification statement. If there is no Layout Primary specification statement (meaning that the input layout system was not GDSII), then the cell name is “drc”. DRC applications will issue a warning if any cell name longer than 32 characters is written to a GDSII-type DRC Results Database.

You can assign GDSII rule check output to specific GDSII layers and datatypes by specifying the DRC Check Map specification statement. Calibre DRC/DRC-H issues a warning for each rule check not having a corresponding DRC Check Map specification statement. By default, it assigns rule check output to layer 0 and datatype 0.

Calibre DRC/DRC-H writes edges and edge clusters to a GDSII DRC results database as 0-width paths, one path per edge in the latter case; clustering is lost.

Result Count Limits

Often, you may want to limit the number of results that DRC places into the DRC results database. This is useful when debugging the rules (on a large database) or during initial checks of new databases.

Calibre DRC allows you to specify an upper bound on the number of DRC results per rule check written to the DRC results database. This limits the number of DRC results added per rule check, not the total number of results generated in the entire

run. If you specify a value for *upper_bound*, whenever DRC generates *upper_bound* results for any single rule check, it adds no further results into the DRC results database for that rule check and issues a warning message.

Limiting the Result Count in Calibre DRC

Limiting the DRC result count in Calibre DRC is accomplished by the rule file DRC Maximum Results specification statement. The *number* parameter (which can be zero) specifies the maximum number of DRC results generated per rule check. ALL simply denotes a very large number (2147483647 to be exact). This value defaults to 1000 if the statement is omitted.

Hierarchical DRC Results Database

The database hierarchy is completely preserved in a GDSII-type DRC results database generated by Calibre DRC-H, and there is no suppression or transformation of DRC results. Cell names are maintained with an optional string appended. This optional string is determined by the DRC Results Database specification statement. Recall that the flat Calibre DRC applications output a GDSII DRC results database with exactly one cell. The GDSII BNF and the semantics of DRC Check Map and DRC Maximum Vertex specification statements are identical to flat Calibre DRC.

Hierarchical Calibre applications internally create additional levels of hierarchy to support their hierarchical algorithms. These new internal cells are called “pseudo cells”. They are named “ICV_*n*” where *n* is incremental so that the cell names unique. By default, DRC results that end up in pseudo cells are transformed up the hierarchy to the first true user cell prior to being instantiated in the DRC results database.

For a GDSII-type DRC results database, no pseudo-hierarchy is instantiated. The user can reverse this suppression of pseudo-hierarchy in the DRC results database by specifying the secondary keyword PSEUDO in the required DRC Results Database specification statement. For an ASCII- or binary-type DRC results database from Calibre DRC-H, use of the PSEUDO keyword acts, in many cases, as an error-suppression mechanism. For GDSII-type DRC results databases used for mask-preparation, as opposed to DRC checking, use of the PSEUDO parameter can reduce output database size.

The DRC Keep Empty specification statement, which regulates retention of empty rule checks in ASCII and binary DRC results databases generated by Calibre DRC applications, also affects GDSII-type DRC results databases written by Calibre DRC-H. when DRC Keep Empty YES is specified in the rule file, then cell and placement records are not written to GDSII-type DRC results databases if the cell, or the placement's cell, respectively, contain no DRC results.

DRC Summary Report

The DRC summary report is created by using the DRC Summary Report specification statement in the rule file. The DRC summary report includes the following information:

- **Heading information**. The first part of the DRC summary report lists general information about the execution. The following is an example:

```
============================================================
=== CALIBRE::DRC-F SUMMARY REPORT
===
Execution Date/Time:      Wed Apr 28 15:04:33 1999
Rule File Pathname:       rule_file
Rule File Title:          Basic DRC Rule File
Layout System:            GDSII
Layout Path(s):           ./layout/basic_drc.gds
Layout Primary Cell:      basic_drc
Current Directory:        /user/johns/drc_example
User Name:                johns
Maximum Results/RuleCheck:1000
Maximum Result Vertices:  4096
DRC Results Database:     ./drc_results_db (ASCII)
Layout Depth:             ALL
Text Depth:               PRIMARY
Summary Report File:      ./drc_summary (REPLACE)
Geometry Flagging:        ACUTE = NO SKEW = NO OFFGRID = NO
                                          NONSIMPLE = NO
Excluded Cells:
CheckText Mapping:        COMMENT TEXT+RULE FILE INFORMATION
Layers:                   MEMORY-BASED
Keep Empty Checks:        YES
```

- **Runtime Warnings**. This section lists any warnings that were generated during execution.
- **Original Layer Statistics**. This section lists the original layers and the number of original geometries processed for that layer.
- **Rule Check Results Statistics**. This section lists the rule checks and the number of results generated. The DRC summary report also lists the rule checks that were not executed.
- **Summary information**. This section shows the total run time, the number of original geometries processed, the number of rule checks executed, and the number of results generated.

Summary report file in Calibre DRC

In Calibre DRC, generation of a summary report file is controlled by the presence of the rule file DRC Summary Report specification statement. When this statement is present, a summary report file will be generated in the specified filename. The keywords REPLACE and APPEND specify whether this summary report file is to be opened in replace mode or append mode. If opened in REPLACE mode, the previous contents of the file, if any, are overwritten. If opened in APPEND mode, an existing summary report file is appended to. The APPEND option is useful in creating a log of DRC runs.

LVS Report

The LVS report contains the results of an LVS run in text form. You can use this report, along with graphical results, to locate discrepancies.

Overall Structure—Flat

A flat LVS report consists of the following information:

- Transcript of connectivity extraction errors and warnings, and stamp errors, if any were found.
- LVS netlist compiler errors and warnings, if any were found.

- An LVS header section, specifying the report file name, the layout and source design names (top-level cell names are indicated in parenthesis when applicable), the rule file name, the rule file title (if a Title statement was specified), the external hcell file name (if specified), the time and date when the report was created, the current working directory, user name, Calibre version and other information.
- Overall Comparison Results section.
- Optional lists of input errors and other problems found in the layout and source.
- Optional list of discrepancies (incorrect elements). This section is divided into subsections for incorrect nets, incorrect ports, and incorrect instances.
- LVS parameters section, showing the LVS settings used.
- Optional information and warnings section.
- Optional detailed instance connections section.
- Optional list of unmatched elements.

The next example shows a flat LVS report:

```
        ##################################################
        ##                                              ##
        ##         C A L I B R E    S Y S T E M         ##
        ##                                              ##
        ##             L V S   R E P O R T              ##
        ##                                              ##
        ##################################################

REPORT FILE NAME:         mix.rep
LAYOUT NAME:              mix.gds
SOURCE NAME:              zmix.net.src ('mix')
RULE FILE:                rules
RULE FILE TITLE:          lvs rules
CREATION TIME:            Mon Jun 22 16:40:47 2000
CURRENT DIRECTORY:        /user/johns/hlvs/test
USER NAME:                johns
CALIBRE VERSION:          v8.7_30.1 Fri Jun 16 12:47:12 PDT 2000


*********************************************************
                  OVERALL COMPARISON RESULTS
*********************************************************

       #   #         ####################
        # #          #                  #
         #           #     INCORRECT    #
        # #          #                  #
       #   #         ####################


  Error:    Different numbers of nets (see below).
  Error:    Different numbers of instances (see below).

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

INITIAL NUMBERS OF OBJECTS
--------------------------

                Layout    Source         Component Type
                ------    ------         --------------
 Nets:               8        12    *
```

```
 Instances:          4         4         mn (4 pins)
                     4         4         mp (4 pins)
                     0         2    *    C (2 pins)
                  ------    ------
 Total Inst:         8        10


NUMBERS OF OBJECTS AFTER TRANSFORMATION
---------------------------------------

                 Layout    Source         Component Type
                 ------    ------         --------------
 Nets:                7        11    *

 Instances:           0         2    *    C (2 pins)
                      2         2         INV (2 pins)
                      1         1         NAND2 (3 pins)
                  ------    ------
 Total Inst:          3         5

*=Number of objects in layout different from number in source.


************************************************************
                      INCORRECT OBJECTS
************************************************************

LEGEND:
-------

  ne  = Naming Error (same layout name found in source
        circuit, but object was matched otherwise).


************************************************************
                    INCORRECT NETS

DISC#  LAYOUT NAME                                ne  SOURCE NAME
************************************************************

  1    ** missing net **                              0/1
```

```
-------------------------------------------------------------

  2    ** missing net **                          0/2

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

  3    ** missing net **                          1/1

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

  4    ** missing net **                          1/2


*************************************************************
                       INCORRECT INSTANCES

DISC#  LAYOUT NAME                              ne  SOURCE NAME
*************************************************************

  5    ** missing instance **                     0/C1  C

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

  6    ** missing instance **                     1/C1  C
```

```
*************************************************************
                      LVS PARAMETERS
*************************************************************

o LVS Setup:

   Component Type Properties:
   Subtype Property:
   Pin Name Properties:
   Power Net Names:                          VCC
   Ground Net Names:                         GROUND
   Non User Name Port:
   Non User Name Net:
   Non User Name Instance:
   Ignore Ports:                             YES
   All Capacitor Pins Swappable:             NO
   Reduce Series Mos Transistors:            NO
   Reduce Parallel Mos Transistors:          YES [TOLERANCE L 0]
   Recognize Gates:                          ALL MIX SUBTYPES
   Reduce Split Gates:                       YES
   Reduce Parallel Bipolar Transistors:      YES
   Reduce Series Capacitors:                 YES
   Reduce Parallel Capacitors:               YES
   Reduce Series Resistors:                  YES
   Reduce Parallel Resistors:                YES
   Reduce Parallel Diodes:                   YES
   Unused Device Filter Options:
   LVS Report Options:
   Reverse WL:                               NO
   Preserve Parametrized Cells:              NO
   Property Resolution Maximum:              8
   Signature Maximum:                        None
   Layout Case:                              NO
   Source Case:                              NO
   Compare Case:                             NO
   Report List Limit:                        50
```

```
*************************************************************
                  INFORMATION AND WARNINGS
*************************************************************

           Matched  Matched  Unmatched  Unmatched  Component
           Layout   Source   Layout     Source     Type
           -------  -------  ---------  ---------  ---------
Nets:             7        7          0          4

Instances:        0        0          0          2   C
                  2        2          0          0   INV
                  1        1          0          0   NAND2
           --------  ------- ---------  ---------
Total Inst:       3        3           0         2

o Statistics:

   2 layout nets had all their pins removed.
   2 source nets had all their pins removed.

   1 net was matched arbitrarily.

o Initial Correspondence Points:

   Nets:         VCC GROUND


*************************************************************
                        SUMMARY
*************************************************************
Total CPU Time:       0 sec
Total Elapsed Time:   0 sec
```

Overall Structure — Hierarchical

A hierarchical LVS report consists of the following information:

- LVS netlist compiler errors and warnings, if any were found.
- LVS header section, specifying the report file name, the layout and source design names (top level cell names are indicated in parenthesis when

applicable), the rule file name, the rule file title (if a Title statement was specified), the external hcell file name (if specified), the time and date when the report was created, the current working directory, user name, Calibre version and other information.

- Overall Comparison Results section, consisting of:
  - Primary comparison status message. This status is `CORRECT` if all individual cells are correct, `INCORRECT` if at least one cell is incorrect, and, otherwise, `NOT COMPARED` if at least one cell is not compared. The usual graphics are provided as well (check mark and smiley face, or X respectively). Refer to section "Overall Comparison Results" for the meaning of individual status messages.
  - Secondary comparison status messages. This is a collection of all secondary comparison status messages reported for individual cells. Refer to section "LVS Netlist Compiler" for the meaning of individual status messages.
  - A Cell Summary, listing the primary comparison status for each individual cell (`CORRECT`, `INCORRECT` or `NOT COMPARED`). The respective layout and source cell names are indicated. Cells that exist only in the layout or only in the source but contain input errors (for example, missing property errors) appear in the `CELL SUMMARY` section as well. In that case, the layout or source cell name is indicated with no corresponding cell name in the other design. This section may be omitted if global problems are found in the design.
  - Optional sections describing global problems found in the design, such as hierarchy cycles.
  - LVS Parameters section. This section shows the LVS program configuration.
- Cell-by-cell Comparison Results section. This section represents each hierarchical correspondence cell (hcell) with a section of its own. The section for each individual cell resembles a complete flat LVS report; including a header, overall comparison results, optional input errors, optional discrepancies, optional information and warnings, optional

detailed instance connections, optional list of unmatched elements for the cell, and so on.

A hierarchical LVS report example is shown below:

```
      ################################################
      ##                                            ##
      ##         C A L I B R E   S Y S T E M        ##
      ##                                            ##
      ##             L V S   R E P O R T            ##
      ##                                            ##
      ################################################

REPORT FILE NAME:         mix.rep
LAYOUT NAME:              z.net ('mix')
SOURCE NAME:              zmix.net.src ('mix')
RULE FILE:                rules
RULE FILE TITLE:          lvs rules
HCELL FILE:               cells
CREATION TIME:            Mon Jun 22 16:42:00 2000
CURRENT DIRECTORY:        /user/johns/hlvs/test
USER NAME:                johns
CALIBRE VERSION:          v8.7_30.1 Fri Jun 16 12:47:12 PDT 2000


                  OVERALL COMPARISON RESULTS


              #   #         ######################
               # #          #                    #
                #           #     INCORRECT      #
               # #          #                    #
              #   #         ######################

  Error:    Different numbers of nets (see below).
  Error:    Different numbers of instances (see below).
```

```
*************************************************************
                       CELL  SUMMARY
*************************************************************

  Result         Layout                        Source
  -----------    -----------                   --------------
  INCORRECT      inv                           inv
  CORRECT        nand                          nand
  CORRECT        mix                           mix

*************************************************************
                        LVS PARAMETERS
*************************************************************
o LVS Setup:

   Component Type Properties:
   Subtype Property:
   Pin Name Properties:
   Power Net Names:                          VCC
   Ground Net Names:                         GROUND
   Non User Name Port:
   Non User Name Net:
   Non User Name Instance:
   Ignore Ports:                             YES
   All Capacitor Pins Swappable:             NO
   Reduce Series Mos Transistors:            NO
   Reduce Parallel Mos Transistors:          YES [TOLERANCE L 0]
   Recognize Gates:                          ALL MIX SUBTYPES
   Reduce Split Gates:                       YES
   Reduce Parallel Bipolar Transistors:      YES
   Reduce Series Capacitors:                 YES
   Reduce Parallel Capacitors:               YES
   Reduce Series Resistors:                  YES
   Reduce Parallel Resistors:                YES
   Reduce Parallel Diodes:                   YES
   Unused Device Filter Options:
   LVS Report Options:
   Reverse WL:                               NO
   Preserve Parametrized Cells:              NO
   Property Resolution Maximum:              8
   Signature Maximum:                        None
   Layout Case:                              NO
   Source Case:                              NO
```

```
   Compare Case:                          NO
   Report List Limit:                     50


                    CELL COMPARISON RESULTS

               #   #         ######################
                # #          #                    #
                  #          #     INCORRECT      #
                # #          #                    #
               #   #         ######################

  Error:    Different numbers of nets (see below).
  Error:    Different numbers of instances (see below).

LAYOUT CELL NAME:         inv
SOURCE CELL NAME:         inv

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

INITIAL NUMBERS OF OBJECTS
--------------------------

                Layout    Source         Component Type
                ------    ------         --------------
 Ports:              3         3

 Nets:               4         6    *

 Instances:          1         1         MN (4 pins)
                     1         1         MP (4 pins)
                     0         1    *    C (2 pins)
                ------    ------
 Total Inst:         2         3


NUMBERS OF OBJECTS AFTER TRANSFORMATION
---------------------------------------

             Layout   Source       Component Type
             ------   ------       --------------
 Ports:           3        3
```

```
 Nets:            4        6    *

 Instances:       0        1    *  C (2 pins)
                  1        1       INV (2 pins: output input)
              ------    ------
 Total Inst:      1         2

*=Number of objects in layout different from number in source.


************************************************************
                      INCORRECT OBJECTS
************************************************************

LEGEND:
-------

  ne  = Naming Error (same layout name found in source
        circuit, but object was matched otherwise).


************************************************************
                      INCORRECT NETS

DISC#  LAYOUT NAME                          ne  SOURCE NAME
************************************************************

  1    ** missing net **                                 1

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

  2    ** missing net **                                 2


************************************************************
                      INCORRECT INSTANCES

DISC#  LAYOUT NAME                            ne  SOURCE NAME
************************************************************

  3    ** missing instance **                          C1  C
```

```
*********************************************************
                   INFORMATION AND WARNINGS
*********************************************************

            Matched  Matched  Unmatched  Unmatched  Component
            Layout   Source   Layout     Source     Type
            -------  -------  ---------  ---------  ---------
   Ports:         3        3          0          0
   Nets:          4        4          0          2
   Instances:     0        0          0          1  C
                  1        1          0          0  INV
            -------  -------  ---------  ---------
   Total Inst:    1        1          0          1

o Statistics:

   2 layout nets were reduced to passthrough nets.
   2 source nets were reduced to passthrough nets.

o Initial Correspondence Points:

   Ports:        VCC GROUND OUT
   Nets:         IN


                  CELL COMPARISON RESULTS


              #       ###################       _   _
            #         #                 #       *   *
          #   #       #     CORRECT     #         |
           # #        #                 #       \___/
            #         ###################


LAYOUT CELL NAME:         nand
SOURCE CELL NAME:         nand

---------------------------------------------------------
```

```
INITIAL NUMBERS OF OBJECTS
--------------------------

                Layout    Source         Component Type
                ------    ------         --------------
 Ports:              4         4
 Nets:               6         6
 Instances:          2         2         MN (4 pins)
                     2         2         MP (4 pins)
                ------    ------
 Total Inst:         4         4


NUMBERS OF OBJECTS AFTER TRANSFORMATION
---------------------------------------

                Layout    Source         Component Type
                ------    ------         --------------
 Ports:              4         4
 Nets:               5         5
 Instances:          1         1         NAND2 (3 pins)
                ------    ------
 Total Inst:         1         1


************************************************************
                    INFORMATION AND WARNINGS
************************************************************


            Matched  Matched  Unmatched  Unmatched  Component
            Layout   Source   Layout     Source     Type
            -------  -------  ---------  ---------  ---------
   Ports:         4        4          0          0
   Nets:          5        5          0          0
```

```
   Instances:    1        1          0          0    NAND2
           -------  -------  ---------  ---------
   Total Inst:   1        1          0          0
```

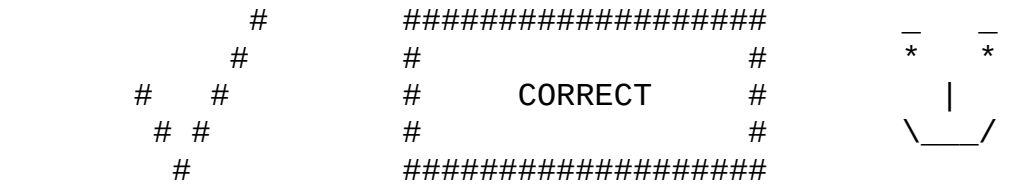
o Statistics:

   2 layout nets were reduced to passthrough nets.
   2 source nets were reduced to passthrough nets.

o Initial Correspondence Points:

   Ports: VCC GROUND I1 I2
   Nets: OUT

CELL COMPARISON RESULTS (TOP LEVEL)



Warning:Ambiguity points were found and resolved arbitrarily.

LAYOUT CELL NAME: mix
SOURCE CELL NAME: mix

NUMBERS OF OBJECTS

	Layout	Source	Component Type
Nets:	4	4	

```
 Instances:          2          2    inv (3 pins): ground vcc out
                     1          1    nand (4 pins)
                  ------     ------
 Total Inst:         3          3


*************************************************************
                     INFORMATION AND WARNINGS
*************************************************************


              Matched   Matched   Unmatched   Unmatched   Component
              Layout    Source    Layout      Source      Type
              -------   -------   ---------   ---------   ---------
   Nets:            4         4           0           0

   Instances:       2         2           0           0   inv
                    1         1           0           0   nand
              -------   -------   ---------   ---------
   Total Inst:      3         3           0           0

o Statistics:

   1 net was matched arbitrarily.

o Initial Correspondence Points:

   Nets:          VCC GROUND


*************************************************************
                              SUMMARY
*************************************************************

Total CPU Time:       0 sec
Total Elapsed Time:   0 sec
```

Overall Structure — SPICE Syntax Check

This kind of report is written only from the -cl and -cs command line options of Calibre LVS (Spice syntax check mode).

A Spice syntax check report consists of:

- LVS netlist compiler errors and warnings, if any were found.
- A header section, specifying the report file name, the layout and/or source design names (top level cell names are indicated in parenthesis when applicable), the rule file name, the rule file title (if a `Title` statement was specified), the time and date when the report was created, the current working directory, user name, Calibre version and other information.
- Overall syntax check results section.

Here is an example:

```
LVS Netlist Compiler - Errors and Warnings for "z2.net"
-------------------------------------------------------

Error: Syntax Error in file "z2.net" at line 7.
  Expected "Cxxx n1 n2 <mname> <c <tc1 <tc2 <scale>>>> <M=m> <L=l> <W=w>
            + <parnam=pval> ... <$SUB=ns> <$[mname] | $.MODEL=mname>
            + <$A=a> <$P=p> <$X=x> <$Y=y>"



                  ##################################################
                  ##                                              ##
                  ##        C A L I B R E   S Y S T E M           ##
                  ##                                              ##
                  ##            L V S   R E P O R T               ##
                  ##                                              ##
                  ##################################################



REPORT FILE NAME:         lvs.rep
LAYOUT NAME:
SOURCE NAME:              z2.net ('top')
RULE FILE:                rules
CREATION TIME:            Wed Apr 17 15:15:19 2002
CURRENT DIRECTORY:        /scratch1/kobi/play
USER NAME:                kobi
CALIBRE VERSION:          v9.1_5.1    Tue Apr 16 22:14:21 PDT 2002
```

```
*******************************************************************************
                              OVERALL SYNTAX CHECK RESULTS
*******************************************************************************

                    #   #         ##########################
                     # #          #                         #
                      #           #  SYNTAX CHECK FAILED    #
                     # #          #                         #
                    #   #         ##########################


            Syntax errors were found in the source.


*******************************************************************************
                                          SUMMARY
*******************************************************************************

Total CPU Time:      0 sec
Total Elapsed Time:  0 sec
```

Analyzing the LVS Report

View the report as a “results summary” of the LVS comparison run as follows:

- Read the error and warning messages at the beginning of the report.
  - The report first shows connectivity extraction errors and warnings, and stamp errors. For example, it will report a warning giving text values and locations if conflicting signal name texts were found on a single net.
  - The report next shows netlist compiler errors and warnings. For example, it would report any syntax errors or undefined subcircuits of a Spice deck if you are using one for a reference netlist.

    Errors reported here cause LVS to abort. Investigate these errors and correct them before running LVS again.

- Look for the following, before the Numbers of Objects After Transformation section:

  - Correct layout and source files, and the correct rules file pathname. These are listed in the report header.

  - Correct or Incorrect errors or warning messages in the Overall Comparison Results section. Investigate the messages to debug an Incorrect run.

- Check the Number of Objects After Transformation report for differences between layout and source counts. Differences are indicated with asterisk (*) characters. The following is an example of this section.

  Ensure that LVS is finding the same kind and number of objects in the layout and source. Differences can indicate a rules problem, setup problem, or viewpoint problem. Consult your rule file writer if necessary.

- Analyze the Incorrect Objects section. This section lists the differences of nets and instances between the layout and source that LVS could not match, or could partially match with some differences.

  - The Incorrect Nets section explains net differences. For example, a net in the source may have connections that a net in the layout does not.

  - The Incorrect Instances section explains instance differences. For example, an instance in the layout may have connections that do not match the connections of the source instance, may be missing, or any other type of discrepancy.

  Often, data will be redundant between the two sections.

- Skim the Information and Warnings section before trying to understand the problem in detail.

  The Information and Warnings section includes statistics and information about the comparison run, including an exact set of numbers on how many objects in both source and layout are identified and not identified.

This section also reports information about isolated nets that are deleted, nets that are reduced to pass-through nets, and initial correspondence points.

The Matched/Unmatched statistics specifies whether correspondence exists between the source and layout.

- Review the Detailed Instance Connections section. This section shows detailed information about matched instances whose pins are listed as part of discrepancies on nets. For each pair of instances, the information includes: the layout instance, corresponding source instance, nets connected to their pins in the layout and source respectively, and corresponding nets in the source and layout, respectively.

Calibre LVS results can be viewed graphically with the Calibre RVE/QDB-H query server when the Mask SVDB Directory specification statement is specified in the rule file. Refer to chapter 15, "RVE/QDB-H and Query Server" chapter for further information.

Errors and Warnings

The following sections discuss possible errors and warnings that can appear in the LVS report.

- **Connectivity Extraction**

  In flat LVS, a transcript of any connectivity extraction errors and warnings that were found during the LVS run appears at the top of the report, prior to the LVS header section. Connectivity extraction errors and warnings are described in the *Standard Verification Rule Format (SVRF) Manual*.

- **Stamp Operation**

  In flat LVS, a transcript of any stamp errors that were found during the LVS run appears at the top of the report, prior to the LVS header section.

  The Stamp operation maps electrical net references from one layer to another. There are two kinds of stamp errors:

- A list of locations where layer X cannot be stamped by layer Y is provided. Each location is a vertex of a polygon on layer x that is not overlapped by any polygon on layer y.

  ```
  Missing connections STAMPing layer X by layer Y.
  ```

- A list of locations where the node reference assignment is ambiguous. For example, where a polygon on layer X is overlapped by two or more polygons on layer Y belonging to different electrical nets. For each location, the net IDs and net names of two of the conflicting nets are provided; in addition, one vertex of the layer x polygon is provided.

  ```
  Conflicting connections STAMPing layer X by layer Y.
  ```

- **LVS Netlist Compiler**

  When a Spice netlist is used as input, a transcript of errors and warnings that are found during compilation appears at the top of the report, prior to the LVS header section. Errors cause LVS to abort, but warnings do not.

LVS Report Listing Conventions

Discrepancies

LVS reports differences between the layout and source circuits as discrepancies. A serial discrepancy number identifies each discrepancy in the LVS report. The layout elements involved in each discrepancy appear on the left hand side of the report. The source elements appear on the right hand side of the report. In the following example, LVS reports a “missing net” discrepancy:

```
------------------------------------------------------------
5    ** missing net **                  /N$716
------------------------------------------------------------
```

The discrepancy number is 5, source net N$716 is missing in the layout.

Net, Instance and Port Identification

- **Eddm or V7.0 erel.** A schematic net or instance, in a Eddm or V7.0 erel design, is identified by its hierarchical pathname. A hierarchical pathname has one of two forms:

  /<instance-name-1>/.../<instance-name-*n*>/<net-name>

  /<instance-name-1>/.../<instance-name-*n*>/<instance-name>

  where "*n*" is zero or more. Instance names are either user given names, or system generated names in the form i$<number>. Net names are either user given names, or system generated names in the form n$<number>. Examples:

  ```
  /CLOCK
  /i$23
  /i$767/MP/MP/i$702
  ```

  A schematic port, in a Eddm or V7.0 erel design, is identified by its name. Example: `INPUT1`

- **Spice netlist.** A hierarchical pathname identifies a net or instance in a Spice netlist. The pathname has one of three forms:

  /*subckt_call_name_1*/ … /*subckt_call_name_n*/*node_name*
  /*subckt_call_name_1*/ … /*subckt_call_name_n*/*element_name*
  /*subckt_call_name_1*/ … /*subckt_call_name_n*/*subckt_call_name*

  where *'n'* is zero or more. In flat LVS, <subckt-call-name-1> through <subckt-call-name-*n*> appear without the preceding 'X'. Examples:

  Flat:

  ```
  NETA
  1/netb
  fred/4/7/R7
  fred/Xabc
  ```

  Hierarchical:

  ```
  NETA
  X1/netb
  ```

```
Xfred/X4/X7/R7
Xfred/Xabc
```

A design port in a Spice netlist is identified by its name. For example:

```
PORT1
```

- **Mask layout.** LVS identifies the following:
  - An extracted layout device instance by its numerical ID followed by an (x,y) layout location.
  - A named layout net by an (x,y) layout location.
  - An unnamed layout net by its numerical ID followed by an (x,y) layout location.

Logic Gate Identification

LVS forms logic gates internally by the logic gate recognition feature. A logic gate instance is identified by its type, in parentheses, followed by a list of transistor instances forming the gate. For example:

```
(INV)

Transistors:
  /I$767/MP/MP/I$702
  /I$767/MN/MN/I$786
```

This shows that transistors /I$767/MP/MP/I$702 and /I$767/MN/MN/I$78 form an inverter.

Instance Pin Identification

LVS identifies an instance pin by the instance name or ID, and location; followed by a colon (:) and the pin name. For example:

```
I$702:G -
```

identifies pin G of schematic instance I$702, and

```
27(230,540):B -
```

identifies pin B of an extracted mask device with an ID of 27, and a location of X=230, Y=540.

Logic Gate Pin Identification

LVS identifies a pin of an internally generated logic gate with the following format:

```
(gate_type): {INPUT | OUTPUT}
```

In the case of an INPUT pin, this identification is followed by a list of transistors, which are part of the logic gate implementation, and whose gate (G) pins form that particular input of the logic gate.

In the case of an OUTPUT pin, this identification is followed by a list of transistors, which are part of the logic gate implementation, and whose source (S) or drain (D) pins form the output of the gate. For example:

```
(INV):OUTPUT
  /I$767/MP/MP/I$702:D
  /I$767/MN/MN/I$786:S
```

This shows the output pin of an inverter (INV). The D (drain) pin of transistor /I$767/MP/MP/I$702 and the S (source) pin of transistor /I$767/MN/MN/I$786 form the inverter.

Unconnected Instance Pin Identification

LVS treats an unconnected instance pin as if it was connected to a virtual net that has no other connections. Such a net is identified by the corresponding pin. For example:

```
Net Pin INST1:IN(34)
```

refers to the “virtual net” that leads to unconnected instance pin

```
INST1:IN(34).
```

Overall Comparison Results

The overall results of the LVS run are listed in the Overall Comparison Results section of the LVS report. In hierarchical LVS, a similar section, titled Cell Comparison Results, is also present for each individual cell, and summarizes the results for that cell.

The following sections show an example, summarize possible messages, and discuss the section.

```
*************************************************************
                 OVERALL COMPARISON RESULTS
*************************************************************
           #   #         ####################
            # #          #                  #
             #           #     INCORRECT    #
            # #          #                  #
           #   #         ####################
  Error:    Different numbers of nets (see below).
  Error:    Different numbers of instances (see below).

-------------------------------------------------------------
INITIAL NUMBERS OF ELEMENTS
---------------------------

               Layout    Source          Component Type
               ------    ------          --------------
 Nets:            966       967    *
 Instances:      1182      1184    *    MP (4 pins)
                 1182      1184    *    MN (4 pins)
               ------    ------
 Total Inst:     2364      2368

NUMBERS OF ELEMENTS AFTER TRANSFORMATION
----------------------------------------
```

```
                Layout    Source          Component Type
                ------    ------          --------------
Nets:              784       785    *
Instances:         471       472    *    MP (4 pins)
                   471       472    *    MN (4 pins)
                   347       348    *    INV (2 pins)
                    59        59         NOR2 (3 pins)
                   123       123         NAND2 (3 pins)
                ------    ------
Total Inst:      1471      1474

     * = Number of elements in layout different from number
         in source.
```

Primary Messages

The overall results section begins with one of the primary comparison status messages listed in Table 14-1.

Table 14-1. Primary Messages

Message	Description
CORRECT	The layout connectivity is equivalent to the source connectivity.
CORRECT except for naming or swap-override errors	The layout connectivity is equivalent to the source connectivity, except for differences in element names, or violations of restrictions on swapping of pins.
INCORRECT	Discrepancies were detected between the two circuits.
NOT COMPARED	LVS aborted prior to the comparison stage because of problems in the layout or source data. The actual problems are listed further down in the report.

Secondary Messages

Secondary messages, if any, follow the primary comparison status message. These messages, listed in Table 14-2, present additional information on the differences between the compared circuits. “Error” conditions cause the overall result to be incorrect, while “warning” conditions do not.

Table 14-2. Secondary Messages- Errors

Error	Description
Error: Connectivity errors.	Connectivity errors were found, for example, incorrect nets, incorrect instances, unmatched elements.
Error: Incorrect names for power/ground nets.	The lvs_power_names and lvs_ground_names application variables specify badly formed power or ground names. This error causes LVS to abort prior to the comparison stage. The “Errors in Names Given for Power/Ground Nets” section lists the badly formed names.
Error: Errors in layout.	LVS found errors in the layout that caused it to abort prior to the comparison stage. The “Layout Input Layers” section lists the actual problems.
Error: Errors in source.	LVS found errors in the source that caused it to abort prior to the comparison stage. The “Source Input Layers” section lists the actual problems.
Error: Power net missing in layout. Power net missing in source. Ground net missing in layout. Ground net missing in source.	Indicates that a power or ground net is missing in the layout or source. These errors cause LVS to abort prior to the comparison stage. These errors are issued only when they are relevant and only when the absence of a power and/or ground net prevents LVS from generating useful results in the comparison stage. For example, if logic gate recognition is requested, power and ground nets are provided, and the source contains logic gates, then a “Power net missing” or “Ground net missing” error is issued for the layout. But if the source contains no logic gates, then those error messages may not be issued. These error messages may be combined together on one line.

Table 14-2. Secondary Messages- Errors [continued]

Error	Description
Error: Power or ground net missing.	Indicates that “Power net missing” or “Ground net missing” errors were issued for one or more cells in hierarchical LVS. This error may appear in the “Overall Comparison Results” section in hierarchical LVS, with more information provided in the report sections pertaining to the individual cells.
Error: Components with non-identical signal pins were found.	The number of signal pins or the signal pin names of some layout components differ from the corresponding schematic components. The “Component Types with Non-Identical Signal Pins” section lists the problematic component types and pin names.
Error: Different numbers of ports (see below).	The number of ports in the layout differ from the number of ports in the source. The actual numbers of ports appear further down in the report.This error is issued only when the lvs_ignore_ports application variable is false.
Error: Different numbers of nets (see below).	The number of nets in the layout differ from the number of nets in the source. The actual numbers of nets appear further down in the report.
Error: Different numbers of instances (see below).	The number of instances, of one or more component type, in the layout differ from the number of instances, of the corresponding component type, in the source. The actual numbers of instances of each component type appear further down in the report.
Error: Instances of different types or subtypes were matched.	Some layout instances were matched to some schematic instances, which had different component types or subtypes. These instances are listed as discrepancies.

Table 14-2. Secondary Messages- Errors [continued]

Error	Description
Error: Cells with non-floating extra pins.	A cell contains instances of other cells that have extra pins in the source or layout, and in those instances the extra pins are connected to other elements (they are not floating). The “Instances of Cells with Non-Floating Extra Pins” section reports these instances as discrepancies.
Error: Property errors.	LVS found differences in values of source and layout properties. The “Property Errors” section lists these instances, with their corresponding property values and error percentages.
Error: Substrate pin errors.	LVS found instances with incorrect substrate connections. The “Incorrect Substrate Connections” section reports these discrepancies. This error is reported only when LVS Soft Substrate Pins YES is indicated in the rule file.
Error: Components with non-identical power or ground pins were found.	The number of power or ground pins, or the power or ground pin names of some layout components differ from the corresponding schematic components. The “Component Types with Non-identical Signal Pins” section lists these component types.
Error: Cell hierarchy contains a cycle	Indicates that the cell hierarchy, consisting of the layout and source hierarchies as well as cell correspondence information, contained a cycle. This error appears only in hierarchical LVS. It is a global error that causes hierarchical LVS to abort without comparing individual cells. Details are provided in a report section titled “Cells in Hierarchy Forming a Cycle”.

Table 14-2. Secondary Messages- Errors [continued]

Error	Description

Error: Many-to-many correspondence in cell names.	Indicates that the cell correspondence specified to hierarchical LVS in the hcell list leads to a many-many relationship. This is a global error that causes hierarchical LVS to abort without comparing individual cells. This error may appear only in hierarchical LVS.
	Here's an example of an hcell list with many-many correspondence:
	
	`aa aa`
	`aa bb`
	`bb bb`

Table 14-3. Secondary Messages- Warnings

Warning	Description
Warning: Bad devices in layout.	Issued only in mask LVS. Means that badly formed devices were found while recognizing devices from layout geometries. The bad devices are listed in the "Information and Warnings" section of the report.
Warning: Components with non-identical power or ground pins were found.	The number of power or ground pins, or the power or ground pin names of some layout components differ from the corresponding schematic components. The "Information and Warnings" section lists these component types.
Warning: Unbalanced smashed mosfets were matched.	The source contains at least one group of MOS transistors connected in parallel. That is, implemented in the layout with only a single transistor or with a group of parallel transistors that consists of a different number of elements. The "Information and Warnings" section lists these transistors.

Table 14-3. Secondary Messages- Warnings [continued]

Warning	Description
Warning: User-names were overridden.	Some layout elements with user-given names were matched to source elements with different user-given names. The “Information and Warnings” section lists these elements.
Warning: Ambiguity points were found and resolved arbitrarily.	The compared circuits contain interchangeable parts. The “Information and Warnings” section lists the elements which were matched arbitrarily.
Warning: Extra ports in layout.	A cell has extra ports in the layout. The “Instances of Cells with Non-Floating Extra Pins” section reports these cells as discrepancies if instances of the cell exist where the extra pins are not floating.
Warning: Extra ports in source.	A cell has extra ports in the source. The “Instances of Cells with Non-Floating Extra Pins” section reports these cells as discrepancies if instances of the cell exist where the extra pins are not floating.
Warning: FY, GY, M, and N filters did not connect some s and d pins.	Unused device filter options FY, GY, M, or N did not connect together the source and drain nets of some transistor devices that were filtered out because the source and drain were connected to different pads. the transistors are listed in the “Information and Warnings” section of the LVS report.
Warning: Source and layout refer to the same data.	Means that the source and layout, as seen by the LVS circuit comparison module, refer to the same data. For example, in Calibre LVS, if the -spice command line option is not used, then the warning appears when Source Path is identical to Layout Path and Source Primary is identical to Layout Primary (or neither Primary name is specified). If the -spice command line option is used, then the indicated -spice file name replaces the original Layout Path in triggering the warning.

Numbers of Elements

The numbers of ports, numbers of nets, the numbers of instances of each component type in the layout and source are specified in the Overall (or Cell) Comparison Results section of the report. The total number of instances are specified as well.

For each component type, the number of pins is indicated. When applicable - for example, when components with the same name but different pin count are mixed together - the number of pins may be replaced with the actual pin names. For example:

```
FOO (5 pins): (a b) c (d e)
```

Logically equivalent or swappable pins are shown in parenthesis. In the above example, pins a and b are swappable, and pins d and e are also swappable. Note however that pin swappability is *not* indicated for logic gates formed by LVS.

When LVS performs logic gate recognition, series or parallel device reduction, filtering of nets or devices, or any other internal transformation of the compared circuits which results in a change in the number of nets or instances, the port, net and instance numbers are shown both for the original circuits (`INITIAL NUMBERS OF OBJECTS section`) and for the new modified circuits (`NUMBERS OF OBJECTS AFTER TRANSFORMATION section`).

Component types that have different number of instances in the source and layout are marked with an asterisk (*). Component types with standard device names but non-standard pin configurations are marked with the text `** non standard device **`.

Here is a complete example of the `OVERALL COMPARISON RESULTS` section.

```
************************************************************
                OVERALL COMPARISON RESULTS
************************************************************


          #   #         ####################
           # #          #                  #
            #           #     INCORRECT    #
           # #          #                  #
          #   #         ####################


  Error:    Different numbers of nets (see below).
  Error:    Different numbers of instances (see below).

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

INITIAL NUMBERS OF OBJECTS
--------------------------

               Layout    Source          Component Type
               ------    ------          --------------
 Nets:            966       967    *

 Instances:      1182      1184    *    MP (4 pins)
                 1182      1184    *    MN (4 pins)
               ------    ------
 Total Inst:     2364      2368

NUMBERS OF OBJECTS AFTER TRANSFORMATION
---------------------------------------

               Layout    Source          Component Type
               ------    ------          --------------
 Nets:            784       785    *

 Instances:       471       472    *    MP (4 pins)
                  471       472    *    MN (4 pins)
                  347       348    *    INV (2 pins)
                   59        59         NOR2 (3 pins)
                  123       123         NAND2 (3 pins)
               ------    ------
 Total Inst:     1471      1474
```

```
* = Number of elements in layout different from number
    in source.
```

Overall SPICE Syntax Check Results

This section appears only in reports created with the -cl or -cs command line options of Calibre LVS (Spice syntax check mode).

The overall syntax check results are shown in the `OVERALL SYNTAX CHECK RESULTS` section of the LVS report. The overall syntax check results section begins with a primary syntax check status which is one of the following:

- `SYNTAX OK`
  Means that no syntax errors were found.

- `SYNTAX CHECK FAILED`
  Means that syntax errors were found.

The primary syntax check status may be followed by one or two secondary syntax check status messages, as follows:

- `Syntax errors were found in the layout.`
  Means that syntax errors were found in the layout netlist (the netlist indicated with Layout Path statement in the rule file).

- `Syntax errors were found in the source.`
  Means that syntax errors were found in the source netlist (the netlist indicated with Source Path statement in the rule file).

Errors in Names Given for Power/Ground Nets

This sub-section contains a list of badly formed power or ground names. A power or ground name is badly formed if it is classified as a non-user-given net name by LVS criteria in both layout and source. Table 14-4 shows the two sub-sections.

Table 14-4. Power/Ground Net Errors

Error	Description
Badly Formed Power/Ground Net Names	Contains a list of badly formed power or ground names found in the LVS Power Name and LVS Ground Name specification statements. A name is badly formed if it is not considered user-given by LVS criteria; namely, if it starts with “n$”, “N$”, “i$”, “I$”, or contains a slash (/) character. Badly formed power or ground names cause LVS to abort prior to the comparison stage.
Contradictory Power/Ground Net Names	Contains a list of names that are specified as both power and ground names; for example, names that appear both in the LVS Power Name and LVS Ground Name specification statements. Contradictory power or ground names cause LVS to abort prior to the comparison stage.

Component Types with Non-Identical Signal Pins

This section is present only if LVS found component types with different numbers of signal pins or different signal pin names in the layout and source. The section lists the component types and pin names, and indicates each pin as missing in the layout or source component type. The following is an example of this section:

```
*************************************************************
        COMPONENT TYPES WITH NON-IDENTICAL SIGNAL PINS
*************************************************************
(Component types that have different numbers of signal pins or
different signal pin names in the layout and source are listed
below. Layout pins missing in the source and source pins
missing in the layout are ignored by the comparison algorithm.
Note that differences in power or ground pins, if any, are
listed separately in the INFORMATION AND WARNINGS section.)
 Layout Pin Name      Source Pin Name         Component Type
 ---------------      -----------------       --------------
 ** missing pin **    B                       MN
 RESET                ** missing pin **       REG5
*************************************************************
```

In this example, component type MN has a pin B in the source but not in the layout, and component type REG5 has a pin RESET in the layout but not in the source. MN and REG5 are component types, not instance names (there can be many instances of MN and REG5).

Input Errors

LVS lists Input errors in two sections of the report: “Layout Input Errors” and “Source Input Errors”. Input errors indicate severe conditions in the layout or source, respectively, that cause LVS to abort prior to the comparison stage. The following lists types of input errors:

Table 14-5. Input Errors

Error	Description
MISSING COMPONENT TYPES	A list of instances whose component types cannot be determined.
MISSING PIN NAMES	A list of instances, grouped by component type, that have pins whose names cannot be determined. Component subtypes are indicated in parentheses, if specified.

Table 14-5. Input Errors

Error	Description
BAD INSTANCES	A list of built-in device type instances that have numbers of pins or pin names that do not follow the LVS conventions. The instances are grouped by component type. Component subtypes are indicated in parentheses, if specified. Refer to the section "Built-in Device Types" in chapter 10.
CONFLICTING INSTANCES	A list of instances with conflicting pin configurations. The instances are grouped by component type. The first instance in each group represents one configuration of pins. The other instances in each group have numbers of pins, pin names, values of the swap_set property, or values of the my_net property, which differ from the first instance. Component subtypes are indicated in parentheses, if specified. Refer to "Instance Pins and Pin Names" in chapter 10.
SERIES PIN NAMES NOT FOUND	This is a list of instances that are subject to a specification statement of the form LVS REDUCE ... SERIES *pin-name-1 pin-name-2* but have no pins called *pin-name-1* or have no pins called *pin-name-2*. For each instance, LVS indicates the component type, optional subtype, instance name and expected pin names (one pin per line).
AMBIGUOUS SERIES PIN NAMES SPECIFIED	This is a list of instances that are subject to a specification statement of the form LVS REDUCE ... SERIES *pin-name-1 pin-name-2* but have more then one pin called *pin-name-1* or more than one pin called *pin-name-2*. For each instance, LVS indicates the component type, optional component subtype, instance name and ambiguous pin names.

Table 14-5. Input Errors

Error	Description
INCORRECT SERIES PIN SWAP SETS	This is a list of instances that are subject to a specification statement of the form LVS REDUCE ... SERIES *pin-name-1 pin-name-2* but in which pins *pin-name-1* and *pin-name-2* of the instance are swappable with other pins (i.e. pins not named in the particular LVS Reduce ... Series statement). For each instance, LVS indicates the component type, optional component subtype, instance name and offending pin names (one pin per line).

Hierarchical Cells Forming a Cycle

This section appears in the hierarchical LVS report if a cycle is found in the cell hierarchy. The cell hierarchy consists of the layout and source hierarchies as well as cell correspondence information. Cycles in the cell hierarchy are global errors that cause hierarchical LVS to abort prior to the comparison stage. This section may appear only in hierarchical LVS reports and does not appear in flat LVS reports.

The report indicates cell names that form the cycle. For example:

```
*************************************************************
            CELLS IN HIERARCHY FORMING A CYCLE
*************************************************************

   Layout Cells                    Source Cells
   ------------                    ------------
   lay3
   lay2
   lay1                            src4
                                   src3
                                   src2
   lay4                            src1
   lay3
```

This means the following:

Layout cell lay3 contains an instance of lay2
    lay2 contains an instance of lay1
    lay1 corresponds to source cell src4 in the hcell list.
Source cell src4 contains an instance of src3
    src3 contains an instance of src2
    src2 contains an instance of src1
    src1 corresponds to layout cell lay4 in the hcell list.
Layout cell lay4 contains an instance of lay3 (closing the cycle).

Note that cycles may be present in the layout hierarchy, or in the source hierarchy, or they may consist of a combination of the layout hierarchy, the source hierarchy, and the cell correspondence as shown in the example above. (Cell correspondence is specified explicitly in the hcell list or implicitly with the `-automatch` command line switch).

Here is another example:

```
Layout Cells                    Source Cells
------------                    ------------
lay3                            src3
                                src2
lay4                            src1
lay3                            src3
```

Here is what happened:

Layout cell lay3 corresponds to source cell src3.
Source cell src3 contains an instance of src2
    src2 contains an instance of src1
    src1 corresponds to layout cell lay4
Layout cell lay4 contains an instance of lay3 (which closes the cycle).

Note that in this example, it is not immediately apparent from the transcript how the cycle is formed. For example, with this transcript, it is possible to go through src1 --> src3 --> lay3 instead of src1 --> lay4 --> lay3. In cases like this you will need to examine the layout and source netlist and/or the hcell list to figure out how the cycle is formed.

Similar information appears in the transcript of the Calibre LVS-H circuit comparison module.

LVS Discrepancy Types

The LVS discrepancy types are described below. For each discrepancy type, the report format and a graphic representation are specified.

- **Short Circuit**: This indicates a short-circuit in the layout. A short-circuit is detected when two source nets are connected together in the layout.

  The report format shows the layout net on the left, and two corresponding source nets on the right. For example:

```
-------------------------------------------
5    Net VDD                 //VDD
                             /N$87
-------------------------------------------
```

  For the above example, source nets //VDD and /N$87 are connected together in the layout to form the single layout net VDD.

- **Open Circuit**: This indicates an open-circuit in the layout. An open circuit is detected when two layout nets should be connected to correspond to a single net in the source.

  The report format shows two layout nets on the left, and the corresponding source net on the right. For example:

```
-------------------------------------------
 5    Net N4                          /SIG1
          N87
-------------------------------------------
```

  For the above example, layout nets N4 and N87 should be connected together to correspond to the single source net /SIG1.

- **Missing Connection**: This indicates a missing connection in a layout or source net. The connection may be to an instance pin or a pin of a logic gate, which was generated internally by LVS. A missing connection occurs when a net in circuit A is connected to more pins of a certain type than the corresponding net in circuit B, and all the connections of this type in the second net have been matched

  The report format shows the layout net and the corresponding source net; followed by a list of the missing connections. Connections are represented by the respective instance pins. The “Detailed Instance Connections” section also lists matched instances whose pins are listed as missing. For example:

```
-------------------------------------------------------
5   Net N716                          /N$781
    ------------------                ------------------
    ** missing connection **          /I$274/XGATE/I$702:S
    ** missing connection **          /I$274/XGATE/I$703:S
-------------------------------------------------------
```

  Layout net N716 was matched to source net /N$781 but the connections to instance pins /I$274/XGATE/I$702.S and /I$274/XGATE/I$703.S in the source net are missing in the layout net.

```
-------------------------------------------------------
5   Net N720                      /N$790
    -------------------           -------------------
    (NAND):INPUT                  ** missing connection **
      MP1:G
      MN1:G
-------------------------------------------------------
```

  Layout net N720 was matched to source net /N$790 but the connections to the NAND input formed by the gate pins of transistors /MP1 and MN1 is missing in the source net. The indicated transistors are a pair of MP and MN transistors which form one input of the NAND gate.

- **Unmatched Connection**: This indicates an unmatched connection in a layout or source net. The connection may be to an instance pin or a pin of a

logic gate generated internally by LVS. An unmatched connection occurs when some of the connections of a net in circuit A are different from the connections of the corresponding net in circuit B, and the LVS algorithm is not able to match these connections or recommend how the connections should be changed.

The report format shows the layout net and the corresponding source net, followed by a list of the unmatched connections. The connections are represented by the respective instance pins. For example:

```
-------------------------------------------------------
5 Net N716                    /N$781
  ------------------          ------------------
  ** unmatched connection **  /I$274/XGATE/I$702:S
  ** unmatched connection **  /I$274/XGATE/I$703:S
  I22:S                       ** unmatched connection **
-------------------------------------------------------
```

Layout net N716 was matched to source net /N$781, but the connection to instance pin I22:S in the layout net and the connections to instance pins /I$274/XGATE/I$702:S and /I$274/XGATE/I$703:S in the source net could not be matched.

- **Missing Net**: This indicates a missing net in the layout or source circuit. A missing net occurs when all nets in one of the circuits have been matched, and there are some unmatched nets left in the other circuit. The unmatched nets are reported as missing. When encountering this discrepancy, check the numbers of nets reported in the “Overall Comparison Results” section.

  The report format shows the missing net. For example:

```
-------------------------------------------------------
  5    ** missing net **               /N$716
-------------------------------------------------------
```

  Source net /N$716 is missing in the layout.

- **Missing Port**: This indicates a missing port in the layout or source circuit. A missing port occurs when all ports in one of the circuits have been matched and there are some unmatched ports left in the other circuit. The unmatched ports are reported as missing.

  The report format shows the missing port. For example:

```
-------------------------------------------------------
 5    ** missing port **             IN2 on net: /IN2
-------------------------------------------------------
```

  Source port IN2 on net /IN2 is missing in the layout.

- **Missing Instance**: This indicates a missing instance in the layout or source circuit. A missing instance occurs when all instances of a particular component type in one of the circuits have been matched and there are some unmatched instances of the same component type left in the other circuit. The unmatched instances are reported as missing. When encountering this discrepancy, check the numbers of instances reported in the “Overall Comparison Results” section.

  The report format shows the missing instance. For example:

```
-------------------------------------------------------
  5    I75  C(A)                 ** missing instance **
-------------------------------------------------------
```

  Layout instance I75 is missing in the source circuit. C is the component type, and A is the component subtype.

- **Missing Gate**: This indicates a missing logic gate in the layout or source circuit. This is similar to the Missing Instance discrepancy, except that it is reported for gates which are generated internally by the logic gate recognition feature of LVS. A missing gate occurs when all instances of a particular logic gate type in one of the circuits have been matched and there are some unmatched instances of the same gate type left in the other circuit. The unmatched gates are reported as missing. When encountering this

discrepancy, check the “Numbers of Elements After Transformation section.

The report format shows the gate type in parentheses followed by a list of transistors forming the gate. For example:

```
-------------------------------------------------------
  5    ** missing gate **      (INV)

                               Transistors:
                                 /I$767/MP/MP/I$702  MP
                                 /I$767/MN/MN/I$786  MN
-------------------------------------------------------
```

The inverter formed by source transistors /I$767/MP/MP/I$702 and /I$767/MN/MN/I$786 is missing in the layout. MP and MN are the component types.

- **No Similar Net**: This indicates a net in the layout or source circuit for which there is no corresponding net with similar connections in the other circuit. This discrepancy is reported only when the LVS algorithm is not able to match the net using its internal error correction mechanisms, and the net can not be classified as missing.

  The report format shows the net. For example:

```
-------------------------------------------------------
  5    ** no similar net **              /N$716
-------------------------------------------------------
```

  Source net /N$716 has no similar net in the layout.

- **No Similar Port**: This indicates a port in the layout or source circuit for which there is no similar corresponding port in the other circuit. The port is usually connected to a net for which a No Similar Net discrepancy was reported. Note that this discrepancy is reported only when the LVS algorithm is not able to match the port or the corresponding net using its

internal error correction mechanisms, and the port can not be classified as missing.

The report format shows the port and its net. For example:

```
-------------------------------------------------------
  5    ** no similar port **          IN2  on net: /IN2
-------------------------------------------------------
```

Source port IN2 on net /IN2 has no similar port in the layout.

- **Bad Component Type**: This indicates that an instance of the wrong cell or device was placed in the layout. It is reported when a layout instance is matched to a source instance with a different component type. The layout instance should be replaced by an instance of a layout component which corresponds to the indicated source component.

  This discrepancy can be reported for logic gates that are generated internally by the logic gate recognition feature of LVS. In this case, a logic gate of the wrong type was implemented in the layout. The layout gate structure should be replaced by a gate of the type indicated for the source.

  LVS matches instances of different component types when they have similar connections in the source and layout circuits.

  The report format shows, for regular instances, the layout instance and the source instance followed by their corresponding component type names. For logic gates generated internally by LVS, the layout and source gate types are indicated in parentheses followed by a list of the transistors forming each gate. For example:

```
-------------------------------------------------------
  5 I75  MP                          /I$34  MN
     bad component type: MP           component type: MN
-------------------------------------------------------
```

Layout instance I75 is an instance of a MP transistor. It corresponds to source instance /I$34 which is an MN transistor. The type of /I$75 in the layout should be changed to MN.

```
-------------------------------------------------------
  6    (NAND)                          (NOR)
        bad component type: NAND        component type: NOR

       Transistors:
         MP27  MP
         MP28  MP
         MN13  MN
         MN14  MN
                                                 /MP6   MP
                                                 /MP7   MP
                                                 /MN10  MN
                                                 /MN12  MN
-------------------------------------------------------
```

The NAND gate formed by layout transistors MP27, MP28, MN13, and MN14 was matched to the NOR gate formed by source transistors /MP6, /MP7, /MN10, and /MN12. The layout NAND structure should be replaced by a NOR. The layout and source transistors are listed on separate lines; this indicates that the transistors were not matched to each other. Component types are indicated next to each transistor.

- **Bad Component Subtype**: This indicates that an instance of the wrong cell or device subtype was placed in the layout. It is reported when a layout instance is matched to a source instance with identical component type but a different component subtype. The layout and source type and subtype names are indicated.

The report format shows the layout and source instances followed by their respective component type and subtype names. The subtype names are indicated in parentheses. For example:

```
-----------------------------------------------------
5  I75  R(X)                    /I$34  R(Y)
   bad component subtype: R(X)  component subtype: R(Y)
-----------------------------------------------------
```

Layout instance I75 is a resistor R with subtype X. It corresponds to source instance /I$34 which is a resistor R with subtype Y.

- **Badly Connected Instance**: This indicates a badly connected layout instance. A Badly Connected Instance is generated only for instances that are not listed elsewhere as part of net discrepancies.

  The report format shows the layout instance and its corresponding source instance on the left and right side of the report, respectively. Next, the correct connections of both instances are listed in the form:

```
layout_pin_name:layout_net_name src_pin_name:src_net_name
```

Next, the bad layout connections are listed in one of four forms:

```
layout_pin_name:layout_net_name        ** src_net_name **
layout_pin_name:layout_net_name        ** missing net **
layout_pin_name:layout_net_name        ** no similar net **
layout_pin_name:layout_net_name        ** unmatched net **
```

In all forms, the layout pin name and the layout net name are indicated on the left. If the layout net is matched, then the corresponding source net name is indicated on the right. Otherwise, the text on the right indicates whether the layout net was classified as a Missing Net discrepancy, a No Similar Net discrepancy, or an unmatched net.

Next, the bad source connections are listed in one of four forms:

```
** layout_net_name **              src_pin_name:src_net_name
** missing net **                  src_pin_name:src_net_name
** no similar net **               src_pin_name:src_net_name
** unmatched net **                src_pin_name:src_net_name
```

In all forms, the source pin name and the source net name are indicated on the right. If the source net is matched, then the corresponding layout net name is indicated on the left. Otherwise, the text on the left indicates whether the source net was classified as a Missing Net discrepancy, a No Similar Net discrepancy, or an unmatched net. For example:

```
-----------------------------------------------------
   I23  BUFF                          /I$34  BUFF
     input: SIGA                        input: /SIGA
     output: NET1                       ** missing net **
     ** SIGB **                        output: /SIGB
-----------------------------------------------------
```

Layout instance I23 corresponds to source instance /I$34 but is badly connected. The input pin in the layout is correctly connected to layout net SIGA, and the input pin in the source is connected to the corresponding source net, /SIGA. The output pin in the layout is connected to layout net NET1, which is missing in the source, while the output pin in the source is connected to source net /SIGB which corresponds to layout net SIGB. BUFF is the component type.

- **Bad Power Supply**: This indicates a logic gate whose power or ground supply in the layout is different from the one in the source. The discrepancy is only reported in logic gates formed by LVS.

  The report format shows the layout gate type and the corresponding source gate type on the left and right side, respectively. Next, the bad power and/or ground connections are listed in the layout and source. Power supplies are indicated with the words “power supply” and ground supplies are indicated with the words “ground supply”. Layout supplies are listed in one of the following forms:

```
power supply: layout_net_name   ** source_net_name **
power supply: layout_net_name   ** missing net **
power supply: layout_net_name   ** no similar net **
power supply: layout_net_name   ** unmatched net **
```

In all forms, the layout net name is indicated on the left. If the layout net is matched, then the corresponding source net name is indicated on the right. Otherwise, the text on the right indicates whether the layout net was classified as a Missing Net discrepancy, a No Similar Net discrepancy, or an unmatched net.

Source supplies are listed in one of the following forms:

```
** layout_net_name **       power supply: source_net_name
** missing net **           power supply: source_net_name
** no similar net **        power supply: source_net_name
** unmatched net **         power supply: source_net_name
```

In all forms, the source net name is indicated on the right. If the source net is matched, then the corresponding layout net name is indicated on the left. Otherwise, the text on the left indicates whether the source net was classified as a Missing Net discrepancy, a No Similar Net discrepancy, or an unmatched net.

Next, the transistors forming the gate in the layout and source are listed. For example:

```
-------------------------------------------------------
 5   (NAND)                          (NAND)
       ground supply: GND1             ** GND1 **
       ** GND2 **                      ground supply: GND2

     Transistors:
       mp20                            2/mp0
       mp21                            2/mp1
       mp22                            2/mp2
       mp23                            2/mp3
-------------------------------------------------------
```

The ground supply to this NAND gate in the layout is GND1 which is matched to net GND1 in the source. The power supply to the corresponding NAND gate in the source is GND2 which is matched to net GND2 in the layout.

- **Instance With Non-Floating Extra Pins**: This indicates an instance of a cell with extra pins in the layout or source. LVS looks for instances where the extra pins are actually connected to other elements (not floating) and reports those instances as discrepancies. Instances where the extra pins are floating are not reported. All discrepancies of this type appear together in a section called Instances of Cells With Non-Floating Extra Pins.

  This type of in-context reporting of extra pins is performed for corresponding cells (hcells) as well as primitive cells like LVS Box cells. It is especially useful in cases where higher level nets in the layout are inadvertently shorted to internal nets in subcells (which results in extra layout pins).

  The report format shows the layout instance on the left followed by its component type in parentheses. The corresponding source instance is shown on the right. This is followed by list of extra pins in the form:

  ```
  pin_name:net_name
  ```

  where pin_name indicates the extra pin, and net_name is the name of a net to which that pin is connected in the containing cell. The string “** missing pin **” appears in the other side to indicate that the pin is missing there. For example:

```
*******************************************************
     INSTANCES OF CELLS WITH NON-FLOATING EXTRA PINS

DISC#  LAYOUT NAME              ne  SOURCE NAME
*******************************************************

  1     x2  mux                     x2  mux
         C:net1                     ** missing pin **
-------------------------------------------------------
```

In this example, cell mux has an extra pin C in the layout. LVS shows instance x2 of cell mux at a higher level of hierarchy. In that instance, the extra pin C is connected to net net1 in the layout (and this net is also connected to other elements).

- **Property Error**: This indicates an instance with different property values in the layout and source. Property checking is driven by trace property rules. Discrepancies of this type are listed together in the “Property Errors” section. The trace property rules are listed in the “LVS Parameters” section.

  The report format shows the layout instance on the left followed by its component type in parentheses. The corresponding source instance is shown on the right. This is followed by the names and values of the properties that are different in both circuits, one property per line, and, for numeric properties, the corresponding error percentages. The values of a particular property are listed only if they are different, and, for numeric properties, the difference exceeds the specified tolerance.

  Devices that are the result of device reduction (such as series or parallel reduction) may own properties with the *unknown* value. *Unknown* values are assigned under certain conditions when an effective property value can not be computed for the device; see “Missing and Unknown Property Values” on page 10-41 for more information. A *unknown* property value is indicated with a question mark and the words "reduced instance". For example,

  ```
  p: ? (reduced instance)
  ```

  indicates that the value of property p is unknown.This could be because p is not one of the standard properties for that device type, or because there was

not sufficient data to compute an effective value for p. For example, in the latter case:

```
*****************************************************
                    PROPERTY ERRORS

DISC#  LAYOUT                    SOURCE         ERROR
*****************************************************
  1    27(230,540)  MD           /I$867  MD
       w: 5.2u                    w = 5u          4%
       l: 2.2u                    l = 2u         10%

  2    35(120,70)  ME            /I$302  ME
       L = 12.2u                  L = 12u         2%
*****************************************************
```

Two discrepancies are listed. Discrepancy number 1 involves layout instance 27 at location (X=230, Y=540), with component type MD, and source instance /I$867. The layout width value is 5.2 microns, the source width value is 5 microns, and the error is 4 percent. The layout length value is 2.2 microns, the source length value is 2 microns, and the error is 10 percent. Discrepancy number 2 involves layout instance 35 at location (X=230, Y=540), with component type ME, and source instance /I$302. The layout length value is 12.2 micron, the source length value is 12 microns, and the error is 2 percent. The width value for this instance is not listed because it is not involved in an error.

- **Split Gate Property Ratio Error**: This indicates a split gate ratio property error. For more information refer to the LVS Split Gate Ratio specification statement. Discrepancies of this type appear in the LVS report in the “Layout Errors” and “Source Errors” sections for layout devices and source devices respectively. The LVS Split Gate Ratio rules are listed in the “LVS Parameters” section of the report.

  The report format shows individual devices in each “row” of the split gate along with respective property values, the computed property ratio, and

error percentage for the row. The base row and all error rows are indicated; correct rows are not listed.

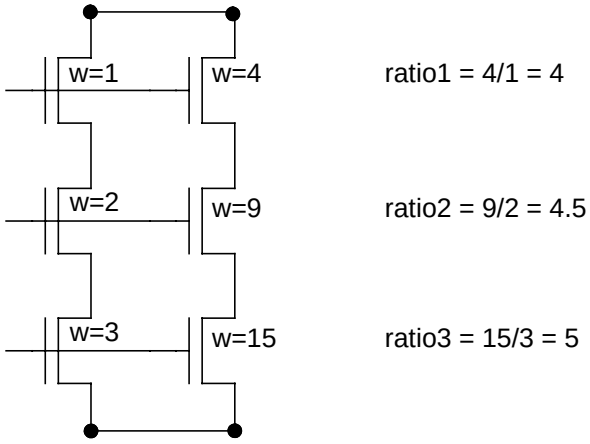


Figure 14-2. Split Gate Property Ratio Error

```
************************************************************
Property Ratio Errors in Split Gates:

1  property  w
   base: m1 MP(P): 1u, m4 MP(P):  4u. ratio: 4
         m2 MP(P): 2u, m5 MP(P):  9u. ratio: 4.5 error: 12.5%
         m3 MP(P): 3u, m6 MP(P): 15u. ratio: 5   error: 25%
************************************************************
```

In the above example LVS checks the property w. The base row consists of transistor m1 with w=1u and transistor m4 with w=4u. The property ratio in the base row is 4/1 = 4. The second row consists of transistors m2 with w=2u and m5 with w=9u. The property ratio in this row is 9/2 = 4.5 and the error percentage is (4.5 - 4)/4 = 12.5%. The third row consists of transistors m3 with w=3u and m6 with w=15u. The property ratio in this row is (5 - 4)/4 = 25%. All transistors have component type MP and subtype P as indicated.

- **Properties Missing on Instances**. This type of discrepancy indicates that a property is missing on an instance in the layout or source. The property is required because it is referenced by Trace Property statements or device reduction Tolerance statements or similar statements, or participates in

effective property calculation for other properties that appear in such statements, or is needed for other reasons. Discrepancies of this type appear in the LVS report in the “Layout Errors” section for layout devices and in the “Source Errors” section for source devices.

Here is an example:

```
*************************************************************
Properties Missing on Instances:
  5    property r              not found on    2/r1 (R)
  6    property c              not found on    2/c2 (C)
*************************************************************
```

Two discrepancies are shown. Discrepancy 5 indicates that property “r” is missing on instance 2/r1 which is of type R. Discrepancy 6 indicates that property “c” is missing on instance 2/c2 which is of type C.

- **Incorrect Substrate Connection**: This indicates an instance with an incorrect substrate connection. This discrepancy, its format and graphic representation are all similar to the Badly Connected Instance discrepancy. Incorrect substrate connections appear in a separate section of the report when the LVS Soft Substrate Pins specification statement is indicated in the rule file. For example:

```
*****************************************************
              INCORRECT SUBSTRATE CONNECTIONS
DISC#  LAYOUT NAME                       ne  SOURCE NAME
*****************************************************
  5    1/m1 MP(P)                           1/m1  MP(P)
         b: vssd                              ** vssd **
         ** no similar net **                b: vssa
```

Information and Warnings

The “Information and Warnings” section of the LVS report provides warnings about conditions that are deemed out of the ordinary but are not considered errors, and additional information that can be useful in verifying a design.

Table 14-6. Information and Warnings

Numbers of Matched And Unmatched Elements	Provides the numbers of matched and unmatched ports and nets, and instances of each component type and subtype in the layout and source. LVS reports Instance counts by component type and subtype. When an instance with a subtype is matched to an instance with no subtype, the one with a specified subtype determines the subtype of both. In case of conflict, the source instance determines the subtype.
Statistics	Provides various statistical information about the LVS run.
Component Types With Non-Identical Power Or Ground Pins	Provides a list of component types that have different power or ground pins in the layout and source. LVS lists the component types and pin names, and indicates each pin as missing in either the layout or source component type. The format is similar to the one used for “Component Types with Non-Identical Signal Pins” on page 14-54. While LVS treats differences in signal pins as errors, differences in power or ground pins are treated as warnings.

Table 14-6. Information and Warnings [continued]

Bad Devices	Provides a list of badly formed layout devices. These can be present only in mask LVS, and are formed when a shape on a device recognition layer in the rule file is not recognized as a valid device because the combination of pin shapes that it touches does not match any of the Device statements for this layer. For each bad device, LVS reports the X and Y layout coordinates of the device shape, plus a list of possible element names for this device. The list contains element_name values from all Device statements in the rule file that use this device recognition layer.
Matched Mosfets Which Have Been Unequally Reduced	Provides a list of MOS transistor groups (component types MN, MP, ME, MD, LDDN, LDDP, LDDE, LDDD), which are connected in parallel in the source, but correspond to single transistors in the layout or groups of parallel transistors that consist of different numbers of elements. See the section “Unequally Reduced Devices” on page 10-40 for more details. In each group, the layout transistors are listed on the left and the source transistors are listed on the right. LVS indicates missing transistors with the string “** missing smashed mosfet **” in the column were they are missing.
Isolated Layout Nets	Provides a list of layout nets which are not connected to any instances, nor connected to any ports of the top-level cell. LVS ignores these nets during comparison, unless they have user-given names that are also present in the source.
Passthrough Layout Nets And Their Ports	Provides a list of layout nets that are connected only to ports of the top-level cell; the ports are also reported. LVS ignores these nets during comparison, unless they or their ports have user-given names that are also present in the source.

Table 14-6. Information and Warnings [continued]

Layout Names That Are Missing In The Source	Provides a list of user-given net, instance, and port names in the layout, which are not present in the source.
Layout Names That Appear On More Than One Element	Provides a list of user-given names that appear on more than one layout net, more than one layout instance, or more than one layout port. LVS does not use these names as initial correspondence points.
Source Names That Appear On More Than One Element	Provides a list of user-given names that appear on more than one source net, more than one source instance, or more than one source port. LVS does not use these names as initial correspondence points.
Conflicting Layout Names	Provides a list of name conflicts, in the layout, caused by the representation of several circuit elements by a single virtual element. For example, all ports found on a single net are represented by a single virtual port, or a group of parallel transistors can be represented by a single virtual transistor. The new virtual element inherits all user-given names from the original elements. Each name listed appears with another name on a single virtual element in the layout, but the two names appear on different elements in the source. LVS does not use these names as initial correspondence points.

Table 14-6. Information and Warnings [continued]

Conflicting Source Names	Provides a list of name conflicts, in the source, caused by the representation of several circuit elements by a single virtual element. For example, all ports found on a single net are represented by a single virtual port, or a group of parallel transistors can be represented by a single virtual transistor. The new virtual element inherits all user-given names from the original elements. Each name listed appears with another name on a single virtual element in the source, but the two names appear on different elements in the layout. LVS does not use these names as initial correspondence points.
Initial Correspondence Points	Provides a list containing pairs of identically named nets, ports, and instances used as initial correspondence points. See the section "Initial Correspondence Points" in chapter 10, for more information.
Cpoints	Pairs of nets that were matched by LVS Cpoint specification statements are reported in this section. Only Cpoints that are actually used by LVS appear in this section. Cpoints that could not be used appear in the Failed Cpoints section. Note that the format used for reporting Cpoint net names in the LVS report is the same as for nets in general. This may be different from the way Cpoints are entered in the rule file. Specifically, flat LVS omits the X subcircuit call designators in hierarchical pathnames in Spice.
Failed Cpoints	Cpoints that could not be used by LVS are reported in this section. Note that the format used for reporting Cpoint net names in the LVS report is the same as for nets in general. This may be different from the way Cpoints are entered in the rule file. Specifically, flat LVS omits the X subcircuit call designators in hierarchical pathnames in Spice.

Table 14-6. Information and Warnings [continued]

Ambiguity Resolution Points	Provides a list containing pairs of nets, instances, and ports, which belong to interchangeable parts of the circuit, and are matched arbitrarily by LVS. For each pair of arbitrarily matched elements, the layout element is reported on the left and the source element is reported on the right. See the section "Resolving Ambiguities" in chapter 10, for more information.
Layout/Source FY, GY, M, and N Filtered Devices That Did Not Connect S And D Pins	These two sections (one for layout and one for source) report transistors for which unused device filter options FY, GY, M, or N could not connect together the source and drain nets because the source and drain were connected to different pads. For each transistor, the report shows the instance name, the respective filter option, and the source and drain nets.
Layout/Source Instances With Undetermined Reduction TOLERANCE Properties	These two sections (one for layout and one for source) report instances that caused device reduction to cease locally because a device reduction TOLERANCE clause was specified but the value for the respective property was unknown on the instance. The instance in question was reduced from other instances, and the effective property value could not be computed. For each instance, LVS reports the instance name, component type, and respective property name.

Detailed Instance Connections

The "Detailed Instance Connections" section of the LVS report provides information about matched instances whose pins are listed as part of discrepancies on nets. For each pair of instances, the information includes: the layout instance, corresponding source instance, nets connected to their pins in the layout and source, respectively, and the corresponding nets in the source and layout, respectively.

The report format is identical to the Badly Connected Instance discrepancy report, except there is no discrepancy number.

Unmatched Elements

Unmatched elements are nets, ports, instances, and internally generated logic gates in the layout or source that cannot be matched to corresponding elements in the other circuit, and can not be classified as any of the available discrepancy types. This can happen when there are elements of similar type in the other circuit that have similar connections, but the LVS algorithm can not make a decision because of a discrepancy nearby. The “Unmatched Elements” section of the LVS report lists unmatched elements. In most cases it is best to correct all discrepancies first, ignoring this section list, and run LVS again. The unmatched elements will usually disappear from the report once all discrepancies are corrected.

- **Unmatched Net**: An unmatched net is a net in the layout or source that cannot be matched to a corresponding net in the other circuit, and cannot be classified as any of the available discrepancy types. In the following example, source net /N$716 cannot be matched.

```
-------------------------------------------------------
** unmatched net **              /N$716
-------------------------------------------------------
```

- **Unmatched Port**: An unmatched port is a port in the layout or source that cannot be matched to a corresponding port in the other circuit, and cannot be classified as any of the available discrepancy types. The port is usually connected to an unmatched net. In the following example, source port IN2 cannot be matched.

```
-------------------------------------------------------
** unmatched port **             IN2
-------------------------------------------------------
```

- **Unmatched Instance**: An unmatched instance is an instance in the layout or source that cannot be matched to a corresponding instance in the other

circuit and can not be classified as any of the available discrepancy types. In the following example, layout instance I75 could not be matched.

```
-------------------------------------------------------
I75                            ** unmatched instance **
-------------------------------------------------------
```

- **Unmatched Gate**: An unmatched gate is a logic gate in the layout or source that cannot be matched to a corresponding gate in the other circuit and cannot be classified as any of the available discrepancy types. This error is reported for gates that are generated internally by the logic gate recognition feature of LVS. The gate type is indicated in parentheses, followed by a list of transistors that form the gate. In the following example, The inverter formed by source transistors /I$767/MP/MP/I$702 and /I$767/MN/MN/I$786 cannot be matched.

```
-------------------------------------------------------
** unmatched gate **               (INV)
                                      /I$767/MP/MP/I$702
                                      /I$767/MN/MN/I$786
-------------------------------------------------------
```

Circuit Extraction Report

The circuit extraction report is written during the circuit extraction phase of Calibre LVS-H. This report contains a summary of circuit extraction warnings and errors that until now appeared only in the Calibre LVS-H transcript or in the extracted layout netlist. Items included in the report are:

- Connectivity extraction errors and warnings, such as short circuits, open circuits, unattached labels.
- Sconnect conflicts.
- Sconnect and LINK usage errors, such as the case when no data is found for a net name.

- Stamp discrepancies.
- Bad devices.
- Top level port name conflicts from the hierarchical Spice netlister.

The report is written to a file named *lvs_report*.ext where *lvs_report* is the name specified in the LVS Report specification statement. If no LVS Report specification statement exists then the report is written to the file lvs.rep.ext in the current working directory.

Mask Results Database

The LVS mask results database is an optional database type. It is the extracted nets and devices resulting from the execution of connectivity related operations and statements contained in a rule file. You use this type when interpreting the results graphically. You can exclude Information from this database by using the NOPROBE and NOCONTACT options, and the -dblayers command line option. You must use a Mask Results Database specification statement when running Calibre MGC.

Cross-Reference Files

The following applications can generate instance and net cross-reference files:

- Calibre LVS
- Calibre MGC
- xCalibre PX-C/PX-RC
- xCalibre MGC

Instance Cross-reference File

The -ixf command line switch instructs the application to generate the file. The reference pages for the applications describe the use of the switch.

The IXF secondary keyword to the Mask SVDB Directory specification statement also creates an instance cross-reference file within the directory specified in the Mask SVDB Directory statement.

Whenever Calibre creates an instance cross-reference file, it is named *layout_primary*.ixf, where *layout_primary* is taken from the Layout Primary specification statement.

The instance cross-reference file contains matched instances, and is in ASCII format. The file contains one line per instance in the following form:

```
layout_id  layout_name  source_id  source_name [ SL | SS ] [X]
```

where:

- ***layout_id*** is a number that represents the instance (ID) in the layout.
- ***layout_name*** is a user-given name that represents the layout instance. The value of *layout_id* is used if no user-given name was specified.
- ***source_id*** is a number that represents the instance (ID) in the source.
- ***source_name*** is a user-given name that represents the source instance. The value of *source_id* is used if no user-given name was specified.
- **SL** indicates a reduced layout device. SL is short for “smashed layout”.
- **SS** indicates a reduced source device. SS is short for “smashed source”.
- **X** indicates a MOS device with swapped source and drain pins.

Mask-mode Instance Coordinates

In Mask-mode extracted devices, the (x,y) location is included with the name in the form *name*(*x*,*y*). For example:

```
1 1(-12.000,-1.000) 4 R1
```

Matched Devices

The instance cross-reference file represents a reduced device by listing its original devices. When a reduced layout device is matched to a reduced source device, all original layout devices are listed in consecutive lines on the left, with the original source device repeated on the right. All the remaining original source devices are listed in consecutive lines on the right, with a representative original layout device repeated on the left. The representative devices are chosen at random.

For example:

```
0 0(-12.000,-1.000) 4 R1
1 1(-18.000,-1.000) 4 R1 SL
0 0(-12.000,-1.000) 5 R2 SS
0 0(-12.000,-1.000) 6 R3 SS
```

In this example, layout devices 0 and 1 are reduced to a single device. The reduced layout device corresponds to source devices R1, R2, and R3, which are also reduced to a single device. Layout device 0 and source device R1 are chosen as representative devices.

Swapped Pins

MOS devices with swapped source and drain pins are indicated with the letter X. For example:

```
2 2(-12.000,-1.000) 5 M1 X
```

X indicates that the source pin of the layout device corresponds to the drain pin of the source device and vice versa. Lines that represent reduced devices (SL or SS) have correct X values as well, with respect to the two devices reported on that particular line.

Logic Gates

The instance cross-reference file represents LVS logic gates by the original transistors that form them, and lists all matched transistors in the layout gate with the corresponding transistors in the source gate.

When you specify the BY GATE secondary keyword to the Mask SVDB Directory specification statement in your rule file, LVS labels transistor pairs that belong to logic gates. The labels are “G” and “GC” and refer to transistors that are the beginning of a gate and a continuation of a gate, respectively. The following example shows a possible output, comments are not part of the output:

```
364 M048 364 M048         // This transistor is not in a gate.
256 M030 256 M030 G       // Beginning of gate.
49 M005 256 M030 SL GC    // Other transistors in the gate
256 M030 49 M005 SS GC    // .
265 M031 265 M031 GC      // .
40 M004 265 M031 SL GC    // .
265 M031 40 M004 SS GC    // .
22 M002 22 M002 G         // Beginning of another gate.
13 M001 13 M001 GC        // Other transistors in the gate.
31 M003 31 M003 GC        // .
4 M000 4 M000 GC          // .
91 M81 91 M91             // This transistor is not in a gate.
```

Net Cross-reference File

The -nxf command line switch instructs the application to generate the file. The reference pages for the applications describe the use of the switch.

The NXF secondary keyword to the Mask SVDB Directory specification statement also creates an net cross-reference file within the directory specified in the Mask SVDB Directory statement.

Whenever Calibre creates a net cross-reference file, it is named *layout_primary*.nxf, where *layout_primary* is taken from the Layout Primary specification statement.

The net cross-reference file contains matched nets, and is in ASCII format. The file contains one line per net in the following form:

```
layout_id  layout_name  source_id  source_name
```

where:

- ***layout_id*** is a number that represents the net (ID) in the layout.

- ***layout_name*** is a user-given name that represents the layout net. The value of *layout_id* is used if no user-given name was specified.
- ***source_id*** is a number that represents the net (ID) in the source.
- ***source_name*** is a user-given name that represents the source net. The value of *source_id* is used if no user-given name was specified.

Mask-mode Net Coordinates

In Mask-mode extracted nets, the (x,y) location is included in the name in the form name-or-id(x,y), with no blanks. For example:

```
1 blip(-12.000,-1.000) 4 VCC
2 2(-12.000,-1.000) 11 up
```

Matched Nets

The net cross-reference file repeats a net when two nets in one database are matched to a single net in the other database. For example:

```
3 aaa(-20.000,-1.000) 5 xyz
4 bbb(-30.000,-1.000) 5 xyx
```

In certain situations, LVS can match several layout nets to one source net, or several source nets to one layout net, or a group of several layout nets (together) to a group of several source nets. This can occur in split gate reduction or when LVS detects an open circuit or short circuit discrepancy. In these cases, a representative net is chosen for the layout side and a representative net is chosen for the source side. The representative pair appears in the net cross reference file. In addition, each of the remaining layout nets appears with the source representative, and each of the remaining source nets appears with the layout representative.

In the following example, layout nets 1, 2 and 3 were matched (as a group) to source nets n1, n2 and n3.

```
1 1(10.000,-1.000) 51 n1
1 1(10.000,-1.000) 52 n2
1 1(10.000,-1.000) 53 n3
2 2(20.000,-1.000) 51 n1
```

```
3 3(30.000,-1.000) 51 n1
```

LVS can be successful, and leave nets unmatched. Examples of nets that are never matched include:

- Nets internal to certain types of logic gates formed by LVS.
- Nets removed because of series device reduction.

Unmatched nets do not appear in the net cross-reference file.

Hierarchical Instance and Net Cross-reference Files

The comparison stage of Calibre LVS-H creates hierarchical instance and net cross-reference files when the rule file specifies the Mask SVDB Directory specification statement. The files establish layout-to-source correspondence in Calibre RVE/QDB-H and xCalibre applications.

The hierarchical cross-reference file formats are similar to the flat formats, except that the information is provided per cell. Each file begins with a SVDB header, see section SVDB Header. After the SVDB header, the file contains one section for each LVS correspondence cells (or hcells). The following example shows the general structure of hierarchical instance and net cross-reference files:

```
# SVDB: header_line
# SVDB: header_line
…
# SVDB: header_line
% layout_cell layout_pin_count source_cell source_pin_count
layout_id  layout_name  source_id  source_name
layout_id  layout_name  source_id  source_name
…
% layout_cell layout_pin_count source_cell source_pin_count
layout_id  layout_name  source_id  source_name
layout_id  layout_name  source_id  source_name
…
…
…
```

Source and Layout Placement Hierarchy Files

The comparison stage of Calibre LVS-H creates source placement hierarchy (sph) and layout placement hierarchy (lph) files when the rule file specifies the Mask SVDB Directory specification statement. The files establish layout-to-source correspondence in Calibre RVE/QDB-H and xCalibre applications, and are in ASCII format.


Note

Each section represents cells, not devices. Pin count is the only connectivity information present.

Each file begins with a SVDB header, see section "SVDB Header". After the SVDB header, the file contains one section for each cell in the hierarchy, not just hcells. The following example shows the general structure of placement hierarchy files:

```
# SVDB: header_line
# SVDB: header_line
…
# SVDB: header_line
% cell_name pin_count
placement_name cell_or_device_name number_of_pins
placement_name cell_or_device_name number_of_pins
…
% cell_name pin_count
placement_name cell_or_device_name number_of_pins
placement_name cell_or_device_name number_of_pins
…
…
…
```

SVDB Header

The instance and net cross-reference files, and the source and layout placement hierarchy files created in the SVDB directory begin with SVDB header lines. This header identifies the type of file and the source of the information used to create

that file. Each line begins with the string "# SVDB: ". The following example shows the SVDB header from a instance cross-reference file:

```
# SVDB: Instance Cross Reference (ixf) (File format 1)
# SVDB: Layout Primary mix
# SVDB: Rules -0 play.rules Wed Dec 10 10:07:38 1997
# SVDB: GDSII -0 (none) (none)
# SVDB: SNL -0 (none) (none)
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
```

The first line identifies the type of file and its format version, and is the only line that differs between files that represent the same design. The following shows first line from each cross-reference and placement hierarchy file:

```
SVDB: Layout Placement Hierarchy (lph) (File format 1)
SVDB: Source Placement Hierarchy (sph) (File format 1)
SVDB: Instance Cross Reference (ixf) (File format 1)
SVDB: Net Cross Reference (nxf) (File format 1)
```

The second line gives the name of the top (primary) cell of the design. The third, fourth, and fifth lines identify the rule file, layout GDSII file, and source net list file. The format for each line is:

```
file_type path_name date_time_stamp
```

where:

- ***file_type*** can be: Rules, GDSII, or SNL, followed by a checksum for the file. The string "-0" specifies that no checksum is present.
- ***path_name*** is the pathname of the *file_type*. The string "(none)" specifies that a path name is not present.
- ***date_time_stamp*** is the time stamp of the file. The string "(none)" specifies that a time stamp is not present. The date_time_stamp takes the form:

  ```
  week_day month day hh:mm:ss year
  ```

Circuit Extraction Report File

The circuit extraction report file is written out during Calibre LVS-H circuit extraction (calibre -spice). This report contains a summary of circuit extraction warnings and errors that also appear in the Calibre LVS-H transcript or in the extracted layout netlist. Items included in the report are:

- Connectivity extraction errors and warnings (short circuit, open circuit, unattached label, and so on).
- Sconnect conflicts.
- Sconnect Link usage errors (for example, no data found for net name).
- Stamp discrepancies.
- Bad devices.
- Top level port name conflicts from the hierarchical Spice netlister.

The report is written to a file named *lvs_report*.ext where *lvs_report* is the name specified in the LVS Report specification statement. If no LVS Report specification statement exists then the report is written to the file lvs.rep.ext in the current working directory.

Binary Polygon File (BPF) Database

The BPF database is used to interface flat Calibre LVS to external tools. It provides access to geometries on layers involved in connectivity extraction and device recognition, and to some other related information. The BPF database is created with the `-bpf` command line option in flat Calibre LVS. The BPF database cannot be created in hierarchical Calibre LVS. The database consists of a set of files, described below.

File names in the BPF database are based on the LVS report name, as specified in the rule file with the LVS Report specification statement. If no LVS Report statement is specified in the rule file then the name “icv” is used for the report prefix.

BPF Binary Polygon Files. BPF files contain polygons from certain layers of interest. The BPF files created have names of the form *lvs_report*.*layer_name*.bpf. The *layer_name* field is the rule file layer name. By default, all Connect and Device seed layers are output. The Calibre LVS `-dblayers` command line option can be used to explicitly select layers for generation.

Polygons in BPF files are annotated with node numbers (for Connect layers, Device pin layers, Stamp layers, and so on) or with device numbers (for Device seed layers). In case of conflict, node numbers are stored and the device numbers are not preserved. For example, if a layer serves as both a pin layer and a device seed layer then the respective BPF file contains node numbers and the device numbers are lost. If you need the device numbers as well then you can copy the seed layer so that separate layers can be used. For example:

```
ngate1 = COPY ngate
DEVICE MN(NFET) ngate1 ngate(G) nact(S) nact(D) dpsub(B)
                <ndif> <poly> (S D)
```

BPF Layout Cross Reference File. The layout cross-reference file, *lvs_report*.lxf, is an ASCII text file that provides a cross reference between internal net numbers and layout texted names.

BPF Ports File. The *lvs_report*.ports file contains information about top level ports. This is an ASCII text file. The file contains one line for each top level port (unattached ports are not output). Each line has the following fields:

port_name node_number node_name port_location port_layer_attached

Where each field is defined as follows:

port_name—The layout name of the port object. For example, the GDSII text string when using Port Layer Text. Or `UNNAMED` if the port is not named.

node_number—The layout node number to which the port is connected.

node_name—The layout node name to which the port is connected; layout node number if the node is unnamed.

port_location—In the form: X Y; in database units. This is the location of the database text object when using Port Layer Text, or a vertex on the port polygon marker when using Port Layer Polygon.

port_layer_attached—Layer of the polygon to which the port got attached. Rule file layer name or rule file layer number if the layer is unnamed. This layer appears in a Connect or Sconnect operation.

Examples:

```
CONF3 5 CONF -98000 -90000 metal
CONF3 5 5 -98000 -90000 metal
<UNNAMED> 5 5 -98000 -90000 metal
CONF 5 CONF -98000 -90000 17
```

Chapter 15
RVE/QDB-H and Query Server

This chapter describes the input requirements, usage, procedures, and commands for Calibre RVE/QDB-H and the Query Server.

Calibre RVE/QDB-H consists of:

- Results viewing environment (RVE): This is the graphical user interface that allows you to debug Calibre LVS/LVS-H results and highlight Calibre DRC/DRC-H results.

- Hierarchical query database (QDB-H): This is the query database. It returns requested data about a design. RVE uses the Query Server to probe the query database.

The Query Server is licensed functionality that allows you to query results database information. RVE serves as a user interface for the Query Server; however, the Query Server may also be run from the command line or your own user interface. The Calibre Connectivity Interface is licensed functionality that uses the Query Server to probe for connectivity information that is intended for backannotation and other uses.

Results Viewing Environment

This section describes the Results Viewing Environment (RVE), a graphical user interface designed to help you browse and debug Calibre results. It includes a product overview, layout editor considerations, invocation, description of the GUIs, and usage procedures.

Refer to "Calibre RVE/QDB-H" on page 2-28 for invocation information.

Calibre RVE/QDB-H requires a calibreqdb or a caldrclvseve license. This allows you to run RVE and the Query Server. Refer to the *Configuring and Licensing Calibre/xCalibre Tools Guide* for more details.

Interface Prerequisites

Use of RVE/QDB-H is dependent on the following:

- Availability of hpux, Solaris, or Linux platforms.
- Access to a supporting layout editor (ICgraph, Calibre WORKbench, Calibre LITHOview, DESIGNrev, Cadence Virtuoso, or Seiko SX9000)
- For investigating DRC/DRC-H results: an ASCII DRC results database (see DRC Results Database in the *SVRF Manual*)
- For investigating ERC results: an ASCII ERC results database (see ERC Results Database in the *SVRF Manual*)
- For investigating flat and hierarchical LVS results: an SVDB directory. This directory is required for accessing query data and is generated when the Mask SVDB Directory specification statement is specified in the rule file.
- For investigating LVS shorts: a short isolation database (see LVS Isolate Shorts in the *SVRF Manual*)
- Familiarity with Query Server concepts such as viewing cell and query instance, as described in section "Viewing, Query, and Query Instance Cells".

RVE Overview

The term *Results Viewing Environment* refers to two interfaces. These are:

- DRC-RVE: This interface allows you to browse DRC results databases, ERC results databases, and LVS short isolation databases.
- LVS-RVE: This interface allows you to investigate and debug your Calibre LVS/LVS-H discrepancies by using the SVDB directory.

RVE provides the following main features:

- The ability to browse DRC, ERC, and short isolation databases on an error-by-error or cell-by-cell basis.
- The ability to highlight connectivity and device information in supporting layout editors. Refer to section "Layout Editor Considerations" on page 15-4 for information.
- The ability to view LVS discrepancies on a cell-by-cell basis, obtain more information by selecting specific errors, and cross-probe between layout views and Spice netlist browsers (layout and source).

RVE uses the Query Server to probe connectivity information. When you use a layout editor to view results, RVE obtains the Query Server results from the hierarchical query database (QDB-H) and sends the data to the layout editor for highlighting. For more information about the query database, refer to the section "Hierarchical Query Database" on page 15-70.

Figure 15-1 shows the data flow associated with RVE:

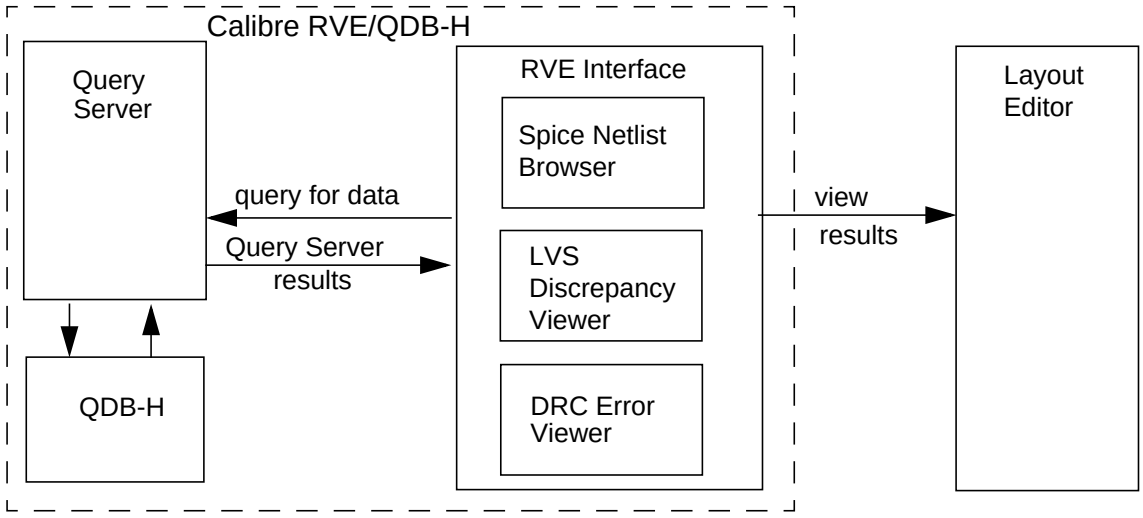


Figure 15-1. Calibre RVE/QDB-H Data Flow Diagram

Refer to section "Calibre RVE/QDB-H" in chapter 2, for the command line invocation syntax. You will probably want to invoke RVE through your layout editor using Calibre Interactive (see "Interface to Calibre RVE" on page 3-21).

Layout Editor Considerations

Calibre RVE can send highlighting information as commands to your layout editor and schematic viewer. The communication is accomplished with socket ports. The layout editor loads custom interface software that enables it to interpret commands sent by RVE. The following sections describe how to install the socket port software into the layout editors that do not support the sockets by default.

IC Station

Within ICgraph, there is a Calibre pulldown menu that includes a selectable button for starting RVE. Refer to the *IC Station User Interface Manual* for more information about ICgraph.

Communication—By default, Calibre RVE uses socket port 9189 to communicate with ICgraph. You can customize the socket port used to receive RVE commands. You can do this through the Setup item on the Calibre pulldown menu or by setting the MGC_CALIBRE_LAYOUT_SERVER environment

variable to *hostname*:*portnumber*. The hostname is optional, in which case you set the environment variable to :*portnumber*. The following statements are equivalent:

```
setenv MGC_CALIBRE_LAYOUT_SERVER sunsvr:9189
setenv MGC_CALIBRE_LAYOUT_SERVER :9189
setenv MGC_CALIBRE_LAYOUT_SERVER 9189
```

If *hostname* is specified, the layout editor interface code ignores it because the layout editor always runs on the local node. The hostname is specified to provide compatibility with RVE's method of parsing this environment variable. RVE uses the hostname to locate the layout viewer.

If your specified port number, or port 9189, is being used by another session of the editor or by another program, the port will not be initialized for RVE use. You can set a different port from the **Calibre > Setup** dialog. Alternatively, the following command allows you to specify a different port without resetting the environment variable and restarting ICgraph:

```
mgc_rve_init_socket socket_port_number
```

Type the command in ICgraph to initialize the specified socket port for RVE communication. In addition, specify the socket port number in RVE by selecting the **Setup > Layout** pulldown menu and specifying the port number in the **Socket Number:** text entry field.

To disable socket communication completely, set the MGC_CALIBRE_LAYOUT_SERVER environment variable to empty. The layout server will not attempt to open the socket.

Highlighting—RVE supports multiple highlighting layers in ICgraph. RVE uses the system layers 4168 (rve_layer_1) through 4177 (rve_layer_10) for its highlighting. See Table 15-4 for more details on error highlighting using RVE.

Other Mentor Graphics Layout Viewers

Mentor Graphics layout viewers such as Calibre WORKbench, Calibre LITHOview, and DESIGNrev behave similarly to ICgraph in terms of the Calibre

pulldown menu and socket port setup. See "IC Station" on page 15-4 for a description. Refer to the specific viewer's documentation for more details on use.

Cadence Virtuoso

You may have the necessary Skill code already installed, especially if you are using Calibre Interactive; the instructions for setting up the Calibre pulldown menu are found in the section "Cadence Virtuoso Interface" on page 3-23. You have this setup if you see a Calibre pulldown menu on the main menu bar of the session window. The section cited in the previous link contains detailed information on using RVE and Calibre Interactive with the Virtuoso Calibre Skill Interface.

Environment Variables for Use With Virtuoso

By default, RVE uses socket port 9189 to communicate with the layout editor. The RVE-Virtuoso interface automatically searches for an available socket if it cannot obtain the default socket. It initializes the first available socket between the numbers 5000 and 9999 and reports the socket number through a dialog box.

You can customize the socket port used to receive RVE commands. Do this by setting the MGC_CALIBRE_LAYOUT_SERVER environment variable to *hostname*:*portnumber*. The hostname is optional, in which case you set the environment variable to :*portnumber*. The following statements are equivalent:

```
setenv MGC_CALIBRE_LAYOUT_SERVER sunsvr:9189
setenv MGC_CALIBRE_LAYOUT_SERVER :9189
setenv MGC_CALIBRE_LAYOUT_SERVER 9189
```

If *hostname* is specified, the layout editor interface code ignores it because the layout editor always runs on the local node. The hostname is specified to provide compatibility with RVE's method of parsing this environment variable. RVE uses the hostname to locate the layout viewer.

If RVE cannot initialize the specified socket from the MGC_CALIBRE_LAYOUT_SERVER environment variable, it will search for another available socket between the values of 5000 and 9999.

Performing one of the following processes allows you to specify a different port without resetting the environment variable and restarting Virtuoso.

- In the Virtuoso layout editor, select **Calibre > Set RVE Socket…** and specify the port number for the server socket in the dialog box that displays.

  or

- In the Virtuoso layout editor, type the command:

  ```
  mgc_rve_init_socket socket_port_number
  ```

  at the CIW prompt to initialize the specified socket port for RVE communication.

In addition to either of these processes, specify the socket port number in RVE by selecting the **Setup > Layout** pulldown menu and specifying the port number in the **Socket Number:** text entry field.

While in the Setup Layout Viewer dialog box, click the **Help** button to display information concerning connection problems between RVE and your layout editor.

To disable socket communication completely, set the MGC_CALIBRE_LAYOUT_SERVER environment variable to empty. The layout server then won't attempt to open the communication socket.

A table of environment and Skill variables is on page 3-33.

Seiko System SX9000

Before using the Seiko SX9000 layout editor with Calibre RVE, you must run the *sxserver* interface program. You can run the program with the following command line:

```
% $MGC_HOME/bin/sxserver
```

You should run sxserver on the same node containing SX9000. This program acts as an interface between Calibre RVE and SX9000, translating highlighting

information from Calibre RVE into SX9000 format. The sxserver program is available on Sun Solaris and HP-UX only.

To set the layers used by sxserver to highlight in SX9000, select the **Setup > Highlight Layers** menu item. The sxserver program sets ten layers by default (layers 201-210).

To specify a location for the temporary files used by sxserver, select the **Setup > Temporary Files** menu item.

This setup information is saved to a file (.sxrvedb) in your home directory and is restored automatically when you restart sxserver.

To use the Seiko SX9000 as layout viewer with Calibre RVE, select it as the layout viewer in Calibre RVE's **Setup > Layout** dialog box.

DRC-RVE Interface

When invoked from your layout editor or the command line, the DRC-RVE session window appears. From this window, you can view your DRC errors in the layout editor. If you don't specify the results database on the command line, RVE prompts you for its pathname upon invocation.

DRC-RVE Session Window

Figure 15-2 shows the session window that appears when you invoke RVE and load DRC (ERC and short isolation are also possible) results database. All actions related to investigating your DRC/DRC-H results are initiated from here.

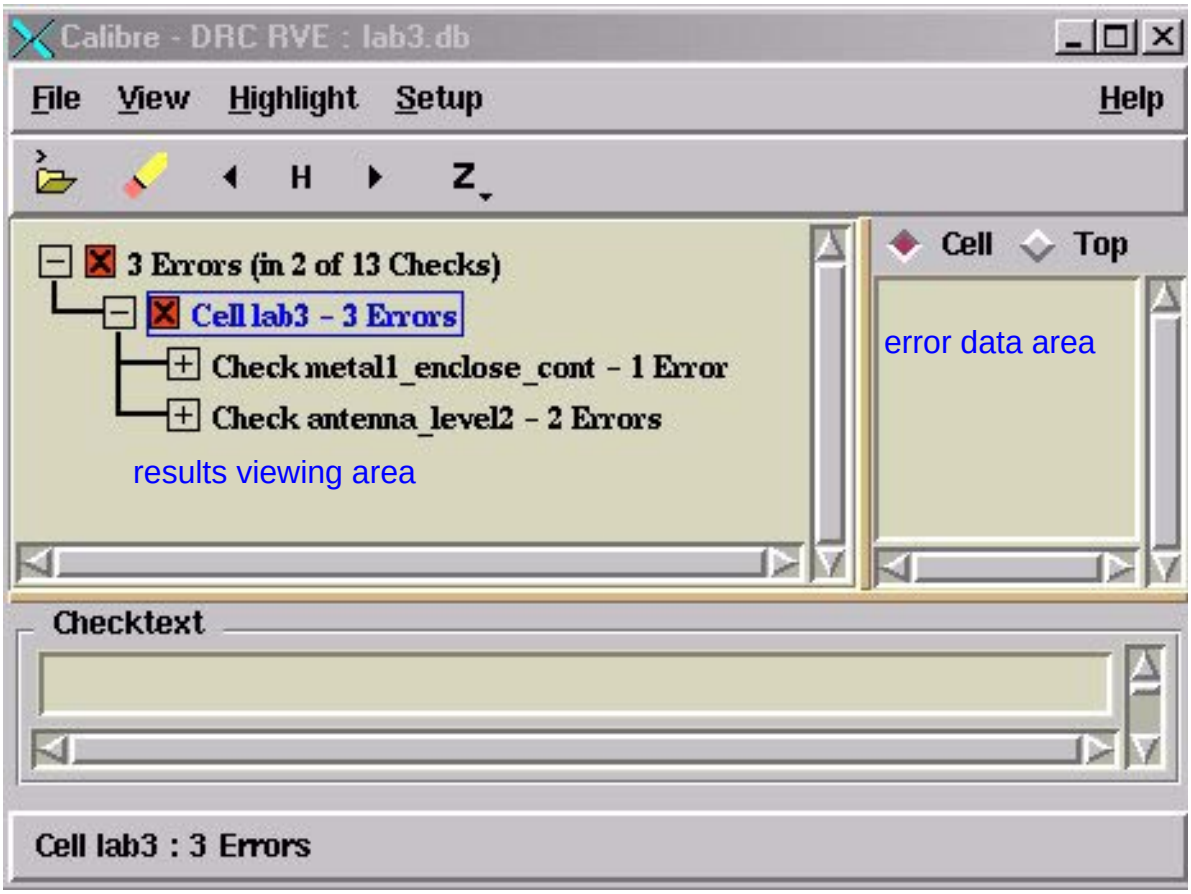


Figure 15-2. DRC-RVE Session Window

For the operating procedures associated with using DRC-RVE, this section uses the following terms to describe window areas, whose locations Figure 15-2 illustrates.

- **Title area**: The *title area* is located at the top of a session window. This area usually contains the name of the application.
- **Main menu bar**: The *main menu bar* appears directly below the title area of a session window. It contains the File, View, Highlight, Setup and Help pulldown menus.
- **Toolbar**: The *toolbar* appears directly below the main menu bar. It displays buttons that represent short cuts to functionality located within the pulldown menus. As applicable, a dialog box appears to prompt you for more information. To preview the button functionality, position the mouse

over a button and pause momentarily. A help message balloon appears that describes the button.

- **Results viewing area**: The *results viewing area* displays a hierarchical tree view of the DRC rules. DRC rules with errors are flagged with red check boxes.

- **Error data area:** The *error data area* displays the coordinates of an error selected in the results viewing area. The **Cell** and **Top** radio buttons allow you to display the selected error's coordinates in either the error-cell's context or in the top-cell's context, respectively. DRC-RVE does not switch between the two contexts if the coordinate transform information is not available in the database.

- **Checktext window:** The *checktext window* displays the design rule check that corresponds to a selected rule in the results viewing area.

- **Message area**: The *message area* is located at the bottom of the session window. This area displays the name of the selected rule check and the currently highlighted error number.

On-line Documentation Help

Both Calibre RVE windows feature a Help menu that allows you to access Calibre on-line documentation. The Help button appears on the session windows on the right end of the main menu bar as shown in Figure 15-2. When you click on the **Help** button, a selection menu appears.

Mentor Graphics uses Adobe Acrobat Exchange as its default on-line documentation viewer. Table 15-1 lists and describes the commands located on the Help pulldown menu.

Table 15-1. Help Pulldown Menu Commands

Command	Description
Open Bookcase	Opens an Adobe Acrobat Exchange session displaying the Calibre Bookcase, which lists all the Mentor Graphics documentation related to the Calibre Verification toolsuite.

Table 15-1. Help Pulldown Menu Commands [continued]

Command	Description
Open User's Manual	Opens an Adobe Acrobat Exchange session displaying the *Calibre Verification User's Manual*.
Open Release Notes	Opens an Adobe Acrobat Exchange session displaying the *Calibre Verification Release Notes*.
Set Up Environment…	Displays the Set Up Online Documentation Environment dialog box, which explains Mentor Graphics on-line documentation system requirements. Use this menu item to set up the Adobe Acrobat reader.
About...	Displays Calibre RVE version information.

File Pulldown Menu

From the DRC-RVE session window, the File pulldown menu appears when you select **File**.

Table 15-2 lists and describes the commands located on the File pulldown menu. The dialog box displayed for a particular entry is described in the section "Usage and Procedures" below. When you open a rule file, report file, or other text file from the File pulldown menu, and that file is already open, the File Viewer window containing the desired file is redisplayed.

Table 15-2. DRC-RVE File Pulldown Menu Commands

Command	Description
Open DB…	Displays the Open Calibre DB dialog box where you specify the pathname to a DRC or ERC results database, LVS (short isolation database), or Spice file. The Open Calibre DB dialog box can also be accessed through the tool bar by selecting the folder icon.

Table 15-2. DRC-RVE File Pulldown Menu Commands [continued]

Command	Description
DRC Rules File…	Displays the File viewer window. A rule file used in the Calibre run (started from Calibre Interactive, for example) that invoked Calibre DRC-RVE is automatically displayed. Otherwise dialog box displays allowing you to select a rule file for viewing. The File Viewer main menu bar contains the **File**, **Edit**, **Options**, and **Windows** pulldown menus, which allow you to edit and browse the rule file, or to open and modify other rule files.
DRC Summary File…	Displays the Open Calibre—DRC Summary Report File dialog box where you specify the pathname to a DRC summary file. The summary file displays in the File viewer window. The File Viewer main menu bar contains the **File**, **Edit**, **Options**, and **Windows** pulldown menus, which allow you to edit and browse the file.
Open Text File…	Displays the Open File dialog box where you specify the pathname to a text file. It displays in the File Viewer window. The File Viewer main menu bar contains the **File**, **Edit**, **Options**, and **Windows** pulldown menus, which allow you to edit and browse the file.
Close	Exits the DRC-RVE session. Terminates the RVE session if this is the last open RVE window.
Exit	Terminates the RVE session.

The behavior of text editing windows is discussed under “Text Editing” on page 3-19.

Database Modification Monitoring

DRC-RVE monitors the modification date/time of the DRC results database, and issues a warning if the database is changed after loading. DRC-RVE gives you a choice of continuing with the current version of the database, reloading the database, or exiting the application. DRC-RVE may not operate reliably if you choose to continue with the current version of the database.

Open Calibre Database Toolbar Button

The Open Calibre Database toolbar button is located at the far left of the toolbar and is depicted by a folder icon. This button is a shortcut to the **File > Open DB** menu item.

Optionally, you can set the MGC_CALIBRE_DB_DIR environment variable to a valid database path. Calibre RVE will attempt to set the working directory to the valid path, specified in this environment variable.

You can reload the current database by clicking the Open Calibre Database button at the same time you press the Control (Ctrl) key on the keyboard.

LVS Short Isolation Database in DRC-RVE

LVS short isolation is handled by a DRC results database accessible through DRC-RVE. This is described in LVS Short Isolation on page 15-69.

View Pulldown Menu

From the DRC-RVE session window, the View pulldown menu appears when you select **View**.

Table 15-3 lists and describes the commands located on the View pulldown menu. The dialog box displayed for a particular entry is described in the section “Usage

and Procedures”. Items on this menu are grayed out if they do not apply to the mode you are in.

Table 15-3. DRC-RVE View Pulldown Menu Commands

Command	Description
Radio button toggle: **By Cell** **By Check**	 Displays a list of all cells in the results viewing area. Displays a list of all design rule checks in the results viewing area.
Error Checks Only toggle button	Alters the list of design rule checks in the results viewing area to show only checks with errors. This is only available when **By Check** is selected.
Sort Cells	Displays a cascading submenu that list options for sorting cells. Choices are: by error count, ASCII alphabetic sorting or dictionary sorting.
Sort Checks	Displays a cascading submenu that list options for sorting design rule checks. Choices are: results database order, by error count, ASCII alphabetic, or dictionary order.
Show Error Data toggle button	Specifies to display the coordinates of an error selected in the results viewing area. Click on any coordinate in the error data area to zoom to it in the layout editor.
Show Error Tips toggle button	As you move the cursor over errors in the results viewing area, specifies to display a popup box with their first coordinates as follows: • Polygons: “P 4 (x,y)” • Edges: “E 2 (x,y)” The toggle does not affect DRC rules that were previously expanded.

Table 15-3. DRC-RVE View Pulldown Menu Commands [continued]

Command	Description
Select Error	Displays a cascading menu that allows you to specify the previous or next error. This is equivalent to the < and > buttons on the toolbar. The same behavior is available from the up and down arrows of your keyboard.
Mark Fixed	Displays a cascading menu having the following toggle button items: Current Error—Specifies to mark current DRC error as corrected. This option is only available when you have a specific error selected. This is for notational purposes only; you must fix errors in the layout editor. This action can also be performed through the right-click popup menu or the alt+F button combination on your keyboard. Current Cluster—Specifies to mark a selected group of DRC errors as corrected. This option is available when you have a design rule check or a specific error selected. This is for notational purposes only; you must fix errors in the layout editor. This action can also be performed through the right-click popup menu or the alt+Shift+F combination on your keyboard. Current Cell—Specifies to mark cell as corrected. This is for notational purposes only; you must fix errors in the layout editor. This action can also be performed through the right-click popup menu or the alt+Ctrl+F combination on your keyboard.

Table 15-3. DRC-RVE View Pulldown Menu Commands [continued]

Command	Description
(cont.)	Cells...—Opens a dialog box that displays fixed cells and allows you to select cells as fixed. Checks...—Opens a dialog box that displays fixed checks and allows you to select checks as fixed.
Mark Waived	Opens a flyout menu that allows you to mark cells, clusters, or individual checks as fixed. These actions can also be initiated by the alt+W, alt+Shift+W, and alt+Ctrl+W keyboard combinations. Click on **Cells...** or **Checks...** to mark errors in cells or checks respectively. You can select discrete multiple entries in the list displayed in the ensuing dialog by Ctrl-left-clicking. Ranges can be selected by Shift-left-clicking. All entries can be selected or unselected by right-clicking and choosing the appropriate command in the pop-up menu. Waived error information is saved in a file in the .rve subdirectory in the directory of the DRC database. This file is named the same as the original DRC database file name, along with a .waived extension. Waived errors remain marked as waived even when the DRC database is regenerated by Calibre DRC.

Highlight Pulldown Menu

From the DRC-RVE session window, the Highlight pulldown menu appears when you select **Highlight**. Table 15-4 lists and describes the commands located on the Highlight pulldown menu. The dialog box displayed for a particular entry is described in the section "Usage and Procedures".

Table 15-4. DRC-RVE Highlight Pulldown Menu Commands

Command	Description
Highlight in Context checkbox	Specifies to highlight errors in the coordinate space of the cell rather than top-level coordinate space. This checkbox is disabled if you did not specify the DRC Cell Name specification statement in the rule file. For details refer to the section "DRC Cell Name Considerations" after this table.
Skip Fixed Errors	Specifies to not highlight errors that have been marked as fixed under View. You can also set this on the **Setup** > **Options** Startup tab.
Skip Waived Errors	Specifies to not highlight errors that have been marked as waived under View. You can also set this on the **Setup** > **Options** Startup tab.
Clear All Highlights	Clears all highlights from the layout editor. By default, highlights accumulate in the layout editor. This action can also be performed through the toolbar button that looks like a pencil eraser. Keyboard shortcut (alt+C).
Current Error	Highlights the current design rule error in the layout editor. This action can also be performed through the H on the toolbar and the right-click popup menu. Keyboard shortcut (alt+H).
Next Error	Highlights the next design rule error in the layout editor. This action can also be performed through the > on the toolbar and the right-click popup menu. Keyboard shortcut (alt+N).

Table 15-4. DRC-RVE Highlight Pulldown Menu Commands

Command	Description
Previous Error	Highlights the previous design rule error in the layout editor. This action can also be performed through the < on the toolbar and the right-click popup menu. Keyboard shortcut (alt+P)
Highlight Error Range...	Displays a dialog box that allows you to highlight (in the layout editor) ranges of errors in a cluster, or all of the errors in a cell (if viewing **By Cell**) or a check (if viewing **By Check**).
Highlight All Errors	Highlights all the design rule errors in the layout editor.
Set Highlight Layers...	Opens a palette that allows you to set the highlight layer index (1-10) for the highlighting layer assigned to individual RuleChecks. Highlight layers for individual RuleChecks can be assigned in your rule file by using this check text comment: `@ RVE Highlight Index: index` where *index* is a highlight layer from 1 to 10.
Export to Layout...	Opens a dialog box that allows you to export error objects on a cell-by-cell basis to the layout editor. You can export all errors from a cell, all errors from a particular check in the cell, or a range of errors from a cell/check pair. Specify the name or number of the layer to export to in the **Export to Layer:** field. (Virtuoso users can specify a layer name or number and an optional layer purpose separated by " " (space). If layer purpose is not specified, it defaults to drawing). Note that RVE does not provide for deletion of such exported objects. They can be deleted through operations provided in the layout editor.

Table 15-4. DRC-RVE Highlight Pulldown Menu Commands

Command	Description
(cont.)	All the settings that affect highlighting also affect the Export operation. These settings are the **Highlight in Context**, **Skip Fixed Errors**, and **Skip Waived Errors** menu items from the Highlight menu. The **Display check names while highlighting errors** setting (**Setup > Options** Text pane) allows for the export of check names along with the error objects.

You can zoom to an error without highlighting it. Press the Ctrl key while clicking on the highlight buttons in the toolbar (<, H, and >).

DRC Cell Name Considerations

The **Highlight > Highlight in Context** menu item described in Table 15-4 depends on the DRC Cell Name specification statement; the syntax is:

```
DRC CELL NAME { YES [CELL SPACE] [XFORM] [ALL] | NO }
```

Refer to the *Standard Verification Reference Format (SVRF) Manual* for a full explanation of the syntax. The following discussion relates to the CELL SPACE and XFORM optional keywords, which affect DRC-RVE behavior.

The results viewing area displays errors by cell (within each rule check) if you specified CELL SPACE.

When you specify CELL SPACE, Calibre DRC-H reports errors in the coordinate space of the cell that the error appears in. If you do not specify CELL SPACE, the errors appear in top-level coordinate space, which is the default. If CELL SPACE is specified, the error data area displays the DRC error coordinates as they appear in the DRC results database. Therefore, these coordinates will be in either cell-space or top-level-space depending on whether you specified CELL SPACE or not.

When you specify the XFORM optional keyword, Calibre DRC-H reports the transformation data for each cell. This keyword affects whether or not you can use DRC-RVE to highlight in both cell-space and top-level-space because it provides the data DRC-RVE requires to transform cell coordinates to top-level coordinates, or vice versa.

For optimal DRC-RVE highlighting behavior specify either:

```
DRC CELL NAME YES CELL SPACE XFORM
```

or

```
DRC CELL NAME YES XFORM
```

The first statement outputs DRC errors in cell coordinate space and provides the transformation data necessary to convert those coordinates to top-level space. The second statement outputs DRC errors in top-level coordinate space and provides the data necessary to convert those coordinates to cell space.

Highlighting With User-defined Keyboard Shortcuts

In the Mentor Graphics ICgraph and Cadence Virtuoso layout editors, the following functions are available:

```
mgc_rve_hl_next_error
mgc_rve_hl_prev_error
mgc_rve_hl_curr_error
```

Following the directions for your layout editor, you can map keyboard shortcuts to these functions to highlight the next, previous, and current errors, respectively.

Prior to using these commands, you must have Calibre DRC-RVE started and a connection to the layout editor socket, established from RVE. These shortcuts allow you to highlight errors without leaving the editing environment. If multiple Calibre DRC-RVE windows are open, these commands communicate with the last window that had input focus.

DRC Highlighting in Virtuoso

The **Highlight > Clear All Highlights** menu item has a corresponding button on the Tool Bar; it is the icon of the pencil and eraser. The following list describes how different clicking scenarios of this button will clear highlights in Virtuoso windows.

- Single-left-click — Clears highlights in the currently selected Virtuoso window.
- Shift-left-click — Clears the highlights in all Virtuoso windows. This is the same action as a double-left-click.
- Ctrl-left-click — Clears highlights corresponding to the error-cell in DRC-RVE.

Error Highlighting of Antenna Violations

DRC-RVE displays antenna ratios associated with each error polygon for Net Area Ratio accumulation output layers if it can read the associated print file. The name of the file is obtained from the check text output to the DRC output database (by specifying DRC Check Text ALL in the rule file). The ratio is displayed along with the coordinates of each error in the error data area of DRC-RVE.

Using Query Help

The Highlight Error Range dialog is the first dialog related to our discussion that has access to Query Help through a button like the one appearing here. We will describe this feature now. To use the Query Help feature, do the following:

1. In any dialog box, click on the **Turn On Query Help** button, located in the lower right corner of the dialog box.

   Query help is enabled and your cursor displays as a question mark (?). The **Turn On Query Help** button is renamed **Turn Off Query Help**. Note that query help is only activated for the dialog box that is displayed.

2. As desired, place the cursor over text entry boxes, buttons, or button options, and click the left mouse button.

   A popup text screen appears that describes a button's function or the type of data to enter into a text entry box.

3. Click the left mouse button again to dismiss the popup text screen.

4. Click on the **Turn Off Query Help** button to deactivate query help.

   Note that you must turn off the query help facility before completing an action or proceeding to another dialog box.

Setup Pulldown Menu

From the DRC-RVE session window, the Setup pulldown menu appears when you select **Setup**. Figure 15-3 shows the Setup pulldown menu items.

Table 15-5 lists and describes the commands located on the Setup pulldown menu.

Table 15-5. DRC-RVE Setup Pulldown Menu Commands

Command	Description
Options…	Displays the Setup DRC-RVE Options dialog box where you specify numerous tool options. These are discussed in detail below.
Layout…	Displays the Setup Layout Viewer dialog box, which connects you to a layout editor. This allows you to view graphical query results. Choices are: ICgraph, Calibre WORKbench (also LITHOview), DESIGNrev, Cadence Virtuoso, and Seiko SX9000. The default is no layout viewer. This dialog also allows you to set your communications socket and your query results file (typically you will use the default settings).
Show Toolbar checkbox	Displays the toolbar that allows you to access commonly used dialog boxes. This is activated by default.

Table 15-5. DRC-RVE Setup Pulldown Menu Commands

Command	Description
Show Tool Tips checkbox	Specifies to display balloon help messages that describe each toolbar button. This is activated by default. Click the right mouse button with the mouse located on a button to view the messages.

Setup > Options...—there are a number of tabs that appear in the Setup DRC-RVE Options dialog:

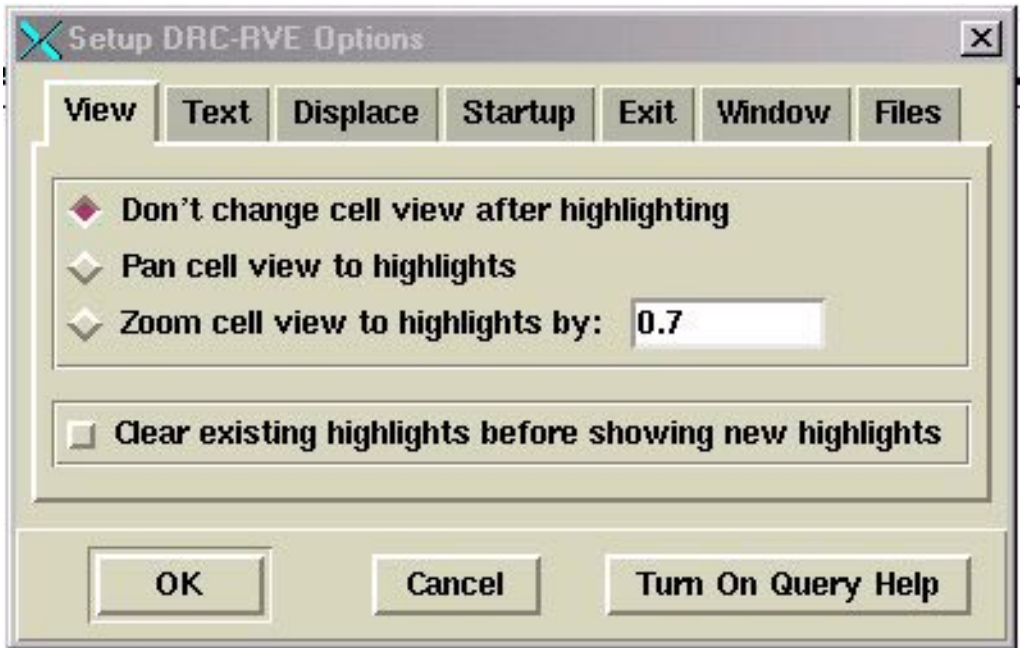


Figure 15-3. DRC-RVE Setup Options

- View—settings on this tab control panning, zooming, and highlighting settings for how RVE interacts with your layout editor. Zoom control can also be accessed by the Z button on the toolbar.

- Text—controls text appearance in short-isolation databases. Also allows you to display RuleCheck names while highlighting errors.

- Displace—controls coordinate offsets for RVE highlighting. Select the Displace coordinates button to perform a displacement. Choices are in top cell or in all cells. The X and Y displacement values go in the **Delta** fields.

- Startup—controls various startup settings. You should visit this tab early in your RVE sessions to select the behaviors you desire.
- Exit—controls exit settings. You should visit this tab early in your RVE sessions to select the behaviors you desire.
- Window—controls position and size of RVE window.
- Files—controls database selection and display features.

Usage and Procedures

This section provides an overview of the DRC-RVE functionality. As applicable, it provides procedural sequences or usage descriptions.

Getting Started

Before investigating your DRC results, you can set the options. All changes you make from the default RVE settings are saved to a .rvedb file located in your working directory. This file is accessed every time you invoke RVE. To return to a default option setting, you must restore it.

To ready DRC-RVE for your queries, do the following, as desired:

1. In the DRC-RVE session window, select **Setup > Options…**

   The Setup DRC-RVE Options dialog box displays, as shown in Figure 15-3. This dialog box contains the window tabs, as shown. You can access each one by clicking on the applicable tab title to bring the window forward. The steps below describe what you can specify in each window.

2. In the **View** tab window, specify the layout editor behavior for viewing the DRC errors. The default is not to change cell view after highlighting.

3. As desired, select the checkbox for clearing existing highlights before showing new highlights.

   This is equivalent to selecting **Clear All Highlights** before each new highlight.

4. Click the **Startup** tab and check its settings. Change defaults as necessary.

5. Click **OK** to enable your changes and dismiss the dialog box.Your changes are saved in the .rvedb file in your home directory.

6. In the DRC-RVE session window, select **Setup > Layout…** to select the layout editor.

   The Setup Layout Viewer dialog box opens.

7. In the Setup Layout Viewer dialog box, specify the following, as applicable:

   - Choose your editor (or none, the default).
   - Hostname the layout editor is running on. Generally, this is localhost if the layout editor is running on the same node as RVE.
   - Socket port the layout editor listens on; default is 9189.

     Refer to section "Layout Editor Considerations" on page 15-4 for information on customizing the port.

     Click the **Help!** button for information about connections between RVE and your layout editor.
   - The intermediate file into which you want your query results stored. The default filename is query_results. Confirm that the path to the specified file is accessible to the layout editor, especially if the layout editor is running on another host.
   - If the layout editor is running but you do not seem to have a connection to RVE, click on the **Connect** button to establish the connection. Make sure the Hostname and Socket Number fields are correct.

Error Browsing in the Layout Editor

As shown in Figure 15-2, the DRC-RVE session window opens with a hierarchical tree view of the DRC rule checks. For rule checks with errors, you can click on any square with a plus sign (+) to view errors numerically. You can

then double-click on a specific error to highlight it. You can use the <, H, and > toolbar buttons or the right-click menu to move from error to error.

By default, errors are listed in DRC results database order and grouped by their design rule check. You can change the order of appearance in RVE by choosing **View > Sort Checks**.

The following procedure acts as an error browsing tutorial to help get you started:

1. Select a design rule check from the results viewing area by clicking on it.

   This is the active rule check. You can investigate all of its errors.

2. Select **Highlight > Next Error**.

   The first error within the active rule check highlights. The status bar indicates this. For example:

   ```
   poly150 : 1 of 39
   ```

   You can also click the > button to highlight the next error.

3. To view the next error select **>** again.

   The previous highlight remains unless you clear it, unless your Setup > Options > View settings indicate to clear existing highlights.

4. Continue highlighting the next error, as desired.

   When you finish one rule check, **Highlight Next Error** automatically begins with the first error of the next rule check listed in the results viewing area.

5. As desired, select **<** to proceed backwards through errors you have already seen.

   As an option, you can also use the up and down keyboard arrows to move the error selection up and down by one error. Shift-up and shift-down move the selection up or down one rule check. The right and left arrows highlight the next and previous errors.

6. As desired, select **Highlight > Highlight Error Range…** to select a range of errors from a particular rule check.

   The errors you specify highlight in the layout editor.

7. As desired, select the **Highlight > Highlight in Context** checkbox to highlight errors in their cell coordinate space rather than top-level coordinate space.

   Refer to section "DRC Cell Name Considerations" on page 15-19 for information about the DRC Cell Name specification statement. This statement is required in your SVRF rule file if you wish to highlight errors in cell coordinate space.

8. As desired, you can use the **Z** button on the Toolbar. When you click on the **Z** button in the toolbar, you see a drop-down menu. The menu allows you to change the zoom settings (to: no view change, pan view to highlights, and zoom view to highlights). You can also control whether existing highlights are deleted before new highlights are drawn. This menu allows access to the same settings as those in the **Setup > Options** dialog in the View tab.

9. As desired, select **View > Mark Fixed** after you fix the DRC errors in your layout editor. You can choose **Current Error**, **Current Cluster**, or **Current Cell**. You can also use the right-mouse-click menu to select items to mark as fixed.This is a notational convenience for you; you must actually correct the DRC error in the layout editor.

   See Figure 15-2. It shows the cell errors as expanded, but not the individual rule checks. To mark individual errors as fixed, select a specific DRC error number (you should completely expand the error tree to see the individual error numbers; this is done by clicking on the + signs at the rule check level of the tree). When you toggle the **Error Fixed** button from the right-click menu (or from **View > Mark Fixed > Current Error**), the selected DRC error appears as fixed (green). As each DRC error is marked as fixed, a count of the fixed errors displays for the entire rule check.

   If you want to mark an entire rule check as fixed, select the rule check in the error tree. When you select the **Cluster Fixed** toggle button, all the DRC errors in a rule check appear as fixed along with a count of total fixed

errors. A rule check cluster will automatically appear as fixed if all errors beneath it have been marked as fixed.

If you want to show a cell as fixed, select the cell name in the error tree. When you select **Cell Fixed**, all the errors for the cell appear as fixed. If all rule checks beneath a particular cell are marked as fixed, the cell will automatically appear as fixed.

You can change the state of any error, cluster, or cell back to the unfixed state.

Fixed error-states are saved in the DRC database as checktext in a check added to the end of the DRC database. This check is named __RVE_ERROR_TAGS__ and contains no errors.

LVS-RVE Interface

This section describes the RVE interface to LVS/LVS-H. It describes the LVS-RVE session window and associated pulldown menus, as well as usage details.

When invoked, the LVS-RVE session window appears. From this window, which is described in section "LVS-RVE Session Window" on page 15-30, the information you can query includes:

- **Cells**: query layout or source cell names; corresponding hcell pairs.
- **Nets**: query net names appearing in the cell (flat or hierarchical); specific source or layout nets; net paths through the hierarchy; layout net shapes corresponding to a specific net; name of the closest net to a given coordinate; and pins and ports on a net.
- **Devices**: device names and instances appearing in the cell (flat or hierarchical); seed shapes on bad devices; specific source or layout devices; device information; name of the closest device to a given coordinate.
- **Ports**: query port names appearing in the cell (flat or hierarchical); port information; name of the closest port to a given coordinate.

- **Placements**: query layout pathnames of each cell placement that meets specific criteria; sub-cell placements in a cell.

LVS-RVE allows you to highlight nets and devices using your layout editor. In addition, RVE includes cross-probing capabilities between the source netlist, layout netlist, and layout editor.

You access LVS Report data through the discrepancy viewer and the Spice netlist with the Spice browser. For example, you can select a net name listed in the discrepancy viewer, then highlight it in the layout editor. The net name also highlights in the layout and source netlists if they are open in their respective browsers. For more information on the discrepancy viewer and Spice Browser, refer to sections "Cross-probing with the Discrepancy Viewer" on page 15-59 and "Cross-probing with the Spice Browser".

If you do not specify the SVDB directory, RVE prompts you for its pathname upon invocation.

The help features for this tool are the same as for DRC-RVE. Balloon help for toolbar items is on by default. The documentation bookcase in accessible from the Help pulldown menu. For detailed descriptions of the dialog box contents, use the Query Help button, which is described in section "Using Query Help" on page 15-21.

Mask SVDB Directory Considerations

LVS-RVE performs different tasks depending on the files located in the specified SVDB directory. The rule file statement:

```
Mask SVDB Directory QUERY
```

produces all necessary files to perform the tasks mentioned above.

However, when you specify the rule file statement:

```
Mask SVDB Directory PHDB
```

Calibre generates only the *layout_primary*.phdb and *layout_primary*.dv files. This statement allows LVS-RVE to only highlight in the layout and the layout netlist,

with cross-probing. It does not allow highlighting or cross-probing to the source netlist.

When you specify the rule file statement:

```
Mask SVDB Directory XDB
```

Calibre generates only the *layout_primary*.xdb and *layout_primary*.dv files. This statement allows LVS-RVE to only highlight in the schematic and the source and layout netlists. It does not allow highlighting in the layout.

LVS-RVE Session Window

Figure 15-4 shows the session window that appears when you invoke RVE through Calibre Interactive, your layout editor, or `calibre -rve` <filename> on the command line. All actions related to investigating your LVS/LVS-H discrepancies are initiated from here.

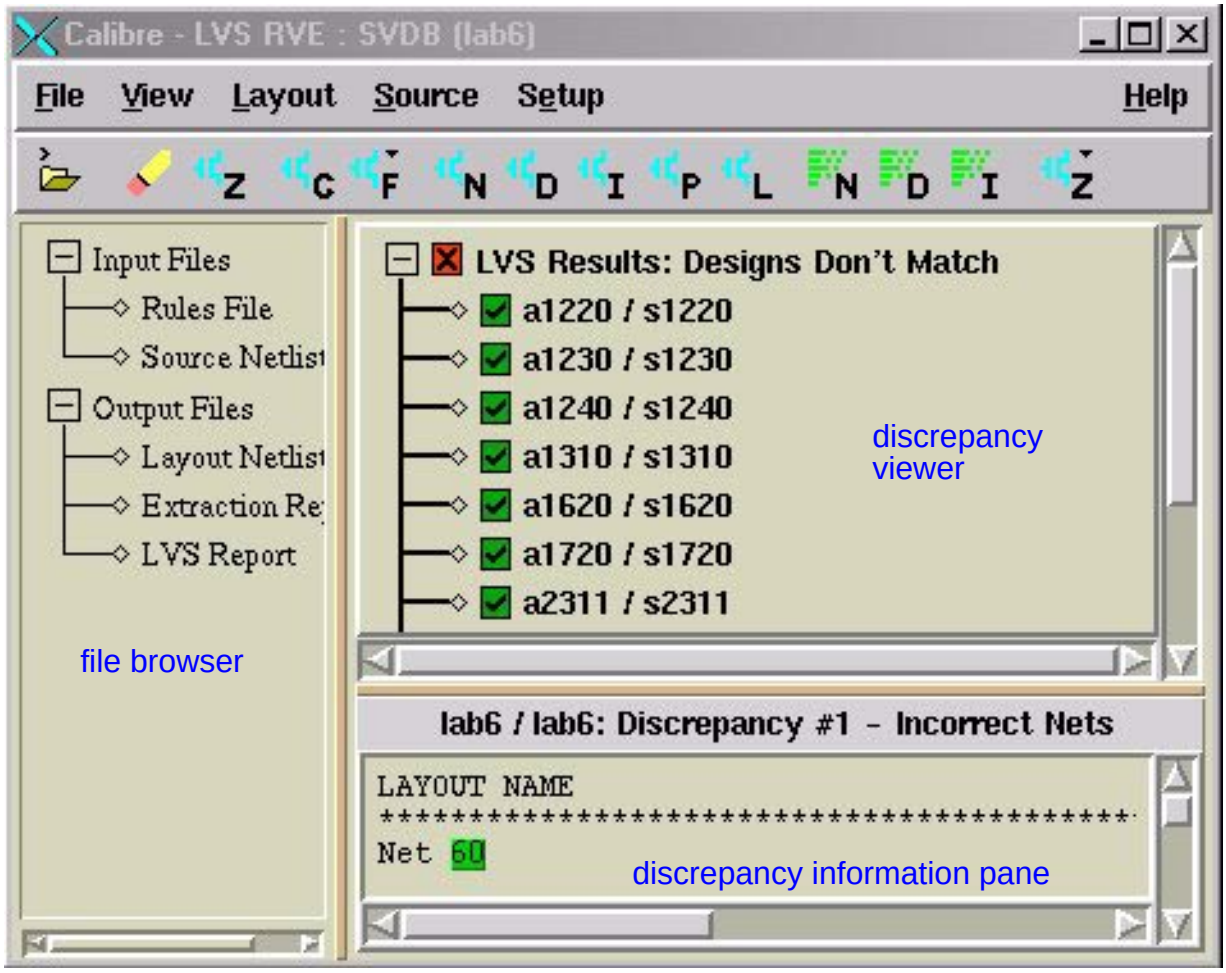


Figure 15-4. LVS-RVE Session Window

For the operating procedures associated with using LVS-RVE, this section uses the following terms to describe window areas, whose locations Figure 15-4 illustrates.

- **Title area**: The *title area* is located at the top of a session window. This area usually contains the name of the application.
- **Main menu bar**: The *main menu bar* appears directly below the title area of a session window. It contains the **File**, **View**, **Layout**, **Source**, **Setup**, and **Help** pulldown menus.
- **Toolbar**: The *toolbar* appears directly below the main menu bar. When you place the cursor over the icons, it displays buttons that represent short cuts

to functionality located within the pulldown menus. As applicable, a dialog box appears to prompt you for more information when you select these buttons.

To preview the button functionality, position the mouse over a button and a help message appears that describes the button.

- **File browser**: The file browser displays a tree view of the input and output files related to the particular LVS run. Selecting the items in the tree with your mouse opens them.

- **Discrepancy viewer**: The discrepancy viewer displays a tree view of discrepancies on a cell-by-cell basis.

- **Discrepancy information pane**: The discrepancy information pane displays an excerpt from the LVS report related to the Discrepancy highlighted in the discrepancy viewer. You can right-click in this pane to display a popup menu with various options depending on the context.

- **Message area**: The message area is located at the bottom of the session window. It provides you with information such as the current query cell.

  When loading the SVDB in LVS-RVE, the message area displays a status bar showing the loading progress.

File Pulldown Menu

From the LVS-RVE session window, the File pulldown menu appears when you select **File**.

Table 15-6 lists and describes the commands located on the File pulldown menu. The dialog box displayed for a particular entry is described in the section “Usage and Procedures” below. When you open a rule file, report file, or other text file

from the File pulldown menu, and that file is already open, the File Viewer window containing the desired file is redisplayed.

Table 15-6. LVS-RVE File Pulldown Menu Commands

Command	Description
Open DB	Displays the Open Calibre DB dialog box where you specify the pathname to a results database. The Open Calibre DB dialog box can also be accessed through the file folder toolbar item.
Rules File...	Displays the rule file in the File viewer window. The File viewer main menu bar contains the **File**, **Edit**, **Options**, and **Windows** pulldown menus, which allow you to edit and browse the rule file. If RVE does not find a rule file, a dialog box prompt displays. This is also accessible from the file browser pane.
Extraction Report...	Opens the extraction report file named *lvs_report_name*.ext in a File window. If the extraction report file is not found, it displays the Open LVS Report File dialog box where you specify the pathname of the extraction report file. This is also accessible from the file browser pane.
LVS Report...	Displays the LVS Report in the File Viewer window. The File Viewer main menu bar contains the **File**, **Edit**, **Options**, and **Windows** pulldown menus, which allow you to edit and browse the LVS Report. This is also accessible from the file browser pane.
Source Netlist...	Opens the Spice netlist in a File Viewer window. If the Spice netlist is not found, displays the Open Source Netlist dialog box where you specify the pathname of the Spice source netlist. See section “Cross-probing with the Spice Browser” on page 15-62 for more information. This is also accessible from the file browser pane.

Table 15-6. LVS-RVE File Pulldown Menu Commands [continued]

Command	Description
Layout Netlist...	Displays the Open Layout Netlist dialog box where you specify the pathname of the layout netlist. The layout netlist displays in a File Viewer window, which allows you to trace nets down the hierarchy and cross-probe with the discrepancy viewer. See section "Cross-probing with the Discrepancy Viewer" on page 15-59 for more information.This is also accessible from the file browser pane.
Exit	Terminates the RVE session.

The behavior of text editing windows is discussed under "Text Editing" on page 3-19.

Open Calibre Database Toolbar Button

The Open Calibre Database toolbar button is located at the far left of the toolbar and is depicted by a folder icon. This button is a shortcut to the **File** > **Open DB** menu item.

You can also reload the current database by clicking the **Open Calibre Database** button at the same time you press the **Ctrl** key on the keyboard.

LVS-RVE monitors the open LVS database for changes. If it detects that the database has changed, LVS-RVE displays a warning dialog that allows for reloading of the database.

View Pulldown Menu

From the LVS-RVE session window, the View pulldown menu appears when you select **View**. This menu allows you to control various viewing features like highlighting, panning, and zooming.

Table 15-7 lists and describes the commands located on the View pulldown menu. The dialog box displayed for a particular entry is described in the section “Usage and Procedures” below.

Table 15-7. LVS-RVE View Pulldown Menu Commands

Command	Description
View Discrepancies	Accesses the discrepancy viewer and displays LVS discrepancies in the results viewing area.
Show All Cells **Show Discrep. Cells**	Displays all cells in the discrepancy viewer. Displays those cells with discrepancies in the discrepancy viewer.
Sort Cells	Displays a cascading submenu that lists options for sorting cells in the discrepancy viewer. Choices are: by discrepancy count, ASCII alphabetical by layout names, ASCII alphabetical by source names, and LVS Report order.
Zoom To Last Highlight	Zooms to the last highlighted net, device, instance, or port using the zoom factor specified in the **Setup > Options** View tab. The default zoom factor is .7. This action can also be performed through the left-most Z button on the toolbar. The zoom options toolbar button (the right-most Z button) also controls zoom features.
Clear Highlights	Clears all highlights from the layout cell being viewed on a layout editor. By default, highlights accumulate in the layout editor. This action can also be performed through the pencil eraser button on the toolbar.

LVS Highlighting in Virtuoso

The following list describes how different clicking scenarios of this button will clear highlights in Virtuoso windows.

- Single-left-click—Clears highlights in the currently selected Virtuoso window.
- Shift-left-click—Clears the highlights in all Virtuoso windows. This is the same action as a double-left-click.
- Ctrl-left-click—Clears highlights corresponding to the query-cell in LVS-RVE.

Layout Pulldown Menu

From the LVS-RVE session window, the Layout pulldown menu appears when you select **Layout**. This menu allows you to perform queries on layout features.

Table 15-8 lists and describes the commands located on the Layout pulldown menu. The dialog box displayed for a particular entry is described in the section "Usage and Procedures" below.

Table 15-8. LVS-RVE Layout Pulldown Menu Commands

Command	Description
Set Query Context...	Displays the Set Layout Context dialog box where you can specify the viewing cell and query instance for your queries. The Set Layout Context dialog box can also be accessed through the C button on the toolbar. Refer to section "Viewing, Query, and Query Instance Cells" on page 15-75 for more information.
Set Query Filters	Displays a flyout menu that allows you to set your layout query filters. Your choices are: **Layers…** and **Devices…** Both options display dialog boxes where you select only those layers or device types you want included in your queries. The submenu can also be accessed through the F button on the toolbar.

Table 15-8. LVS-RVE Layout Pulldown Menu Commands

Command	Description
Net Queries...	Displays the Query Layout Nets in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box. This dialog box has numerous highlighting options for you to choose from. This dialog box can also be accessed through the N button with the transistor symbol by it on the toolbar.
Device Queries...	Displays the Query Layout Devices in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box. This dialog box has several highlighting options for you to choose from.This dialog box can also be accessed through the D button with the transistor symbol by it on the toolbar.
Instance Queries...	Displays the Query Layout Instance in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box. This dialog box can also be accessed through the I button with the transistor symbol by it on the toolbar.
Port Queries...	Displays the Query Layout Ports in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context For Queries dialog box. This dialog box can also be accessed through the P button on the toolbar.
Location Queries...	Displays the Query Layout Location in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box. This dialog box has several highlighting options for you to choose from. This dialog box can also be accessed through the L button on the toolbar.

Table 15-8. LVS-RVE Layout Pulldown Menu Commands

Command	Description
Unconnected nets...	Displays the Unconnected Nets in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box. This dialog box allows you to locate instances in which a specified target cell and net does not connect to a reference net in the query cell.
Find Cell Instances...	Displays the Find Layout Instances in *layout_cell* dialog box, where *layout_cell* is the query cell as specified in the Set Layout Context dialog box.
Layout Cells...	Displays the Layout Cells dialog box where you can view a list of layout cell names and their corresponding source cell names.

Source Pulldown Menu

From the LVS-RVE session window, the Source pulldown menu appears when you select **Source**.

Table 15-9 lists and describes the commands located on the Source pulldown menu. The dialog box displayed for a particular entry is described in the section "Usage and Procedures" below.

Table 15-9. LVS-RVE Source Pulldown Menu Commands

Command	Description
Net Queries...	Displays the Query Source Nets in *source_cell* dialog box, where *source_cell* is the source cell that corresponds to the query cell specified in the Set Layout Context dialog box. The Query Source Nets dialog box can also be accessed through the N button with the netlist symbol by it on the toolbar.

Table 15-9. LVS-RVE Source Pulldown Menu Commands

Command	Description
Device Queries...	Displays the Query Source Devices in *source_cell* dialog box, where *source_cell* is the source cell that corresponds to the query cell specified in the Set Layout Context dialog box. The Query Source Devices dialog box can also be accessed through the D button with the netlist symbol by it on the toolbar.
Instance Queries...	Displays the Query Source Instances in *source_cell* dialog box, where *source_cell* is the source cell that corresponds to the query cell specified in the Set Layout Context dialog box. This dialog box can also be accessed through the I button with the netlist symbol by it on the toolbar.
Source Cells	Displays the Source Cells dialog box where you can view a list of source cell names and their corresponding layout cell names.

Setup Pulldown Menu

From the LVS-RVE session window, the Setup pulldown menu appears when you select **Setup**.

Table 15-10 lists and describes the commands located on the Setup pulldown menu. Some of the dialog boxes displayed for particular entries are described in the section “Usage and Procedures”.

Table 15-10. LVS-RVE Setup Pulldown Menu Commands

Command	Description
Options...	Displays the Setup LVS-RVE Options dialog box where you specify setup options such as location query filter distance, marker size, and how you want the highlighted query results displayed. Refer to Table 15-14, Parameter Commands, for details about Query Server filters and markers.
Layout...	Displays the Setup Layout Viewer dialog box, which connects you to a layout editor. This allows you to view graphical query results. Choices are: Mentor Graphics editors, Cadence Virtuoso, and Seiko SX9000. The default is no layout viewer.
Schematic...	Displays the Setup Schematic Viewer dialog box, which connects you to either the Design Architect or Cadence Composer schematic viewer. This allows you to view schematic graphical query results. The default is no schematic viewer.
Show Toolbar checkbox	Displays the toolbar that allows you to access commonly used dialog boxes. This is activated by default.
Show Tool Tips checkbox	Specifies to display balloon help messages that describe each toolbar button. This is activated by default. Click the right mouse button with the mouse located on a button to view the messages.

Setup > Options... opens the Setup LVS-RVE Options dialog box has the following tabs:

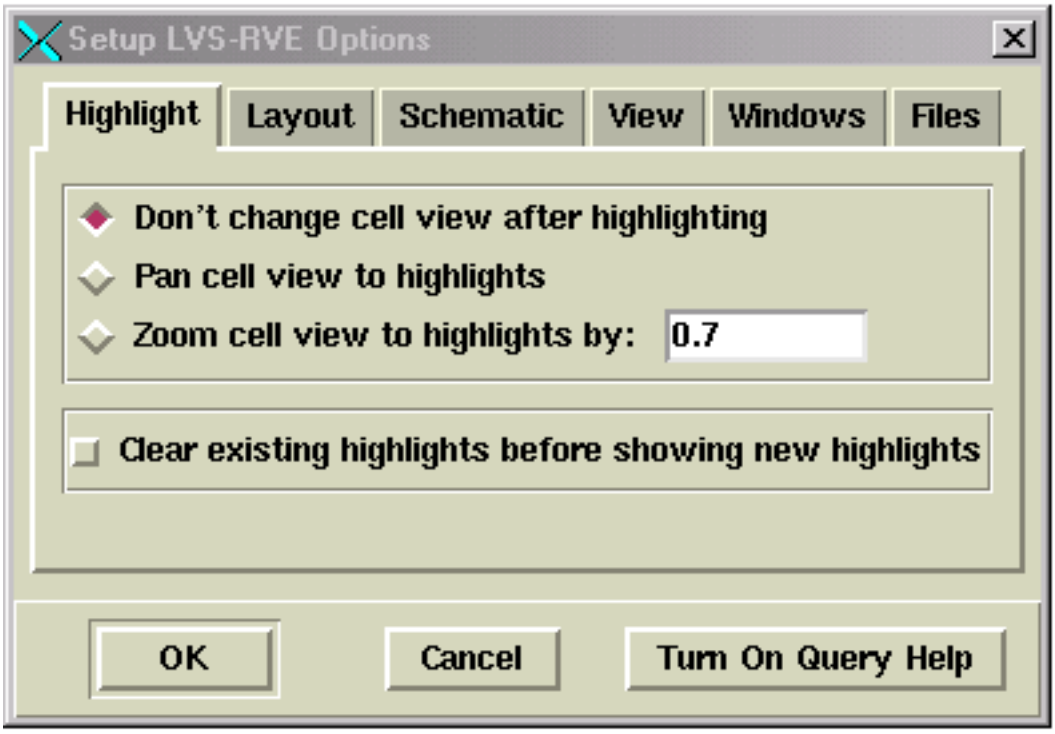


Figure 15-5. LVS-RVE Setup Options

- Highlight—settings on this tab control panning, zooming, and highlighting settings for how RVE interacts with your layout editor. These controls can also be accessed by the rightmost Z button on the toolbar.

- Layout—controls layout highlighting and viewing preferences. It has the following tabs:

  - Filters—allows you to specify layer and device distance filters. These are used for location queries.

  - Highlight—controls whether highlights are shown in the discrepancy cell or the top-cell. When highlighting in the top cell, LVS-RVE will choose a representative instance of the discrepancy cell in the top cell and set the query context to that instance. Layout highlighting of objects within the discrepancy cell will then be displayed in the top-cell within the boundary of the chosen instance.

  - Displace—controls coordinate offsets for RVE highlighting. Select the Displace coordinates button to perform a displacement. Choices are in top cell or in all cells. The X and Y displacement values go in the **Delta** fields.

  - Misc.—controls marker size and maximum vertices for polygon output.

- Schematic—controls whether you use a Spice prefix when highlighting a schematic device (true is the default).

- View—controls startup and exit settings, toolbar button appearance, database warnings, and browse settings for pseudo cells and deviceless cells. You should visit this tab early in your RVE sessions to select the behaviors you desire.

- Windows—controls position and size of RVE window.

- Files—controls database selection filters.

Usage and Procedures

This section provides an overview of the LVS-RVE functionality. As applicable, it provides procedural sequences or usage descriptions.

Getting Started

Before investigating your LVS-H results, you can set the query options, layout editor, schematic editor, and filters as desired. This section provides a procedure to help you tailor your RVE environment to your needs.

All changes you make to the RVE settings are saved to a .rvedb file located in your working directory. This file is accessed every time you invoke RVE. To return a setting to its default, you must specify the default explicitly.

To ready LVS-RVE for your queries, do any or all of the following steps:

1. In the LVS-RVE session window, select **Setup > Options** (see Figure 15-5)

2. In the **Highlight** tab, specify the layout editor behavior for viewing the LVS errors. By default, the cell view does not change during highlighting. Alternatively, you can pan the cell view or zoom to the cell view on highlight.

3. The **Layout** tab enables you to control various layout presentation features using the following tabs:

   a. Select the **Filters** tab if you want to specify filters. You can specify the following:

      - The filter distance in user units from the location point of a location-based query beyond which a port, device, or net shape is ignored.

      - The layers to include in your queries by selecting the **Layer Filters** button. All layers are queried by default.

      - The device types you want searched in device queries by selecting the **Device Filters** button. All devices are queried by default.

      You can also set your layer and device type filters through the session window **Layout > Set Query Filters** menu item or the toolbar. When you set filters, the layer names and device types display in the status bar at the bottom of the session window.

   b. Select the **Misc.** tab if you want to change the size of the marker used to mark pin and seed shape locations, or set the maximum vertex count for polygons output at highlighted data.

      The **Misc.** tab comes forward and you can specify the marker size in the **Marker Size:** text entry box or a customized vertex count in the **Custom:** text entry box. By default, the vertex count is 4096 or the value specified by the optional DRC Maximum Vertex specification statement in the rule file.

   c. Select the **Displace** tab if you want to displace LVS-RVE highlighting coordinates by an X or Y increment while highlighting in a layout viewer. This might be used, for example, if the GDSII produced from

the viewer has been displaced by a corresponding increment from its original location in the layout editor.

Click the **Displace coordinates** check box to enable this feature. You can choose to displace highlighting in just the top cell, or in all cells. The increments are specified in user units. These increments are added to each highlighting coordinate generated by LVS-RVE.

4. Click the **View** tab if you want to change the opening and closing behaviors of LVS-RVE. You can also choose to browse pseudocells (created during seed promotion, for instance) and deviceless cells (like vias) from this tab.

5. Click the **Windows** tab if you want to customize the size and location for the Calibre LVS-RVE session, source netlist, and layout netlist windows.

   The **Windows** tab comes forward and you can record window positions for future use. First, arrange the windows as you desire, then click the **Record Positions** button.

6. Click **OK** to enable your changes and dismiss the dialog box.

   Your changes are saved in the .rvedb file in your home directory.

7. In the LVS-RVE session window, select **Setup > Layout…** to select the layout editor.

   The Setup Layout Viewer dialog box opens.

8. In the Setup Layout Viewer dialog box, specify the following, as applicable:

   - Mentor Graphics, Cadence Virtuoso, Seiko SX9000, or no layout viewer.

   - Hostname the layout editor is running on. Generally, this is localhost if the layout editor is running on the same node as RVE.

   - Socket port the layout editor listens on, which defaults to 9189.

Refer to section “Layout Editor Considerations” on page 15-4 for information on customizing the port.

Click the **Help** button for information about connections between RVE and your layout editor.

- The intermediate file into which you want your query results stored. The default filename is query_results. Confirm that the path to the specified file is accessible to the layout editor, especially if the layout editor is running on another host.

- If the layout editor is running but you do not seem to have a connection to RVE, click on the **Connect** button to create the connection. Make sure the Hostname and Socket Number fields are correct.

9. Click **OK** to enable your changes and dismiss the dialog box.

10. In the LVS-RVE session window, select **Setup > Schematic…** to select the Mentor Graphics or Cadence schematic viewer. The Setup Schematic Viewer dialog box opens.

    - In the Setup Schematic Viewer dialog box, select the desired schematic viewer to enable highlighting in schematics. You can either reuse the layout connection to the layout editor, or you can set up an independent socket to the schematic viewer. You do this by specifying a hostname and socket port number for the schematic viewer.

    - If the schematic viewer is running but you do not seem to have a connection to RVE, click on the **Connect** button to create the connection. Make sure the Hostname and Socket Number fields are correct.

11. Click **OK** to enable your changes and dismiss the dialog box.

Browse Button for Query Dialog Boxes

For all of the query dialog boxes we are about to discuss, there is a **Browse** button that opens an instance browser. The hierarchy is displayed on a cell basis. For each unique cell instanced, a node will display the current instance number for

that cell. You can sequence through to the next instance of that cell by left-clicking on the instance number string (for example, click on **1 of 2 instances**). You can also right-click on this string to display a popup menu that allows you scan through and access a random instance index in the sequence.

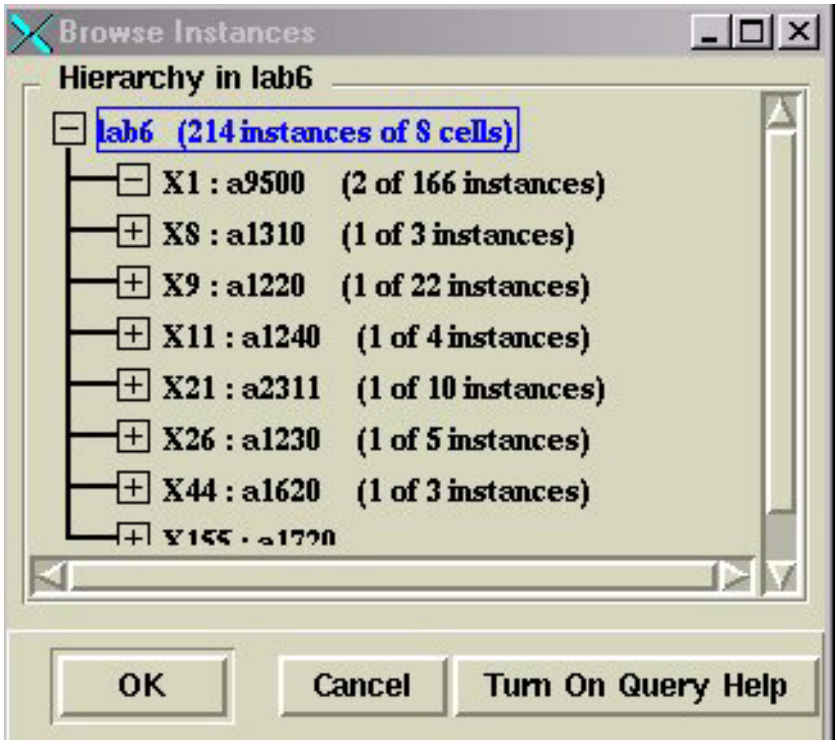


Figure 15-6. LVS-RVE Browse Instances Dialog

This behavior is true of all instance browsers in LVS-RVE.

Net Queries

This section describes the various options you have for querying net information.

You can zoom to a desired instance in the schematic editor by highlighting the instance within RVE. The schematic editor zooms to the top-level instance if the top-level is displayed in the window. If the top-level schematic is not displayed, the view is not adjusted.

You query nets through the Query Layout Nets in *layout_cell* and Query Source Nets in *source_cell* dialog boxes, which this section describes.

Query Layout Nets in *layout_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Nets in *layout_cell* dialog box displays when you:

- Select **Layout > Net Queries…**

  or

- Click on the toolbar button designated with a layout **N**.

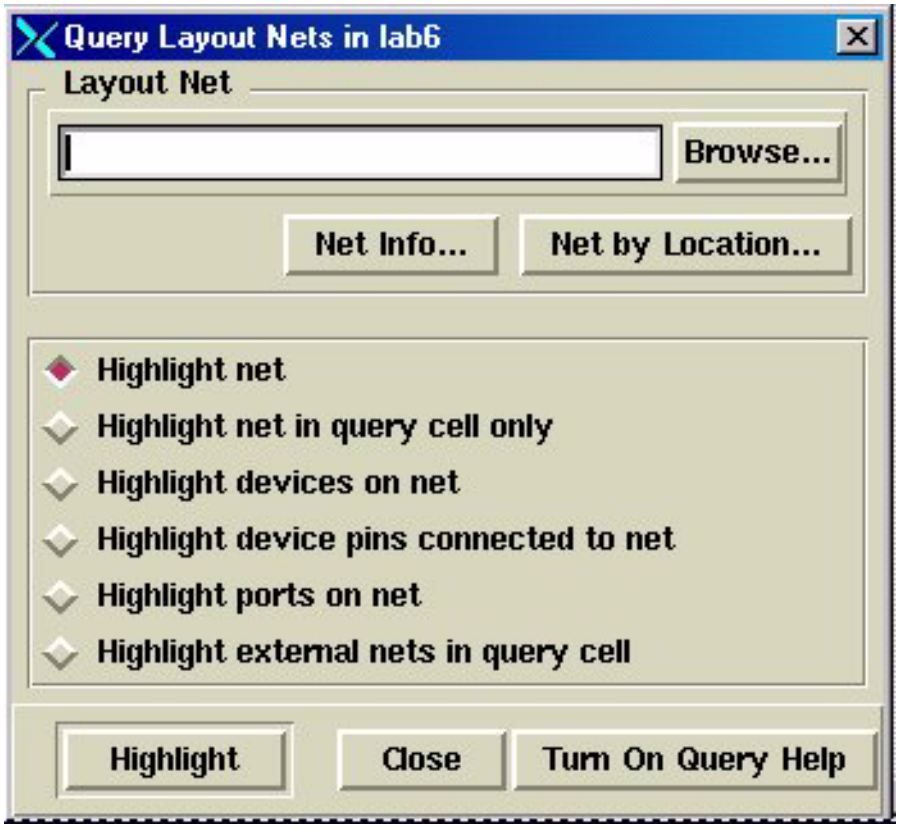


Figure 15-7. Query Layout Nets

Given a layout net name that you specify in the **Layout Name** text entry field, you can do the following:

- Click the **Net Info…** button to view detailed information about the net.

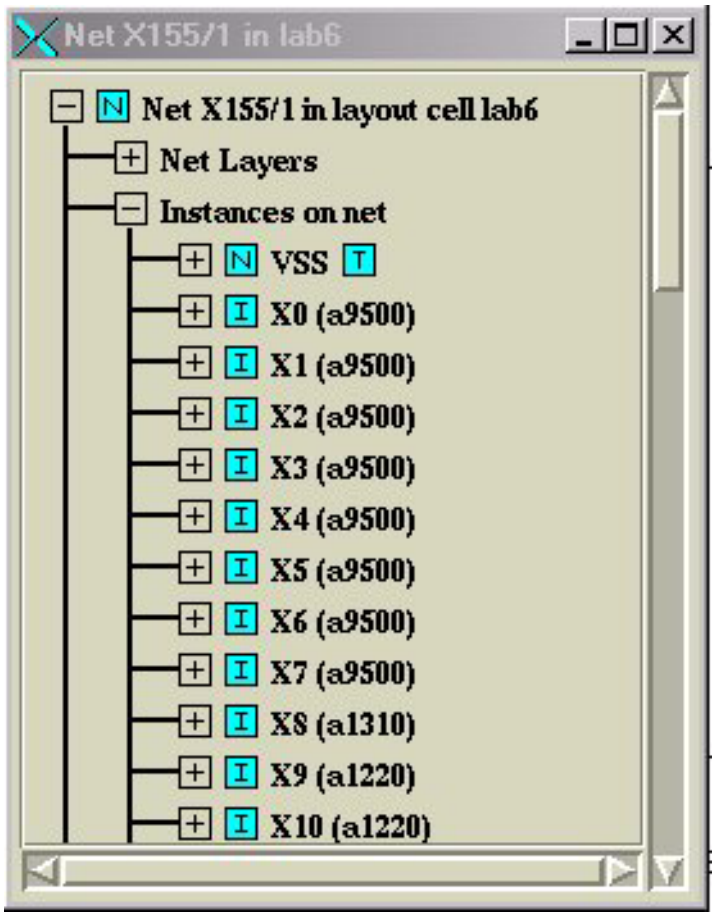


Figure 15-8. Net Info Browser

A browser opens that displays a tree with connectivity information about the net (N icon). Click on any square with a plus sign (+) to access the next data branch. The tree includes information about the Net Layers (L icon), Instances on net (I icon), Devices on net (D icon), Ports on net (P icon), Corresponding Source Nets, and Hierarchical Names. A blue icon indicates a layout item, a green one indicates a source item. Selecting any of the small icons in the tree will highlight the associated item in your layout editor.

For example, The first entry in that panel is the Net Layers node. Expand this node to display all the layers that contribute shapes to the net. Left-click on the icon next to the layer name to highlight the net on that layer. You may also right-click on the net name to display a menu that will allow you to highlight the net on that layer.

As shown in Figure 15-8, the **Instances on net** branch displays the net's name in the instance and a list of instances that the net visits. By clicking on the T icon to the right of the net's name, you can highlight the net in the layout viewer in either the query cell or the instance itself.

- Click the **Net by Location…** button to display the Query Layout Location in *layout_cell* dialog box.

Figure 15-9. Query Location Dialog

Enter the X-Y coordinates in the appropriate fields, or select **Click in layout for point...** and then select a point in your layout editor view.

Selecting **Zoom to point** will zoom the view in your layout editor to the selected point by the current **Setup** > **Options** > **View** zoom factor setting.

Click the **Find closest overlapping:** button to return a list of nets that overlap the location of interest.

Click on the **Highlight** button to see the results of your location query. Objects that overlap the location you specify will be displayed. Select the **List top-level** only check box if you want results in top-cell coordinates.

You can highlight any item in the list by selecting the item and pressing the H key on your keyboard, or by right-clicking the item and selecting **highlight**.

- Select from among the radio button choices in the Query Layout Nets dialog. Click **Highlight** to see the results of your query in the layout editor.

Query Source Nets in *source_cell* Dialog Box. From the LVS-RVE session window, the Query Source Nets in *source_cell* dialog box displays when you:

- Select **Source > Net Queries…**

  or

- Click on the toolbar button designated with a netlist **N**.

Figure 15-10. Query Source Nets Dialog

Given a source net name that you specify in the **Source Net** text entry field, you can do the following:

- Click the **Net Info…** button to view detailed information about the net. A browser similar to Figure 15-8 appears.

  The Net *net_name* in *cell_name* dialog box opens and displays a tree with connectivity information about the net. Click on any square with a + to access the next data branch. The tree includes the associated layout net name and associated connectivity information. Selecting an icon from the tree highlights it in your layout editor.

  The Net *net_name* in *cell_name* also allows you to view parasitic information from Spice files extracted by xCalibre by expanding the **xCalibre PEX Info** node. To view the RC net model for the net, left-click the **Show RC Net Model** label. To view the lumped capacitors on the net, left-click on the **Lumped C** subnode. These actions will ask

you for the extracted netlist file name. To change the associated filename ctrl-left-click on the nodes.

- Click the **Probe net in schematic…** button to use the schematic viewer to select a net.

  This button is only selectable if the schematic viewer's session window is open. When you press this button, LVS-RVE waits for you to click on a net in the schematic. The net then highlights in the layout editor and Spice browser, if open.

- Select the **Highlight** button in the Query Source Nets dialog to highlight the net in the layout editor.

The various dialog boxes and browsers for the other query tools discussed in this section are very similar to the ones discussed above. Screen shots for them are not shown.

Device Queries

This section describes the various options you have for querying device information. You query devices through the Query Layout Devices in *layout_cell* and Query Source Devices in *source_cell* dialog boxes, which this section describes.

Query Layout Devices in *layout_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Devices in *layout_cell* dialog box displays when you:

- Select **Layout > Device Queries…**

  or

- Click on the toolbar button designated with a layout **D**.

  Given a layout device name that you specify in the **Layout Device** text entry field, you can do the following:

  - Click the **Show Device Info…** button to view detailed information about the device.

A browser opens that displays a tree with connectivity information about the device. Click on any square with a + to access the next data branch. Selecting any of the icons in the tree displays the associated item in your layout editor.

- Click the **Device by Location…** Query Layout Location in *layout_cell* dialog box.

  Enter the X-Y coordinates in the appropriate fields, or select **Click in layout for point...** and then select a point in your layout editor view. Selecting **Zoom to point** will zoom the view in your layout editor to the selected point by the current **Setup > Options > View** zoom factor setting.

  Click the **Find closest overlapping:** button to return a list of devices nearest the location of interest. Note that device locations are defined by seed shapes that mark the lowest of the left-most vertices

  Click on the **Highlight** button to see the results of your location query. Select the **List top-level** only check box if you want results in top-cell coordinates.

  You can highlight any item in the list by selecting the item and pressing the H key on your keyboard, or by right-clicking the net and selecting **highlight**.

- Choose from among the three radio button settings in the Query Layout Devices in *layout_cell* dialog and click **Highlight** to show your query results.

Query Source Devices in *source_cell* Dialog Box. From the LVS-RVE session window, the Query Source Devices in *source_cell* dialog box displays when you:

- Select **Source > Device Queries…**

  or

- Click on the toolbar button designated with a netlist **D**.

Given a source device name that you specify in the **Source Device** text entry field, you can do the following:

- Click **Device Info…** to view detailed information about the device.

  A browser opens that displays a tree with connectivity information about the device and its marker location in the layout. Click on any square with a + to access the next data branch.

- Click **Probe device in schematic…** to use the schematic viewer to select an instance.

  This button is only selectable if the schematic viewer's session window is open. When you press this button, LVS-RVE waits for you to click on a device in the schematic. Calibre LVS-RVE uses this information to determine the Spice suffix for the device. If multiple device suffixes match, you will be given a list to choose from.

- Click the **Highlight** button to highlight your query in the layout editor.

Instance Queries

This section describes the various options you have for querying instance information.

You can zoom to a desired instance in the schematic editor by highlighting the instance within RVE. The schematic editor zooms to the top-level instance if the top-level schematic is displayed in the window. If the top-level schematic is not displayed, the view is not adjusted.

You query instances through the Query Layout Instances in *layout_cell* and Query Source Instances in *source_cell* dialog boxes, which this section describes.

Query Layout Instances in *layout_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Instances in *layout_cell* dialog box displays when you:

- Select **Layout > Instance Queries…**

  or

- Click on the button designated with a layout **I**.

  Given a layout instance name that you specify in the **Layout Instance Path** text entry field, you can do the following:

  - Click the **Instance Info…** button to view detailed information about the instance.

    A browser opens that displays a tree with the instance's cell name and its marker location in the layout. Click on any square with a + to access the next data branch. Selecting any icon in the tree highlights the associated item in the layout editor.

  - Click the **Instance by Location…** Query Layout Instances in *layout_cell* dialog box.

    Enter the X-Y coordinates in the appropriate fields, or select **Click in layout for point...** and then select a point in your layout editor view. Selecting **Zoom to point** will zoom the view in your layout editor to the selected point by the current **Setup > Options > View** zoom factor setting.

    Click the **Find closest overlapping:** button to return a list of instances nearest the location of interest.

    Click the **Highlight** button to see the results of your query. Select the **List top-level** only check box if you want results in top-cell coordinates.

    You can highlight any item in the list by selecting the item and pressing the H key on your keyboard, or by right-clicking the item and selecting **highlight**.

  - Click **Highlight** In the Query Layout Instances dialog to show your query results.

Query Source Instances in *source_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Instances in *source_cell* dialog box displays when you:

- Select **Source** > **Instance Queries…**

  or

- Click on the first toolbar button from the right, designated with a netlist **I**.

  Given a source instance name that you specify in the **Source Instance** text entry box, you can do the following:

  - Click the **Instance Info…** button to view detailed information about the instance.

    A browser opens that displays a tree with the instance's cell name and its marker location in the source. Click on any square with a + to access the next data branch. Selecting any icon in the tree highlights the associated item in the layout editor.

  - Click the **Probe inst. in schematic…** button to use the schematic viewer to select an instance.

    This button is only selectable if the schematic viewer's session window is open. When you press this button, LVS-RVE waits for you to click on an instance in the schematic. The instance then highlights in the layout editor and Spice browser, if open.

  - Click the **Highlight** button to highlight the instance marker in your layout editor.

Port Queries

This section describes the various options you have for querying port information. You query ports through the Query Layout Ports in *layout_cell* dialog box, which this section describes.

Query Layout Ports in *layout_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Ports in *layout_cell* dialog box displays when you:

- Select **Layout** > **Port Queries…**

  or

- Click on the fifth toolbar button from the right, designated with a layout **P**.

  Given a layout port name that you specify in the **Layout Port** text entry box, you can do the following:

  - Click the **Show Port Info…** button to view detailed information about the port.

    A browser opens that displays a tree with connectivity information about the port and its marker location in the layout. Click on any square with a + to access the next data branch. Selecting any icon in the tree highlights the associated item in the layout editor.

  - Click the **Port by Location…** button to display the Query Layout Location in *layout_cell* dialog box, which is also accessible through the toolbar.

    Enter the X-Y coordinates in the appropriate fields, or select **Click in layout for point...** and then select a point in your layout editor view. Selecting **Zoom to point** will zoom the view in your layout editor to the selected point by the current **Setup** > **Options** > **View** zoom factor setting.

    Click the **Find closest overlapping:** button to return a list of ports nearest your location of interest. Note that port locations are defined by seed shapes that mark the lowest of the left-most vertices

    Click the **Highlight** button to see the results of your query. Select the **List top-level** only check box if you want results in top-cell coordinates.

    You can highlight any item in the list by selecting the item and pressing the H key on your keyboard, or by right-clicking the item and selecting **highlight**.

  - Click **Highlight** In the Query Layout Ports dialog to show your query results.

Location Queries

This section describes the various options you have for querying cell location, or placement, information. You query cell placements through the Query Layout Location in *layout_cell* dialog box, which this section describes. This dialog is accessible from all of the other Query Layout dialog boxes by selecting the item by Location... button.

Query Layout Location in *layout_cell* Dialog Box. From the LVS-RVE session window, the Query Layout Location in *layout_cell* dialog box displays when you:

- Select **Layout > Location Queries…**

  or

- Click toolbar button designated with a layout **L**. See Figure 15-9.

  Given X-Y coordinates that you specify in the **Viewing Cell Coordinates** text entry boxes, you can do the following:

  - Select the **Nets**, **Devices**, **Instances**, or **Ports** radio button and then click the **Find closest overlapping:** button to return the nets, devices, instances, or ports overlapping the specified coordinate. The nets, devices, instances, or ports appear in the text box below the radio buttons. Note that device and port locations are defined by seed shapes that mark the lowest of the left-most vertices.

    Click the **Highlight** button to highlight the net, device seed shape, instance, or port seed shape returned by the **Find closest overlapping:** command.

  - You can specify the viewing cell coordinates by selecting the **Click in layout for point** button, then clicking in the layout editor. The coordinates corresponding to that location in the layout appear in the **Viewing Cell Coordinates** text entry boxes.

    Click **Zoom to point** to change the focus of the layout editor to the point specified in the dialog box.

Zoom Settings Toolbar Button

The rightmost Z toolbar button facilitates easy changes of zoom settings. When you click on the rightmost **Z** button in the toolbar, you see a drop-down menu. The menu allows you to select zoom settings. You can also control whether existing highlights are deleted before new highlights are drawn. This menu allows access to similar settings as those in the **Setup > Options** dialog Highlight tab.

Cross-probing with the Discrepancy Viewer

The discrepancy viewer allows you to investigate LVS Report discrepancies. This section describes its capabilities and how to use it to cross-probe the layout, source netlist, and layout netlist.

To use the discrepancy viewer you must have a *layout_primary*.dv file in your SVDB directory, where *layout_primary* is the name specified by the Layout Primary specification statement in the rule file. This file is generated when you run Calibre LVS with the Mask SVDB Directory specification statement in the rule file.

To use the discrepancy viewer, follow these steps:

1. In the LVS-RVE session window, select the **View** pulldown menu and click on the **View Discrepancies** radio button (this is selected by default).

   LVS discrepancies display in the discrepancy viewing pane, as shown in Figure 15-4.

   By default, the discrepancies appear in the order the cells appear in the LVS Report. You can select **View > Sort Cells** to choose a different sort mode.

   Each cell with a discrepancy has a discrepancy branch. When you expand the branch and select individual discrepancies in the branch, the associated connectivity information appears in the Discrepancy Information pane. This discrepancy is considered active.

   In the discrepancy information pane, note that certain data is highlighted with colored text boxes, which indicate that you can access further information. Click on a colored text box to make it active, then click your

right mouse button. A popup menu appears, and from the popup menu you can, for example, highlight a selected net in your layout editor or access its connectivity information. You can also double click on a net to highlight nets.

2. Select **File > Layout Netlist…** (or select this item from the file browser)

   A file viewer window opens with the layout netlist. Position it so that you can view the discrepancy viewer, layout editor, and layout netlist. As you highlight items (nets, devices, or instances, for example) in the layout editor, they also highlight in the layout netlist. This functions in reverse also; an item you select in the layout netlist highlights the corresponding item in the layout editor.

   In the layout netlist, double click on a net to highlight the net in the layout editor.

3. Select **File > Source Netlist…** (or select this item from the file browser)

   Another file viewer window opens to display the source netlist. Refer to section "Cross-probing with the Spice Browser" on page 15-62 for detailed information about using the browser. Position it so you can see all four open windows. Note that as you highlight nets, they highlight in both the layout and source netlists.

   In the source netlist, double click on a net to highlight the net in the layout editor and layout netlist.

4. In the discrepancy viewer, as desired, select **View > Show Discrep. Cells**. This shows only the cells with discrepancies. The default view mode is **Show All Cells**.

5. In the results viewing area, click on any square with a + to display details about the mismatched cells.

6. Select a discrepancy to activate it and click your right mouse button.

   A popup menu displays.

7. From the discrepancy viewer popup menu you can do the following:

   - Click **Set Context to Cell** to set the RVE context to the cell containing the current active discrepancy.

   - Click the **View Report File Entry** button to view the portion of the LVS Report that lists the active discrepancy.

     The file viewer window opens to display the LVS Report. The window contains four pulldown menus that allow you to save the report, edit it, reformat it, and execute searches.

   - Click the **List All Discreps in Cell** button to display all the LVS discrepancies associated with the applicable layout cell.

     A new window opens that displays the LVS Report INCORRECT NETS section for that layout cell.

     Note the colored text boxes, which are described in step 1.

   - Click the **Discrepancy Fixed** toggle button to note that you have corrected a discrepancy.

     This is simply a notational convenience for you; you must actually correct the discrepancy in the layout editor. Tagged discrepancies are saved in the *layout_primary*.dv file.

     To use the toggle button, select a specific discrepancy. Anytime you specify a corrected discrepancy, a green box appears next to it. You mark all discrepancies within a type or cell at once by selecting the type or cell and clicking the toggle button.

     When all discrepancies of a certain type within a cell are fixed, a green box also displays next to the discrepancy type's listing. As each discrepancy is marked as corrected, a count of the fixed discrepancies displays for the type as well as the cell. When all discrepancies within a cell are fixed, the red X icon displayed next to the cell listing changes to a transparent X icon.

You can change the state of a discrepancy or group or discrepancies by toggling the **Discrepancy Fixed** button back to its off position.

8. In the discrepancy information pane, select a net, instance, or coordinate, as desired. They are denoted by their colored text boxes. A black underscore appears when an item is active.

   If you double-click a colored text box, the item is highlighted in the layout editor and any netlist browsers you have open.

9. In the discrepancy information pane, click the right mouse button.

   A popup window displays. Its menu items depend on the active item. For example, if you selected a net, you can highlight it in the layout, access further net information, or do a text search. If you selected a coordinate, you can zoom to it in your layout, or highlight the closest net, device, or port.

Cross-probing with the Spice Browser

The Spice browser allows you to investigate a Spice netlist and highlight its elements in your layout editor. This section describes its capabilities and how to use it to cross-probe the layout editor and the other Spice browser (layout or source), if open.

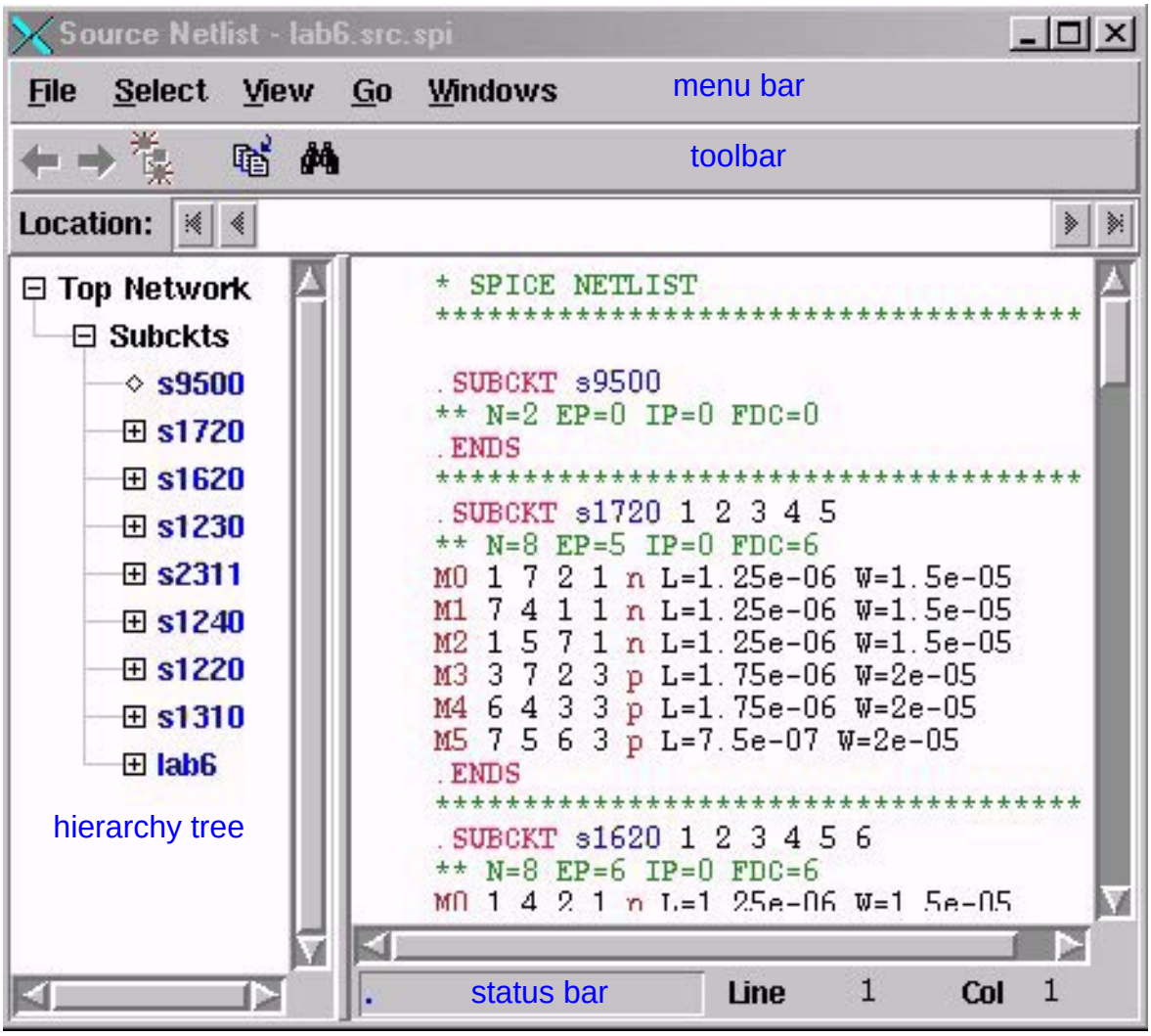


Figure 15-11. Spice Netlist File Viewer

Table 15-11 provides an overview of the Spice browser menu bar items. Many of these menu bar items also appear on the right-click popup menus that appear when

you place the cursor either over empty window space, an item in the hierarchy tree, or an item in the Spice netlist and you right-click.

Table 15-11. Spice Browser Menu Items

Command	Description
File	**Open…** Opens a text file. **Current Files…** Opens a dialog box that shows .include files. You can display any of the listed files in the Spice browser's netlist pane. This is also accessible through the toolbar files icon. **Close** Closes the Spice browser.
Select	**Select by Name...** Allows you to select devices and nets by name in the Spice netlist. You can also add paths to your selection environment. **Clear Selection Environment** Clears the selection environment. **Unselect All** Clears all highlights in the Spice netlist and layout editor (as applicable).
View	**Sort Hierarchy by Name** Sorts the subcircuits, instances, and non-instances listed in the hierarchy tree in alpha-numeric order. **Sort Hierarchy by Called Subckt** Sorts instances in the hierarchy tree by called subcircuit name. **Hide Empty Subckts** Shows/hides empty subcircuits. **Hierarchy Window** Shows/hides the hierarchy window on the left side of the browser. **Line Numbers** Shows/hides line numbers. **Line Wrapping** Activates/deactivates line wrapping. **Font** Adjusts font size. **Status Bar** Shows/hides the status bar.

Table 15-11. Spice Browser Menu Items [continued]

Command	Description
Go	**Search…** Searches the netlist for text you specify. This is also accessible through the toolbar binoculars icon. **Go to line…** Goes to the line you specify. **Go to Select Env** Goes to the selection environment you have chosen, if any, under Select > Select by Name... S**earch Hierarchy for Instance...** Opens the Search Hierarchy for Instance dialog. You can specify which subckt to search in (the "top" subckt) by clicking on that subckt's primary entry in the hierarchy pane. Type in the name of the instance to search for in the text field in the dialog. • The **Forward** and **Backward** buttons allow you search up or down for the instance from the currently selected position. • Select **Find From Top** if you want to begin at the top level subckt and search in a depth-wise fashion. Select **Find Next** if you want to find the next instance (either forward or backward). When an instance is found, the hierarchy pane is opened to that particular instance and the path to the instance is displayed in the Location field (below the toolbar). Right click in the Location area to display a menu that now allows you to set the select environment to that location. **Back** Go back through locations in the hierarchy tree. This is also accessible through the toolbar. **Forward** Go forward through locations in the hierarchy tree. This is also accessible through the toolbar buttons. **Clear History** Clears the search history used by Back and Forward functions.

Table 15-11. Spice Browser Menu Items [continued]

Command	Description
Windows	**Arrange File Viewer Windows:** Changes the File Viewer window display to the specified choice. **Cascade** arranges files in an overlapping cascade. **Tile Horizontally** arranges windows horizontally, not overlapping. **Tile Vertically** arranges windows vertically, not overlapping. **Tile in a 2X2 Grid** arranges files in two columns and two rows, not overlapping. **Original Locations** Reverts to the original File Viewer window display.

The following procedure steps through the various actions you can take with the Spice browser. After step 1, you can highlight and browse as you desire; the sequence is for instructional purposes only.

1. In the LVS-RVE session window, select **File > Source Netlist…** or choose Source Netlist from the file browser.

   The File viewer window displays with the Spice netlist in the righthand window and a hierarchy tree of Spice subcircuits in the lefthand window, similar to that shown in Figure 15-11.

   As shown above, you can display the cell hierarchy by clicking on any square with a + to access the next data branch. The + turns to a - sign.

   Notice that the **Location:** text box displays the path of your location in the subcircuit hierarchy tree. You can select any element in this path to view it in the Spice netlist. Forward and backward arrows at either end of the text box allow you to move along the path when it is longer than the text box itself. You can also select any item in the hierarchy tree. Selecting items in either the path or the tree scrolls the netlist to the item you have selected and places a >> marker by it in the netlist.

   When you click on any net name, number, or instance in the Spice netlist, all references to it in the applicable subcircuit highlight. This also

highlights the corresponding objects in all open views. With your cursor over a highlighted object in a netlist, you can right-click and choose **Unselect Object** to clear the currently selected object highlights. Alternatively, right-click and choose **Unselect all in Subckt** to unselect all highlights in the selected subcircuit. You can also right-click in the empty Spice browser window space to the right of the netlist text and choose **Unselect All** to unselect all highlights.

You can highlight elements in the current Spice netlist only (that is, no other views will display the highlighted objects) by right-clicking over an element. In the popup menu that appears you can then choose either **Select Object** or **Select By Name...**

2. To highlight a net or instance in the layout editor, click the net or instance name.

   Note that if you highlight an object in the layout editor from a Spice browser, the corresponding source cell name highlights in the Spice browser. In addition, the object highlights in all associated open windows. For example, highlighting a layout netlist object will also highlight the corresponding object in the source netlist, if a corresponding object is found. In this way, you can do a netlist-to-netlist comparison of corresponding items. Selecting discrepancies in the discrepancy viewer causes the corresponding objects to highlight in all open layout and netlist windows associated with the discrepancy.

3. To view an item in the hierarchy tree from the corresponding item shown in the Spice netlist, you have many options:

   - Right-click the cursor over an item in the Spice netlist. Selecting **Find in Hierarchy** from the right-click menu will open the hierarchy tree to the object you select. If you right-click a .Subckt call and choose **Go to Called Subckt**, the hierarchy tree and Spice browser will go to the called subcircuit.

   - Click on the **Back** and **Forward** arrows in the toolbar to move through previously highlighted locations in the hierarchy tree.

Each time you change location within the hierarchy tree, LVS-RVE adds it to its history of locations. The **Back** and **Forward** buttons move through this location history, which you can clear with the **Go > Clear History** pulldown menu item.

4. To follow a net's connection through hierarchy, do the following:

   a. In the Spice netlist window, select the net of interest by selecting it with the left mouse button.

      The net is highlighted everywhere it appears in the subcircuit.

   b. In the hierarchy tree window, find the net's subcircuit and click the + to display the instances within it.

      The subcircuit name is listed in parenthesis beside the instance name.

   c. In the hierarchy tree window, pick the instance/subcircuit pair that you want to trace the highlighted net into by placing the mouse over it and clicking the right mouse button.

      A popup menu displays.

   d. Select **Add to Select Env**.

      This command specifies to follow the connectivity of the net, device or instance into the instantiated subcircuit. A red square appears around the instance and subcircuit name to indicate that it has been added to the selection environment. A corresponding red S appears next to selected instances in the netlist.

      Once you add a subcircuit to the selection environment, you can return to it in the hierarchy tree at any time by selecting the **Go to selection environment** item on the toolbar.

      You can also add a subcircuit to the selection environment through the **Add path to select environment** button in the **Select > Select by Name** dialog box.

e. In the hierarchy tree window, double-click on the selected subcircuit name.

   The selected subcircuit displays in the Spice netlist window with the connections of the net, device, or instance highlighted.

   You can follow the net, device, or instance up to the next level of hierarchy by adding successive subcircuits to the selection environment. The connections highlight as you do this.

5. As desired, in the Netlist window, select **Select > Clear Selection Env** to clear the selection environment.

LVS Short Isolation

Calibre LVS generates a short isolation database if you specify LVS Isolate Shorts YES in your rule file. The database is a DRC database and is named *lvs_report_name*.shorts, where *lvs_report_name* is the report name specified by the LVS Report statement in your rule file.

You can open the short isolation database by selecting **File > Open DB...** . Navigate to the short isolation database and select **OK**. The short isolation database you specify will open in a DRC-RVE window. When you select a check in the DRC-RVE window, right-click and choose **Highlight All Check Errors**. The geometry associated with the short will highlight in your layout editor window.

Short isolation databases contain information about shorted text labels and their locations. DRC-RVE can parse this information and optionally display the text while highlighting each short.

Short isolation databases represent each short as a group of polygons in a check. When you highlight all the check polygons from DRC-RVE, the shorted texts are also drawn in the layout. You can control the size of text in the Text tab of the Setup/Options dialog. You can also turn off the drawing of the text in that dialog.

The shorted text for each short (check) is displayed in the Checktext pane at the bottom of the DRC-RVE window. If you click on the location for the text in this pane, DRC-RVE will zoom the layout window to that location.

Text Selection in LVS-RVE Text Fields and Listboxes

Whenever you select any item in a text field or in a listbox, that selection is automatically copied to the X selection and can be pasted by a middle mouse button click. You can turn off this feature by setting the MGC_UI_NO_EXPORT_SELECTION environment variable before starting LVS-RVE.

Hierarchical Query Database

The information in this section describes the Query Server from the perspective of the user who wants to access query database information from the command line. This is not required knowledge for RVE users.

This section describes the Query Server employed by RVE to return connectivity information.

A PHDB is a persistent hierarchical database that contains selected hierarchical geometries, connectivity, extracted devices, #ifdef statement definitions, and Variable specification statement definitions. It also stores environment variable settings during rule file compilation of an original Calibre run and will reuse the stored values when the rule file is restored by the Query Server. It is created when the compiled rules operate on a GDSII layout and is stored in the SVDB.

The Query Server acts as an intermediary between a persistent hierarchical database (PHDB) and a program, such as a Perl script, layout editor, or a user. It is primarily for program-to-program communication, such as between RVE and a PHDB. However, you can also use it interactively to examine a PHDB.

The Query Server returns requested data about a design, which can be used to construct netlists, investigate the details of discrepancies listed in a hierarchical LVS report or for xCalibre debugging. The Query Server can also provide information such as:

- A list of all cells or hierarchical cells in a layout design.
- The geometric representation of a net or device.

- A list of all devices attached to a given net.
- Alist of all nets attached to a given device.
- Cell, net, and device correspondences between the layout and source.

In addition, the Query Server can produce standard format files that contain this type of information through the Calibre Connectivity Interface (CCI), which is discussed later in this chapter.

Upon invocation or when accessed by RVE, the Query Server reads the PHDB, the GDSII file specified by the Layout System specification statement in the rule file, and, if available, reads in the source-to-netlist cross-reference information contained in the cross-reference database (XDB). It then announces that it is ready to respond to query commands.


Note

The rule file must include the Mask SVDB Directory specification statement to use the Query Server. In addition, source-to-netlist cross-reference data is required when querying source information. These must be obtained by running Calibre LVS-H with the *-hcell* option.

Query Server commands are in ASCII format. Commands either control the Query Server or request design information. They are read as standard in. Query Server indirect responses are in ASCII DRC results database file format, and can be viewed with IC Station. Case-sensitivity is driven by the Layout Case and Source Case specification statements in the rule file.

SVDB Database

The Standard Verification Database (SVDB) is composed of several separate databases that are generated during Calibre LVS. The Persistent Hierarchical Database (PHDB) contains information produced during the LVS Extraction (calibre -spice) phase of execution. Cross Reference information produced during the LVS comparison phase of execution may also be generated in several forms.

The Mask SVDB Directory rule file statement controls the types of information written to the SVDB database. For additional information, see the *SVRF Manual*.

The PHDB database contains the following information:

- Cell and cell placement information (hierarchical mode only).
- The complete text of the rule file(s) used for extraction.
- Summary extracted device information including device coordinates, types, calculated properties, pin and pin location information.
- Net connectivity information through the hierarchy of cell placements.
- Geometry information for database layers that appear in Connect and Sconnect operations, serve as Device seed or pin layers, or are target layers of Stamp operations (second argument of Stamp).
- Device seed shape information.

The XDB (Cross Reference Database) contains information generated after LVS comparison associating devices and nets with one another. This database contains:

- Net cross reference for nets matched during LVS including nets matched to multiple nets.
- Unmatched incorrect nets and undecided nets.
- Instance cross reference for instances matched during LVS including smashed instances and gate membership information.
- Unmatched instances (except filtered instances).
- Original netlist placement hierarchy information (for hierarchical mode only).

Information from each database is available independently. Some commands rely on both databases to generate results.

Query Server

Upon invocation, the Query Server reads the GDSII file specified in the rule file, and if available, reads in the source-to-netlist cross-reference database (XDB). It then announces that it is ready to respond to client query commands:

```
OK: Ready to serve.
```

Client Context

The server is aware that the application has multiple client contexts. From the server's point of view client contexts are created, go through phases of being active and inactive and eventually are deleted. Exactly one client context is active at any given time and is called the current client.

Commands are provided for the application to create new client contexts, specify which client context is currently active, and delete client contexts. The server treats all queries as having come from the currently active client context. The server assigns client ID numbers (non negative integers) to the clients. Client ID 0 represents the application itself as a client context. It initially exists, is active, can become inactive, but can never be deleted. A client context can only be inactivated by activating another client context. Only an inactive client context can be deleted. Thus there is always exactly one active client context.

The server maintains a small table, the client table, of information about each client context. Commands exist which allow a client to modify its client table information. Note that there is no scanner state information in the client table. The server does not directly support scanning paradigms. Any scanning state information must be maintained by the application.

Table 15-12 describes the data stored as the current context for each client:

Table 15-12. Client Table

Entry	Description
ID	Client ID number of this client context.

Table 15-12. Client Table [continued]

Entry	Description
Query Cell	The name of a cell (not a cell instance) about which queries are currently being made. The default value is the name of the top cell of the design.
Viewing Cell	The name of a cell (not a cell instance) whose coordinate space is used for the return of geometric results. The value must either be the query cell or a cell containing an instance (possibly separated by several levels of hierarchy) of the query cell.
Query Instance	If the viewing and query cells are the same, this value is NULL. Otherwise it is the pathname relative to the viewing cell of an instance of the query cell.
Response Mode	The method in which the server will respond: direct (standard out) or indirect (response file). NULL if this client is receiving direct responses, otherwise the pathname of the user-specified response file for this client. The default value is NULL.
Marker Size	The width in user units of the geometric squares used to mark pin and seed shape locations in responses. The default value is 0.25.
Filter Distance	The distance from the location point of a location-based query beyond which a port, device, or net shape is ignored. The default value is 0, which means the port or device coordinate must exactly match the location point and the edge of a net shape must pass through the location point. In normal use, this value should be set a larger value.
Filter Layers	A list of layer names and/or numbers. In NET LOCATION or PORT LOCATION queries, ports or net shapes not on one of these layers are ignored. This filter has no effect on any other queries. The default value is ALL, which represents all possible layers.

Table 15-12. Client Table [continued]

Entry	Description
Filter Devices	A list of device type numbers. In DEVICE LOCATION queries, devices whose types are not present on the list are ignored. The default value is ALL, which represents all possible device types. See section Device Tables for more details.
Filter Windows	A set of rectangular viewing windows beyond which location-based query responses are ignored. Setting filter windows can decrease response time of a command.
Filter Cull	The (x, y) dimensions specifying the minimum extent dimensions required for a polygon to be reported in results sets. If a polygon is not either wider than x, or taller than y, it is not reported.

Viewing, Query, and Query Instance Cells

A hierarchical LVS report contains a layout-to-source cell comparison for each corresponding hierarchical cell, or hcell. The hcell about which queries are currently being made is the query cell. In commands and Query Server responses, pathnames are always relative to the query cell. For example, if the query cell is ADDER2 and the command contains the net path X2/CLK.Then the net being specified is the net named CLK in the cell instance named X2 in the cell named ADDER2.

Geometric results are either returned in the coordinate system of the query cell or in the coordinate system of a cell containing an instance of the query cell. The cell in whose coordinate system the results are returned is called the viewing cell. If the viewing cell is not the same as the query cell, then it must contain an instance (possibly separated by several levels of hierarchy) of the query cell. One such instance is designated as the query instance and the geometric results from the query cell are mapped onto that instance in the viewing cell’s coordinate system.

In commands which require the specification of a geometric location using (x,y) coordinates, the coordinates are those of the viewing cell, even though the object

being specified is in the query instance. The coordinates are specified in user units.

In summary, when the viewing cell and query cell are distinct, pathnames are relative to the query cell but geometric coordinates are relative to the viewing cell.

To illustrate these concepts, consider the design in Figure 15-12. Top cell A contains instances, X1 and X2 of cell B and nets N3 and N4. Cell B also contains nets named N3 and N4. Net N3 in cell A connects nets N3 and N4 of instance X1 and net N3 of instance X3. Net N4 of cell A connects to net N4 of instance X2.

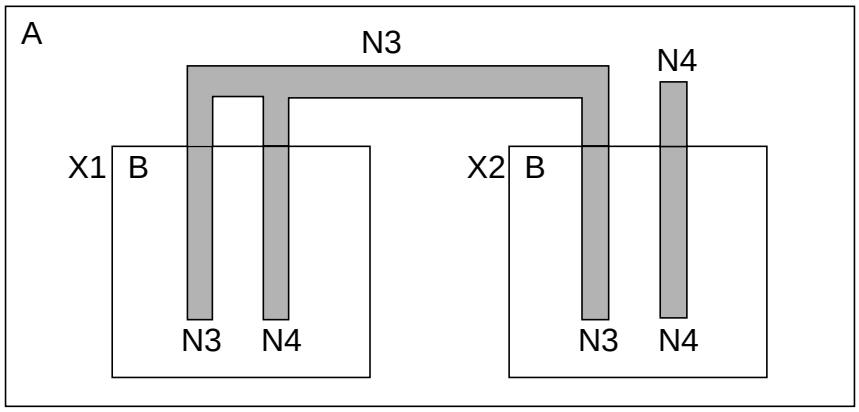


Figure 15-12. Top Cell A

Consider the query “NET SHAPES N3” in the following three examples.

Example 1: The viewing and query cells are both A (see Figure 15-13). In this case the user is viewing cell A and asking about objects relative to cell A. Thus, N3 in the query refers to net N3 of cell A. The results are mapped into cell A.

The Query Server will return the gray shapes shown below in the coordinate system of A:

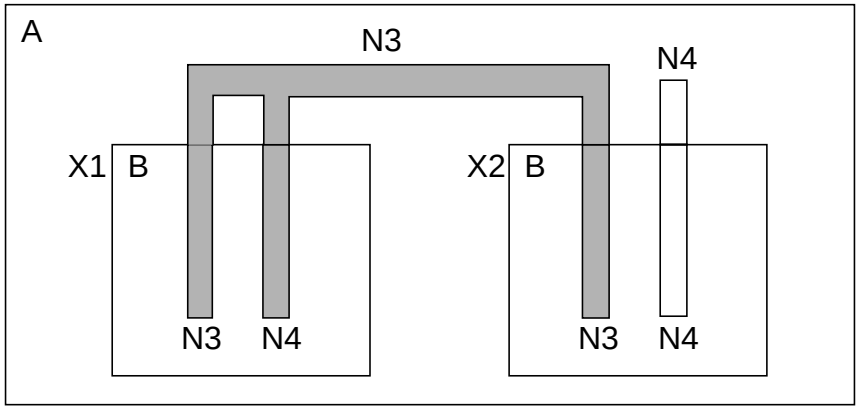


Figure 15-13. Viewing and Query Cells are both A

Example 2: The viewing cell is A, the query cell is B and the query instance is X2 (see Figure 15-14). In this case the user is viewing cell A but making queries relative to cell B and expects the results to be mapped on instance X2. Thus, N3 in the query refers to net N3 of cell B. The Query Server will return the gray shape shown below in the coordinate system of A:

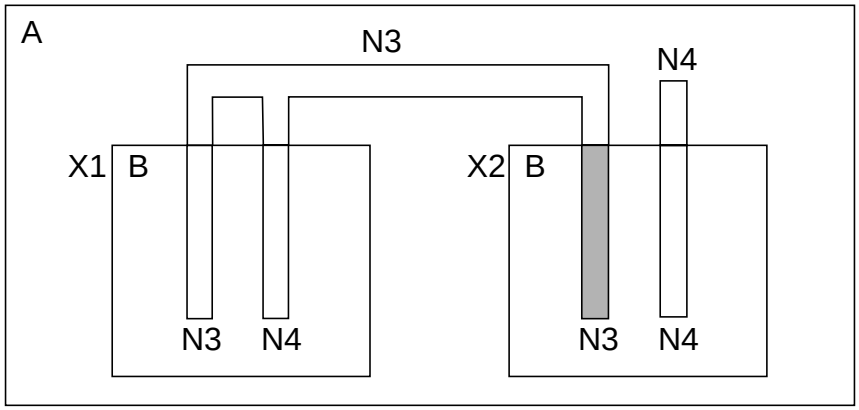


Figure 15-14. Viewing Cell is A, Query Cell is B, Query Instance is X2

Example 3: The query and viewing cells are both B (see Figure 15-15). In this case the user is viewing cell B and asking about objects relative to cell B. Thus, N3 in the query refers to net N3 of cell B. The results are to be mapped into cell B.

The Query Server will return the gray shape shown below in the coordinate system of B:

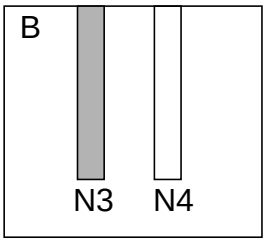


Figure 15-15. Viewing and Query Cells are both B

Server-Client Communication

The server communicates with the client application by reading standard in to receive commands and writing standard out to return acknowledgements. Direct responses are written to standard out immediately following the acknowledgment. Indirect responses are written to the current response file where the application can read it directly.

When a response file is established for a client ID, it is used for all responses until another response file is established or the client context's response mode is set to direct. It is the application's responsibility to delete these files when they are no longer needed. Each time a response is generated, the server opens the file, writes the response into the file, and closes the file. This has several implications. First, any previous contents of the file are overwritten by the new response. Second, after receiving the "OK" acknowledgment, the application can rename or delete the file if it wishes. The server will use the file again for the next response. The application can establish a new response file whenever it wishes. Thus it can have each response directed to a different response file or can have all responses directed to the same file. In the latter case, new responses will overwrite previous responses unless the application renames the file between responses.

Command Format

Commands are used by the applications to control the server. Blank lines and lines beginning with "#" are ignored and may be used in test scripts for documentation. All other lines are treated as command lines. Here are some examples (the "//"

comments are not part of the command and would not be allowed on the command line):

```
NET SHAPES X12/X45/CLOCK1
// app asks for geometries of a net relative to the current
// query cell
DEVICE INFO X95/X4/X312/C4
// app asks for information about a device relative to the
// current query cell
ACTIVATE 12
// app asks server to deactivate the current context
// and activate client id 12
```

Each command is a single string of ASCII text terminated by a return character. The command consists of one or more fixed command words followed by any additional information required by the command such as numbers or path names. The words chosen to represent a command are a compromise between brevity and human legibility. Later sections cover each command in detail. Although they are shown in upper case in this document, the server is not sensitive to the case of the command words. Case sensitivity of the arguments depends upon the rule file Layout Case and Source Case statements.

Acknowledgements

Each command between application and server is replied to by an acknowledgment. An acknowledgment is a single string of ASCII text beginning with three spaces and terminated by a carriage return character. A command which generates responses will generate a response if and only if its acknowledgment begins with “OK”.

Since some acknowledgement lines contain pathnames and other design information, there is no guaranteed limit on the length of an acknowledgement line. The application must be prepared to accept acknowledgements of arbitrary length and to reformat them for display if necessary. Here are some examples (the “//” comments are not part of the acknowledgment):

OK. // command succeeded

OK: 7 // command succeeded; returned value 7

NOK(1): There are no pins on layout net FOO. // command failed for given reason

ERROR(108): Client id 0 can never be disconnected.
// command failed for given reason

The four examples above illustrate the four types of acknowledgment. They are

- Simple success: This acknowledges that the command succeeded. It is always the string “OK.”. Any additional information generated by the command is returned in a response.

- Success with value: This acknowledges that the command succeeded and returns some information produced by the command. It always consists of the string “OK:” followed by the information. This form of acknowledgment is used with commands which generate a small amount of information which can fit on a single line. It is possible (but not currently done) that a command could given a success with value acknowledgment and also return information in a response.

- Failure: This acknowledges that the command was performed but failed to produce any of the requested information. In particular no response was generated. It is always of the form “ NOK(<num>): <reason>”, where <num> is a number in the range 1-99 and <reason> is a sentence explaining the reason for failure. The reason is formatted for human consumption and may be displayed to the human client. The <num> is provided so that it is easy for a computer program to identify the message without parsing the reason. This type of acknowledgment is only used for queries which request design information.

- Error: This acknowledgment is given when a problem has arisen which is more serious or which could have been avoided. It is always of the form “ERROR(<num>): <reason>”, where <num> is a number in the range 101-199 and <reason> is a sentence explaining the reason for the error. It may represent a failure of the environment (such as the inability to open or write to the response file) or misuse of the communication channel by the application (such as an unknown command or a command which is inappropriate but could have been avoided).

Response Format

A response contains the design information returned by the server in response to a query and is formatted as an ASCII DRC Results Database file. This is true even in direct response mode where the response is not written to a file. The response format allows for return of both textual and graphical information. The definition of the ASCII DRC Results Database is found in "ASCII and Binary DRC Results Databases" on page 14-11. The coordinate system used in a response is determined by the current context.

Responses are formatted as follows:

- Header line followed by one or more check sections. The header line contains a cell name followed by the precision of the design.
- Check sections beginning with a check title line containing the name of the check, which may contain embedded white space.
- Count line containing the number geometries currently in the check, the number of geometries originally in the check, the number of text lines in the check, and the date. The two geometry counts are always equal. However, response readers must ignore the original geometry count which is reserved for future expansion.
- Zero or more text lines, the number of lines having been given in the text line count field of the count line.
- Zero or more polygons, the number of polygons having been given in the current geometry count field of the count line. Each polygon consists of a polygon header line containing the character p, a polygon number (in responses the polygons are numbered sequentially from 1 upward), and the vertex count of the polygon.
- Vertex lines contain the vertices of the polygon in counterclockwise order, one vertex per line, each line containing the x followed by the y coordinates of the vertex. Note that the edge cluster form of geometries is not currently used in responses.

- End of response check title and count lines. Because it is always present and always the same, the END OF RESPONSE response is omitted from the command descriptions later in this document.

Below is a visual representation of the response format. The angle-bracketed items on each line are separated by spaces, and with the exceptions of the <check_name> and <text_line> items contain no white space.

```
<response_name> <precision>// leader line
<check_name>// first check title line
<curr_count> <orig_count> <text_count> <date>// count line
<text_line>// first text line
…  // more text lines
p <poly_number> <vertex_count>// polygon header line
<x> <y>// first vertex line
…  // more vertex lines
…  // more polygons
…  // more check sections
END OF RESPONSE// end check title line
0 0 0 <date>// end check count line
```

Device Tables

The device table consists of an indexed set of entries containing information about each device type in the design. There is only one such table for the entire design and it is independent of context. The indices range from 0 to *n*-1, where there are *n* Device specification statements in the rule file. These indices are called device type numbers. Each entry contains the following information:

device type	pin names
element name	pin layers
model name	pin swap group numbers
netlist element name	auxiliary layers
netlist model name	property names

The device type is one of the following constants: “DIODE”, “RESISTOR”, “CAPACITOR”, “MOS”, “BIPOLAR”, “USER”.

Responses that describe devices or device pins use the indices into the device table to reduce the size of the response. The DEVICE TABLE command allows the user to view the device table to interpret the device type numbers. For example in a response giving the device pins on a net, each pin is described by the following five items:

- <device_path> path name ending with device instance name
- <device_type> the 0-based index of the device entry in the device table
- <pin_index> the 0-based index of the pin in the pin lists of the device entry
- <x> the x coordinate of the pin in the current view cell
- <y> the y coordinate of the pin in the current view cell

Additional information about the device, such as the values of its properties or the nets to which its other pins are connected, can be obtained by using the DEVICE INFO command.Pin and property names are lowercase. Layer names have the same case as the rule file.

Net Names

Nets in a command may be referred to by name or by number. Nets returned in acknowledgements or responses are always returned by name if possible, otherwise by number.

PHDB and XDB Only Modes

The Query Server normally provides information gathered during circuit extraction (PHDB) and LVS comparison (XDB). If one or the other database is missing, a subset of commands which require only information from the available database will continue to work. So, for example, the NET SHAPES command will work even if no LVS comparison has occurred (and no XDB database has been written). The DEVICE SOURCE command will work for Spice to Spice comparisons where no PHDB has been written.

Flat Database Mode

Flat LVS runs can generate an SVDB database which can be read by the Query Server. The SVDB for a flat LVS run contains no cell or placement information since all layout and source information is flattened during the LVS run. All devices and nets appear to be in a single top-level cell. Commands which involve placement arguments or involve multiple cells are not available from the Query Server in FLAT mode.

Limitations

The Query Server has the following limitations:

- It has no knowledge of the LVS discrepancy report and thus cannot respond to discrepancy number based queries.
- It will not retain the type of state information which allows it to produce sequencing or scanning behavior. For example, there is no “return the next device on this net” query. All such sequencing must be performed by the application. (The browse operations help with this type of operation).
- No progress information is generated as the response to a query is being generated.
- Filter Devices only affects the DEVICE NAMES and DEVICE LOCATION queries. It could also be made to affect NET PINS by returning only pins of device types on the filter list.

Commands and Queries

Tables 15-13 through 15-21 describe the commands the Query Server accepts. Unless specified otherwise, commands and their arguments pertain to the current query cell, and pathnames are relative to the current query cell. Nets in a command can be referred to by name or by number. Nets returned in responses are always returned by name if possible, otherwise by number. Responses are shown for commands that issue responses. If a command does not issue a response, none is indicated in the command descriptions.

Communication and Control Commands

The commands described in Table 15-13 are used for communication and control between the Query Server and the client.

Table 15-13. Communication and Control Commands

Command	**Acknowledgments**
Description	
ACTIVATE *client_id*	OK. ERROR(106), ERROR(107)
Description: Deactivates the current client and activates the given client and its current context. Notes: In case of error, the current client remains active.	
CONNECT	OK: *new_client_id*
Description: Deactivates the current client context, creates an unused client ID number, initializes the client table entries for that context to their default values, makes the new client context the active context, and returns the new client ID. Notes: a) Client IDs are non-negative integers. b) There is no reasonable limit to the number of client contexts which may be defined at any given time, but only one may be the current active client context. c) When the server is initiated it creates client id zero and activates it. This client context can never be deleted. It is suggested it be reserved for use by the application acting on its own behalf when multiple client contexts are being maintained. For example some application implementations may wish to cache information such as the device table so they can use it directly without requesting it of the server for each operation.	

Table 15-13. Communication and Control Commands [continued]

Command	**Acknowledgments**
Description	
DISCONNECT *client_id*	OK. ERROR(106), ERROR(107), ERROR(108), ERROR(109)
Description: Deletes the stored current context about an *inactive* client, which makes the client id undefined for the rest of the Query Server session.	
ECHO	OK: ECHO {ON | OFF}
Description: Intended only for use in testing the Query Server, this toggles the echo state. Normally the server does not echo standard in to standard out. However in testing the server with a script, it is convenient that it do this so that the output contains a complete transcript of the session including comments and blank lines in the input. When the echo state is ON, all input is echoed to output. The echo state is initially OFF. Notes: a) The acknowledgement shows the new echo state. b) The echo state applies to all clients.	

Table 15-13. Communication and Control Commands [continued]

Command	Acknowledgments
Description	
HELP COMMANDS	OK. Error(1)
Description: Returns a list of the valid Query Server commands and arguments (including the Calibre Connectivity Interface). Each command is listed together with its arguments on a separate line. Alternate forms are listed on separate lines. The list is intended to serve as a crib sheet for people already familiar with the command set and its function. It does not attempt to explain the commands. Response: Help_Commands <precision> // “Help_Commands” and precision Commands: // “Commands:” 0 0 <n> <date> // n lines of text follow (no geometries) protocol version // first command description help commands // next command description . . . // additional command descriptions device info <layout_device_path> // last command description	

Table 15-13. Communication and Control Commands [continued]

Command	Acknowledgments
Description	
PROTOCOL VERSION	OK: major_version,minor_version
Description: Returns the communication protocol version used by the Query Server. This allows socket clients to verify that they are dealing with a protocol they understand. Protocol here refers to the command set, and the actions it invokes in the system including the content of acknowledgment and responses. Notes: a) The protocol version consists of two integers separated by a period. b) When a protocol change is made, if the new protocol is a strict superset of the old, the minor_version number is incremented but the major_version number is retained. Otherwise the major_version is incremented and the minor_version is set to zero. Thus a client which understands protocol k.m, can use any protocol k.n where n greater than or equal m, but could not use protocol j.l where j is unequal to k. (For an exception, see the next note.) c) Note that changes in the text explanations returned in failure and error acknowledgements are considered to produce a superset of the previous protocol. The would cause the minor_version to be incremented, but not the major. Any client which is dependent on the content of these text explanations would not be compatible with the new minor version.	
RESPONSE DIRECT	OK.
Description: Sets the Query Server response mode to direct (standard out). If a response file exists at the time this command is processed, it is abandoned but not deleted. Notes: a) If there was a response file established at the time this command is received, the server abandons it. It is the application's responsibility to delete it.	

Table 15-13. Communication and Control Commands [continued]

Command	Acknowledgments
Description	
RESPONSE [MULTIPLE] **FILE** ***response_file_path***	OK. ERROR(101), ERROR(102)
Description: Establishes file as the response file for the current client context. If the MULTIPLE keyword is used, multiple sequential responses are all appended to the file in DRC Results format. Normal response header and trailer information is not issued for subsequent commands. The MULTIPLE setting is useful for viewing responses to several commands simultaneously. Response: Response_File <precision> // “Response_File” and design precision File: <response_file_path> // “File:” followed by the response file path name 0 0 0 <date> // no geometries and no lines of text follow Notes: a) There is no information content in the response. However it may be read by the client as a confirmation that the response mechanism has been properly set up. Of course the client should first check that the acknowledgment was “OK.”. b) The response file may already exist in which case the response written into it at the front of the file, overwriting any exiting content. c) If there was a current response file established at the time this command is received, the server abandons it. It is the client’s responsibility to delete it. d) If this command fails with an error, the response mode is set to direct as if the command had been RESPONSE DIRECT. This is because any current response file may no longer be appropriate. That is, the client may have deleted or renamed it as well as the directory containing it before issuing this command. The client must watch for this error since it changes the nature of acknowledgment channel traffic.	

Table 15-13. Communication and Control Commands [continued]

Command	Acknowledgments
Description	
e) Once the file_path_name has been validated as descibed above, the server will attempt to use for all following commands until another RESPONSE DIRECT or RESPONSE FILE command is issued. If for whatever reason the server is no longer able to write to this file path, each command which attempts to generate a response will fail with an “ERROR(101): File <response_file_path> could not be opened for writing.” error. That is, once the RESPONSE FILE command has accepted the path, the only way to change it is with a RESPONSE DIRECT or RESPONSE FILE command.	
TERMINATE	OK: Terminating.
Description: Causes Query Server to terminate and exit.	

Parameter Commands

The commands described in Table 15-14 are used to set and inquire about various parameters and settings that affect the results of queries.

Table 15-14. Parameter Commands

Command	**Acknowledgments**
Description	
CONTEXT *view_cell* [*query_instance_path*]	OK: Query cell: *query_cell* ERROR(103), ERROR(104)
Description: Given a cell name and optionally a cell instance path relative to that cell, change the client table context entries for the current client. That is, establish the view cell, query cell and query instance path if any. Any existing filter windows are deleted. Notes: a) If OK is returned, the following changes are made to the client table for the current client: the viewing cell entry is set to view_cell. If query_instance_path was absent, the query cell entry is set to view_cell and the query instance entry is set to NULL, otherwise the query instance entry is set to query_instance_path and the query cell entry is set to the name of the cell of which that is an instance. b) The acknowledgement returns the name of the query cell so that the client may confirm the query_instance_path lead to the desired type of cell, or in the case where a human user issued the query path through an editor, the editor easily determine the newly established query cell. c) The instance path must be of the form Xn 1 /Xn 2 /.../Xn k , where each n i is an integer. d) If an error is returned, the current context remains unchanged.	

Table 15-14. Parameter Commands [continued]

Command	Acknowledgments
Description	
FILTER CULL *x y*	OK. ERROR(117)
Description: Given a minimum width X, and a minimum height Y, this command causes result shapes which do not exceed X in width or Y in height to be filtered from the results presentation. Notes: a) Marker squares are never filtered. b) FILTER CULL 0 0 eliminates this filtering (default behavior).	
FILTER DEVICES {ALL | *device_list*} **or** **FILTER DEVICENAMES** ***element_name* [*(model_name)*]...**	OK. ERROR(115), ERROR(126)
Description: Given the keyword ALL or a list of device type numbers or names, these commands set the filter device list for the current client to those values. The new list will be effective beginning with the next response. See section “Device Tables” for more details. Notes: a) The device_list> consists of a white space separated list of device type numbers. Valid numbers range from 0 to one less than the number DEVICE statements in the rule file. The meaning of a given number can be determined from the device table. b) For FILTER DEVICENAMES, all devices that match the element_name (model_name) combination are added to the filtered list. c) If an invalid list is given, the former list is retained. d) The value ALL represents all possible device type numbers, in effect causing no filter-ing. e) The default value for new clients is ALL.	

Table 15-14. Parameter Commands [continued]

Command	Acknowledgments
Description	
FILTER DISTANCE *distance*	OK. ERROR(113)
Description: Given a non-negative number, this command sets the filter distance for the current client to that value. The new distance will be effective beginning with the next response. Notes: a) The filter distance is expressed in user units and may be represented as a decimal integer or fraction. It must be non-negative. It is not checked for a reasonable maximum size, but large values may significantly delay the response. b) If an invalid distance is given, the former value is retained. c) The default value for new clients is 0. d) Large values of the filter distance cause the Query Server to search more of the design space. The prudent client will use the smallest reasonable value. e) In general, the filter distance may be set to 0 as long as the user is aware that the hierarchy is searched only for placements which overlap the search point directly (not placements which are nearby).	

Table 15-14. Parameter Commands [continued]

Command	Acknowledgments
Description	
FILTER LAYERS {ALL | *layer_list}*	OK. ERROR(114)
Description: Given the keyword ALL or a list of layer names and numbers, this command sets the filter layer list for the current client to those values. The new list will be effective beginning with the next response. Notes: a) The layer_list consists of a white space separated list of layer names or numbers as they are used in the rule file. b) If an invalid list is given, the former list is retained. c) The value ALL represents all possible layers, in effect causing no filtering. d) The default value for new clients is ALL. e) In any given command, layers in the list which are not relevant to that command will be ignored. For example, the NET LOCATION x y command will ignore any non-connectivity layers. f) If a layer has a name then the layer number can not be used.	

Table 15-14. Parameter Commands [continued]

Command	**Acknowledgments**
Description	
FILTER WINDOW {INCLUDE *x1 y1 x2 y2* | EXCLUDE *x1 y1 x2 y2* | NONE }	OK. ERROR(117)
Description: Given a set of rectangular coordinates, these commands manipulate a set of viewing windows which apply to coordinate based queries. Notes: a) FILTER WINDOW INCLUDE causes results from the specified area to be included in results. b) FILTER WINDOW EXCLUDE causes results from the specified area to be excluded from results. c) Successive calls to FILTER WINDOW INCLUDE and FILTER WINDOW EXCLUDE build up complex viewing area consisting of multiple rectangular areas that are either included or excluded. d) FILTER WINDOW NONE clears all windows and returns to the default state (results from the entire set of coordinates are presented). e) The SET CONTEXT command eliminates all FILTER WINDOWS (same as FILTER WINDOW NONE).	
MARKER SIZE *size*	OK. ERROR(110)
Description: Given a marker size, this command sets the marker size for the current client to that value. The new size will be effective beginning with the next response.	
MAXIMUM VERTEX COUNT *count*	OK. ERROR(119)
Description: Given a value, this command sets the maximum vertex count for results for all clients. The new size will be effective beginning with the next response.	

Table 15-14. Parameter Commands [continued]

Command	Acknowledgments
Description	
STATUS [CLIENT | VIEW CELL | QUERY INSTANCE | QUERY CELL | RESPONSE MODE | MARKER SIZE | MAXIMUM VERTEX COUNT | FILTER DISTANCE | FILTER LAYERS | FILTER DEVICES | FILTER CULL | FILTER WINDOWS]	If no entry_name was given, the acknowledgement is “OK.” and information is returned in a response. If an entry_name was supplied, the acknowledgement is of the form “OK: entry_name: entry_value” where entry value is the value of the entry for the current client. Note the colon separating the entry_name from the entry_value.
Description: Responds with the status of the current client for the specified argument, whether it be a numeric value or a text string. If no argument is specified, responds with the values for all the arguments in the client’s current context. Response: If an entry_name was given, there is no response. Otherwise the response is the following: Status <precision> // “Status” followed by design precision Entries: // “Entries:” 0 0 <n> <date> // n lines of text follow (no geometries) <entry_name_1>: <entry_value_1> // first entry name and value . . . // intermediate names and values <entry_name_n>: <entry_value_n> // last name an value Notes: a) The <entry_name_n>’s are the same text strings as listed above in the Command section but in lower case. Note that some are a single word and others are multiple words. The value is separated from the last word of the name by a colon and white space. b) If the response mode is currently direct, the value for response mode will be “(direct)”.	

Table 15-14. Parameter Commands [continued]

Command	Acknowledgments
Description c) If the view and query cells are the same, the value of query instance will be “(null)”. d) If the filter layers list contains all layers, the value of filter layers will be “(all)”. e) If the filter devices list contains all device types, the value of filter devices will be“(all)”. f) The order in which the entry_name_n: entry_value_n pairs appear in the response is not guaranteed. Program clients who wish to parse this response should do so on the bases of the value of entry_name_n, not on an assumed position of a given item in the list. (Such clients will have a longer and more graceful life!) g) MAXIMUM VERTEX is global, not client-specific	

Cell Query Commands

The commands described in Table 15-15 are used to query cell information.

Table 15-15. Cell Query Commands

Command	**Acknowledgment**
Description	
CELLS CORRESPONDING	OK. NOK(21), ERROR(102)
Description: Returns a list of corresponding hcell pairs as determined by LVS. Each pair consists of a layout cell name and the corresponding source cell name. (Also see the commands CELL CORRESPONDING SOURCE and CELL CORRESPONDING LAYOUT.) Response: Corresponding_Cells <precision> // “Corresponding_Cells” and precision Correspondences: // “Correspondences:” 0 0 <n> <date> // n lines of text follow (no geometries) <layout_cell_name_1> <source_cell_name_1> ... <layout_cell_name_n> <source_cell_name_n> Notes: a) Cell are listed in top to bottom order. That is, if cell A directly or indirectly contains an instance of cell B, then cell A will appear earlier in the list than cell B. Thus the first cell in the list is the top cell of the design. b) This list contains only the cells which correspond between layout and source. The complete list of layout cells or source cells can be obtained using the CELLS LAYOUT or CELLS SOURCE commands. c) Due to the match by name option for hierarchical lvs, this list may be more comprehensive than the “hcells” list given to LVS. That is, it will also include all cells matched by name.	

Table 15-15. Cell Query Commands

Command	**Acknowledgment**
Description	
CELL CORRESPONDING LAYOUT ***source_cell_name***	OK: *corresponding_layout_cell_name* NOK(21), NOK(26)
Description: Given the name of a source cell, returns the name of the corresponding layout cell. The correspondence is the same as that returned by the CELLS CORRESPONDING command. That command returns all cell correspondences as a response. This returns only the single layout cell corresponding to a single source cell as an acknowledgement. This is more efficient for the client who is only interested in a single correspondence. (See also the commands CELLS CORRESPONDING and CELL CORRESPONDING SOURCE.)	
CELL CORRESPONDING SOURCE ***layout_cell_name***	OK: *corresponding_source_cell_name* NOK(2), NOK(21)
Description: Given the name of a layout cell, returns the name of the corresponding source cell. The correspondence is the same as that returned by the CELLS CORRESPONDING command. That command returns all cell correspondences as a response. This returns only the single source cell corresponding to a single layout cell as an acknowledgement. This is more efficient for the client who is only interested in a single correspondence. (See also the commands CELLS CORRESPONDING and CELL CORRESPONDING LAYOUT.)	

Table 15-15. Cell Query Commands

Command	**Acknowledgment**
Description	
CELLS LAYOUT	OK. ERROR(101), ERROR(102)
Description: Responds with a list of all layout cell names in a design. Response: Cells_Layout <precision> // “Cells_Layout” followed by design precision Cells: // “Cells:” 0 0 <n> <date> // n lines of text follow (no geometries) <cell_name_1> // top cell in layout design . . . // intermediate cells in layout design <cell_name_n> // last cell in layout design Notes: a) Cell are listed in top to bottom order. That is, if cell A contains an instance of cell B, then cell A will appear earlier in the list than cell B. Thus the first cell in the list is the top cell of the design. b) The list of cell names contains all cells in the layout, not just those which correspond to source cells. c) The list returned can be used by the client to set up a scan over all cells in the layout design.	

Table 15-15. Cell Query Commands

Command	**Acknowledgment**
Description	
CELLS SOURCE	OK. NOK(21), ERROR(101), ERROR(102)
Description: Responds with a list of all source cell names in a design. Cells are listed top-to-bottom. Response: Cells_Source <precision> // “Cells_Source” followed by design precision Cells: // “Cells:” 0 0 <n> <date> // n lines of text follow (no geometries) <cell_name_1> // top cell in source design . . . // intermediate cells in source design <cell_name_n> // last cell in source design Notes: a) Cell are listed in top to bottom order. That is, if cell A contains an instance of cell B, then cell A will appear earlier in the list than cell B. Thus the first cell in the list is the top cell of the design. b) The list of cell names contains all cells in the source, not just those which correspond to layout cells. c) The list returned can be used by the client to set up a scan over all cells in the source design.	
CELL TOP	OK: *top_cell_name*
Description: Returns the name of the top-level cell.	

Browse Pseudo or Deviceless Cells

The commands described in Browse Pseudo or Deviceless Cells Commands allow for simpler browsing functionality within the Query Server by peeking through certain uninteresting cells (pseudo cells and/or cells that do not contain devices--like vias).

Table 15-16. Browse Pseudo or Deviceless Cells Commands

Command	**Acknowledgments**
Description	
BROWSE PSEUDO CELLS { NO | YES}	OK.

Description: Causes the Query Server's BROWSE commands to peek/not peek through artificially-created pseudo cells. The Calibre hierarchical database causes creation of various pseudo cells with names like "ICV_nn" (where nn is a number). By default, pseudo cells are peeked through for the purposes of BROWSE commands (BROWSE PSEUDO CELLS NO).
Notes:
By default, pseudo cells are peeked through by the BROWSE commands.
Ouput with NO specified (pseudo cells are transparent):
`NET BROWSE DEVICES GND X34`
`Net_Browse_Devices 1000`
`Devices:`
`0 0 6 Dec 11 10:27:34 2001`
`X34/X0/M6 1`
`X34/X0/M7 1`
`X34/X0/M8 1`
`X34/X1/M6 1`
`X34/X1/M7 1`
`X34/X1/M8 1`
`Placements:`
`0 0 0 Dec 11 10:27:34 2001`
`END OF RESPONSE`
`0 0 0 Dec 11 10:27:34 2001`
`OK.`	

Table 15-16. Browse Pseudo or Deviceless Cells Commands

Command	Acknowledgments
Description	

Output with YES specified:
`NET BROWSE DEVICES GND X34`
`Net_Browse_Devices 1000`
`Devices:`
`0 0 0 Dec 11 10:27:34 2001`
`Placements:`
`0 0 2 Dec 11 10:27:34 2001`
`X34/X0 ICV_22`
`X34/X1 ICV_22`
`END OF RESPONSE`
`0 0 0 Dec 11 10:27:34 2001`
`OK.`
Here ICV_22 is a pseudo cell.	

Table 15-16. Browse Pseudo or Deviceless Cells Commands

Command	**Acknowledgments**
Description	
BROWSE DEVICELESS CELLS { NO | YES}}	OK.

Description: Causes the Query Server's BROWSE commands to peek/not peek through cells that do not contain any devices hierarchically (via cells, for example).
Notes:
By default, deviceless cells are peeked through by the BROWSE commands.
Output with NO specified (deviceless cells are transparent):
`NET BROWSE SHAPES gnd x649`
`Net_Browse_Shapes 1000`
`M1`
`1 1 0 Dec 11 10:27:34 2001`
`p 1 8`
`364900 13800`
`365700 13800`
`365700 19700`
`368800 19700`
`368800 13800`
`369600 13800`
`369600 21900`
`364900 21900`
`VIA1 <<< (shapes on layer VIA1 come from cell VIA12)`
`2 2 0 Dec 11 11:22:27 2001`
`p 1 4`
`364850 13750`
`365750 13750`
`365750 19750`
`364850 19750`
`p 2 4`
`365300 20350`
`369200 20350`
`369200 21250`
`365300 21250`	

Table 15-16. Browse Pseudo or Deviceless Cells Commands

Command	Acknowledgments
Description	
Placements: 0 0 1 Dec 11 11:22:27 2001 X649/X4 NAND END OF RESPONSE 0 0 0 Dec 11 11:22:27 2001 OK. Output with YES specified: NET BROWSE SHAPES gnd x649 Net_Browse_Shapes 1000 M1 1 1 0 Dec 11 10:27:34 2001 p 1 8 364900 13800 365700 13800 365700 19700 368800 19700 368800 13800 369600 13800 369600 21900 364900 21900 Placements: 0 0 3 Dec 11 10:27:34 2001 X649/X2 VIA12 X649/X3 VIA12 X649/X4 NAND END OF RESPONSE 0 0 0 Dec 11 10:27:34 2001 OK. VIA12 is a deviceless cell.	

Cell Query Placement Commands

The commands described in Table 15-17 are used to query cell placement information.

Table 15-17. Cell Query Placement Commands

Command	**Acknowledgment**
Description	
PLACEMENT BROWSE DEVICES ***placement_name***	OK. NOK(18), ERROR(32), ERROR(33)
Description: List devices within a particular cell placement. Response: Placement_Browse_Devices <precision> // Placement_Browse_Devices and design precision 0 0 n <date_stamp> // 0 0 <number_of_devices> date/time stamp <device_name> ... // <device_name> of <placement_cell> ...additional devices	
PLACEMENT BROWSE NETS ***placement_name***	OK. NOK(16), ERROR(32), ERROR(33)
Description: List nets within a particular cell placement. Response: Placement_Browse_Nets <precision> // Placement_Browse_Nets and design precision 0 0 n <date_stamp> // 0 0 <number_of_nets> date/time stamp <net_name> ... // <net_name> of <placement_cell> ...additional nets	

Table 15-17. Cell Query Placement Commands [continued]

Command	Acknowledgment
Description	
PLACEMENT BROWSE PLACEMENTS ***placement_name***	OK. NOK(22), ERROR(32), ERROR(33)

Description: List cell placements within a particular cell placement.
Response:
Placement_Browse_Placements <precision>
// Placement_Browse_Placements and design precision
<placement_cell>
// The following placements are of cell <placement_cell>
0 0 n <date_stamp>
// 0 0 <number_of_placements> date/time stamp
<placement_name> ...
// <placement_name> of <placement_cell>
...additional cells
...additional placements	

PLACEMENT BROWSE PORTS ***placement_name***	OK. NOK(10), ERROR(32), ERROR(33)
Description: List ports within a particular cell placement. Response: Placement_Browse_Ports <precision> // Placement_Browse_Ports and design precision 0 0 n <date_stamp> // 0 0 <number_of_ports> date/time stamp <port_name> ... // <port_name> of <placement_cell> ...additional ports	

Table 15-17. Cell Query Placement Commands [continued]

Command	Acknowledgment
Description	
PLACEMENT INFO ***placement_name***	OK. NOK(32), ERROR(102), ERROR(104)
Description: Given the name of a cell instance placement in the current query cell, returns the name of the cell and its extent. If a device instance placement name is provided, this has the same effect as using the DEVICE INFO command. Response: Placement_Info 1000 // “Placement_Info” and design precision cell_name // placement_name is a placement of cell_name 1 1 0 <date> // 1 polygon in response p 1 4 ... // 4 sets of vertices marking the extent of the placement	
PLACEMENT LAYOUT ***source_placement***	OK. NOK(2), NOK(8), NOK(21)
Description: Given the name of a source placement, returns the name of the corresponding layout placement. Since device and placement correspondence are maintained in the same data structures, this command is simply an alias for the Device Layout command.	

Table 15-17. Cell Query Placement Commands [continued]

Command	Acknowledgment
Description	
PLACEMENT LOCATION { *cell_name x y ll_x ll_y ur_x ur_y* | *m n* [FLAT] }	OK: *placement_path1*… *placement_pathn* NOK(24)
Description: Given a placement's cell name, origin, and lower-left and upper-right coordinates, returns the path name of each matching placement in the layout. See the notes for the matching criteria. The first form of this command is expected to be used primarily by clients which are layout editors and have knowledge of the origins and extents of cell placements. It is necessary because, given a placement of a cell in its data base, an editor cannot on its own determine the layout path by which that placement is known to the Query Server. This is because it cannot know the numbering system used in the layout from which the Query Server works and because that layout may have had additional cells and placement added to the design by hierarchical injection. A second form of the command is expected to be used directly by end users. It returns all placements of any cell placed under a specified point with coordinates *m n*. Hierarchy is descended and all placements at any point in the hierarchy are reported if the FLAT option is included. Filter: FILTER DISTANCE Response: Form 1: None Form 2: Placement_Location 1000 // "Placement_Location" and design precision placement_name cell_name // placement_name of cell cell_name ... Notes: a) The *cell_name* is the layout name of the cell whose placement is being sought. b) The point *x y* is the origin of the placement in viewing cell coordinates.	

Table 15-17. Cell Query Placement Commands [continued]

Command	Acknowledgment
Description	
c) The points *ll_x ll_y ur_x ur_y* are the lower left and upper right corners of the placement extent. d) A placement is selected by the first form of this command if it meets the following criteria: - It is a placement of cell *cell_name* in the layout hierarchy in or below the current query cell. - The rectangular extent of the placement in the layout database overlaps the rectangular extent given in the command. - The origin of the placement in viewing cell coordinates is within the current filter distance from the point specified by *x y*. e) The layout path names of each placement meeting the above criteria are returned in the acknowledgement. See the next note for the ordering. f) The placement extents known to the Query Server are based on the actual geometries used for the placement when the hierarchical data base was created from the original GDSII file. For this reason they may differ from the extent known to the editor. The matching placements are returned in order of decreasing overlap between the hierarchical data base extent and the editor extent. That is, the first placement path returned is the most likely to be the desired placement because it has the greatest overlap with the editor's placement extent specified as *ll_x ll_y ur_x ur_y* in the command.	

Table 15-17. Cell Query Placement Commands [continued]

Command	**Acknowledgment**
Description	
PLACEMENT NAMES [FLAT]	OK. NOK(22), NOK(23), NOK(33) ERROR(101), ERROR(102)
Description: Responds with a list of all cell placement names that appear in the current query cell. If FLAT is specified, all cell placements in the flattened query cell are reported. Response: Placement_Names <precision> // “Placement_Names” and design precision Placements: // “Placements:” 0 0 <n> <date> // n lines of text follow <placement_name_1> // first placement name in current context . . . // intermediate placement names in current context <placement_name_n> // last placement name in current context Notes: a) If the optional command word FLAT is used, all cell placements in the flattened query cell are reported. Otherwise only cell placements appearing directly in the query cell are reported. b) The order in which the placements are listed is arbitrary.	
PLACEMENT SOURCE ***layout_placement***	OK. NOK(2), NOK(7), NOK(21)
Description: Given the name of a layout placement, returns the name of the corresponding source placement. Since device and placement correspondence are maintained in the same data structures, this command is simply an alias for the Device Source command.	

Table 15-17. Cell Query Placement Commands [continued]

Command	Acknowledgment
Description	
PLACEMENT TRANSFORM ***placement_name***	OK: x-offset y-offset reflection rotation NOK(32)
Description: Given the name of a placement in the current query cell, returns the transform information (x offset, y offset, reflection, rotation) about the placement. Offsets are in database units, reflection is 0 for no reflection and 1 for reflection, rotation is in degrees.	
PLACEMENT VALID {**SOURCE** | **LAYOUT**} ***device_path***	OK: *device_path* NOK(7), NOK(8), NOK(9)
Description: Returns with a message stating whether the Query Server recognizes the path name to the placement. This is the same as the Device Valid command.	
PLACEMENTS OF ***cell_name*** [FLAT]	OK. NOK(22), NOK(23), NOK(33) ERROR(101), ERROR(102)
Description: Responds with the placements of the specified *cell_name*. If FLAT is specified, all placements of *cell_name* in the flattened query cell are reported. Otherwise, only placements that appear in the query cell are reported. The order in which the placements are listed is arbitrary. Response: Placements_Of <precision> // “Placements_Of” and design precision Placements: // “Placements:” 0 0 <n> <date> // n lines of text follow <placement_name_1> // first placement name rooted in current context . . . // intermediate names rooted in current context <placement_name_n> // last placement name rooted in current context Notes: a) If the optional command word FLAT is used, all placements of in the flattened query cell are reported. If FLAT is omitted, only placements appearing directly in the query cell are reported. b) The order in which the placements are listed is arbitrary.	

Query Port Commands

The commands described in Table 15-18 are used to query port information.

Table 15-18. Query Port Commands

Command	**Acknowledgments**
Description	
PORT INFO *port_name*	OK. NOK(11), ERROR(101), ERROR(102)
Description: Given the name of a port in the current query cell, returns the name of the net to which the port is attached, the signal direction of the port, and for each location associated with the port, a marker square centered at that location on the appropriate layer. Response: Port_Info <precision> // “Port_Info” and design precision Port: <port_name> // constant “Port:” followed by port name 0 0 2 <date> // two lines of text follow <net_name> // net attached to port <signal_direction> // signal direction number: 0 = in, 1 = out, 2 = in/out Layer_name_1 // first layer on which port has a presence k k 0 <date> // k squares on this layer p 1 4 // first square on this layer . . . // the four vertices of the first square . . . // remaining k-1 squares on this layer . . . // remaining layers on which port has a presence. Notes: a) The size of the marker squares is determined by the current value of the MARKER SIZE parameter.	

Table 15-18. Query Port Commands [continued]

Command	Acknowledgments
Description	
PORT LOCATION *x y*	OK: *port_name* NOK(13), ERROR(116)
Description: Given a location in the current viewing cell coordinate system, returns the pathname of the closest port in the current query instance to that location. Filter: FILTER LAYERS Notes: a) Ports of cells with placements in the query cell are ignored. That is, only ports of the query cell itself are examined. b) Ports which are not on a layer in the current client's filter layer list are ignored. c) If two or more ports are equidistant from the location, one is arbitrarily selected and the others are ignored.	

Table 15-18. Query Port Commands [continued]

Command	Acknowledgments
Description	
PORT NAMES [FLAT]	OK. NOK(10), NOK(15), NOK(33) ERROR(101), ERROR(102)
Description: Returns a list of all port names in the current (optionally flattened) query cell. Response: Port_Names <precision> // “Port_Names” and design precision Ports: // “Ports:” 0 0 <n> <date> // n lines of text follow <port_name_1> // first port name in current context . . . // intermediate port names in current context <port_name_n> // last port name in current context Notes: a) If the optional command word FLAT is used, all ports in the flattened query cell are re-ported. That is, ports of cells placed in the query cell at any level are reported with a query cell based path name leading to the cell instance containing the port. If FLAT is omitted, only ports appearing directly in the query cell are reported. b) The ports are listed in alphabetically increasing order. c) Ports with more than one location are listed only once.	

Table 15-18. Query Port Commands [continued]

Command	Acknowledgments
Description	
PORT TEXT MAP [INVALID]	OK. NOK(47)
Description: Generates a list of port names corresponding to Calibre LVS generated net numbers. Response: Port_Text_Map <precision> // “Port_Text_Map” and design precision Port Texts: 0 0 <n> <date> // n lines of text follow <port_text> <net_number> // port_text is the name of a port connected to net net_number Notes: a) Certain texts are not used for Spice netlist net names by Calibre. These texts are not included in the map. b) The INVALID option causes only those net names that are invalid for spice netlisting by Calibre to be shown.	

Query Net Commands

The commands described in Table 15-19 are used to query net information.

Table 15-19. Query Net Commands

Command	**Acknowledgments**
Description	
NET BROWSE DEVICES ***layout_net_path*** [*placement_name*]	OK. NOK(5), NOK(32), NOK(33), NOK(34), NOK(35)
Description: Given a layout net path, return devices on that net which are in a particular placement (the current query context by default). In addition, indicate placements which contain additional devices on that net down the hierarchy. Filter: FILTER DEVICES Response: Net_Browse_Devices <precision> // “Net_Browse_Devices” and design precision Devices: // “Devices:” 0 0 k <date> // 0 0 number_of_devices datestamp <device_name_1> <device_index_1> // name of the device, index into the device table ... additional devices ... // additional names and indices Placements: // “Placements” 0 0 k <date> // 0 0 number_of_placements datestamp <placement_name_1><cell_name> // placements containing additional devices. ... additional placements ... // additional placements. Notes: a) layout_net_path is traced upward to its highest hierarchical representative before browsing occurs.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET BROWSE NETS ***layout_net_path*** [*placement_name*]	OK. NOK(5), NOK(32), NOK(33), NOK(35)
Description: Filter: FILTER LAYERS Response: Net_Browse_Nets <precision> // “Net_Browse_Nets” and design precision Nets: // “Nets:” 0 0 k <date> // 0 0 number_of_nets datestamp <net_name_1> // name of the net ... additional nets ... // additional names Placements: // “Placements” 0 0 k <date> // 0 0 number_of_placements datestamp <placement_name_1> <cell_name> // placements containing additional nets. ... additional placements ... // additional placements and cells. Notes: a) layout_net_path is traced upward to its highest hierarchical representative before browsing occurs.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET BROWSE PORTS ***layout_net_path*** [*placement_name*]	OK. NOK(5), NOK(12), NOK(32), NOK(33), NOK(35)
Description: Given a net name, return ports on that net which are in a particular placement (the current query context by default). In addition, indicate placements which contain additional ports on that net down the hierarchy. Filter: FILTER LAYERS Response: Net_Browse_Ports <precision> // “Net_Browse_Ports” and design precision Ports: // “Ports:” 0 0 k <date> // 0 0 number_of_ports datestamp <port_name_1> <device_index_1> // name of the port ... additional ports ... // additional names Placements: // “Placements” 0 0 k <date> // 0 0 number_of_placements datestamp <placement_name_1> <cell_name> // placements containing additional ports. ... additional placements ... // additional placements and cells. Notes: a) layout_net_path is traced upward to its highest hierarchical representative before browsing occurs.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET BROWSE SHAPES ***layout_net_path*** [*placement_name*]	OK. NOK(5), NOK(29), NOK(32), NOK(33), NOK(35)
Description: Given a net name, return shapes making up that net that are in a particular placement (the current query context by default). In addition, indicate placements which contain additional shapes on that net down the hierarchy. Filter: FILTER LAYERS Response: Net_Browse_Shapes <precision> // “Net_Browse_Shapes” and design precision Layer_name_1 // first layer on which the net has a presence <k> <k> 0 <date> // k polygons on this layer p 1 4 // first polygon on this layer . . . // the vertices of the first polygon . . . // remaining k-1 polygons on this layer . . . // remaining layers on which net has a presence. Placements: // “Placements” 0 0 k <date> // 0 0 number_of_placements datestamp <placement_name_1><cell_name> // placements containing additional devices. ... additional placements ... // additional placements. Notes: a) layout_net_path is traced upward to its highest hierarchical representative before browsing occurs.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET DEVICENAMES *net_name*	OK. NOK(5), NOK(33), NOK(34)
Description: Give a net name, return all devices on that net. Filter: FILTER DEVICES Response: Net_Devicenames <precision> // “Net_Devicenames” and design precision Devices: // “Devices:” 0 0 k <date> // 0 0 number_of_devices datestamp <device_name_1> <device_index_1>// name of the device, index into the device table ... additional devices ... // additional names and indices	
NET EXTERNAL SHAPES	OK. NOK(29), NOK(33), ERROR(101), ERROR(102)
Description: Determines which nets in the current query cell connect upwards in the design hierarchy. Flattens all of these nets into the coordinate system of the current context and returns them as a response. Filters: FILTER LAYERS, FILTER WINDOW, FILTER CULL Response: Net_Shapes <precision> // “Net_Shapes” and design precision Layer_name_1 // first layer on which shapes were found <k> <k> 0 <date> // k polygons on this layer p 1 4 // first polygon on this layer . . . // the vertices of the first polygon . . . // remaining k-1 polygons on this layer . . . // remaining layers on which shapes were found. Notes: a) Only shapes on the layers allowed by the current setting of the FILTER LAYERS parameter will be returned. b) Only pins allowed by the current setting of the FILTER WINDOW parameter will be returned.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET LAYERS *layout_net_path*	OK. NOK(5)
Description: Given the path name of a net relative to the current query cell, returns the names of layers with shapes on the net. Response: Net_Layers <precision> // “Net_Shapes” and design precision Layers: // first layer on which shapes were found 0 0 n <date> // n unique layers on this net Layer_name_1 // first layer on this net . . . // remaining n-1 layers on this net Notes: a) The net path name specified need not be the highest representative of the net in query cell. It will be traced through out the query cell, not just downward from the given path.	
NET LAYOUT *source_net_path*	OK: *layout_net_path1…layout_net_path_n* NOK(2), NOK(4), NOK(21), NOK(44)
Description: Given the path name of a net in the source cell corresponding to the current query cell, returns the path name of all corresponding layout nets in the query cell. Notes: a) LVS may create one to many or many to many correspondences between nets. The Acknowledgment returns a list of all layout nets corresponding to the given source net. The first net in the list is the primary corresponding net according to the cross reference. The remaining nets are listed in an arbitrary order. b) See the NET SOURCE command for a description of circumstances under which LVS will remove nets for comparison purposes. c) The cross reference database (XDB) includes nets from the "INCORRECT NETS" section of the report that are listed with "no similar net" as well as those listed as under "UNMATCHED OBJECTS" as "unmatched net".	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET LOCATION ***x y***	OK: *net_path_name* NOK(14), NOK(116)
Description: Given a location in the current viewing cell coordinate system, returns the pathname of the net in the current query instance overlapping that location. Filter: FILTER LAYERS Notes: a) The search for a closest net considers not only shapes in the query cell but all shapes obtained by flattening the hierarchy directly below the query point. b) Shapes which are not on a layer in the current client's filter layer list are ignored. c) Layers in the filter layer list which are not connectivity layers are ignored. d) If two or more net shapes overlap the location and are equally distant from it, one is arbitrarily selected and the others are ignored.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET NAMES [FLAT]	OK. NOK(15), NOK(16), NOK(33), ERROR(101), ERROR(102)
Description: Returns a list of all net names in the current (optionally flattened) query cell. Response: Net_Names <precision> // “Net_Names” and design precision Nets: // “Nets:” 0 0 <n> <date> // n lines of text follow <net_name_1> // first net name in current context . . . // intermediate net names in current context <net_name_n> // last net name in current context Notes: a) If the optional command word FLAT is used, all nets in the flattened query cell are reported. That is, nets of cells placed in the query cell at any level are reported with a query cell based net name leading to the cell instance containing the net. The resulting list will contain the same design wide net under several names. For example if the net CLK in the top cell connects to the net CLK1 in cell instance X23, the resulting list will contain both “CLK” and “X23/CLK”. b) If FLAT is omitted, only nets appearing directly in the query cell are reported. c) The order in which the nets are listed is arbitrary.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET NOT CONNECTED ***reference_net*** [<*reference_net*> ...] ***target_cell target_net***	OK. NOK(5), NOK(33), NOK(36), NOK(37), NOK(38), NOK(39)
Description: Given a set of reference net paths, a target cell which resides farther down the hierarchy, and a target net within that cell, returns a list of all placements of the target cell for which the target net is not connected to any reference net. Response: Net_Not_Connected <precision>// “Net_Not_Connected” and design precision Placements: // “Placements:” 0 0 <n> <date> // n lines of text follow <placement_name_1> // first placement name rooted in current context . . . // intermediate names rooted in current context <placement_name_n> // last placement name rooted in current context Notes: a) The order in which the placements are listed is arbitrary. b) if target_net is connected to reference_net throughout the hierarchy, an empty list is returned. c) A net path may be used. A path which extends upward in the hierarchy is searched from its highest point in the hierarchy. For example if net X1/X3/2 is used as the reference net, and this net connects to the net GND in the current query cell, the commands NET NOT CONNECTED X1/X3/2 NANDCELL GND and NET NOT CONNECT-ED GND NANDCELL GND are equivalent. Example: NET NOT CONNECTED GND NANDCELL GND When executed from the top level cell, finds placements of NANDCELL where the GND net in the cell NANDCELL is not connected to top level GND net. NET NOT CONNECTED VCC VCCA NANDCELL VCC When executed from the top level cell, finds placements of NANDCELL where the VCC net in the cell NANDCELL is not connected to top level VCC net and not connected to the VCCA net.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET PINS *layout_net_path*	OK. NOK(1), NOK(5), NOK(33), ERROR(101), ERROR(102)
Description: Given the path name of a net relative to the current query cell, return the intentional device pins attached to the net. Each pin will be represented as a device path name, the device type number (that is, the 0-based index of the device in the device table), the 0-based index of the pin in the device table entry, and a marker square centered at the pin location on the pin layer. Filters: FILTER LAYERS, FILTER WINDOW Response: Net_Pins <precision> // “Net_Pins” and design precision Pins: // “Pins” 0 0 <n> <date> // n lines of text follow (n pins were found) <pin_info_1> // pin number 1 (see notes below for definition of pin_info) . . . // more pins <pin_info_n> // pin number n Layer_name_1 // first layer on which pins are present <k> <k> 0 <date> // k pins on this layer p <i> 4 // i is the number of this pin in the “Pins:” check section . . . // the four vertices of the first square . . . // remaining m-1 pins on this layer . . . // remaining layers on which pins are present Notes: a) Only pin shapes on the layers allowed by the current setting of the FILTER LAYERS parameter will be returned. b) Only pins allowed by the current setting of the FILTER WINDOW parameter will be returned. c) The pins are listed in the “Pins:” check section in an arbitrary order.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description d) Each pin_info_k line is a white space separated list of the following items: pin_number // number of pin in the list of pins device_path // instance path name ending with device instance name device_type // the index of the device in the device table pin_index // the index of the pin in the ordered list of device pins See the Device Table section for the meaning of device_number and pin_index. e) The pin_number included in the Pins: check is not logically necessary since it simply counts up by 1 for each line, but is included for convenience. f) The location on which the square is centered is the lowest of the left most points where the pin shape touches or overlaps the device seed shape. The actual shape of the pin is not known to the Query Server. g) Under certain circumstances the device recognizer will assign more than one logical device pin to the same physical pin shape. In this case, a marker will be present for each of the logical device pins, but they will all center on the same location. h) The size of the marker squares is determined by the current value of the MARKER SIZE parameter.	
NET PORTNAMES *net_name*	OK. NOK(5), NOK(12), NOK(33)
Description: Given a net name, return all ports on that net. Filter: FILTER LAYERS Response: Net_Portnames <precision> // “Net_Portnames” and design precision Ports: // “Ports:” 0 0 k <date> // 0 0 number_of_ports datestamp <port_name_1> // name of the port ... additional ports ... // additional ports	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET PORTS *net_path*	OK. NOK(5), NOK(12), NOK(33), ERROR(101), ERROR(102)
Description: Given the pathname of a net in the current query cell, returns the names of the ports attached to that net and a marker square for each port. Filters: FILTER LAYERS, FILTER WINDOW Response: Net_Ports <precision> // “Net_Ports” and design precision Ports: // “Ports:” 0 0 <n> <date> // n lines of text follow (n ports were found) 1 <port_name_1> // index of first port and first port name . . . // more indices and ports <n> <port_name_n> //index of last port last port name Layer_name_1 // first unfiltered layer on which ports are present <k> <k> 0 <date> // k ports on this layer p <i> 4 // i is the number of this port in the “Ports:” check section . . . // the four vertices of the first square . . . // remaining m-1 ports on this layer . . . // remaining unfiltered layers on which ports are present Notes: a) Only ports on the layers allowed by the current setting of the FILTER LAYERS parameter will be returned. b) Only ports allowed by the current setting of the FILTER WINDOW parameter will be returned. c) A port may have more than one location. Each location is listed separately in the Ports: check and a separate marker is given on the appropriate layer for each location. d) The ports returned are not only those in the query cell, but those in any cell placement into which the net extends. e) The ports are listed in an arbitrary order. f) The number <i> which appears in the polygon entry for each marker square is the index number of the corresponding port in the Ports: check. g) Information on any given port may be obtained by the PORT INFO command.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET SHAPES *layout_net_path*	OK. NOK(5), NOK(29), NOK(33) ERROR(101), ERROR(102)
Description: Given the layout path name of a net relative to the current query cell, return the shapes of the net flattened into the coordinate system of the current context. The path name specified need not be the highest representative of the net in query cell. It will be traced through out the query cell, not just downward from the given path. Filters: FILTER LAYERS, FILTER WINDOW, FILTER CULL Response: Net_Shapes <precision> // "Net_Shapes" and design precision Layer_name_1 // first layer on which the net has a presence <k> <k> 0 <date> // k polygons on this layer p 1 4 // first polygon on this layer . . . // the vertices of the first polygon . . . // remaining k-1 polygons on this layer . . . // remaining layers on which net has a presence. Notes: a) Only shapes on the layers allowed by the current setting of the FILTER LAYERS parameter will be returned. b) Only pins allowed by the current setting of the FILTER WINDOW parameter will be returned. c) Polygons in the response are limited to a maximum number of vertices. Polygons which would have a more than this number of vertices are segmented into polygons which obey the maximum vertex limitation. See the next note for how to determine the maximum. d) The maximum vertex count of response polygons can be set by using the MAXIMUM VERTEX <v> statement in the rules file. If no such statement is found, the maximum value will be 4096. e) Any layer which contains node numbers, including CONNECT layers and those that get connectivity from STAMP operations and through node-preserving layer constructors (NOT, AND, topological operations, and so on) will be returned if it is on the net requested.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET SOURCE *layout_net_path*	OK: *source_net_path1…source_net_path_n* NOK(2), NOK(3), NOK(21), NOK(45)
Description: Given the path name of a net relative to the current query cell, returns the path names of all corresponding nets in the corresponding source cell. Notes: a) LVS may create one to many or many to many correspondences between nets. The Acknowledgment returns a list of all source nets corresponding to the given layout net. The first net in the list is the primary corresponding net according to the cross reference. b) LVS may discard certain nets during comparison. The Query Server has limited ability to cross reference nets which have been discarded during comparison. Nets that can be traced down in the layout database will be matched to a lower level net if an unambiguous correspondence point can be found (PHDB database must be present). c) Net names may also be lost during comparison for the following reasons: i) a lower level net is flattened to a higher level in the hierarchy where it connects to a higher level net--if so, LVS will use the name of the higher level net. ii) A net is filtered out for comparison purposes if it only connects devices that are combined into a structure for comparison purposes (device reduction and gate recognition). iii) A net is filtered out for comparison purposes if it does not connect to more than one device (floating). This includes nets that are left floating after device filtering. iv) The cross reference database (XDB) includes nets from the "INCORRECT NETS" section of the report that are listed with "no similar net" as well as those listed as under "UNMATCHED OBJECTS" as "unmatched net".	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET TRACE *layout_net_path*	OK: *high_net_path net_name1…net_namen* NOK(5)
Description: Given the path name of a net relative to the current query cell, returns the path name of the highest representative of that net in the query cell as well as all intermediate net path names between the highest net and the given net. Notes: a) Consider the following command and acknowledgment: NET TRACE X1/X2/X3/X4/X5/clk5 OK: X1/X2/clk2 X1/X2/X3/clk3 X1/X2/X3/X4/clk4 X1/X2/X3/X4/X5/clk5 This response means the following: Net X1/X2/clk2 is the highest representative of X1/X2/X3/X4/X5/clk5 relative to the current query cell. That is, net X1/X2/clk2 is not connected to an external pin of instance X1/X2. Net X1/X2/clk2 connects to net X1/X2/X3/clk3 Net X1/X2/X3/clk3 connects to net X1/X2/X3/X4/clk4 Net X1/X2/X3/X4/clk4 connects to net X1/X2/X3/X4/X5/clk5 b) Note that the trace proceeds upward from the given net until it reaches its highest hierarchical representative within the current query cell. This does not imply that the net does not also descend into other below the query cell instances, or that it does not descend into the instances on the given path by more than one pin, or that it does not connect through an external pin of the query cell into higher levels of the design. For instance, in the example above net X1/X2/clk2 may very well be connected not only to net X1/X2/X3/clk3 but also to net X1/X2/X3/sync by a second pin in instance X1/X2/X3.	

Table 15-19. Query Net Commands [continued]

Command	Acknowledgments
Description	
NET TEXT MAP [INVALID]	OK. NOK(46)
Description: Generates a list of net names corresponding to Calibre LVS generated net numbers. Response: Net_Text_Map <precision> // “Net_Text_Map” and design precision Net Texts: 0 0 <n> <date> // n lines of text follow <net_text> <net_number>// net_text is the name of net net_number Notes: a) Certain texts are not used for Spice netlist net names by Calibre. These texts are not included in the map. b) The INVALID option causes only those net names that are invalid for spice netlisting by Calibre to be shown.	
NET VALID **{SOURCE | LAYOUT}** ***net_path***	OK: *net_path* NOK(3), NOK(4), NOK(5)
Description: Given the name of a net, indicates if it is valid. Notes: If a PHDB is available it is searched for a layout net name. If running in XDB only mode, the XDB is searched for the layout name. Source names are always searched for in the XDB.	

Query Device Commands

The commands described in Table 15-20 are used to query device information.

Table 15-20. Query Device Commands

Command	**Acknowledgments**
Description	
DEVICE BAD	OK. NOK(20), ERROR(101), ERROR(102)
Description: Return a seed shape for each bad device in the query cell. Response: Device_Bad <precision> // “Device_Bad” and design precision <seed_layer_name_1> // name of first seed shape containing bad devices <n> <n> 0 <date> // <n> seed shapes follow p 1 n // first bad device seed shape . . . // the vertices of the first shape . . . // remaining n-1 shapes on this layer . . . // remaining seed layers on which bad devices are present Notes: a) Only bad devices in the query cell itself are reported. That is, no bad devices from lower level cell placements are reported. To obtain all the bad devices in the design, each cell must be queried for bad devices.	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description	
DEVICE INFO ***layout_device_path***	OK. NOK(9), NOK(33), ERROR(101), ERROR(102)
Description: Given the path name of a device relative to the current query cell, return its device type number, a list of the nets attached to its pins, a list of its property values, and the seed shape. See also PLACEMENT INFO. Response: Device_Info <precision> // "Device_Info" and design precision Info: // "Info:" 0 0 <n> <date> // n lines of text follow <device_type_number> // 0-based index of this device type in device table <layout_net_path_1> // net connected to first pin in device table . . . // nets connected to other pins in device table order <property_value_1> // value of first device property in device table . . . // value of other properties in device table order <seed_layer_name>: // name of layer containing seed shape followed by a colon 1 1 0 <date> // one seed shape follows p 1 n // the seed shape with n vertices . . . // the n vertices of the seed shape Notes: a) The number of device pins and properties must be obtained from the device table. b) The layout net paths listed appear in the same order as the pin names in the device table. The property names must be obtained from that table. c) The property values listed appear in the same order as the property names in the device table. The property names must be obtained from that table. d) The layout_net_path_i's are reported at the same hierarchical level as the device. That is, they are not shortened to their highest point in the query cell hierarchy.	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description	
DEVICE LAYOUT ***source_device_path***	OK: *source_device_path* NOK(2), NOK(8), NOK(21)
Description: Given the path name of a device relative to the source cell corresponding to the current query cell, returns the path names of all corresponding layout devices relative to the current query cell. Notes: a) LVS may create many to many correspondences between devices. The acknowledgement returns a list of all layout devices corresponding to the given source device. The first device in the list is the primary corresponding device according to the cross reference.	
DEVICE LOCATION ***x y***	OK: *device_path_name* NOK(6), ERROR(116)
Description: Given a location in the current viewing cell coordinate system, returns the pathname of a device composed of a seed shape directly under location. Filters: FILTER DEVICES, FILTER LAYERS Notes: a) Devices whose device type number is not on the client's filter device list are ignored. b) Devices whose seed shape layers are not in the clients filter layer list are ignored. c) If two or more device seed shapes overlap the location, one is arbitrarily selected and the others are ignored.	

Table 15-20. Query Device Commands [continued]

Command	**Acknowledgments**
Description	
DEVICE NAMES [FLAT}	OK. NOK(18), NOK(19), ERROR(101), ERROR(102)
Description: Returns a list of all devices in the current (optionally flattened) query cell. Filter: FILTER DEVICES Response: Device_Names <precision> // “Device_Names” and design precision Devices: // “Devices:” 0 0 <n> <date> // n lines of text follow <device_name_1> <device_type_1> // first device name and type . . . // intermediate device names and types <device_name_n> <device_type_n> // last device name and type Notes: a) If the optional command word FLAT is used, all devices in the flattened query cell are reported. That is, devices contained in cell instances placed in the query cell at any level are reported with a query cell based path name leading to the cell instance containing the device. If FLAT is omitted, only devices appearing directly in the query cell are reported. b) The devices are listed in an arbitrary order.	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description	
DEVICE PINS *layout_device_path*	OK. NOK(18), NOK(19), ERROR(101), ERROR(102)
Description: Given the path name of a device relative to the current query cell, return its device type number, a list of the nets attached to its pins, and the pin shapes. Pin shapes reported within an area equal to the rectangular extent of the seed shape plus the marker size. Response: Device_Pins <precision> // “Device_Pins” and design precision Info: // “Info:” 0 0 <n> <date> // n lines of text follow <device_type_number> // 0-based index of this device type in device table <layout_net_path_1> // net connected to first pin in device table . . . // nets connected to other pins in device table order <pin_layer_name> // name of layer containing pin shape np np 0 <date> // np pin shapes follow p 1 n // the seed shape with n vertices . . . // the n vertices of the seed shape ... // additional pin shapes <pin_layer_name> // name of layer containing last pin shape ... // polygon information for last pin shape Notes: a) The layout net paths listed appear in the same order as the pin names in the device table. The property names must be obtained from that table. b) The layout_net_path_i’s are reported at the same hierarchical level as the device. That is, they are not shortened to their highest point in the query cell hierarchy.	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description	
DEVICE SOURCE ***layout_device_path***	OK: *source_dev_path1*…*source_dev_pathn* NOK(2), NOK(7), NOK(21)
Description: Given the path name of a device relative to the current query cell, returns the path names of all corresponding devices in the corresponding source cell. Notes: a) LVS may create one to many or many to many correspondences between devices. The acknowledgment returns a list of all source devices corresponding to the given layout device. The first device in the list is the primary corresponding device according to the cross reference.	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description	
DEVICE TABLE	OK. ERROR(101), ERROR(102)
Description: Responds with the device table. See the section "Device Tables" for more information. Response: Device_Table <precision> // "Device_Table" and design precision Table Count // "Table Count" 0 0 1 <date> // 1 line of text follows <k> // k device entries are defined (indexed 0 through k-1) Device Entry 0 // constant "Device Entry" followed by the device number 0 0 <n> <date> // n lines of text follow <device_layer_name> // name of seed layer for this device <device_type> // basic type of device (see notes for values) <element_name> // the element name of the device <model_name> // the model name of the device; "(null)" if none <netlist_element_name> // the netlist element name of the device; "(null)" if none <netlist_model_name> // the netlist model name of the device; "(null)" if none <pin_count> // number of pins of device <pin_info_0> // information about first pin (see notes) . . . // information about remaining pins <aux_layer_count> // number of auxiliary layers associated with device <aux_layer_0> // name of first auxiliary layer . . . // remaining auxiliary layers <property_count> // number of device properties <property_name_0> // first property name . . . // remaining property names . . . // remaining device entries 1 through k-1 Notes: a) The device table is design wide and independent of the current context. b) device_type is one of the following constants: "DIODE", "RESISTOR", "CAPACITOR", "MOS", "BIPOLAR", "USER".	

Table 15-20. Query Device Commands [continued]

Command	Acknowledgments
Description c) The pin_info_i lines contain the following entries separated by white space <pin_name> <pin_layer> <swap_group>. d) Pin and property names have been lower cased. e) If any of <model_name>, <netlist_element_name>, or <netlist_model_name> were not specified in the rule file, they are represented in the response by the constant string “(null)”. f) The layer names have the same case as in the rule file.	
DEVICE VALID {SOURCE | LAYOUT}*device_path*	OK: *device_path* NOK(7), NOK(8), NOK(9)
Description: Given the name of a device, indicates if it is valid. This command is identical to PLACEMENT VALID. Notes: If a PHDB is available it is searched for a layout net name. If running in XDB only mode, the XDB is searched for the layout name. Source names are always searched for in the XDB.	

Query Rule File Commands

The commands described in Table 15-21 are used to query rule file information.

Table 15-21. Query Rule File Commands

Command	**Acknowledgments**
Description	
RULES FILE NAME	OK: *rule_file_name* NOK(43)
Description: Returns the file name of the rule file.	
RULES LAYOUT NETLIST	OK: *netlist_path_name*
Description: Returns the path name of the netlist used for LVS comparison. This may be either the path given in the Source Path statement in the rule file (for spice-to-spice comparison) or the argument to the calibre -spice command line switch (for GDSII-to-spice comparison).	
RULES LVS REPORT	OK: *lvs_report_file_name* NOK(27), NOK(43)
Description: Returns the file name from the LVS Report specification statement in the rule file.	
RULES [SOURCE | LAYOUT] PATH	OK: *source_path* or *layout_path* NOK(30), NOK(31)
Description: Returns the source or layout path.	
RULES [SOURCE | LAYOUT] SYSTEM	OK: *source_system* or *layout_system*
Description: Returns the source or layout system as specified it the rule file.	
RULES SVDB DIRECTORY	OK: *svdb_directory_path* NOK(28)
Description: Returns the pathname from the Mask SVDB Directory specification statement in the rule file.	

Examples

The following command asks for geometries of net CLOCK1 relative to the current query cell:

```
NET SHAPES X12/X45/CLOCK1
```

The following command asks for information about device C4 relative to the top cell:

```
DEVICE INFO X95/X4/X312/C4
```

The following command asks the Query Server to deactivate the current client and activate client 12:

```
ACTIVATE 12
```

When executed from the top level cell, the following command finds placements of NANDCELL where the GND net in the cell is not connected to top level GND:

```
NET NOT CONNECTED GND NANDCELL GND
```

The following two examples show the information available from the Query Server given various Calibre LVS-H and Query Server invocations and specified rule file statements. Given

```
LAYOUT PRIMARY top
MASK SVDB DIRECTORY svdb QUERY
```

the following commands provide layout and source information:

```
Calibre LVS-H: calibre -spice lay.net -lvs -hier -hcell cells rules
Query Server: calibre -query svdb top
```

```
Calibre LVS-H:calibre -spice lay.net rules.ext
        calibre -lvs -hier -hcell cells rules.cmp
Query Server:calibre -query svdb top
```

and the following commands provide layout information only (because *-lvs* was not specified):

```
Calibre LVS-H: calibre -spice lay.net rules
Query Server: calibre -query svdb top
```

Given

```
LAYOUT PRIMARY top
MASK SVDB DIRECTORY /scratch/svdb PHDB
```

the following commands provide layout information only (because the rule file specifies PHDB):

```
Calibre LVS-H:calibre -spice lay.net -lvs -hier -hcell cells rules
Query Server:calibre -query /scratch/svdb top
```

```
Calibre LVS-H:calibre -spice lay.net rules
Query Server:calibre -query /scratch/svdb top
```

Calibre Connectivity Interface

The following commands govern the creation of files that are part of the Calibre Connectivity Interface (CCI). The CCI provides a way for downstream tools, including third party tools, to gain access to layout connectivity information extracted by Hierarchical Calibre LVS. This information includes layout geometries, nets, devices, and corresponding schematic or source names. Third party parasitic extractors are an example of potential clients for this interface.

The interface consists of a set of files with documented format. These files are created upon request by the Calibre Query Server. This is a command-driven interactive application which reads the SVDB results database created by Calibre LVS. All data required for CCI interface is enabled when the Mask SVDB Directory rule file statement is specified with the CCI option. Conventions for CCI commands are unlike other Query Server commands in that these are intended to dump data from the databases that the Query Server manages. Issues of client context are not important here. Also, the file formats produced are standard formats and so do not conform to the usual presentation of results in DRC results format. These commands all require a “calibreci” license to operate.

This interface consists of the following file formats:

- Annotated GDSII Files (AGF) from the layout database

  These files are GDSII stream files which use the PROPATTR and PROPVALUE fields in GDSII to specify device ids, node numbers and

instance placement names. Your Mask SVDB Directory statement must include the GDSII, or CCI options to generate AGF files.

- Hierarchical Layout Netlist

  This is a Spice netlist representing network connectivity in the AGF file.

  your Mask SVDB Directory statement must include NETLIST, GDSII, or CCI options to generate a Layout Netlist.

- Layout Netlist Names (LNN)

  This provides a mapping between net names and net numbers used in the Hierarchical Layout Netlist. Your Mask SVDB Directory statement must include NETLIST, GDSII, or CCI options to generate a Layout Netlist Names file.

- Port Table (PORTS)

  This provides port location information.

  Your Mask SVDB Directory statement must include NETLIST, GDSII, or CCI options to generate a Ports file.

- Cross Reference files

  These indicate correspondence as determined by LVS between nets and instances between source and layout netlists.

  Your Mask SVDB Directory statement must include XDB, QUERY, or CCI options to generate any cross reference files.

Customized Layout Netlist Generation

The following Query Server commands produce customized netlists from the SVDB results database generated during Calibre LVS. The LAYOUT NETLIST

WRITE command creates a Spice netlist. The construction of this netlist can be tailored by using the following commands:

- LAYOUT NETLIST DEVICE LOCATION to control device coordinate presentation
- LAYOUT NETLIST EMPTY CELLS to present cells (like vias) that do not contain devices
- LAYOUT NETLIST HIERARCHY to expand various cells within the netlist
- LAYOUT NETLIST NAMES to use layout names, source names, or no names
- LAYOUT NETLIST PIN LOCATIONS to present pin locations of devices
- LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS to present primitive device cells
- LAYOUT NETLIST TRIVIAL PINS to present pins that do not have devices attached
- FILTER DEVICES | FILTER DEVICENAMES (see page 15-92) to select which devices are presented.

These commands are discussed in Table 15-22.

Table 15-22. Layout Netlist Generation Commands

Command	Acknowledgments
Description	
LAYOUT NETLIST DEVICE LOCATION {VERTEX | CENTER}	OK ERROR(130)
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command by giving device locations. The netlist may be generated with $X= $Y= comments to indicate either a vertex or the center of the device seed shape. Notes: a) By default, the layout netlist is written with vertex locations. b) This option requires use of either the CCI or ANNOTATE DEVICES option to the Mask SVDB Directory rule file statement. c) If CENTER is specified and the CENTER of a device seed shape lies outside the seed shape (on a C-shaped device, for example), then a vertex is used for the $X= $Y= location.	
LAYOUT NETLIST EMPTY CELLS {YES | NO}	OK
Description: This command alters the netlist generated by the LAYOUT NETLIST WRITE command to give calls to empty cells. The netlist may be generated with calls to cells that do not contain devices (for example, via cells). Notes: a) By default, the layout netlist is written with no calls to cells that do not contain devices.	

Table 15-22. Layout Netlist Generation Commands

Command	**Acknowledgments**
Description	
LAYOUT NETLIST HIERARCHY {ALL | FLAT}	OK ERROR(128)
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command so that it provides the specified hierarchy. The netlist may be generated with certain cells expanded. ALL implies that no cells are expanded. FLAT implies that all cell placements are expanded into the top-level cell. Notes: a) If Layout Netlist Names SOURCE is not specified, the layout netlist is written with ALL cells by default. b) If Layout Netlist Names SOURCE is specified, the layout netlist is written FLAT by default. c) If Layout Netlist Names SOURCE is specified, the LAYOUT NETLIST HIERARCHY ALL command is invalid and will be rejected.	

Table 15-22. Layout Netlist Generation Commands

Command	**Acknowledgments**
Description	
LAYOUT NETLIST NAMES {LAYOUT | NONE | SOURCE} [NETS | INSTANCES]	OK ERROR(129)
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command to provide the specified netlist names. The netlist may be generated with original layout net names for texted nets (generated net numbers for untexted nets), with generated net numbers or with source names. By default, the command applies to both NETS and INSTANCES. The optional [NETS | INSTANCES] keywords cause it to apply specifically to NETS or INSTANCES, respectively. Notes: a) By default, the layout netlist is written with layout names. b) If Layout Netlist Names SOURCE is executed and Layout Netlist Hierarchy is currently set to ALL, the Query Server will automatically change the HIERARCHY setting to FLAT and issue a NOTE to that effect (see LAYOUT NETLIST HIERARCHY command). c) Unmatched nets are identified using a prefix when SOURCE is specified. The prefix is typically _Layout_, but if there are source names that begin with _Layout_, additional "_" characters are added to the prefix to insure that it is unique.	

Table 15-22. Layout Netlist Generation Commands

Command	**Acknowledgments**
Description	
LAYOUT NETLIST PIN LOCATIONS {YES|NO}	OK
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command to provide pin locations. This command configures the netlist to include the locations of pins for all devices. The pin location information includes a device template for each device type that shows the ordering of pins and a $PIN_XY comment after each device instance showing the coordinates of the pins. The device template has the following format: *.DEVTMPLT <d> <el>([<mod>])<seed> <pl_1>(<pn_1>) [<pl_2>(pl_2)...] *.DEVTMPLT indicates the device template comment record. <d> represents the device template number. Each device is identified with a $dt=<n> comment field that matches it with a specific DEVTMPLT. <el> is the element name specified in the rule file DEVICE statement. <mod> is the model name, when present, specified in the rule file DEVICE statement. <seed> is the device seed layer specified in the rule file DEVICE statement. <pl_n> is the n^{th} device pin layer specified in the rule file DEVICE statement. <pn_n> is the n^{th} device pin name specified in the rule file DEVICE statement. The device instances in this netlist have the following additional comment line: <regular_device_record> +$dt=<d> $PIN_XY=X1,Y1[,X2,Y2...] $dt=<d> shows that this is a device corresponding to *.DEVTMPLT d $PIN_XY= specifies ordered X,Y coordinates for each pin in the order shown on the *.DEVTMPLT record (this order can be different than the pin order specified in the netlist instance when the device is a standard MOS device).	

Table 15-22. Layout Netlist Generation Commands

Command	Acknowledgments
Description	
For example: *.DEVTMPLT 0 mp() PSEED GPIN(g) SDPIN(s) SDPIN(d) represents provides pin layer and ordering information for all netlist devices that include the $dt=0 comment. It represents an MP device with no model name. The seed layer is PSEED, the pin layers are GPIN, SDPIN and SDPIN for the pin names g, s, and d, respectively. The following device instance in the netlist: M0 2 4 1 p L=5e-06 W=3e-06 $X=-6000 $Y=1000 +$dt=0 $PIN_XY=-4000,3500,-6000,2500,-2000,2500 corresponds with *DEVTMPLT 0 above. Pin locations for this device are as follows (recall that Spice standard syntax puts pins in the order d g s): GPIN(g) connected to net 4: (-4000,3500) SDPIN(s) connected to net 1: (-6000,2500) SDPIN(d) connected to net 2: (-2000,2500) Notes: a) By default, the layout netlist is written with no pin location information. b) The CENTER option requires use of either the “CCI” or “ANNOTATE DEVICES” option to the Mask SVDB Directory rule file statement. c) If CENTER is specified and the CENTER of a device seed shape lies outside the seed shape (on a C shaped device, for example), then a vertex is used for the $X= $Y= location.	
LAYOUT NETLIST PRIMITIVE DEVICE SUBCKTS {YES | NO}	OK
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command to provide primitive device subcircuits. The netlist may be generated with or without .SUBCKT statements for user-defined primitive devices. Normally, empty primitive device subckts are included for all user defined devices defined in the rule file. This command configures the Query Server to leave them out.	

Table 15-22. Layout Netlist Generation Commands

Command	**Acknowledgments**
Description	
LAYOUT NETLIST RESET	OK
Description: Resets layout netlist generation information to default values. Notes: a) This command does not reset FILTER DEVICES/DEVICENAMES (see page 15-92). Use FILTER DEVICES ALL to reset the device filter.	
LAYOUT NETLIST TRIVIAL PINS {YES | NO}	OK
Description: Alters the netlist generated by the LAYOUT NETLIST WRITE command to provide trivial pins. The netlist may be generated with pins not connected to devices. Notes: a) By default, the layout netlist includes only pins connected to other pins or to devices.	
LAYOUT NETLIST WRITE *file_path*	OK NOK(33), ERROR(101), ERROR(102)
Description: Given the name of an output file ***file_path***, writes a Spice netlist describing devices, placements, and network connectivity present in the layout. Several supporting commands configure the netlist. Response: writes layout netlist file Notes: a) Details of the layout netlist format are included below. b) By default, the netlist written is similar to the netlist produced by calibre -spice. c) The command may generate warnings as it executes. These are sent to stdout and are prefixed by the string “WARNING”. d) Requires calibreci and calibreqdb licenses.	

Table 15-22. Layout Netlist Generation Commands

Command	Acknowledgments
Description	
STATUS LAYOUT NETLIST	OK

Description: Presents current layout netlist generation options.
Response:
Status Layout Netlist <prec>
// "Status Layout Netlist" followed by design precision
Entries: // "Entries:"
0 0 <n> <date> // n lines of text follow
Hierarchy: <hierarchy_setting>
// setting for LAYOUT NETLIST HIERARCHY
Names: <names_setting>
// setting for LAYOUT NETLIST NAMES
Device Location: <dev_loc>
// setting for LAYOUT NETLIST DEVICE LOCATION
Filter Devices: <filter_info>
// setting for FILTER {DEVICES | DEVICENAMES}
END OF RESPONSE	

Layout Netlist File Format

This is a Spice netlist representing the layout connectivity.

The netlist conforms to the extended Spice netlist format used in Calibre LVS and described in the "Spice Format" chapter. Note that comment-coded fields, when used, are ignored by most Spice simulators.

By default, the netlist is identical in structure to the netlist produced by calibre -spice. Various configuration options alter the structure of the netlist produced.

- Subcircuit definitions

  .SUBCKT definitions in the layout netlist represent original cells in the input database or pseudo-cells created by Calibre hierarchical pre-processing. For original database cells, subcircuit names in the layout

netlist are identical to cell names in the input database. An exception occurs when a cell name in the input database begins with a '$' character. Cell names that start with '$' usually have the prefix "__". In general, to avoid conflicts with database names, the prefix consists of N underscores, where N is one larger than the number of leading underscores in any database cell named "___*$.*" (that is, any database cell whose name begins with at least two underscores followed by a '$'). This prefix is necessary to make those cell names valid in Spice; recall that a leading '$' denotes a comment in Spice.

Subcircuit definitions indicate which nets in the cell serve as pins of the cell. All nets in a cell that make connections to nets outside of the cell appear as pins in the .SUBCKT line for the cell.

For example:

```
.SUBCKT ABC 1 2
R1 1 2 50
.ENDS
```

describes cell ABC with two pins: 1, 2. These numbers are net numbers in the cell.

- Subcircuit calls

  Subcircuit calls are represented by standard Spice subcircuit call notation 'X'. Comments provide some additional information.

  For example, the line:

  ```
  X3 5 7 ABC $T=-375 13750 1 0 $X=2250 $Y=2000
  ```

  describes a placement of cell ABC with nets 5, and 7 connected to the first and second pins of ABC respectively. The comment field $T represents the transform of the placement (X translation, Y translation, reflection (0 or 1, indicating false or true, with the X-axis being the axis of symmetry) and rotation (0, 90, 180 or 270 degrees, counter-clockwise), and $X and $Y represent coordinates of the placement.

- Devices

  When possible, devices in the layout netlist are represented with regular Spice elements (M, R, C, D, Q, and so on). In other cases (for example, when the device is not a standard Spice element), devices in the layout netlist are represented with subcircuit calls. In the latter case, the layout netlist will contain an empty subcircuit definition describing the device. Device lines provide the type of device, pin connections, model name and parameters or properties. Location of the device is presented as an X Y coordinate pair in database units encoded as a comment at the end of the device.

  For example, the line:

  ```
  M2 4 5 6 7 p L=1 W=2 $X=-6000 $Y=1000
  ```

  describes a MOS device instance of type P (Calibre device MP) with drain, gate, source and bulk connected to nets 4, 5, 6 and 7 respectively and W/L as specified at the coordinates (-6000,1000) in the current cell. The line:

  ```
  X2 7 8 9 10 11 Q3E $X=1000 $Y=1000
  ```

  describes a device instance of type Q3E with five pins. In this case, the netlist will also contain a subcircuit definition such as this:

  ```
  .SUBCKT Q3E C B E1 E2 E3
  .ENDS
  ```

  Actual seed and pin device geometries that make up the device may occur at various levels above and below the device coordinates. Their actual location is governed by the LVS circuit extraction Seed Promotion functionality. The device coordinates appear at a vertex of the seed shape after translation into the current cell. Device coordinates can also be configured to appear at the center of the seed shape when the coordinates of the center lie on the seed shape using the Layout Netlist Device Location Query Server command.

Some built-in devices make use of optional substrate pins defined by Calibre. They are specified as:

```
R0 1 2 50 $SUB=3
```

where R0 is a resistor with terminals on nets 1 and 2 and substrate pin on net 3. This feature is controlled with the LVS Netlist Comment Coded Substrate specification statement. If LVS Netlist Comment Coded Substrate NO is specified in the rule file, the above resistor is output as a call to a subcircuit:

```
.SUBCKT R pos neg sub
.ENDS

X0 1 2 3 r=50
```

Devices included in the netlist are affected by the FILTER DEVICES setting currently active in the Query Server. Only devices of FILTERED types are included in the netlist.

- Nets

  The nets in the layout netlist can be specified in one of three ways:

  a. Using a combination of layout net names available from text in the layout database (default) and net numbers for untexted nets.

  b. Pure net numbers.

  c. Source names where source names are available, layout names prepended by a prefix where matching names are not available. The prefix is typically _Layout_, but if there are source names that begin with _Layout_, additional "_" characters are added to the prefix to insure that it is unique.

  Name format is controlled with the Query Server's Layout Netlist Names command.

- Net numbers

  Net numbers are determined arbitrarily by Calibre during circuit extraction. The combination of subcircuit calls and subcircuit definitions in the layout netlist describes how electrical nets traverse levels of hierarchy in the design.

- Expanded placements

  The Layout Netlist Hierarchy command can be used to expand cells in place within the netlist. When cells are expanded, the names of cell placements are of the form XXi/[Xj...]/Xk.

  Note that all device and cell placement names have exactly one extra letter at the front when compared with names in the unexpanded netlist (Layout Netlist Hierarchy ALL). This extra letter insures that the generated netlist is compatible with Spice readers and that the generated names can all be easily mapped to regular Calibre hierarchical names by simply removing the first letter. The extra letter is applied within all cells (those that have expanded placements as well as those that do not).

  For example, consider the following unflattened circuit:

```
.SUBCKT CELL_1 14 13
M0 11 IN 10 13 p L=1e-06 W=1.8e-05 $X=3000 $Y=9500
M1 9 IN 8 14 n L=1e-06 W=1.8e-05 $X=3000 $Y=-10500
X2 23 13 24 14 26 CELL_2
.ENDS

.SUBCKT CELL_2 I1 13 OUT 14 I2
M0 13 I1 OUT 13 p L=1e-06 W=1e-05 $X=-20000 $Y=20500
M1 13 I2 OUT 13 p L=1e-06 W=1e-05 $X=0 $Y=24500
M2 6 I2 14 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-50500
M3 OUT I1 6 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-29500
.ENDS

.SUBCKT TOP 14 13
M0 13 4 5 13 p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
M1 3 2 14 14 n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
M2 5 4 14 14 n L=2e-06 W=1e-05 $X=49000 $Y=-42000
```

```
X3 14 13 CELL_1 $T=-104000 -58000 0 0 $X=-108000
$Y=-76000
.ENDS
```

when CELL_1 is expanded, the circuit is represented as:

```
.SUBCKT CELL_2 I1 13 OUT 14 I2
MM0 13 I1 OUT 13 p L=1e-06 W=1e-05 $X=-20000 $Y=20500
MM1 13 I2 OUT 13 p L=1e-06 W=1e-05 $X=0 $Y=24500
MM2 6 I2 14 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-50500
MM3 OUT I1 6 14 n L=1e-06 W=1e-05 $X=-10000 $Y=-29500
.ENDS

SUBCKT TOP 14 13
MM0 13 4 5 13 p L=3e-06 W=1.3e-05 $X=47000 $Y=-2000
MM1 3 2 14 14 n L=1e-06 W=9e-06 $X=-52000 $Y=-35000
MM2 5 4 14 14 n L=2e-06 W=1e-05 $X=49000 $Y=-42000
MX3/M0 X3/11 X3/1 X3/10 6 p L=1e-06 W=1.8e-05 $X=3000
$Y=9500
MX3/M1 X3/9 X3/1 X3/8 1 n L=1e-06 W=1.8e-05 $X=3000
$Y=-10500
XX3/X2 X3/23 13 X3/24 14 X3/26 CELL_2 $T=0 -22000 0 0
$X=-40000 $Y=-102000
.ENDS
```

See the LAYOUT NETLIST HIERARCHY command for more details on types of flattening available.

- Example use

  To generate a flat netlist consisting only of a specific diode device, D(probe), using source net names, with device locations at the center of the seed shape, use the following sequence of operations:

  a. Ensure that you have access to a calibreci license as well as calibreqdb, calibrelvs, and calibrehlvs licenses.

  b. Use the following rule file statement to enable full CCI functionality within the PHDB:

     ```
     MASK SVDB DIRECTORY "svdb_dir" CCI
     ```

Run Calibre as you normally would:

```
calibre -spice lay.net -lvs -hier -hcell cells rules
```

Run the Query Server on svdb_dir generated in step ii).

```
calibre -query svdb_dir
```

c. After getting the “OK: Ready to serve.” prompt, issue the following Query Server commands:

```
FILTER DEVICENAMES D(probe)
LAYOUT NETLIST HIERARCHY FLAT
LAYOUT NETLIST DEVICE LOCATION CENTER
LAYOUT NETLIST NAMES SOURCE NETS
LAYOUT NETLIST WRITE probe.net
TERMINATE
```

This will create the desired netlist in the file probe.net.

Annotated GDSII File Generation

The following Query Server commands produce annotated GDSII output that use GDSII PROPATTR and PROPVALUE records to present connectivity information with geometry information.

The GDS WRITE command can be configured using the following commands:

- GDS MAP controls or lists layer name to layer number and datatype mapping
- GDS [DEVPROP | NETPROP | PLACEPROP] NUMBER configures PROPATTR record
- GDS RESET resets the default parameters for AGF file generation
- GDS SEED PROPERTY controls the output of device seed geometries.
- LAYOUT NAMETABLE WRITE associates names with net IDs used in the AGF file

- PORT TABLE presents port information in a separate file

These commands are discussed in the following table:

Table 15-23. Annotated GDSII Generation Commands

Command	**Acknowledgments**
Description	
GDS MAP [*layer_name gdsii_layer*] [*gdsii_datatype*] [{DEVICE | ORIGINAL}]	OK ERROR(114), ERROR(123), ERROR(124), ERROR(133)
Description: Returns the present layer name to layer number/datatype mapping in effect. Given a layer name or number and a desired GDSII layer number and an optional GDSII datatype, this command configures the GDS WRITE command to use the desired layer number and datatype for geometries associated with this layer. If GDS SEED PROPERTY DEVICE ORIGINAL is specified, the DEVICE or ORIGINAL keyword must be supplied in order to configure the proper layer. Response: Gds_Map <precision> // “Gds_Map” and design precision Layers: // “Layers:” 0 0 <n> <date> // n lines of text follow <name> <layer> <datatype> // mapping for layer <name> ... continuing for all meaningful layers. Notes: a) GDSII standard specifies layer and datatypes are numbers between 0-63. Values specified between -32768 and +32768 will be accepted and used. Values beyond that range are not representable in GDSII and will not be accepted. b) Multiple layers may be configured with successive invocations of the command. c) By default, layers are arbitrarily numbered sequentially starting with 1.	

Table 15-23. Annotated GDSII Generation Commands

Command	**Acknowledgments**
Description	
GDS [DEVPROP | NETPROP | PLACEPROP] NUMBER *number*	OK
Description: Changes the value of the PROPATTR value inserted in the GDSII output of the GDS WRITE command. GDS NETPROP NUMBER changes the PROPATTR value for net numbers, GDS PLACEPROP NUMBER changes the PROPATTR value for placement names. GDS DEVPROP NUMBER changes the PROPATTR value for device names. Notes: a) Default PROPATTR values are 1 for device IDs, net IDs and placement names.	
GDS RESET	OK
Description: Causes default settings to take effect for the GDS WRITE command. Notes: a) Affects the GDS MAP, GDS NETPROP, and GDS PLACEPROP commands. b) Does not affect the MAXIMUM VERTEX COUNT.	
GDS SEED PROPERTY {[DEVICE] [ORIGINAL]}	OK
Description: Given the DEVICE and/or ORIGINAL argument, causes either the original database shapes (without device annotations) or the recognized seed shapes (with device annotations) to be written. Both arguments cause both layers to be written on different output layers. Notes: a) Due to seed promotion, original layer geometries may occur at a different level than the recognized seed shapes. b) If net ids are available on the ORIGINAL layer, they will be annotated using the NETPROP annotations.	

Table 15-23. Annotated GDSII Generation Commands

Command	**Acknowledgments**
Description	
GDS WRITE *file_path*	OK NOK(33), ERROR(102), ERROR(103), ERROR(130)
Description: Writes annotated layout geometry in GDSII format to the specified file. Filters: FILTER LAYERS, FILTER DEVICES/DEVICENAMES Response: Writes an Annotated GDSII File (AGF) Notes: a) Net numbers are written to GDSII property fields using PROPATTR and PROPVALUE records on BOUNDARY elements representing geometry contained in the design. b) Device ids are written to GDSII property fields using PROPATTR and PROPVALUE records on BOUNDARY elements representing seed geometry for the device. c) Placement names are written to GDSII property fields using PROPATTR and PROPVALUE records on SREF elements representing cell placements in the design. d) The FILTER DEVICES and FILTER LAYERS filters both apply to which device seed shapes are written to the file. e) Details of the AGF format are provided below. f) By default, the net, device, and placement number PROPATTR value is 1. The value for attributes may be changed using the GDS NETPROP NUMBER, GDS DEVPROP and GDS PLACEPROP commands. g) By default, the maximum vertex count is 200 (same as GDSII standard). It may be adjusted up or down with the MAXIMUM VERTEX COUNT command. h) Requires calibreci license in addition to calibreqdb license.	

Table 15-23. Annotated GDSII Generation Commands

Command	Acknowledgments
Description	
LAYOUT NAMETABLE WRITE ***file_path***	OK NOK(33), ERROR(102), ERROR(103), ERROR(130)
Description: Writes Layout Netlist Names (LNN) file describing correspondence of generated net numbers to original layout names. Response: Writes Layout Netlist Names file (discussed below) Notes: a) Details of the Layout netlist names format are provided below. b) The command may generate warnings as it executes. These transcript to the standard out and are prefixed by the string “WARNING.” c) Requires calibreci license in addition to calibreqdb license.	
PORT TABLE [CELLS] WRITE ***file_path***	OK ERROR(101), ERROR(102), ERROR(130)
Description: Writes a port table file. Response: Without the CELLS option, writes a port table file for the top-level ports in the design (discussed below). If the CELLS option is specified, writes a port table file for ports in all cells in the design. Notes: a) Requires calibreci license in addition to calibreqdb license.	

Annotated GDSII File (AGF) Format

This file provides geometric and connectivity information for the design.

This file is a GDSII file that uses the GDSII PROPERTY record as a mechanism for attaching net numbers to geometry (BOUNDARY) records and placement names to instance (SREF or AREF) records stored in the file.

The AGF file is designed to be used in conjunction with a layout netlist and LNN file generated in the following manner:

```
LAYOUT NETLIST NAMES NONE
LAYOUT NETLIST TRIVIAL PINS YES
LAYOUT NETLIST EMPTY CELLS YES
LAYOUT NETLIST WRITE <filename.net>
LAYOUT NETLIST NAMES WRITE <filename.lnn>
```

The layout netlist is required in order to follow nets through levels of hierarchy in the AGF.

The Annotated GDSII File contains net numbers but does not contain net names, even for texted nets. The correspondence between layout net numbers and layout net names (for texted nets) is given in the Layout Net Name file (LNN).

The Annotated GDSII File consists of a collection of GDSII Records in the following format:

A HEADER record.
A BGNLIB record including a current timestamp.
A LIBNAME record (the name is always lvs.db).
A UNITS record (based on rule file PRECISION and UNIT LENGTH parameters.

For each Cell in the design:

a) A BGNSTR record with current time stamp.
b) A STRNAME record with the name of the current cell.
c) Cell instances [optional]. Also called “cell placements” or “placements”. Cell instances are presented as GDSII SREF elements with cell names written as STRNAME records. Transformation information is included in an STRANS record if applicable. Location information for the placement is written as an XY record. The range of instance numbers (including device and cell instance numbers) is well-behaved. Instance numbers normally begin with a zero, tend to be consecutive, and are usable as array indices.
d) Placement names are written as GDSII PROPATTR and PROPVALUE records. These placement names correspond with the

placement names written in the Layout Netlist file, for example “X3”. (The names do not correspond when LAYOUT NETLIST NAMES SOURCE is specified before generating the layout netlist).

e) [Optional] Polygons are written as GDSII BOUNDARY records. Polygons on all layers of interest are present. Layers of interest are specified using the Query Server’s FILTER LAYERS command. The layers of interest must be layers that are saved in the PHDB database, that is, layers that appear in Connect and Sconnect operations, serve as Device seed or pin layers, or are target layers of Stamp operations (second argument of Stamp). Layers that are not in one of these sets are not available. These BOUNDARY records include LAYER and DATATYPE records which can be controlled with the Query Server’s GDS MAP command.

f) An XY record represents coordinate information as part of the BOUNDARY element. An array of (X,Y) locations in database units, in the coordinate space of the cell. Maximum number of XY pairs can be controlled by the Query Server’s MAXIMUM VERTEX COUNT command.

g) The PROPATTR and PROPVALUE fields in the BOUNDARY record are used to represent net number (also called node number) in layers that are not device seed layers. These records are present if this polygon belongs to a layer that carries connectivity. These net numbers correspond to the net numbers specified in the layout netlist when the LAYOUT NETLIST NAMES NONE option is used. The PROPATTR attribute number may be specified using the Query Server’s GDS NETPROP NUMBER command. The PROPVALUE record indicates the number of the electrical net to which the polygon belongs in the cell. Net numbers are local to the cell and are unique within the cell. Net numbers are assigned by Calibre LVS during circuit extraction.

h) Some nets in the AGF may be omitted from the layout netlist. Specifically, nets that are not connected to devices nor to placement pins and that do not serve as pins of the cell that contains them are omitted.

i) Most nets are represented by simple net numbers. The range of these net numbers is well-behaved. That is, net numbers normally start with 1, tend to be consecutive, and are safe for use as array indices.

j) Sometimes nets and instances are represented by short hierarchical

strings which represent entities at lower levels of hierarchy which are not present in the AGF file. For example, X3/2 or X4/X1/3. Each of these strings represents a net unique in the cell. These hierarchical paths come from expanded cells as described below.

k) In device seed layers, the PROPATTR and PROPVALUE fields represent the device ID string as seen in the layout netlist file. (Note that device ID annotations are available only if the Mask SVDB Directory statement in the rule file contains the CCI option or the ANNOTATE DEVICES option). The PROPATTR attribute number may be specified using the Query Server's GDS DEVPROP NUMBER command. The PROPVALUE record indicates the device ID associated with the seed shape as assigned by Calibre LVS during circuit extraction.

l) Devices are represented by the same types of strings that are used in the Spice netlist. For example "M0", "D1", "C3". Like net numbers, they may be represented by hierarchical pathnames like "X1/M3".

- Expanded Cells

  Certain cell placements are expanded in place in the AGF file output. When this occurs, geometry from the expanded placement is transformed upward into the containing cell. If geometry is connected to a net within the containing cell, the higher level net number is used in the PROPVALUE record for each transformed polygon. If the net does not propagate upward from the placement, it is represented with a hierarchical string of the form "Xn/[[Xm/]...]i. These paths may be used in conjunction with the cross reference files described below. They are formed using placement numbers available in the LPH file. Only cells generated by Calibre during circuit extraction are expanded in this manner.

Layout Netlist Names (LNN) File Format

The Layout Net Name file provides correspondence between net numbers and user-specified net names, for texted layout nets. It is intended for use with the LAYOUT NETLIST NAMES NONE option to the LAYOUT NETLIST WRITE command and the AGF file by tying net numbers in the AGF file and netlist to layout net names specified using text in the layout database. The file begins with a number of "SVDB" header lines indicated with #. The header format is described

later in this section. Following that there is a section for each cell in the layout hierarchy (not just corresponding cells or hcells, but all cells). The first line in each section consists of a percent “%” sign, a space, the cell name, a space, and finally the number of pins of the cell. The format is:

```
% cell_name pin_count
```

Each of the following lines represents a net within the current cell and consists of the net number (local to the cell), a space, and the user-specified net name. The format is:

```
number name
```

Only texted nets have entries in the Layout Net Name file.

In summary, the format of the Layout Net Name file is this:

```
# SVDB: header_line
# SVDB: header_line
....
# SVDB: header_line
% cell_name pin_count
number name
number name
....
....
....
....
```

Cells with no named nets will appear in the file with no number-name pairs following. Cells that have been renamed in the netlist due to the presence of a “$” character at the beginning of the cell name will have the modified name printed inside brackets at the end of the line.

Here's an example of a Layout Net Name file:

```
# SVDB: ....
....
# SVDB: ....
% NAND 5
4 I1
5 I2
6 OUT
% TOP 0
2 VDD
3 VSS
17 CLOCK
% $VIA2 1 [__$VIA2]
```

The Layout Net Name file is required in order to translate layout net numbers (of texted layout nets) into net names for lookup in the hierarchical Net Cross Reference file (NXF).

Port Table File Format

The port table identifies locations of identified ports in the design. The file contains one line for each port. (Unattached ports are not output). Each line has the following fields:

port-name node-number node-name location layer-attached [cell-name]

Where:

port-name—layout name of the port object, for example the GDSII text string when using Port Layer Text, or <UNNAMED> if the port is not named.

node-number—layout node number to which the port is connected.

node-name—layout node name to which the port is connected, or layout node number if the node is unnamed.

location—in the form: X Y; in database units. This is the location of the database text object when using Port Layer Text, or a vertex on the port polygon marker when using Port Layer Polygon.

layer-attached—layer of the polygon to which the port got attached. Rule file layer name or rule file layer number if the layer is unnamed. This layer appears in a Connect or Sconnect operation.

cell-name—cell name field is present only when the file is produced by the PORT TABLE CELLS WRITE command. It specifies the name of the cell in which the port resides. The PORT TABLE WRITE command only writes ports from the top-level cell.

Examples:

PORT TABLE WRITE *file_path* may produce the following output:

```
CONF3 5 CONF -98000 -90000 metal
CONF3 5 5 -98000 -90000 metal
<UNNAMED> 5 5 -98000 -90000 metal
CONF 5 CONF -98000 -90000 17
```

PORT TABLE CELLS WRITE *file_path* may produce the following output:

```
CONF3 5 CONF -98000 -90000 metal cellA
CONF3 5 5 -98000 -90000 metal cellA
<UNNAMED> 5 5 -98000 -90000 metal cellB
CONF 5 CONF -98000 -90000 17 CHIP
```

Cross Reference File Generation

The following table discusses commands used in cross reference file generation:

Table 15-24. Cross Reference File Generation Commands

Command	**Acknowledgments**
Description	
INSTANCE XREF WRITE ***file_path***	OK NOK(33), ERROR(101), ERROR(102)
Description: Writes instance correspondence as determined by LVS. Response: Writes an instance cross reference file Notes: a) Details of the Instance Cross Reference format are provided below. b) The format and content of this file is similar to the one generated by the calibre -ixf switch and the “Mask SVDB Directory *file* IXF” SVRF rule file statement. c) Requires calibreci license in addition to calibreqdb license.	
{LAYOUT | SOURCE} HIERARCHY WRITE ***file_path***	OK NOK(33), ERROR(101), ERROR(102)
Description: writes Source or Layout Placement Hierarchy file describing original netlist placement hierarchy. Response: Writes a Placement Hierarchy file. Notes: a) Details of the Placement Hierarchy format are provided below. b) A similar file is generated by Calibre when the Mask SVDB Directory <dir> SLPH option is set in the rule file. c) Requires calibreci license in addition to calibreqdb license.	

Table 15-24. Cross Reference File Generation Commands

Command	Acknowledgments
Description	
NET XREF WRITE *file_path*	OK NOK(33), ERROR(101), ERROR(102)
Description: Writes net correspondence as determined by LVS. Response: Writes a net cross reference file Notes: a) Details of the Net Cross Reference format are provided below. b) The format and content of this file is similar to the one generated by the calibre -nxf switch and the “Mask SVDB Directory *file* NXF” SVRF rule file statement. c) Requires calibreci license in addition to calibreqdb license.	

Cross Reference System File Formats

In this section we describe how individual instances and nets are identified in the IXF, NXF, LPH and SPH cross reference files. Untexted layout nets are identified with their net number, for example, 17. Texted layout nets are identified with their user-specified names, for example, CLOCK. Layout cell instances are identified with the letter X followed by the instance number, for example, X25. Layout device instances are identified with a letter designating their device type (for example M, R, C, D, Q, X, and so on), followed by the instance number, for example, M5. Source instances and nets are identified with names as they appear in the original source netlist. Note that this identification scheme is somewhat different from the one used in the AGF and the Layout Netlist. In particular, note that all nets in the AGF and the Layout Netlist are identified solely by their numbers. The correspondence between numbers and user-specified names, for texted layout nets, is provided in the Layout Net Name file (LNN).

- Placement Hierarchy (LPH/SPH) File Format

  The layout placement hierarchy file (LPH) and the source placement hierarchy file (SPH) describe the hierarchical structure of the layout and source respectively. They map cell instances to cell names in the layout and

source. These files complement the IXF and NXF cross reference files. Note: the terms “placement” and “instance” are used interchangeably in this discussion.

Each placement hierarchy file begins with a number of “SVDB” header lines indicated with #. The header format is described later in this document. Following that there is a section for each cell in the hierarchy (not just corresponding cells or hcells, but all cells). The first line in each section consists of a percent “%” sign, a space, the cell name, a space, and finally the number of pins of the cell. The format is:

% cell_name pin_count

Each of the following lines represents a cell instance (placement) or device instance within the current cell and consists of the instance identifier (local to the cell), a space, the name of the cell or device being instantiated, a space and the number of pins on the instance. The format is:

instance_identifier cell_or_device_name number_of_pins

Note that only cells are represented by sections in the file. Devices are not. Note also that no connectivity information is present other than pin counts. Also, cells that do not contain devices or placements of other cells (that is, vias) are not represented.

In summary, the file format is this:

```
# SVDB: header_line
# SVDB: header_line
....
# SVDB: header_line
% cell_name pin_count
instance_identifier cell_or_device_name number_of_pins
instance_identifier cell_or_device_name number_of_pins
....
%cell_name pin_count
instance_identifier cell_or_device_name number_of_pins
instance_identifier cell_or_device_name number_of_pins
....
```

....
....

Example:

```
# SVDB: ....
....
# SVDB: ....
% NAND 5
M1 MP 4
M2 MP 4
M3 MN 4
M4 MN 4
% TOP 0
X1 NAND 5
X2 NAND 5
```

- Net/Instance Cross Reference File (IXF/NXF) File Format

  There are two hierarchical cross reference files:

  IXF: Instance Cross Reference File
  NXF: Net Cross Reference File

  Each hierarchical cross reference file begins with a number of “SVDB” header lines indicated with #. The header format is described later in this document. The remainder of each file contains sections for individual correspondence cells or “hcells.” These cells exist in both layout and source netlists. There is one section for each hcell. Each hcell section begins with a % line specifying the layout cell name and pin count and the corresponding source cell name and pin count. This is followed by lines of corresponding layout-source elements in the cell. The general structure of the file is shown below.

  # SVDB: header_line
  # SVDB: header_line
  ....
  # SVDB: header_line
  % layout_cell layout_pin_count source_cell source_pin_count layout_id layout_path source_id source_path

layout_id layout_path source_id source_path
....
% layout_cell layout_pin_count source_cell source_pin_count
layout_id layout_path source_id source_path
layout_id layout_path source_id source_path
....
....

The layout_path and source_path fields contain hierarchical pathnames rooted in the particular hcell that owns them. Note that the IXF and NXF files produced by the Query Server always have “layout ID” and “source id” values of 0. More information about individual instance and net lines is provided below.

- Instance Cross Reference File (IXF) Line Format

  The instance cross reference file contains matched instances. This includes device instances as well as cell instances. There is one line per instance in the following form:

  layout_id layout_path source_id source_path [SL | SS] [X]

  The layout_path and source_path fields identify corresponding layout and source instances. Instances are identified by hierarchical pathnames rooted in the particular hcell to which this line belongs. A instance pathname consists of zero or more cell instance identifiers, followed by a device instance identifier, separated by “/” characters. The layout_id and source_id fields are for internal use and should be ignored. In particular, note that the layout_id field does not contain layout instance numbers. For example:

  ```
  % ALU 25 ALU 25
  0 X2/X3/M1 0 MFOO
  0 D2 0 X3/X1/D8
  ```

  Reduced devices (such as those created by series or parallel device reduction) are represented in the file by all their original constituents. The original devices appear in consecutive lines in the file, with the corresponding device from the other design repeated in each line. When

a reduced layout device is matched to a reduced source device, then all the original layout devices are listed in consecutive lines on the left, with a representative original source device repeated on the right; all the other original source devices are listed in consecutive lines on the right, with a representative original layout device repeated on the left. The representative devices are chosen arbitrarily. Reduced layout devices (other then the representative) are indicated with the string “SL” which stands for “smashed layout.” Reduced source devices (other then the representative) are indicated with the string “SS” which stands for “smashed source.” For example:

```
0 R10 0 R1
0 R11 0 R1 SL
0 R10 0 R2 SS
0 R10 0 R3 SS
```

In this example, layout devices R10 and R11 were reduced to a single device, and correspond to source devices R1, R2 and R3 that were also reduced to a single device. Layout device R10 and source device R1 were chosen as representatives.

MOS devices with swapped source/drain pins are indicated with the letter X. For example:

```
0 M10 0 M1 X
```

X indicates that the source pin of the layout device corresponds to the drain pin of the source device and vice versa. Lines that represent reduced devices (SL or SS) have correct X values as well, with respect to the two devices reported on that particular line. LVS logic gates are represented by the original transistors that form them. All matched transistors in the layout gate are listed, along with the corresponding transistors in the source gate.

- Net Cross Reference File (NXF) Line Format

  The net cross reference file contains matched nets. There is one line per net in the following form:

  layout_id layout_path source_id source_path

The layout_path and source_path fields identify corresponding layout and source nets. Nets are identified by hierarchical pathnames rooted in the particular hcell to which this line belongs. A net pathname consists of zero or more cell instance identifiers, followed by a net identifier, separated by "/" characters. The layout_id and source_id fields are for internal use and should be ignored. In particular, note that the layout_id field does not contain layout net numbers. For example:

```
% ALU 25 ALU 25
0 X2/X3/7 0 SIG1
0 13 0 X3/X1/8
```

In certain situations, LVS may match several layout nets to one source net, or several source nets to one layout net, or a group of several layout nets (together) to a group of several source nets. This may occur, for example, in split gate reduction or when LVS detects an open circuit or short circuit discrepancy. In these cases, a representative net is chosen for the layout side and a representative net is chosen for the source side. The representative pair appears in the net cross reference file. In addition, each of the remaining layout nets appears with the source representative, and each of the remaining source nets appears with the layout representative. In the following example, layout nets 1, 2 and 3 were matched (as a group) to source nets n1, n2 and n3.

```
0 1 0 n1
0 1 0 n2
0 1 0 n3
0 2 0 n1
0 3 0 n1
```

Note that some nets are unmatched even when LVS is successful. Examples of nets that are never matched are: nets internal to certain types of logic gates formed by LVS; nets removed because of series device reduction. Unmatched nets do not appear in the net cross reference file.

- SVDB Header Description

  As mentioned, the IXF, NXF, LPH and SPH files each begin with a number of SVDB header lines. The acronym SVDB stands for Standard

Verification Data Base. The header identifies the type of file and the source of the information used to create the file. The header consists of 10 lines. Each line begins with the string "# SVDB:". Here is an example of the SVDB header from a IXF file:

```
# SVDB: Instance Cross Reference (ixf) (File format 1)
# SVDB: Layout Primary mix
# SVDB: Rules -0 play.rules Wed Dec 10 10:07:38 1997
# SVDB: GDSII -0 (none) (none)
# SVDB: SNL -0 (none) (none)
# SVDB:
# SVDB:
# SVDB:
# SVDB:
# SVDB: End of header.
```

The first line in the header identifies the type of file and its format version and is the only line which differs between files representing the same design. Here are the first lines from each of the four different files mentioned above:

```
#SVDB: Layout Placement Hierarchy (lph) (File format 1)
#SVDB: Source Placement Hierarchy (sph) (File format 1)
#SVDB: Instance Cross Reference (ixf) (File format 1)
#SVDB: Net Cross Reference (nxf) (File format 1)
```

The second line in the header gives the name of the top (primary) cell of the design. The third, fourth, and fifth lines identify the Calibre rule file, layout GDSII file, and source netlist file used in creating the file. The entries on these lines are the type of file (Rules, GDSII or SNL respectively, the latter standing for Source NetList) followed by a checksum for the file (or -0 if no checksum is present), the pathname to the file at the time the information was captured (or "(none)" if a pathname is not present) and a time stamp of the file which was used (or "(none)" if a time stamp is not present). A time stamp, when specified, is of the form:

```
week_day month day hh:mm:ss year
```

- Step by Step Example

  The following example describes a step by step process for generating full CCI output from Calibre.

  1) Ensure that the calibre rule file includes the line Mask SVDB Directory “output_dir” CCI

  2) Run hierarchical Calibre LVS.

  3) Run the Calibre Query Server on the SVDB directory generated during step 1.

  Use the following commands to generate CCI output from the Query Server:

```
###################################
##### Generate the Annotated GDSII File
###################################
### set the value used for the PROPATTR record in GDSII
### BOUNDARY element to indicate node id
gds netprop number 5

### set the value used for the PROPATTR record in GDSII
### SREF element to indicate placement name
gds placeprop number 6

### set the value used for the PROPATTR record in GDSII
### BOUNDARY element to indicate device name
gds devprop number 7

### Set layer name to number mappings if desired
gds map LayerRed 2
gds map BlueSeed 3
gds map GreenPin 4

### write out the gds map showing layer name to number
### mappings
response file output/gds_map
gds map
response direct
```

```
### write the agf file in annotated GDSII format
gds write output/svdb.agf

###################################
##### Generate the Layout Netlist
###################################
### Include trivial pins and empty cells in the layout
### netlist
layout netlist trivial pins YES
layout netlist empty cells YES

### Use only node numbers for netlists
layout netlist names NONE

### write the layout netlist
layout netlist write output/svdb.spi

#######################################
##### write the net to name mapping file (LNN)
#######################################
layout nametable write output/svdb.lnn
#######################################
###### Generate Cross Reference Information
#######################################
### write the source and layout placement hierarchy
### files (sph,lph)
### not necessary if the Mask SVDB Directory <dir> SLPH
### option is used
source hierarchy write output/svdb.sph
layout hierarchy write output/svdb.lph

### write the net and instance cross reference
### Mask SVDB Directory <dir> NXF IXF options are used
net xref write output/svdb.nxf
instance xref write output/svdb.ixf

#######################################
##### Generate the Port Table File
#######################################
### write the port table
port table write output/svdb.ports
```

Query Server Error and Failure Messages

Error Messages

Table 15-25 describes the errors that may be returned as acknowledgments. Errors generally involve avoidable problems, such as unknown commands, missing arguments, or environment failures.

Table 15-25. Error Messages

Message	Description
ERROR(101)	File *response_file_path* could not be opened for writing.
ERROR(102)	Problems writing to file *response_file_path*.
ERROR(103)	There is no layout cell named *cell_name*. (Current context remains unchanged.)
ERROR(104)	Layout cell *viewing_cell* has no instance *query_instance*. (Current context remains unchanged.)
ERROR(105)	Due problems parsing cross-reference files, the Query Server cannot be initialized.
ERROR(106)	This client id is negative or malformed: *client_id*.
ERROR(107)	There is no client with id: *client_id*.
ERROR(108)	Client 0 can never be disconnected.
ERROR(109)	The active client cannot be disconnected.
ERROR(110)	Marker size value is negative or malformed: *size*.
ERROR(111)	Unknown command or wrong number of arguments.
ERROR(112)	Command not yet implemented.
ERROR(113)	Filter distance is negative or malformed: *distance*.
ERROR(114)	Unknown layer name or number: *layer_name_or_number*.
ERROR(115)	Unknown device type number: *device_type_number*.

Table 15-25. Error Messages [continued]

Message	Description
ERROR(116)	The x y location is malformed: *coordinate*.
ERROR(117)	Invalid or malformed window coordinates: *x1 x2 y1 y2*
ERROR(118)	Invalid or malformed cull distances: *x y*
ERROR(119)	The maximum vertex count was malformed or less than 4.
ERROR(120)	Query instance argument *query_instance* not valid in XDB only mode.
ERROR(121)	Zoom factor is negative or malformed: *z_factor* (RVE only).
ERROR(122)	Command not available in FLAT mode.
ERROR(123)	Invalid arguments to GDS MAP [gdsii_layer][gdsii_datatype].
ERROR(124)	Value specified <value> is not representable in GDSII.
ERROR(125)	Invalid cell name/pin count combination: <layout_cell> <pin_count> <source_cell> <pin_count>
ERROR(126)	Unknown device element name: <name>
ERROR(127)	(not currently used).
ERROR(128)	Layout netlist hierarchy must be FLAT when layout netlist source is specified.
ERROR(129)	SVDB Database revision does not support this feature, rerun Calibre.
ERROR(130)	Incorrect MASK SVDB DIRECTORY options for this function.
ERROR(131)	CALIBRE CONNECTIVITY INTERFACE license is unavilable.
ERROR(132)	Delta XY setting is malformed <x> <y>.

Table 15-25. Error Messages [continued]

Message	Description
ERROR(133)	Must specify DEVICE or ORIGINAL layer.

Failure Messages

Table 15-26 describes the failures that may be returned as acknowledgments. They indicate the command (usually a request for design information) was performed but failed to produce the requested response.

Table 15-26. Failure Messages

Message	Description
NOK(1)	There are no unfiltered pins on layout net *layout_net_path*.
NOK(2)	No source cell corresponds to layout cell *cell_name*.
NOK(3)	Either layout cell *layout_cell* has no net *layout_net_path* or it was removed before comparison.
NOK(4)	Either source cell *source_cell* has no net *source_net_path* or it was removed before comparison.
NOK(5)	Layout cell *query_cell* has no net named *layout_net_path*.
NOK(6)	No device of the filtered type found within the filter distance.
NOK(7)	Either layout cell *query_cell* has no device *layout_device_path* or it was filtered.
NOK(8)	Either source cell *source_cell* has no device *source_dev_path* or it was filtered.
NOK(9)	Layout cell *query_cell* has no device *layout_device_path*.
NOK(10)	Layout cell *query_cell* has no ports.
NOK(11)	Layout cell *query_cell* has no port named *port_name*.

Table 15-26. Failure Messages [continued]

Message	Description
NOK(12)	There are no ports on layout net *layout_net_path*.
NOK(13)	No port found on the filter layers.
NOK(14)	No net found on the filter layers.
NOK(15)	Flattened layout cell *query_cell* has no ports.
NOK(16)	Layout cell *query_cell* has no nets.
NOK(17)	Flattened layout cell *query_cell* has no nets.
NOK(18)	Layout cell *query_cell* has no devices.
NOK(19)	Flattened layout cell *query_cell* has no devices.
NOK(20)	Layout cell *query_cell* has no bad devices.
NOK(21)	Cross-reference commands are disabled due to missing cross-reference.
NOK(22)	Layout cell *query_cell* has no placements.
NOK(23)	Flattened layout cell *query_cell* has no placements.
NOK(24)	No placements were selected.
NOK(25)	Layout cell *query_cell* has no such placement.
NOK(26)	No layout cell corresponds to source cell *cell_name*.
NOK(27)	No LVS report was specified in the rules.
NOK(28)	No SVDB directory was specified in the rules.
NOK(29)	Layout cell *query_cell* has no unfiltered shapes on net *net_names*.
NOK(30)	No source directory was specified in the rules.
NOK(31)	No layout file was specified in the rules.
NOK(32)	Layout cell *query_cell* has no placement *placement_name*.

Table 15-26. Failure Messages [continued]

Message	Description
NOK(33)	Interrupted.
NOK(34)	Layout cell *query_cell* had no unfiltered devices on net *net_name*.
NOK(35)	Layout cell *query_cell* had no net *net_name* extending into *placement_name*.
NOK(36)	There is no layout cell named *target_cell*.
NOK(37)	Target net *target_net* must be a top level net of *target_cell*.
NOK(38)	Layout cell *target_cell* is topologically higher than net *reference_net*.
NOK(39)	No placement of *target_cell* lies within the current query context.
NOK(40)	Layout database query commands are disabled due to missing PHDB database.
NOK(41)	*source_device* was not matched to a layout device.
NOK(42)	*layout_device* was not matched to a source device.
NOK(43)	File name not available.
NOK(44)	Net *source_net_path* was not matched to a layout net.
NOK(45)	Net *layout_net_path* was not matched to a source net.
NOK(46)	Layout cell <cell_name> has no [INVALID] net texts.
NOK(47)	Layout cell <cell_name> has no [INVALID] port texts.

When the Query Server or RVE restore the PHDB database, the SVRF rule file stored in the PHDB is recompiled. There are circumstances that will cause this compilation to fail even when it was successful during the original Calibre run.

The query Server and RVE issue a diagnostic message when the rule file compilation fails similar to:

```
RESTORATION OF PHDB FAILED.
PHDB STATUS IS -13
Rules file is invalid or incomplete.
Error ENV1 on line 233 of rules1.4497 - undefined or
empty environment variable: DESIGN_DIR.
```

Appendix A
Application Notes

This appendix lists the application notes available for the Calibre Verification, Calibre RET, and xCalibre products.

Application notes are created outside of the documentation group, as an addendum to the manual. Normally, they provide procedural information to a depth greater than can be covered within the product manuals. Application notes may not include the latest functionality of the toolset.

The following list shows the titles of the application notes available. Following the list of titles there are instructions on accessing these documents.

- *Antenna Rule Checks with Calibre DRC*
- *Bent Gate Device Extraction With Calibre*
- *Debugging Electrical Shorts*
- *How to Create Scattering Bars Using Calibre*
- *Single-Layer Runs with Calibre DRC*
- *What to Look for in the Calibre LVS Transcript*
- *Calibre RVE Debugging of HLVS Results*
- *Multithreaded Hierarchical Calibre Performance Characteristics*
- *Calibre RVE Debugging of HLVS Results*
- *File Input/Output of Calibre DRC*

- *Conditionals and Scripting in SVRF*
- *Calibre Architectural Features for High Performance*
- *Advanced Rule-Based OPC with Calibre SVRF*
- *Common situations encountered with Calibre LVS*
- *Converting Dracula Antenna Checks to Calibre*
- *Dracula ERC Translations*
- *Writing OPC Rules with Calibre DRC*
- *How Calibre LVS Uses Text*
- *Benchmarking and Optimizing Calibre DRC*
- *Converting Dracula LVS Files to Calibre*
- *Post Processing the Dracula Rule File Converter*
- *xCalibrate and Local Interconnect*
- *xCalibrate and Planar/nonPlanar Processes*

You can obtain these application notes by navigating with a web browser to the Mentor Graphics *Customer Access* site. If your PDF viewer is connected to your web browser, the following address will take you directly to the application note site, however, if your PDF viewer is not connected, you can enter the address manually.

http://www.mentor.com/dsm/customer/appnotes

Customer Access is a website designed specifically for Calibre customers to bring you the latest technical information and software updates.

This information is part of your support contract and you must be a registered user to access the website. To obtain your login, please submit this “sign up” webform.

http://www.mentor.com/dsm/cust_signup.cfm

You will receive your login within one business day.

Appendix B
Calibre Interactive Files

This appendix contains files which are useful for configuring and using Calibre Interactive.

Runset File Example

Runsets are ASCII files that set up Calibre Interactive for a Calibre run. They contain only information that differs from the default configuration of Calibre Interactive. There is a one-to-one correspondence between entry lines in the runset file and fields and button items in the Calibre Interactive user interface. Here is as example of a DRC runset:

```
*drcRulesFile: rule_file
*drcRulesFileLastLoad: 1009224452
*drcLayoutPaths: ./lab3.gds
*drcLayoutPrimary: lab3
*drcResultsFile: ./lab3.db
*drcSummaryFile: drc_report
*drcRunTurbo: 0
*drcRunRemoteOn: Cluster
*drcRemoteLICENSEFILEName: MGLS_LICENSE_FILE
*drcRemoteLICENSEFILEValue: /scratch1/mgls/mgclicenses
*drcDontWaitForLicense: 0
```

Default Configuration

The default settings for Calibre Interactive are stored in the file $MCG_HOME/pkgs/icv/userware/default/cgi/options.tcl. There is a one-to-one

correspondence between entry lines in this file and fields and button items in the Calibre Interactive user interface. Here is a copy of the settings section of that file:

```
#
#     Copyright Mentor Graphics Corporation 2002
#             All Rights Reserved.
#  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
#     WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION
#       OR ITS LICENSORS AND IS SUBJECT TO LICENSE TERMS.
#
#

namespace eval options {
    variable opts_vars

    namespace export init readFile saveFile areOptionsChanged
resetOptions

    set opts_vars(drc_opts) [list \
        [list RulesFile             "rules"] \
        [list RulesFileLastLoad     0] \
        [list RunDir                "."] \
        [list LayoutPaths           ""] \
        [list LayoutSystem          GDSII] \
        [list LayoutPrimary         ""] \
        [list LayoutGetFromViewer   0] \
        [list ResultsFile           drc.results] \
        [list ResultsFormat         ASCII] \
        [list CellName              0] \
        [list CellNameSpace         1] \
        [list CellNameXform         1] \
        [list CellNameAll           0] \
        [list CellText              0] \
        [list DRCMaxResultsAll      0] \
        [list DRCMaxResultsCount    1000] \
        [list AutoHalo              1] \
        [list HaloSize              0.0] \
        [list LayoutPrecision       1000] \
        [list StartRVE              1] \
        [list WriteSummary          1] \
        [list SummaryFile           drc.summary] \
        [list AppendSummary         0] \
        [list ViewSummary           1] \
```

```
    [list RunHier               1] \
    [list Run64                 0] \
    [list RunTurbo              1] \
    [list UseAllProcessors      1] \
    [list NumProcessors         ""] \
    [list TranscriptFile        ""] \
    [list TranscriptFileAppend  0] \
    [list TranscriptEchoToFile  0] \
    [list ShowChecksAlpha       0] \
    [list ShowGroupsAlpha       0] \
    [list ShowGroupsHier        0] \
    [list ShowGroupsTop         0] \
    [list DontWaitForLicense    1] \
    [list DRCCheckArea          0] \
    [list DRCAreaCoords         ""] \
    [list IncludeFiles          ""] \
    [list EnvVars               ""] \
    [list Template_LP           "%l.gds"] \
    [list Template_RD           "%l.drc.results"] \
    [list Template_SF           "%l.drc.summary"] \
    [list Template_IL           1] \
]

set opts_vars(lvs_opts) [list \
    [list RulesFile             "rules"] \
    [list RulesFileLastLoad     0] \
    [list RunDir                "."] \
    [list LayoutPaths           ""] \
    [list LayoutSystem          GDSII] \
    [list LayoutPrimary         ""] \
    [list LayoutGetFromViewer   0] \
    [list LayoutPrecision       1000] \
    [list SourcePath            ""] \
    [list SourceSystem          SPICE] \
    [list SourcePrimary         ""] \
    [list SourceGetFromViewer   0] \
    [list ReportFile            lvs.report] \
    [list ViewReport            1] \
    [list CreateSVDB            1] \
    [list SVDBContents          QUERY] \
    [list SVDBphdb              0] \
    [list SVDBxdb               0] \
    [list SVDBinxf              0] \
```

```
[list SVDBslph              0] \
[list SVDBcci               0] \
[list SVDBbygate            0] \
[list SVDBanndev            0] \
[list SVDBdv                0] \
[list SVDBDir               svdb] \
[list SVDBNoflat            0] \
[list StartRVE              1] \
[list WriteMaskDB           0] \
[list MaskDBFile            maskdb] \
[list WriteINXF             0] \
[list WriteBPF              0] \
[list WriteNL               0] \
[list RunHier               1] \
[list RunWhat               LVN] \
[list SpiceFile             lay.net] \
[list AutoMatch             0] \
[list UseHCells             0] \
[list HCellsFile            hcells] \
[list Run64                 0] \
[list RunTurbo              1] \
[list UseAllProcessors      1] \
[list NumProcessors         ""] \
[list TranscriptFile        ""] \
[list TranscriptFileAppend  0] \
[list TranscriptEchoToFile  0] \
[list ReportMaximumAll      0] \
[list ReportMaximumCount    50] \
[list AbortOnSoftchk        0] \
[list AbortOnSupplyError    1] \
[list IgnorePorts           0] \
[list PowerNames            ""] \
[list GroundNames           ""] \
[list ShowSeedPromotions    0] \
[list ShowSeedPromotionsMax 50] \
[list RecognizeGates        ALL] \
[list RecognizeGatesMixSubtypes  0] \
[list RunERC                1] \
[list ERCDatabase           ""] \
[list ERCSummaryFile        ""] \
[list ERCAppendSummary      0] \
[list ERCMaxResultsAll      0] \
[list ERCMaxResultsCount    1000] \
```

```
    [list ERCMaxVertexAll       0] \
    [list ERCMaxVertexCount     4096] \
    [list IsolateShorts         0] \
    [list IsolateShortsByLayer  0] \
    [list IsolateShortsFlat     0] \
    [list IsolateShortsInTopCell    1] \
    [list IsolateShortsOp       AND] \
    [list IsolateShortsAllNames 1] \
    [list IsolateShortsNames    ""] \
    [list VConnectColon         0] \
    [list VConnectReport        0] \
    [list VConnectNamesState    "NONE"] \
    [list VConnectNames         ""] \
    [list VConnectBoxColon      0] \
    [list VConnectBoxNamesState "NONE"] \
    [list VConnectBoxNames      ""] \
    [list ShowChecksAlpha       0] \
    [list ShowGroupsAlpha       0] \
    [list ShowGroupsHier        0] \
    [list ShowGroupsTop         0] \
    [list DontWaitForLicense    1] \
    [list IncludeFiles          ""] \
    [list EnvVars               ""] \
    [list Template_LP           "%l.gds"] \
    [list Template_SP           "%s.src.net"] \
    [list Template_LR           "%l.lvs.report"] \
    [list Template_ES           "%l.lay.net"] \
    [list Template_ED           "%l.erc.results"] \
    [list Template_EU           "%l.erc.summary"] \
    [list Template_SV           "svdb"] \
    [list Template_MD           "%l.maskdb"] \
    [list Template_HF           "hcells"] \
    [list Template_IL           1] \
    [list Template_IS           0] \
]


set opts_vars(pex_opts) [list \
    [list RulesFile             "rules"] \
    [list RulesFileLastLoad     0] \
    [list RunDir                "."] \
    [list LayoutPaths           ""] \
    [list LayoutSystem          GDSII] \
```

```
[list LayoutPrimary          ""] \
[list LayoutGetFromViewer    0] \
[list Precision              1000] \
[list SourcePath             ""] \
[list SourceSystem           SPICE] \
[list SourcePrimary          ""] \
[list SourceGetFromViewer    0] \
[list PexNetlistType         DIST] \
[list PexNetlistFormat       SPICE] \
[list PexNetlistNameSource   SOURCE] \
[list PexNetlistFile         ""] \
[list ViewPexNetlist         1] \
[list PexReport              0] \
[list PexReportFormat        ASCII] \
[list PexReportFile          ""] \
[list ViewPexReport          0] \
[list ExcludeNets            0] \
[list ExcludeNetsValue       ""] \
[list GroundName             0] \
[list GroundNameValue        0] \
[list Loop                   0] \
[list Separator              0] \
[list SeparatorValue         "/"] \
[list Location               0] \
[list Rlocation              0] \
[list Rwidth                 0] \
[list Rlayer                 0] \
[list CoupledRunType         0] \
[list LumpRunType            0] \
[list LumpFile                   ""] \
[list LumpFormat                 ""] \
[list LumpSizeMultiplier         "1e-06"] \
[list LumpNameSource             ""] \
[list LumpGroundName             0] \
[list LumpGroundNameValue        ""] \
[list LumpLoop                   0] \
[list LumpSeparator              0] \
[list LumpSeparatorValue         ""] \
[list LumpLocation               0] \
[list LumpRlocation              0] \
[list LumpRwidth                 0] \
[list LumpRlayer                 0] \
[list LumpGenerateReport         0] \
```

```
[list LumpReportFile           ""] \
[list LumpReportFormat         ASCII] \
[list LumpExcludeNets          0] \
[list LumpExcludeNetsValue     ""] \
[list DistRunType              0] \
[list DistFile                 ""] \
[list DistFormat               ""] \
[list DistSizeMultiplier       "1e-06"] \
[list DistNameSource           ""] \
[list DistGroundName           0] \
[list DistGroundNameValue      ""] \
[list DistLoop                 0] \
[list DistSeparator            0] \
[list DistSeparatorValue       ""] \
[list DistLocation             0] \
[list DistRlocation            0] \
[list DistRwidth               0] \
[list DistRlayer               0] \
[list DistGenerateReport       0] \
[list DistReportFile           ""] \
[list DistReportFormat         ""] \
[list DistExcludeNets          0] \
[list DistExcludeNetsValue     ""] \
[list FDBName                  "myfdb"] \
[list FDBPolygonCount          "1000"] \
[list FDBExcludeNets           0] \
[list FDBExcludeNetsValue     ""] \
[list CreateSVDB           1] \
[list SVDBContents         "XCALIBRE"] \
[list SVDBDir              svdb] \
[list SVDBcustom           ""] \
[list SVDBNoflat           0] \
[list StartRVE             0] \
[list WriteMaskDB          0] \
[list MaskDBFile           maskdb] \
[list WriteINXF            0] \
[list WriteBPF             0] \
[list WriteNL              0] \
[list MergeDatabase        0] \
[list RunHier              1] \
[list RunExt               1] \
[list SpiceFile            lay.net] \
[list RunCmp               1] \
```

```
        [list AutoMatch             0] \
        [list TranscriptFile        ""] \
        [list TranscriptFileAppend  0] \
        [list TranscriptEchoToFile  0] \
        [list ShowGroupsTop     0] \
        [list DontWaitForLicense    1] \
        [list ViaReductionLimit         0] \
        [list ViaReductionLimitValue    "0"] \
        [list FMTGlobal                 0] \
        [list FMTGlobalValue            ""] \
        [list NameFilterMode            0] \
        [list KeywordUpcase             0] \
        [list NetNameSeparator          0] \
        [list NetNameSeparatorValue     "_"] \
        [list UseShortNames             0] \
        [list IncludeFiles          ""] \
    ]


    set opts_vars(cmn_opts) [list \
        [list HostName              "localhost"] \
        [list HostPort              9189] \
        [list ShowToolTips          1] \
        [list ProcessLocalLinks     1] \
        [list DRCMaxVertexAll       0] \
        [list DRCMaxVertexCount     4096] \
        [list RunRemote             0] \
        [list RunRemoteOn           Host] \
        [list RemoteHostName        ""] \
        [list RemoteQueueCmd        ""] \
        [list RemoteUserIsCurrentUser 1] \
        [list RemoteUserName        ""] \
        [list RemoteShellDefault    1] \
        [list RemoteShellName       ""] \
        [list RemoteMGCHOMEUseCurrent   1] \
        [list RemoteMGCHOME         ""] \
        [list RemoteLICENSEFILEName ""] \
        [list RemoteLICENSEFILEValue    ""] \
        [list WarnRunsetNotWritable 1] \
        [list ReadIOFromRules       1] \
    ]
}
```

Appendix C
V2LVS BNF

This differs from the BNF described in the IEEE standard in order to remove ambiguities that cannot be handled by Berkeley YACC.

```
%start source_text
%token YYID
%token YYINUMBER
%token YYRNUMBER
%token YYSTRING
%token YYALLPATH
%token YYALWAYS
%token YYAND
%token YYASSIGN
%token YYBEGIN
%token YYBUF
%token YYBUFIF0
%token YYBUFIF1
%token YYCASE
%token YYCASEX
%token YYCASEZ
%token YYCMOS
%token YYCONDITIONAL
%token YYDEASSIGN
%token YYDEFAULT
%token YYDEFPARAM
%token YYDISABLE
%token YYELSE
%token YYEDGE
%token YYEND
%token YYENDCASE
%token YYENDMODULE
%token YYENDFUNCTION
%token YYENDPRIMITIVE
%token YYENDSPECIFY
%token YYENDTABLE
%token YYENDTASK
%token YYENUM
%token YYEVENT
%token YYFORCE
%token YYFOR
```

```
%token YYFOREVER
%token YYFORK
%token YYFUNCTION
%token YYGEQ
%token YYHIGHZ0
%token YYHIGHZ1
%token YYIF
%token YYIFNONE
%token YYINITIAL
%token YYINOUT
%token YYINPUT
%token YYINTEGER
%token YYJOIN
%token YYLARGE
%token YYLEADTO
%token YYLOGAND
%token YYCASEEQUALITY
%token YYCASEINEQUALITY
%token YYLOGNAND
%token YYLOGNOR
%token YYLOGOR
%token YYLOGXNOR
%token YYLOGEQUALITY
%token YYLOGINEQUALITY
%token YYLSHIFT
%token YYMACROMODULE
%token YYMEDIUM
%token YYMODULE
%token YYNAND
%token YY_LF_ARROW
%token YYNEGEDGE
%token YYNMOS
%token YYNOR
%token YYNOT
%token YYNOTIF0
%token YYNOTIF1
%token YYOR
%token YYOUTPUT
%token YYPARAMETER
%token YYPMOS
%token YYPOSEDGE
%token YYPRIMITIVE
%token YYPULL0
%token YYPULL1
%token YYPULLUP
%token YYPULLDOWN
%token YYRCMOS
%token YYREAL
%token YYREG
%token YYRELEASE
%token YYREPEAT
```

```
%token YYRIGHTARROW
%token YYRNMOS
%token YYRPMOS
%token YYRSHIFT
%token YYRTRAN
%token YYRTRANIF0
%token YYRTRANIF1
%token YYSCALARED
%token YYSMALL
%token YYSPECIFY
%token YYSPECPARAM
%token YYSTRONG0
%token YYSTRONG1
%token YYSUPPLY0
%token YYSUPPLY1
%token YYTABLE
%token YYTASK
%token YYTIME
%token YYREALTIME
%token YYTRAN
%token YYTRANIF0
%token YYTRANIF1
%token YYTRI
%token YYTRI0
%token YYTRI1
%token YYTRIAND
%token YYTRIOR
%token YYuTYPE
%token YYTYPEDEF
%token YYVECTORED
%token YYTRIREG
%token YYWAIT
%token YYWAND
%token YYWEAK0
%token YYWEAK1
%token YYWHILE
%token YYWIRE
%token YYWOR
%token YYXNOR
%token YYXOR
%token YYsysSETUP
%token YYsysID
%token YYsysHOLD             /* $hold */
%token YYsysPERIOD           /* $period */
%token YYsysRECOVERY         /* $recovery */
%token YYsysSETUP            /* $setup */
%token YYsysSETUPHOLD        /* $setuphold */
%token YYsysSKEW             /* $skew */
%token YYsysWIDTH            /* $width */
%token YYsysNOCHANGE         /* $nochange */
%token YYCONDITIONING_AND    /* &&& */
```

```
%token LEX_ERROR           /* not used in the grammar anywhere, so it can
produce an error */
%right '?' ':'
%left YYOR
%left YYLOGOR
%left YYLOGAND
%left YYLOGNAND
%left '|'
%left '&' '^' YYLOGXNOR
%left YYLOGEQUALITY YYLOGINEQUALITY YYCASEEQUALITY YYCASEINEQUALITY
%left '<' YY_LF_ARROW '>' YYGEQ
%left YYLSHIFT YYRSHIFT
%left '+' '-'
%left '*' '/' '%'
%right '~' '!' YYUNARYOPERATOR
%type <desc> source_text
%type <desc> description
%type <modu> module_declaration
%type <prim> udp_declaration
%type <lstp> list_of_ports_opt list_of_ports
%type <port> port
%type <lstidr> port_expression_opt port_expression
%type <lstidr> port_ref_list
%type <idrng> port_reference
%type <rang> port_reference_arg
%type <rangdcl> module_item
%type <rangdcl> module_item_declaration
%type <lstd> module_item_clr
%type <paradcl> parameter_declaration
%type <rangdcl> input_declaration output_declaration inout_declaration
%type <rangdcl> reg_declaration
%type <netdcl>  net_declaration
%type <bascdcl> time_declaration realtime_declaration event_declaration
%type <bascdcl> integer_declaration real_declaration
%type <type> gate_instantiation
%type <type> user_instance
%type <paradcl> parameter_override
%type <lsta>   continuous_assign
%type <prcstmt> initial_construct always_construct
%type <task> task_declaration
%type <func> function_declaration
%type <lstidr> variable_list
%type <lstp> udp_port_declaration_eclr
%type <lstp> udp_body
%type <lstp> table_entries combinational_entry_eclr sequential_entry_eclr
%type <ntry> combinational_entry sequential_entry
%type <lstp> input_list level_symbol_or_edge_eclr
%type <ival> output_symbol state next_state
%type <ival> level_symbol edge level_symbol_or_edge edge_symbol
%type <type> udp_port_declaration
%type <lstp> tf_item_declaration_clr tf_item_declaration_eclr
```

```
%type <lstp> statement_opt
%type <rngtyp>  range_or_type_opt range_or_type
%type <rang> range
%type <type> tf_item_declaration
%type <lsta> assignment_list
%type <rang> range_opt
%type <expr> expression expression_opt
%type <ival> drive_strength_opt drive_strength drive_strength_rest
%type <ival> charge_strength_opt charge_strength
%type <ntype> net_type
%type <dlay> delay_opt delay
%type <lstp> register_variable_list
%type <lstp> name_of_event_list
%type <idrng>  identifier function_identifier
%type <idrng>  register_variable
%type <charp>  name_of_register
%type <idrng>  name_of_event
%type <ival> strength0 strength1
%type <assign> assignment
%type <lstp> drive_delay_clr
%type <lstg> gate_instance_list
%type <gtype> gatetype
%type <ival> drive_delay
%type <gtinst> gate_instance
%type <lste> terminal_list
%type <name> name_of_module name_of_instance
%type <idrng> name_of_gate_instance
%type <lstt> instance_list
%type <dlay> parameter_value_assignment parameter_value_assignment_opt
%type <mod_inst> instance named_instance
%type <lstpc> module_connection_list module_port_connection_list
%type <lstpc> named_port_connection_list
%type <connect> module_port_connection named_port_connection
named_port_connection_name
%type <type> statement
%type <lstp> statement_clr case_item_eclr
%type <caseitem> case_item
%type <dlay> delay_control
%type <evnt> event_control
%type <lval> lvalue
%type <lste> expression_list
%type <seqstmt>  seq_block
%type <parstmt>  par_block
%type <idrng> name_of_block
%type <lstp> block_declaration_clr
%type <type> block_declaration
%type <taskenablestmt>  task_enable
%type <lste> concatenation
%type <expr> multiple_concatenation
%type <expr> mintypmax_expression
%type <expr> constant_primary
```

```
%type <expr> primary
%type <expr> function_call
%type <evnt> event_expression ored_event_expression
%%

/* A.1 Source text */
source_text
        :
        | source_text description
        ;

description
        : module_declaration
        | udp_declaration
        ;

module_declaration
        : module_keyword YYID
          list_of_ports_opt ';'
          module_item_clr
          YYENDMODULE
        ;

module_keyword
        : YYMODULE
        | YYMACROMODULE
        ;

list_of_ports_opt
        :
        | '(' list_of_ports ')'
        ;

list_of_ports
        : port
        | list_of_ports ',' port
        ;

port
        : port_expression_opt
        | '.' YYID
        '(' port_expression_opt ')'
        ;

port_expression_opt
        :
        |  port_expression
        ;

port_expression
        : port_reference
```

```
        | '{' port_ref_list '}'
        ;

port_ref_list
        : port_reference
        | port_ref_list ',' port_reference
        ;

port_reference
        : YYID
          port_reference_arg
        ;

port_reference_arg
        :
        | '[' expression ']'
        | '[' expression ':' expression ']'
        ;

module_item_clr
        :
        | module_item_clr module_item
        ;

module_item
        : module_item_declaration
        | gate_instantiation
        | user_instance
        | parameter_override
        | continuous_assign
        | specify_block
        | initial_construct
        | always_construct
        ;

module_item_declaration
        : parameter_declaration
        | input_declaration
        | output_declaration
        | inout_declaration
        | net_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | realtime_declaration
        | event_declaration
        | task_declaration
        | function_declaration
        ;
```

```
parameter_override
        : YYDEFPARAM assignment_list ';'
        ;

/* A.2 Declarations */

parameter_declaration
        : YYPARAMETER range_opt param_assignment_list ';'
        ;

param_assignment_list
        : param_assignment
        | param_assignment_list ',' param_assignment
        ;

param_assignment
        : identifier '=' mintypmax_expression
        ;


assignment_list
        : assignment
        | assignment_list ',' assignment
        ;

assignment
        : lvalue type_action '=' expression
        | lvalue type_action YY_LF_ARROW expression
        ;

type_action
        :
        ;

input_declaration
        : YYINPUT range_opt variable_list ';'
        ;

output_declaration
        : YYOUTPUT range_opt variable_list ';'
        ;

inout_declaration
        : YYINOUT range_opt variable_list ';'
        ;

variable_list
        : identifier
        | variable_list ',' identifier
        ;
```

```
reg_declaration
        : YYREG range_opt register_variable_list ';'
        ;

time_declaration
        : YYTIME register_variable_list ';'
        ;

integer_declaration
        : YYINTEGER register_variable_list ';'
        ;

real_declaration
        : YYREAL variable_list ';'
        ;

realtime_declaration
        : YYREALTIME variable_list ';'
        ;

event_declaration
        : YYEVENT name_of_event_list ';'
        ;

name_of_event_list
        : name_of_event
        | name_of_event_list ',' name_of_event
        ;

name_of_event
        : YYID
        ;

register_variable_list
        : register_variable
        | register_variable_list ',' register_variable
        ;

register_variable
        : name_of_register
        | name_of_register '[' expression ':' expression ']'
        ;

name_of_register
        : YYID
        ;

range_opt
        :
        | range
        ;
```

```
range
        : '[' expression ':' expression ']'
        ;

net_declaration
        : net_type range delay_opt variable_list ';'
        | net_type vectored_or_scalared range delay_opt variable_list ';'
        | net_type delay_opt variable_list ';'
        | net_type vectored_or_scalared delay_opt variable_list ';'
        | YYTRIREG vectored_or_scalared_opt charge_strength_opt range_opt
delay_opt variable_list ';'
        | net_type drive_strength range delay_opt assignment_list ';'
        | net_type vectored_or_scalared drive_strength range delay_opt
assignment_list ';'
        | net_type range delay_opt assignment_list ';'
        | net_type vectored_or_scalared range delay_opt assignment_list
';'
        | net_type drive_strength delay_opt assignment_list ';'
        | net_type vectored_or_scalared drive_strength delay_opt
assignment_list ';'
        | net_type delay_opt assignment_list ';'
        | net_type vectored_or_scalared delay_opt assignment_list ';'
        ;

vectored_or_scalared_opt
        :
        | vectored_or_scalared
        ;

vectored_or_scalared
        : YYVECTORED
        | YYSCALARED
        ;

/* all net types are handled as wires except supply0 and supply1 */
net_type
        : YYWIRE
        | YYTRI
        | YYTRI1
        | YYSUPPLY0
        | YYWAND
        | YYTRIAND
        | YYTRI0
        | YYSUPPLY1
        | YYWOR
        | YYTRIOR
        ;

drive_strength_opt
        :
```

```
        | drive_strength
        ;

drive_strength
        : '(' strength0 ',' strength1 ')'
        | '(' strength1 ',' strength0 ')'
        ;

strength0
        : YYSUPPLY0
        | YYSTRONG0
        | YYPULL0
        | YYWEAK0
        | YYHIGHZ0
        ;

strength1
        : YYSUPPLY1
        | YYSTRONG1
        | YYPULL1
        | YYWEAK1
        | YYHIGHZ1
        ;

charge_strength_opt
        :
        | charge_strength
        ;

charge_strength
        : '(' YYSMALL ')'
        | '(' YYMEDIUM ')'
        | '(' YYLARGE ')'
        ;

delay_opt
        :
        | delay
        ;

delay
        : '#' YYINUMBER
        | '#' YYRNUMBER
        | '#' identifier
        | '#' '(' mintypmax_expression ')'
        | '#' '(' mintypmax_expression ',' mintypmax_expression ')'
        | '#' '(' mintypmax_expression ',' mintypmax_expression ','
mintypmax_expression ')'
        ;
```

```
function_declaration
        : YYFUNCTION range_or_type_opt YYID
          ';' tf_item_declaration_eclr statement
          YYENDFUNCTION
        ;

range_or_type_opt
        :
        | range_or_type
        ;

range_or_type
        : range
        | YYINTEGER
        | YYREAL
        | YYTIME
        | YYREALTIME
        ;

tf_item_declaration_clr
        :
        | tf_item_declaration_clr tf_item_declaration
        ;

tf_item_declaration_eclr
        : tf_item_declaration
        | tf_item_declaration_eclr tf_item_declaration
        ;

tf_item_declaration
        : parameter_declaration
        | input_declaration
        | output_declaration
        | inout_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | realtime_declaration
        | event_declaration
        ;


task_declaration
        : YYTASK YYID
          ';' tf_item_declaration_clr statement_opt
          YYENDTASK
      ;


/* A.3 Primitive Instances */
```

```
gate_instantiation
        : gatetype drive_delay_clr gate_instance_list ';'
        ;

drive_delay_clr
        :
        | drive_delay_clr drive_delay
        ;

drive_delay
        : drive_strength
        | delay
        ;

gatetype
        : YYAND
        | YYNAND
        | YYOR
        | YYNOR
        | YYXOR
        | YYXNOR
        | YYBUF
        | YYBUFIF0
        | YYBUFIF1
        | YYNOT
        | YYNOTIF0
        | YYNOTIF1
        | YYPULLDOWN
        | YYPULLUP
        | YYNMOS
        | YYPMOS
        | YYRNMOS
        | YYRPMOS
        | YYCMOS
        | YYRCMOS
        | YYTRAN
        | YYRTRAN
        | YYTRANIF0
        | YYRTRANIF0
        | YYTRANIF1
        | YYRTRANIF1
        ;

gate_instance_list
        : gate_instance
        | gate_instance_list ',' gate_instance
        ;

gate_instance
        : '(' terminal_list ')'
```

```
        | name_of_gate_instance '(' terminal_list ')'
        ;

name_of_gate_instance
        : YYID range_opt
        ;

terminal_list
        : expression
        | terminal_list ',' expression
        ;

/* A.4 Module Instantiation */
user_instance
        : name_of_module '(' drive_strength_rest
parameter_value_assignment_opt instance_list ';'
        | name_of_module '(' module_connection_list ')' ';'
        | name_of_module parameter_value_assignment instance_list ';'
        | name_of_module named_instance instance_list ';'
        ;

drive_strength_rest
        : strength0 ')'
        | strength0 ',' strength1 ')'
        | strength1 ')'
        | strength1 ',' strength0 ')'
        ;


name_of_module
        : YYID
        ;

name_of_instance
        : YYID
        ;

/* A.5 UDP declaration */
udp_declaration
        : YYPRIMITIVE YYID
          '(' list_of_ports ')' ';'
          udp_port_declaration_eclr
          udp_body
          YYENDPRIMITIVE
        ;

udp_port_declaration_eclr
        : udp_port_declaration
        | udp_port_declaration_eclr udp_port_declaration
        ;
```

```
udp_port_declaration
        : output_declaration
        | reg_declaration
        | input_declaration
        ;

udp_body
        : initial_construct_opt YYTABLE table_entries YYENDTABLE
        ;

table_entries
        : combinational_entry_eclr
        | sequential_entry_eclr
        ;

combinational_entry_eclr
        : combinational_entry
        | combinational_entry_eclr combinational_entry
        ;

combinational_entry
        : input_list ':' output_symbol ';'
        ;

sequential_entry_eclr
        : sequential_entry
        | sequential_entry_eclr sequential_entry
        ;

sequential_entry
        : input_list ':' state ':' next_state ';'
        ;

input_list
        : level_symbol_or_edge_eclr
        ;

level_symbol_or_edge_eclr
        : level_symbol_or_edge
        | level_symbol_or_edge_eclr level_symbol_or_edge
        ;

level_symbol_or_edge
        : level_symbol
        | edge
        ;

edge
        : '(' level_symbol level_symbol ')'
        | edge_symbol
        ;
```

```
state
        : level_symbol
        ;

next_state
        : output_symbol
        | '-'
        ;

output_symbol
        : '0'
        | '1'
        | 'x'
        | 'X'
        ;

level_symbol
        : '0'
        | '1'
        | 'x'
        | 'X'
        | '?'
        | 'b'
        | 'B'
        ;

edge_symbol
        : 'r'
        | 'R'
        | 'f'
        | 'F'
        | 'p'
        | 'P'
        | 'n'
        | 'N'
        | '*'
        ;

/* A.6 Behavioral statements */
continuous_assign
        : YYASSIGN {}
          drive_strength_opt delay_opt assignment_list ';'

        ;

initial_construct_opt
        :
        | initial_construct
        ;
```

```
initial_construct
        : YYINITIAL {} statement
        ;

always_construct
        : YYALWAYS {} statement
        ;

statement_opt
        :
        | statement
        ;

statement_clr
        :
        | statement_clr statement
        ;


statement
        : ';'
        | assignment ';'
        | YYIF '(' expression ')' statement
        | YYIF '(' expression ')' statement YYELSE statement
        | YYCASE '(' expression ')' case_item_eclr YYENDCASE
        | YYCASEZ '(' expression ')' case_item_eclr YYENDCASE
        | YYCASEX '(' expression ')' case_item_eclr YYENDCASE
        | YYFOREVER statement
        | YYREPEAT '(' expression ')' statement
        | YYWHILE '(' expression ')' statement
       | YYFOR '(' assignment ';' expression ';' assignment ')' statement
        | delay_control statement
        | event_control statement
        | lvalue type_action '=' delay_control expression ';'
        | lvalue type_action '=' event_control expression ';'
       | lvalue type_action '=' YYREPEAT '(' expression ')' event_control
expression ';'
        | lvalue type_action YY_LF_ARROW delay_control expression ';'
        | lvalue type_action YY_LF_ARROW event_control expression ';'
        | lvalue type_action YY_LF_ARROW YYREPEAT '(' expression ')'
event_control expression ';'
        | YYWAIT '(' event_expression ')' statement
        | YYRIGHTARROW name_of_event ';'
        | seq_block
        | par_block
        | task_enable
        | system_task_enable
        | YYDISABLE identifier ';'
        | YYASSIGN assignment ';'
        | YYFORCE assignment ';'
        | YYDEASSIGN lvalue ';'
```

```
        | YYRELEASE lvalue ';'
        ;

delay_control
        : '#' YYINUMBER
        | '#' YYRNUMBER
        | '#' identifier
        | '#' '(' mintypmax_expression ')'
        ;

event_control
        : '@' identifier
        | '@' '(' event_expression ')'
        | '@' '(' ored_event_expression ')'
        ;


event_expression
        : expression
        | YYPOSEDGE expression
        | YYNEGEDGE expression
        | YYEDGE expression
        ;

ored_event_expression
        : event_expression YYOR event_expression
        | ored_event_expression YYOR event_expression
        ;

case_item_eclr
        : case_item
        | case_item_eclr case_item
        ;

case_item
        : expression_list ':' statement
        | YYDEFAULT ':' statement
        | YYDEFAULT statement
        ;

seq_block
        : YYBEGIN statement_clr YYEND
        | YYBEGIN ':' name_of_block block_declaration_clr statement_clr
YYEND
        ;

par_block
        : YYFORK statement_clr YYJOIN
        | YYFORK ':' name_of_block block_declaration_clr statement_clr
YYJOIN
        ;
```

```
name_of_block
        : YYID
        ;

block_declaration_clr
        :
        | block_declaration_clr block_declaration
        ;

block_declaration
        : parameter_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | event_declaration
        ;


task_enable
        : identifier ';'
        | identifier '(' expression_list ')' ';'
        ;

system_task_enable
        : name_of_system_task ';'
        | name_of_system_task '(' expression_list ')' ';'
        ;

name_of_system_task
        : system_identifier
        ;

system_identifier
        : YYsysID
        ;

/* A.7 Specify Section */

specify_block
        : YYSPECIFY
             specify_item_list YYENDSPECIFY
        ;

specify_item_list
        : /* empty */
        | specify_item_list specify_item
        ;

specify_item
```

```
        : specparam_declaration
        | path_declaration
        | system_timing_check
        ;

specparam_declaration
        : YYSPECPARAM list_of_specparam_assignments ';'
        ;

list_of_specparam_assignments
        : specparam_assignment
        | list_of_specparam_assignments ',' specparam_assignment
        ;

specparam_assignment
        : YYID '=' mintypmax_expression
        | YYID '=' '(' mintypmax_expression_list_2 ')'
        ;

path_declaration
        : path_conditional_opt '(' edge_identifier_opt
list_of_path_terminals
          polarity_operator_opt arrow list_of_path_terminals ')'
          '=' path_delay_value ';'
        | path_conditional_opt '(' edge_identifier_opt
list_of_path_terminals
          polarity_operator_opt arrow
          '(' list_of_path_terminals polarity_operator_opt ':' expression
')' ')'
          '=' path_delay_value ';'
        ;

path_conditional_opt
        : /* empty */
        | YYIF '(' expression ')'
        | YYIFNONE
        ;

arrow
        : YYLEADTO
        | YYALLPATH
        ;

list_of_path_terminals
        : specify_terminal_descriptor
        | list_of_path_terminals ',' specify_terminal_descriptor
        ;

specify_terminal_descriptor
        : YYID
        | YYID '[' expression ']'
```

```
        | YYID '[' expression ':' expression ']'
        ;

path_delay_value
        : mintypmax_expression_list
        | '(' mintypmax_expression_list_2 ')'
        ;

edge_identifier_opt
        : /* empty */
        | YYPOSEDGE
        | YYNEGEDGE
        ;

mintypmax_expression_list
        : mintypmax_expression
        | mintypmax_expression_list ',' mintypmax_expression
        ;

mintypmax_expression_list_2
        : mintypmax_expression ',' mintypmax_expression
        | mintypmax_expression_list_2 ',' mintypmax_expression
        ;

system_timing_check
        : YYsysSETUP '(' timing_check_event ',' timing_check_event ','
          mintypmax_expression notify_register ')' ';'
        | YYsysHOLD '(' timing_check_event ',' timing_check_event ','
          mintypmax_expression notify_register ')' ';'
        | YYsysPERIOD '(' controlled_timing_check_event ','
          mintypmax_expression notify_register ')' ';'
        | YYsysWIDTH '(' controlled_timing_check_event ','
          mintypmax_expression ')' ';'
        | YYsysWIDTH '(' controlled_timing_check_event ','
          mintypmax_expression ',' mintypmax_expression notify_register
')' ';'
        | YYsysSKEW '(' timing_check_event ',' timing_check_event ','
          mintypmax_expression notify_register ')' ';'
        | YYsysRECOVERY '(' controlled_timing_check_event ','
          timing_check_event ',' mintypmax_expression notify_register ')'
';'
       | YYsysSETUPHOLD '(' timing_check_event ',' timing_check_event ','
          mintypmax_expression ',' mintypmax_expression ')' ';'
       | YYsysSETUPHOLD '(' timing_check_event ',' timing_check_event ','
         mintypmax_expression ',' mintypmax_expression ',' sys_arglist ')'
';'
        | YYsysNOCHANGE '(' timing_check_event ',' timing_check_event ','
          mintypmax_expression ',' mintypmax_expression notify_register
')' ';'
        ;
```

```
timing_check_event
        : timing_check_control_opt specify_terminal_descriptor
        timing_check_condition
        ;

controlled_timing_check_event
        : timing_check_control specify_terminal_descriptor
        timing_check_condition
        ;

timing_check_control_opt
        : /* empty */
        | timing_check_control
        ;

timing_check_control
        : YYPOSEDGE
        | YYNEGEDGE
        | YYEDGE '[' edge_descriptor_list ']'
        ;

edge_descriptor_list
        : edge_descriptor
        | edge_descriptor_list ',' edge_descriptor
        ;

edge_descriptor
        : YYINUMBER
        ;

timing_check_condition
        : /* empty */
        | YYCONDITIONING_AND expression
        ;

notify_register
        : /* empty */
        | ',' YYID
        ;

sys_arglist
        : expression
        | sys_arglist ',' expression
        ;

polarity_operator_opt
        :
        | polarity_operator
        ;

polarity_operator
```

```
        : '+'
        | '-'
        ;

parameter_value_assignment_opt
        : /* empty */
        | parameter_value_assignment
        ;

parameter_value_assignment
        : '#' '(' expression_list ')'
        | '#' YYINUMBER
        | '#' identifier
        ;

instance
        : named_instance
        | '(' module_connection_list ')'
        ;


instance_list
        : /* empty */
        | instance
        | instance_list ',' instance
        ;

named_instance
        : name_of_instance
          range_opt '(' module_connection_list ')'
        ;

module_connection_list
        : module_port_connection_list
        | named_port_connection_list
        ;

module_port_connection_list
        : module_port_connection
        | module_port_connection_list ',' module_port_connection
        ;

named_port_connection_list
        : named_port_connection
        | named_port_connection_list ',' named_port_connection
        ;

module_port_connection
        :
        | expression
        ;
```

```
named_port_connection
        : '.' named_port_connection_name '(' expression ')'
        | '.' named_port_connection_name '(' ')'
        ;

named_port_connection_name
        : YYID

/* A.8 Expressions */
lvalue
        : identifier
        | identifier '[' expression ']'
        | identifier '[' expression ':' expression ']'
        | concatenation
        ;

mintypmax_expression
        : expression
        | expression ':' expression ':' expression
        ;

expression_list
        : expression_opt
        | expression_list ',' expression_opt
        ;

expression_opt
        :
        | expression
        ;

expression
        : primary
        | '+' primary %prec YYUNARYOPERATOR
        | '-' primary %prec YYUNARYOPERATOR
        | '!' primary %prec YYUNARYOPERATOR
        | '~' primary %prec YYUNARYOPERATOR
        | '&' primary %prec YYUNARYOPERATOR
        | '|' primary %prec YYUNARYOPERATOR
        | '^' primary %prec YYUNARYOPERATOR
        | YYLOGNAND primary %prec YYUNARYOPERATOR
        | YYLOGNOR primary %prec YYUNARYOPERATOR
        | YYLOGXNOR primary %prec YYUNARYOPERATOR
        | expression '+' expression
        | expression '-' expression
        | expression '*' expression
        | expression '/' expression
        | expression '%' expression
        | expression YYLOGEQUALITY expression
        | expression YYLOGINEQUALITY expression
```

```
        | expression YYCASEEQUALITY expression
        | expression YYCASEINEQUALITY expression
        | expression YYLOGAND expression
        | expression YYLOGOR expression
        | expression YYLOGNAND expression
        | expression '<' expression
        | expression '>' expression
        | expression '&' expression
        | expression '|' expression
        | expression '^' expression
        | expression YY_LF_ARROW expression
        | expression YYGEQ expression
        | expression YYLSHIFT expression
        | expression YYRSHIFT expression
        | expression YYLOGXNOR expression
        | expression '?' expression ':' expression
        | YYSTRING
        ;

constant_primary
        : YYINUMBER
        | YYRNUMBER
        | identifier
        | concatenation
        | multiple_concatenation
        ;

primary
        : constant_primary
        | identifier '[' expression ']'
        | identifier '[' expression ':'  expression ']'
        | function_call
        | '(' mintypmax_expression ')'
        | YYsysID
        | YYsysID '(' expression_list ')'
        ;

concatenation
        : '{' expression_list '}'
        ;

multiple_concatenation
        : '{' expression '{' expression_list '}' '}'
        ;

function_call
        : function_identifier '(' expression_list ')'
        ;

function_identifier
        : YYID
```

```
        | selected_name
        ;

identifier
        : YYID
        | selected_name
        ;

selected_name
        : YYID '.' YYID
        | selected_name '.' YYID
        ;
%start source_text
%token YYID
%token YYINUMBER
%token YYRNUMBER
%token YYSTRING
%token YYALLPATH
%token YYALWAYS
%token YYAND
%token YYASSIGN
%token YYBEGIN
%token YYBUF
%token YYBUFIF0
%token YYBUFIF1
%token YYCASE
%token YYCASEX
%token YYCASEZ
%token YYCMOS
%token YYCONDITIONAL
%token YYDEASSIGN
%token YYDEFAULT
%token YYDEFPARAM
%token YYDISABLE
%token YYELSE
%token YYEDGE
%token YYEND
%token YYENDCASE
%token YYENDMODULE
%token YYENDFUNCTION
%token YYENDPRIMITIVE
%token YYENDSPECIFY
%token YYENDTABLE
%token YYENDTASK
%token YYENUM
%token YYEVENT
%token YYFORCE
%token YYFOR
%token YYFOREVER
%token YYFORK
%token YYFUNCTION
```

```
%token YYGEQ
%token YYHIGHZ0
%token YYHIGHZ1
%token YYIF
%token YYINITIAL
%token YYINOUT
%token YYINPUT
%token YYINTEGER
%token YYJOIN
%token YYLARGE
%token YYLEADTO
%token YYLEQ
%token YYLOGAND
%token YYCASEEQUALITY
%token YYCASEINEQUALITY
%token YYLOGNAND
%token YYLOGNOR
%token YYLOGOR
%token YYLOGXNOR
%token YYLOGEQUALITY
%token YYLOGINEQUALITY
%token YYLSHIFT
%token YYMACROMODULE
%token YYMEDIUM
%token YYMODULE
%token YYNAND
%token YYNBASSIGN
%token YYNEGEDGE
%token YYNMOS
%token YYNOR
%token YYNOT
%token YYNOTIF0
%token YYNOTIF1
%token YYOR
%token YYOUTPUT
%token YYPARAMETER
%token YYPMOS
%token YYPOSEDGE
%token YYPRIMITIVE
%token YYPULL0
%token YYPULL1
%token YYPULLUP
%token YYPULLDOWN
%token YYRCMOS
%token YYREAL
%token YYREG
%token YYRELEASE
%token YYREPEAT
%token YYRIGHTARROW
%token YYRNMOS
%token YYRPMOS
```

```
%token YYRSHIFT
%token YYRTRAN
%token YYRTRANIF0
%token YYRTRANIF1
%token YYSCALARED
%token YYSMALL
%token YYSPECIFY
%token YYSPECPARAM
%token YYSTRONG0
%token YYSTRONG1
%token YYSUPPLY0
%token YYSUPPLY1
%token YYTABLE
%token YYTASK
%token YYTIME
%token YYREALTIME
%token YYTRAN
%token YYTRANIF0
%token YYTRANIF1
%token YYTRI
%token YYTRI0
%token YYTRI1
%token YYTRIAND
%token YYTRIOR
%token YYuTYPE
%token YYTYPEDEF
%token YYVECTORED
%token YYTRIREG
%token YYWAIT
%token YYWAND
%token YYWEAK0
%token YYWEAK1
%token YYWHILE
%token YYWIRE
%token YYWOR
%token YYXNOR
%token YYXOR
%token YYsysSETUP
%token YYsysID
%token YYsysHOLD             /* $hold */
%token YYsysPERIOD           /* $period */
%token YYsysRECOVERY         /* $recovery */
%token YYsysSETUP            /* $setup */
%token YYsysSETUPHOLD        /* $setuphold */
%token YYsysSKEW             /* $skew */
%token YYsysWIDTH            /* $width */
%token YYsysNOCHANGE         /* $nochange */
%token YYCONDITIONING_AND /* &&& */
%token LEX_ERROR           /* not used in the grammar anywhere, so it can
produce an error */
```

```
source_text
        :
        | source_text description
        ;

description
        : module_declaration
        | udp_declaration
        ;

module_declaration
        : module_keyword YYID
        ;

module_keyword
        : YYMODULE
        | YYMACROMODULE
        ;

list_of_ports_opt
        :
        | '(' list_of_ports ')'
        ;

list_of_ports
        : port
        | list_of_ports ',' port
        ;

port
        : port_expression_opt
        | '.' YYID
        ;

port_expression_opt
        :
        |  port_expression
        ;

port_expression
        : port_reference
        | '{' port_ref_list '}'
        ;

port_ref_list
        : port_reference
        | port_ref_list ',' port_reference
        ;

port_reference
        : YYID
```

```
        ;

port_reference_arg
        :
        | '[' expression ']'
        | '[' expression ':' expression ']'
        ;

module_item_clr
        :
        | module_item_clr module_item
        ;

module_item
        : module_item_declaration
        | gate_instantiation
        | module_instantiation
        | parameter_override
        | continuous_assign
        | specify_block
        | initial_construct
        | always_construct
        ;

module_item_declaration
        : parameter_declaration
        | input_declaration
        | output_declaration
        | inout_declaration
        | net_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | realtime_declaration
        | event_declaration
        | task_declaration
        | function_declaration
        ;

parameter_override
        : YYDEFPARAM assignment_list ';'
        ;

parameter_declaration
        : YYPARAMETER range_opt param_assignment_list ';'
        ;

param_assignment_list
       : param_assignment
       | param_assignment_list ',' param_assignment
```

```
        ;

param_assignment
        : identifier '=' expression
        ;

assignment_list
        : assignment
        | assignment_list ',' assignment
        ;

assignment
        : lvalue type_action '=' expression
        | lvalue type_action YYNBASSIGN expression
        ;

type_action
        :
        ;

input_declaration
        : YYINPUT range_opt variable_list ';'
        ;

output_declaration
        : YYOUTPUT range_opt variable_list ';'
        ;

inout_declaration
        : YYINOUT range_opt variable_list ';'
        ;

variable_list
        : identifier
        | variable_list ',' identifier
        ;

reg_declaration
        : YYREG range_opt register_variable_list ';'
        ;

time_declaration
        : YYTIME register_variable_list ';'
        ;

integer_declaration
        : YYINTEGER register_variable_list ';'
        ;

real_declaration
        : YYREAL variable_list ';'
```

```
        ;

realtime_declaration
        : YYREALTIME variable_list ';'
        ;

event_declaration
        : YYEVENT name_of_event_list ';'
        ;

name_of_event_list
        : name_of_event
        | name_of_event_list ',' name_of_event
        ;

name_of_event
        : YYID
        ;

register_variable_list
        : register_variable
        | register_variable_list ',' register_variable
        ;

register_variable
        : name_of_register
        | name_of_register '[' expression ':' expression ']'
        ;

name_of_register
        : YYID
        ;

range_opt
        :
        | range
        ;

range
        : '[' expression ':' expression ']'
        ;

net_declaration
        : net_type range delay_opt variable_list ';'
        | net_type vectored_or_scalared range delay_opt variable_list ';'
        | net_type delay_opt variable_list ';'
        | net_type vectored_or_scalared delay_opt variable_list ';'
        | YYTRIREG vectored_or_scalared_opt charge_strength_opt range_opt
delay_opt variable_list ';'
        | net_type drive_strength range delay_opt assignment_list ';'
```

```
        | net_type vectored_or_scalared drive_strength range delay_opt
assignment_list ';'
        | net_type range delay_opt assignment_list ';'
        | net_type vectored_or_scalared range delay_opt assignment_list
';'
        | net_type drive_strength delay_opt assignment_list ';'
        | net_type vectored_or_scalared drive_strength delay_opt
assignment_list ';'
        | net_type delay_opt assignment_list ';'
        | net_type vectored_or_scalared delay_opt assignment_list ';'
        ;

vectored_or_scalared_opt
        :
        | vectored_or_scalared
        ;

vectored_or_scalared
        : YYVECTORED
        | YYSCALARED
        ;

net_type
        : YYWIRE
        | YYTRI
        | YYTRI1
        | YYSUPPLY0
        | YYWAND
        | YYTRIAND
        | YYTRI0
        | YYSUPPLY1
        | YYWOR
        | YYTRIOR
        ;

drive_strength_opt
        :
        | drive_strength
        ;

drive_strength
        : '(' strength0 ',' strength1 ')'
        | '(' strength1 ',' strength0 ')'
        ;

strength0
        : YYSUPPLY0
        | YYSTRONG0
        | YYPULL0
        | YYWEAK0
        | YYHIGHZ0
```

```
        ;

strength1
        : YYSUPPLY1
        | YYSTRONG1
        | YYPULL1
        | YYWEAK1
        | YYHIGHZ1
        ;

charge_strength_opt
        :
        | charge_strength
        ;

charge_strength
        : '(' YYSMALL ')'
        | '(' YYMEDIUM ')'
        | '(' YYLARGE ')'
        ;

delay_opt
        :
        | delay
        ;

delay
        : '#' YYINUMBER
        | '#' YYRNUMBER
        | '#' identifier
        | '#' '(' mintypmax_expression ')'
        | '#' '(' mintypmax_expression ',' mintypmax_expression ')'
        | '#' '(' mintypmax_expression ',' mintypmax_expression ','
mintypmax_expression ')'
        ;

function_declaration
        : YYFUNCTION range_or_type_opt YYID
        ;

range_or_type_opt
        :
        | range_or_type
        ;

range_or_type
        : range
        | YYINTEGER
        | YYREAL
        | YYTIME
        | YYREALTIME
```

```
        ;

tf_item_declaration_clr
        :
        | tf_item_declaration_clr tf_item_declaration
        ;

tf_item_declaration_eclr
        : tf_item_declaration
        | tf_item_declaration_eclr tf_item_declaration
        ;

tf_item_declaration
        : parameter_declaration
        | input_declaration
        | output_declaration
        | inout_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | realtime_declaration
        | event_declaration
        ;

task_declaration
        : YYTASK YYID
      ;

gate_instantiation
        : gatetype drive_delay_clr gate_instance_list ';'
        ;

drive_delay_clr
        :
        | drive_delay_clr drive_delay
        ;

drive_delay
        : drive_strength
        | delay
        ;

gatetype
        : YYAND
        | YYNAND
        | YYOR
        | YYNOR
        | YYXOR
        | YYXNOR
        | YYBUF
```

```
        | YYBUFIF0
        | YYBUFIF1
        | YYNOT
        | YYNOTIF0
        | YYNOTIF1
        | YYPULLDOWN
        | YYPULLUP
        | YYNMOS
        | YYPMOS
        | YYRNMOS
        | YYRPMOS
        | YYCMOS
        | YYRCMOS
        | YYTRAN
        | YYRTRAN
        | YYTRANIF0
        | YYRTRANIF0
        | YYTRANIF1
        | YYRTRANIF1
        ;

gate_instance_list
        : gate_instance
        | gate_instance_list ',' gate_instance
        ;

gate_instance
        : '(' terminal_list ')'
        | name_of_gate_instance '(' terminal_list ')'
        ;

name_of_gate_instance
        : YYID range_opt
        ;

terminal_list
        : expression
        | terminal_list ',' expression
        ;

module_instantiation
        : name_of_module module_option_clr
        ;

name_of_module
        : YYID
        ;

module_option_clr
        :
        | module_option_clr module_option
```

```
        ;

module_option
        : drive_strength
        | parameter_value_assignment
        ;

udp_declaration
        : YYPRIMITIVE YYID
        ;

udp_port_declaration_eclr
        : udp_port_declaration
        | udp_port_declaration_eclr udp_port_declaration
        ;

udp_port_declaration
        : output_declaration
        | reg_declaration
        | input_declaration
        ;

udp_body
        : initial_construct_opt YYTABLE table_entries YYENDTABLE
        ;

table_entries
        : combinational_entry_eclr
        | sequential_entry_eclr
        ;

combinational_entry_eclr
        : combinational_entry
        | combinational_entry_eclr combinational_entry
        ;

combinational_entry
        : input_list ':' output_symbol ';'
        ;

sequential_entry_eclr
        : sequential_entry
        | sequential_entry_eclr sequential_entry
        ;

sequential_entry
        : input_list ':' state ':' next_state ';'
        ;

input_list
        : level_symbol_or_edge_eclr
```

```
        ;

level_symbol_or_edge_eclr
        : level_symbol_or_edge
        | level_symbol_or_edge_eclr level_symbol_or_edge
        ;

level_symbol_or_edge
        : level_symbol
        | edge
        ;

edge
        : '(' level_symbol level_symbol ')'
        | edge_symbol
        ;

state
        : level_symbol
        ;

next_state
        : output_symbol
        | '-'
        ;

output_symbol
        : '0'
        | '1'
        | 'x'
        | 'X'
        ;

level_symbol
        : '0'
        | '1'
        | 'x'
        | 'X'
        | '?'
        | 'b'
        | 'B'
        ;

edge_symbol
        : 'r'
        | 'R'
        | 'f'
        | 'F'
        | 'p'
        | 'P'
        | 'n'
```

```
        | 'N'
        | '*'
        ;

continuous_assign
        : YYASSIGN {}
        ;

initial_construct_opt
        :
        | initial_construct
        ;

initial_construct
        : YYINITIAL {} statement
        ;

always_construct
        : YYALWAYS {} statement
        ;

statement_opt
        :
        | statement
        ;

statement_clr
        :
        | statement_clr statement
        ;

statement
        : ';'
        | assignment ';'
        | YYIF '(' expression ')' statement
        | YYIF '(' expression ')' statement YYELSE statement
        | YYCASE '(' expression ')' case_item_eclr YYENDCASE
        | YYCASEZ '(' expression ')' case_item_eclr YYENDCASE
        | YYCASEX '(' expression ')' case_item_eclr YYENDCASE
        | YYFOREVER statement
        | YYREPEAT '(' expression ')' statement
        | YYWHILE '(' expression ')' statement
        | YYFOR '(' assignment ';' expression ';' assignment ')' statement
        | delay_control statement
        | event_control statement
        | lvalue type_action '=' delay_control expression ';'
        | lvalue type_action '=' event_control expression ';'
        | lvalue type_action '=' YYREPEAT '(' expression ')' event_control
expression ';'
        | lvalue type_action YYNBASSIGN delay_control expression ';'
        | lvalue type_action YYNBASSIGN event_control expression ';'
```

```
        | lvalue type_action YYNBASSIGN YYREPEAT '(' expression ')'
event_control expression ';'
        | YYWAIT '(' event_expression ')' statement
        | YYRIGHTARROW name_of_event ';'
        | seq_block
        | par_block
        | task_enable
        | system_task_enable
        | YYDISABLE YYID ';'
        | YYASSIGN assignment ';'
        | YYFORCE assignment ';'
        | YYDEASSIGN lvalue ';'
        | YYRELEASE lvalue ';'
        ;

delay_control
        : '#' YYINUMBER
        | '#' YYRNUMBER
        | '#' identifier
        | '#' '(' mintypmax_expression ')'
        ;

event_control
        : '@' identifier
        | '@' '(' event_expression ')'
        | '@' '(' ored_event_expression ')'
        ;

event_expression
        : expression
        | YYPOSEDGE expression
        | YYNEGEDGE expression
        | YYEDGE expression
        ;

ored_event_expression
        : event_expression YYOR event_expression
        | ored_event_expression YYOR event_expression
        ;

case_item_eclr
        : case_item
        | case_item_eclr case_item
        ;

case_item
        : expression_list ':' statement
        | YYDEFAULT ':' statement
        | YYDEFAULT statement
        ;
```

```
seq_block
        : YYBEGIN statement_clr YYEND
        | YYBEGIN ':' name_of_block block_declaration_clr statement_clr
YYEND
        ;

par_block
        : YYFORK statement_clr YYJOIN
        | YYFORK ':' name_of_block block_declaration_clr statement_clr
YYJOIN
        ;

name_of_block
        : YYID
        ;

block_declaration_clr
        :
        | block_declaration_clr block_declaration
        ;

block_declaration
        : parameter_declaration
        | reg_declaration
        | integer_declaration
        | real_declaration
        | time_declaration
        | event_declaration
        ;

task_enable
        : identifier ';'
        | identifier '(' expression_list ')' ';'
        ;

system_task_enable
        : name_of_system_task ';'
        | name_of_system_task '(' expression_list ')' ';'
        ;

name_of_system_task
        : system_identifier
        ;

system_identifier
        : YYsysID
        ;

specify_block
        : YYSPECIFY
        ;
```

```
specify_item_list
        : /* empty */
        | specify_item_list specify_item
        ;

specify_item
        : specparam_declaration
        | path_declaration
        | system_timing_check
        ;

specparam_declaration
        : YYSPECPARAM list_of_specparam_assignments ';'
        ;

list_of_specparam_assignments
        : specparam_assignment
        | list_of_specparam_assignments ',' specparam_assignment
        ;

specparam_assignment
        : YYID '=' mintypmax_expression
        | YYID '=' '(' mintypmax_expression_list_2 ')'
        ;

path_declaration
   : path_conditional_opt '(' edge_identifier_opt list_of_path_terminals
   | path_conditional_opt '(' edge_identifier_opt list_of_path_terminals
   ;

path_conditional_opt
        : /* empty */
        | YYIF '(' expression ')'
        ;

arrow
        : YYLEADTO
        | YYALLPATH
        ;

list_of_path_terminals
        : specify_terminal_descriptor
        | list_of_path_terminals ',' specify_terminal_descriptor
        ;

specify_terminal_descriptor
        : YYID
        | YYID '[' expression ']'
        | YYID '[' expression ':' expression ']'
        ;
```

```
path_delay_value
        : mintypmax_expression_list
        | '(' mintypmax_expression_list_2 ')'
        ;

edge_identifier_opt
        : /* empty */
        | YYPOSEDGE
        | YYNEGEDGE
        ;

mintypmax_expression_list
        : mintypmax_expression
        | mintypmax_expression_list ',' mintypmax_expression
        ;

mintypmax_expression_list_2
        : mintypmax_expression ',' mintypmax_expression
        | mintypmax_expression_list_2 ',' mintypmax_expression
        ;

system_timing_check
        : YYsysSETUP'(' timing_check_event ',' timing_check_event ','
        | YYsysHOLD'(' timing_check_event ',' timing_check_event ','
        | YYsysPERIOD'(' controlled_timing_check_event ','
        | YYsysWIDTH'(' controlled_timing_check_event ','
        | YYsysWIDTH'(' controlled_timing_check_event ','
        | YYsysSKEW'(' timing_check_event ',' timing_check_event ','
        | YYsysRECOVERY'(' controlled_timing_check_event ','
timing_check_event ','
        | YYsysSETUPHOLD'(' timing_check_event ',' timing_check_event ','
        | YYsysSETUPHOLD'(' timing_check_event ',' timing_check_event ','
        | YYsysNOCHANGE '('timing_check_event ',' timing_check_event ','
        ;

timing_check_event
        : timing_check_control_opt specify_terminal_descriptor
        ;

controlled_timing_check_event
        : timing_check_control specify_terminal_descriptor
        ;

timing_check_control_opt
        : /* empty */
        | timing_check_control
        ;

timing_check_control
        : YYPOSEDGE
```

```
        | YYNEGEDGE
        | YYEDGE '[' edge_descriptor_list ']'
        ;

edge_descriptor_list
        : edge_descriptor
        | edge_descriptor_list ',' edge_descriptor
        ;

edge_descriptor
       :YYINUMBER
        ;

timing_check_condition
        : /* empty */
        | YYCONDITIONING_AND expression
        ;

notify_register
        : /* empty */
        | ',' YYID
        ;

sys_arglist
        : expression
        | sys_arglist ',' expression
        ;

polarity_operator_opt
        :
        | polarity_operator
        ;

polarity_operator
        : '+'
        | '-'
        ;

parameter_value_assignment
        : '#' '(' expression_list ')'
        ;

module_instance_list
        : module_instance
        | module_instance_list ',' module_instance
        ;

module_instance
        : identifier
        ;
```

```
module_connection_list
        : module_port_connection_list
        | named_port_connection_list
        ;

module_port_connection_list
        : module_port_connection
        | module_port_connection_list ',' module_port_connection
        ;

named_port_connection_list
        : named_port_connection
        | named_port_connection_list ',' named_port_connection
        ;

module_port_connection
        :
        | expression
        ;

named_port_connection
        : '.' named_port_connection_name '(' expression ')'
        | '.' named_port_connection_name '(' ')'
        ;

named_port_connection_name
        : YYID

lvalue
        : identifier
        | identifier '[' expression ']'
        | identifier '[' expression ':' expression ']'
        | concatenation
        ;

mintypmax_expression
        : expression
        | expression ':' expression ':' expression
        ;

expression_list
        : expression_opt
        | expression_list ',' expression_opt
        ;

expression_opt
        :
        | expression
        ;

expression
```

```
        : primary
        | '+' primary %prec YYUNARYOPERATOR
        | '-' primary %prec YYUNARYOPERATOR
        | '!' primary %prec YYUNARYOPERATOR
        | '~' primary %prec YYUNARYOPERATOR
        | '&' primary %prec YYUNARYOPERATOR
        | '|' primary %prec YYUNARYOPERATOR
        | '^' primary %prec YYUNARYOPERATOR
        | YYLOGNAND primary %prec YYUNARYOPERATOR
        | YYLOGNOR primary %prec YYUNARYOPERATOR
        | YYLOGXNOR primary %prec YYUNARYOPERATOR
        | expression '+' expression
        | expression '-' expression
        | expression '*' expression
        | expression '/' expression
        | expression '%' expression
        | expression YYLOGEQUALITY expression
        | expression YYLOGINEQUALITY expression
        | expression YYCASEEQUALITY expression
        | expression YYCASEINEQUALITY expression
        | expression YYLOGAND expression
        | expression YYLOGOR expression
        | expression YYLOGNAND expression
        | expression '<' expression
        | expression '>' expression
        | expression '&' expression
        | expression '|' expression
        | expression '^' expression
        | expression YYLEQ expression
        | expression YYNBASSIGN expression
        | expression YYGEQ expression
        | expression YYLSHIFT expression
        | expression YYRSHIFT expression
        | expression YYLOGXNOR expression
        | expression '?' expression ':' expression
        | YYSTRING
        ;

constant_primary
        : YYINUMBER
        | YYRNUMBER
        | identifier
        | concatenation
        | multiple_concatenation
        ;

primary
        : constant_primary
        | identifier '[' expression ']'
        | identifier '[' expression ':'  expression ']'
        | function_call
```

```
        | '(' mintypmax_expression ')'
        | YYsysID
        | YYsysID '(' expression_list ')'
        ;

concatenation
        : '{' expression_list '}'
        ;

multiple_concatenation
        : '{' expression '{' expression_list '}' '}'
        ;

function_call
        : function_identifier '(' expression_list ')'
        ;

function_identifier
        : YYID
        | selected_name
        ;

identifier
        : YYID
        | selected_name
        ;

selected_name
        : YYID '.' YYID
        | selected_name '.' YYID
        ;
```

Index

Index (cont.)

B

Index (cont.)

C

Index (cont.)

Index (cont.)

Index (cont.)

E

Index (cont.)

F

Index (cont.)

Index (cont.)

Index (cont.)

Index (cont.)

M

Index (cont.)

N

O

Index (cont.)

P

Q

Index (cont.)

R

Index (cont.)

S

Index (cont.)

T

Index (cont.)

Index (cont.)

W

Z

Trademark Information

Mentor Graphics Trademarks

The following names are trademarks, registered trademarks, and service marks of Mentor Graphics Corporation:

3D Design™, A World of Learning℠, ABIST™, Arithmetic BIST™, AccuPARTner™, AccuParts™, AccuSim®, ADEPT™, ADVance™ MS, ADVance™ RFIC, AMPLE™, Analog Analyst™, Analog Station™, AppNotes℠, ARTgrid™, ArtRouter™, ARTshape™, ASICPlan™, ASICVector Interfaces™, Aspire™ Assess2000℠, AutoActive®, AutoCells™, AutoDissolve™, AutoFilter™, AutoFlow™, AutoLib™, AutoLinear™, AutoLink™, AutoLogic™, AutoLogic BLOCKS™, AutoLogic FPGA™, AutoLogic VHDL®, AutomotiveLib™, AutoPAR®, AutoTherm®, AutoTherm Duo™, AutoThermMCM™, AutoView™, Autowire Station™, AXEL™, AXEL Symbol Genie™, BISTArchitect™, BIST Compiler℠, BIST-In-Place℠, BIST-Ready℠, Board Architect™, Board Designer™, Board Layout™, Board Link™, Board Process Library™, Board Station®, Board Station Consumer™, BOLD Administrator™, BOLD Browser™, BOLD Composer™, BSDArchitect™, BSPBuilder™, Buy on Demand™, Cable Analyzer™, Cable Station™, CAECO Designer™, CAEFORM™, Calibre®, Calibre CB™, Calibre DESIGNrev™, Calibre DRC™, Calibre DRC-H™, Calibre FRACTUREh™, Calibre FRACTUREj™, Calibre FRACTUREk™, Calibre FRACTUREm™, Calibre FRACTUREt™, Calibre Interactive™, Calibre LITHOview™, Calibre LVS™, Calibre LVS-H™, Calibre MDPview™, Calibre MGC™, Calibre OPCpro™, Calibre OPCsbar™, Calibre ORC™, Calibre PRINTimage™, Calibre PSMgate™, Calibre PSMcheck™, Calibre RVE™, Calibre TDopc™, Calibre WORKbench™, Calibre xRC™, CAM Station™, Capture Station®, CAPITAL™, CAPITAL Analysis™, CAPITAL Bridges™, CAPITAL Documents™, CAPITAL H™, CAPITAL Harness™, CAPITAL Harness Systems™, CAPITAL H the complete desktop engineer®, CAPITAL Insight™, CAPITAL Integration™, CAPITAL Manager™, CAPITAL Manufacturer™, CAPITAL Support™, CAPITAL Systems™, Cell Builder™, Cell Station®, CellFloor™, CellGraph™, CellPlace™, CellPower™, CellRoute™, Centricity™, CEOC™, ChaseX™, CheckMate™, CHEOS™, Chip Station®, ChipGraph™, CommLib™, CommLib BMC™, Concurrent Board Process℠, Concurrent Design Environment™, Connectivity Dataport™, Continuum™, Continuum Power Analyst™, CoreAlliance™, CoreBIST™, Core Builder™, Core Factory™, Co-Verification Environment™, CTIntegrator™, DataCentric Model™, DataFusion™, Datapath™, Data Solvent™, dBUG™, Debug Detective™, DC Analyzer™, Design Architect®, Design Architect Elite™, DesignBook®, Design Capture™, Design Manager™, Design Station®, DesignView™, DesktopASIC™, Destination PCB®, DFTAdvisor™, DFTArchitect™, DFTInsight™, DirectConnect℠, DSV™, Direct System Verification™, Documentation Station™, DSS (Decision Support System)™, DSV™, E3LCable™, ECO Immunity℠, EDGE (Engineering Design Guide for Excellence)℠, EDT™, Eldo™, EldoNet™, ePartners™, EParts®, Empowering Solutions™, Engineer's Desktop™, EngineerView™, ENRead™, ENWrite™, ESim™, Exemplar™, Exemplar Logic™, Expedition™, Expert2000℠, Explorer CAECO Layout™, Explorer CheckMate™, Explorer Datapath™, Explorer Lsim™, Explorer Lsim-C™, Explorer Lsim-S™, Explorer Ltime™, Explorer Schematic™, Explorer VHDLsim™, ExpressI/O™, FabLink™, Falcon®, Falcon Framework®, FastScan™, FastStart™, FastTrack Consulting℠, First-Pass Design Success™, First-Pass success℠, FlexSim™, FlexTest™, FDL (Flow Definition Language)™, FlowTabs™, FlowXpert™, FORMA™, FormalPro™, FPGA Advantage®, FPGAdvisor™, FPGA BoardLink™, FPGA Builder™, FPGASim™, FPGA Station®, FrameConnect™, Galileo®, Gate Station®, GateGraph™, GatePlace™, GateRoute™, GDT®, GDT Core®, GDT Designer™, GDT Developer™, GENIE™, GenWare™, Geom Genie™, HDL2Graphics™, HDL Architect™, HDL Architect Station™, HDL Author™, HDL Designer™, HDL Designer Series™, HDL Detective™, HDL Inventor™, HDL Pilot™, HDL Processor™, HDL Sim™, HDLWrite™,Hardware Modeling Library™, HIC rules™, Hierarchical Injection™, Hierarchy Injection™, HotPlot®, Hybrid Designer™, Hybrid Station®, IBD™, IC Design Station™, IC Designer™, IC Layout Station™, IC Station®, ICbasic™, ICblocks™, ICcheck™, ICcompact™, ICdevice™, ICextract™, ICGen™, ICgraph™, ICLink™, IClister™, ICplan™, ICRT Controller Lcompiler™, ICrules™, ICtrace™, ICverify™, ICview™, ICX™, ICX Active™, ICX Custom Model™, ICX Custom Modeling™, ICX Plan™, ICX Pro™, ICX Project Modeling™, ICX Sentry™, ICX Standard Library™, ICX Verify™, ICX Vision™, IDEA Series™, Idea Station®, INFORM®, IFX™, Inexia™, Integrated Product Development®, Integra Station™, Integration Tool Kit™, INTELLITEST®, Interactive Layout™, Interconnect Table™, Interface-Based Design™, IntraStep℠, Inventra™, InventraIPX™, Inventra Soft Cores™, IP Engine ™, IP Evaluation Kit™, IP Factory™, IP -PCB™, IP QuickUse™, IPSim™, IS_Analyzer™, IS_Floorplanner™, IS_MultiBoard™, IS_Optimizer™, IS_Synthesizer™, ISD Creation℠, ITK™, It's More than Just Tools℠, Knowledge Center℠, Knowledge-Sourcing℠, LAYOUT™, LNL™, LBIST™, LBISTArchitect™, Language Neutral Licensing™, Lc™, Lcore™, Leaf Cell Toolkit™, Led™, LED LAYOUT™, Leonardo®, LeonardoInsight™, LeonardoSpectrum™, LIBRARIAN™, Library Builder™, Logic Analyzer on a Chip℠, Logic Builder™, Logical Cable™, LogicLib™, *logio*™, Lsim™, Lsim DSM™, Lsim-Gate™, Lsim Net™, Lsim Power Analyst™, Lsim-Review™, Lsim-Switch™, Lsim-XL™, Mach PA™, Mach TA™, Manufacture View™, Manufacturing Advisor™, Manufacturing Cable™, MaskCompose™, MaskPE®, MBIST™, MBISTArchitect™, MBIST Full-Speed™, MBIST Flex™, MBIST Manager™, MCM Designer™, MCM Station®, MDV™, MegaFunction™, Memory Builder™, Memory Builder Conductor™, Memory Builder Mozart™, Memory Designer™, Memory Model Builder™, Mentor™, Mentor Graphics®, Mentor Graphics Support CD℠, Mentor Graphics SupportBulletin℠, Mentor Graphics SupportCenter℠, Mentor Graphics SupportFax℠, Mentor Graphics SupportNet-Email℠, Mentor Graphics SupportNet-FTP℠, Mentor Graphics SupportNet-Telnet℠, Mentor Graphics We Mean Business™, MicroPlan™, MicroRoute™, Microtec®, Mixed-Signal Pro™, ModelEditor™, Model*Sim*®, Model*Sim* LNL™, Model*Sim* VHDL™, Model*Sim* VLOG™, Model*Sim* SE™, ModelStation®, Model Technology™, ModelViewer™, ModelViewer*Plus*™, *MODGEN*™, Monet®, Mslab™, Msview™, MS Analyzer™, MS Architect™, MS-Express™, MSIMON™, MTPI℠, Nanokernel®, NetCheck™, NETED™, Nucleus™, Online Knowledge Center℠, OpenDoor℠, Opsim™, OutNet™, P&RIntegrator™, PACKAGE™, PARADE™, ParallelRoute-Autocells™, ParallelRoute-MicroRoute™, PathLink™, Parts SpeciaList™, PCB-Gen™, PCB-Generator™, PCB IGES™, PCB Mechanical Interface™, PDLSim™, Personal Learning Program™, Physical Cable™, Physical Test Manager:SITE™, PLA Lcompiler™, Platform Express™, PLDSynthesis™, PLDSynthesis II™, Power Analyst™, PowerAnalyst Station™, Power To Create®, Precision™, Precision Synthesis™, Precision HLS™, Precision PNR™, Precision PTC™, Pre-Silicon™, ProjectXpert™, ProtoBoard™, ProtoView™, QNet™, QualityIBIS™, QuickCheck™, QuickConnect™, QuickFault™, QuickGrade™, QuickHDL™, QuickHDL Express™, QuickHDL Pro™, QuickPart Builder™, QuickPart Tables™, QuickParts™, QuickPath™, QuickSim™, QuickSimII™, QuickStart™, QuickUse™, QuickVHDL®, RAM Lcompiler™, RC-Delay™, RC-Reduction™, RapidExpert™, REAL Time Solutions!™, Registrar™, Reinstatement 2000℠, Reliability Advisor™, Reliability Manager™, REMEDI™, Renoir™, RF Architect™, RF Gateway™, RISE™, ROM Lcompiler™, RTL X-Press™, Satellite PCB Station™, ScalableModels™, Scaleable Verification™, SCAP™, Scan-Sequential™, Scepter™, Scepter DFF™, Schematic View Compiler, SVC™, Schemgen™, SDF™ (Software Data Formatter), SDL2000 Lcompiler™, Seamless®, Seamless C-Bridge™, Seamless Co-Designer™, Seamless CVE™, Seamless Express™, Selective Promotion™, SignaMask OPC™, Signal Spy™, Signal Vision™, Signature Synthesis™, Simulation Manager™, SimExpress™, SimPilot™, SimView™, SiteLine2000℠, SmartMask™, SmartParts™, SmartRouter™, SmartScripts™, Smartshape™, SNX™, SneakPath Analyzer™, SOS Initiative™, Source Explorer™, SpeedGate™, SpeedGate DSV™, SpiceNet™, SST Velocity®, Standard Power Model Format (SPMF)™, Structure Recovery™, Super C™, Super IC Station™, Support Services BaseLine℠, Support Services ClassLine℠, Support Services Latitudes℠, Support Services OpenLine℠, Support Services PrivateLine℠, Support Services SiteLine℠, Support Services TechLine℠, Support Services RemoteLine℠, Symbol Genie™, Symbolscript™, SYMED™, SynthesisWizard™, System Architect™, System Design Station™, System Modeling Blocks™, Systems on Board Initiative™, System Vision™, Target Manager™, Tau®, TeraCell™, TeraPlace™, TeraPlace-GF™, TechNotes™, The Ultimate Tool for HDL Simulation™, TestKompress™, Test Station®, Test Structure Builder™, The Ultimate Site For HDL Simulation™, TimeCloser™, Timing Builder™, TNX™, ToolBuilder™, TrueTiming™, Vlog™, V-Express™, V-Net™, VHDLnet™, VHDLwrite™, Verinex™, ViewCreator™, ViewWare®, Virtual Library™, Virtual Target™, Virtual Test Manager:TOP™, VR-Process℠, VRTX®, VRTXmc™, VRTXoc™, VRTXsa™, VRTX32®, Waveform DataPort™, We Make TMN Easy™, Wiz-o-matic™, WorkXpert™, xCalibre™, xCalibrate™, Xconfig™, XlibCreator™, Xpert™, Xpert API™, XpertBuilder™, Xpert Dialogs™, Xpert Profiler™, XRAY®, XRAY MasterWorks®, XSH®, Xtrace®, Xtrace Daemon™, Xtrace Protocol™, Zeelan®, Zero Tolerance Verification™, Zlibs™

Third-Party Trademarks

The following names are trademarks, registered trademarks, and service marks of other companies that appear in Mentor Graphics product publications:

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Exchange, FrameMaker, FrameViewer, PostScript,and Reader are registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Altera, ByteBlaster, Excalibur, and Quartus are trademarks or registered trademarks of Altera Corporation in the United States and other countries.

AM188, AMD, AMD-K6, and AMD Athlon Processor are trademarks of Advanced Micro Devices, Inc.

Apple and Laserwriter are registered trademarks of Apple Computer, Inc.

ARIES is a registered trademark of Aries Technology.

AMBA, ARM, ARMulator, ARM7TDMI, ARM7TDMI-S, ARM9TDMI, ARM9E-S, ARM946E-S, ARM966E-S, EmbeddedICE, StrongARM, TDMI, and Thumb are trademarks or registered trademarks of ARM Limited.

ASAP, Aspire, C-FAS, CMPI, Eldo-FAS, EldoHDL, Eldo-Opt, Eldo-UDM, EldoVHDL, Eldo-XL, Elga, Elib, Elib-Plus, ESim, Fidel, Fideldo, GENIE, GENLIB, HDL-A, MDT, MGS-MEMT, MixVHDL, Model Generator Series (MGS), Opsim, SimLink, SimPilot, SpecEditor, Success, SystemEldo, VHDeLDO and Xelga are registered trademarks of ANACAD Electrical Engineering Software, a unit of Mentor Graphics Corporation.

Avant! and Star-Hspice are trademarks of Avant! Corporation.

AVR is a registered trademark of Atmel Corporation.

Cadence, Affirma signalscan, Allegro, Analog Artist, Composer, Concept, Design Planner, Dracula, GDSII, GED, HLD Systems, Leapfrog, Logic DP, NC-Verilog, OCEAN, Physical DP, Pillar, Silicon Ensemble, Spectre, Verilog, Verilog XL, Veritime, and Virtuoso are trademarks or registered trademarks of Cadence Design Systems, Inc.

CAE+Plus and ArchGen are registered trademarks of Cynergy System Design.

CalComp is a registered trademark of CalComp, Inc.

Canon is a registered trademark of Canon, Inc. BJ-130, BJ-130e, BJ-330, and Bubble Jet are trademarks of Canon, Inc.

Centronics is a registered trademark of Centronics Data Computer Corporation.

ColdFire and M-Core are registered trademarks of Motorola, Inc.

Ethernet is a registered trademark of Xerox Corporation.

Foresight and Foresight Co-Designer are trademarks of Nu Thena Systems, Inc.

FLEXlm is a trademark of Globetrotter Software, Inc.

GenCAD is a trademark of Teradyne Inc.

Hewlett-Packard (HP), LaserJet, MDS, HP-UX, PA-RISC, APOLLO, DOMAIN and HPare registered trademarks of Hewlett-Packard Company.

HCL-eXceed and HCL-eXceed/W are registered trademark of Hummingbird Communications. Ltd.

HyperHelp is a trademark of Bristol Technology Inc.

Installshield is a registered trademark and service mark of InstallShield Corporation.

IBM, PowerPC, and RISC Systems/6000 are trademarks of International Business Machines Corporation.

I-DEAS and UG/Wiring are registered trademarks of Electronic Data Systems Corporation.

IKON is a trademark of Tahoma Technology.

IKOS and Voyager are registered trademarks of IKOS Systems, Inc.

Imagen, QMS, QMS-PS 820, Innovator, and Real Time Rasterization are registered trademarks of MINOLTA-QMS Inc. imPRESS and UltraScript are trademarks of MINOLTA-QMS Inc.

ImageGear is a registered trademark of AccuSoft Corporation.

Infineon, TriCore, and C165 are trademarks of Infineon Technologies AG.

Intel, i960, i386, and i486 are registered trademarks of Intel Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Linux is a registered trademark of Linus Torvalds.

MemoryModeler MemMaker are trademarks of Denali Software, Inc.

MIPS is a trademark of MIPS Technologies, Inc.

MS-DOS, Windows 95, Windows 98, Windows 2000, and Windows NT are registered trademarks of Microsoft Corporation.

MULTI is a registered trademark of Green Hills Software, Inc.

NEC and NEC EWS4800 are trademarks of NEC Corp.

Netscape is a trademark of Netscape Communications Corporation.

Novas, Debussy, and nWave are trademarks or registered trademarks of Novas Software, Inc.

OakDSPCore is a registered trademark for DSP Group, Inc.

Oracle, Oracle8i, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation.

OSE is a registered trademark of OSE Systems.

PKZIP is a registered trademark of PKWARE, Inc.

Pro/CABLING and HARNESSDESIGN are trademarks or registered trademarks of Parametric Technology Corporation.

Quantic is a registered trademark of Quantic EMC Inc.

QUASAR is a trademark of ASM Lithography Holding N.V.

End-User License Agreement

IMPORTANT - USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE

This license is a legal "Agreement" concerning the use of Software between you, the end-user, either individually or as an authorized representative of the company purchasing the license, and Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Mentor Graphics (Singapore) Private Limited, and their majority-owned subsidiaries ("Mentor Graphics"). USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. If you do not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within 10 days after receipt of Software and receive a full refund of any license fee paid

END-USER LICENSE AGREEMENT

1. **GRANT OF LICENSE**. The software programs you are installing, downloading, or have acquired with this Agreement, including any updates, modifications, revisions, copies, and documentation ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics or its authorized distributor grants to you, subject to payment of appropriate license fees, a nontransferable, nonexclusive license to use Software solely: (a) (in machine-readable, object-code form; (b) for your internal business purposes; and (c) on the computer hardware or at the site for which an applicable license fee is paid, or as authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics' then-current standard policies, which vary depending on Software, license fees paid or service plan purchased, apply to the following and are subject to change: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be communicated and technically implemented through the use of authorization codes or similar devices); (c) eligibility to receive updates, modifications, and revisions; and (d) support services provided. Current standard policies are available upon request.

2. **ESD SOFTWARE**. If you purchased a license to use embedded software development (ESD) Software, Mentor Graphics or its authorized distributor grants to you a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESD compilers, including the ESD run-time libraries distributed with ESD C and C++ compiler Software that are linked into a composite program as an integral part of your compiled computer program, provided that you distribute these files only in conjunction with your compiled computer program. Mentor Graphics does NOT grant you any right to duplicate or incorporate copies of Mentor Graphics' real-time operating systems or other ESD Software, except those explicitly granted in this section, into your products without first signing a separate agreement with Mentor Graphics for such purpose.

3. **BETA CODE**

   3.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to you a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

   3.2. If Mentor Graphics authorizes you to use the Beta Code, you agree to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. You will contact Mentor Graphics

periodically during your use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of your evaluation and testing, you will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

3.3. You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceives or makes during or subsequent to this Agreement, including those based partly or wholly on your feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this subsection shall survive termination or expiration of this Agreement.

4. **RESTRICTIONS ON USE**. You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than your employer's employees and contractors, excluding Mentor Graphics' competitors, whose job performance requires access. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by the European Union Software Directive or local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it without Mentor Graphics' prior written consent. The provisions of this section shall survive the termination or expiration of this Agreement.

5. **LIMITED WARRANTY**

5.1. Mentor Graphics warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LOANED TO YOU FOR A LIMITED TERM OR AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

5.2. THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. **LIMITATION OF LIABILITY**. EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE STATUTE OR REGULATION, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER

LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER.

7. **LIFE ENDANGERING ACTIVITIES**. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH SUCH USE.

8. **INFRINGEMENT**

   8.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright in the United States, Canada, Japan, Switzerland, Norway, Israel, Egypt, or the European Union. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the claim, provided that you: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the claim; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the claim.

   8.2. If an infringement claim is made, Mentor Graphics may, at its option and expense, either (a) replace or modify Software so that it becomes noninfringing, or (b) procure for you the right to continue using Software. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

   8.3. Mentor Graphics has no liability to you if the alleged infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you design or market; (f) any Beta Code contained in Software; or (g) any Software provided by Mentor Graphics' licensors which do not provide such indemnification to Mentor Graphics' customers.

   8.4. THIS SECTION 8 STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

9. **TERM**. This Agreement remains effective until expiration or termination. This Agreement will automatically terminate if you fail to comply with any term or condition of this Agreement or if you fail to pay for the license when due and such failure to pay continues for a period of 30 days after written notice from Mentor Graphics. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics' reasonable satisfaction.

10. **EXPORT**. Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export in any manner any Software or direct product of Software, without first obtaining all necessary approval from appropriate local and United States government agencies.

11. **RESTRICTED RIGHTS NOTICE**. Software has been developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable. Contractor/manufacturer is Mentor Graphics Corporation, 8005 Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

12. **THIRD PARTY BENEFICIARY**. For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth in this Agreement.

13. **CONTROLLING LAW**. This Agreement shall be governed by and construed under the laws of Ireland if the Software is licensed for use in Israel, Egypt, Switzerland, Norway, South Africa, or the European Union, the laws of Japan if the Software is licensed for use in Japan, the laws of Singapore if the Software is licensed for use in Singapore, People's Republic of China, Republic of China, India, or Korea, and the laws of the state of Oregon if the Software is licensed for use in the United States of America, Canada, Mexico, South America or anywhere else worldwide not provided for in this section

14. **SEVERABILITY**. If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

15. **MISCELLANEOUS**. This Agreement contains the entire understanding between the parties relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions, except valid license agreements related to the subject matter of this Agreement which are physically signed by you and an authorized agent of Mentor Graphics. This Agreement may only be modified by a physically signed writing between you and an authorized agent of Mentor Graphics. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse. The prevailing party in any legal action regarding the subject matter of this Agreement shall be entitled to recover, in addition to other relief, reasonable attorneys' fees and expenses.