This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are 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 provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (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 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: www.mentor.com/supportnet
Contact Your Technical Writer: www.mentor.com/supportnet/documentation/reply_form.cfm

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics’ trademarks may be viewed at: www.mentor.com/terms_conditions/trademarks.cfm.
# Table of Contents

## Chapter 1
Licensing ................................................................. 1-1

## Chapter 2
Overview ................................................................. 2-1
Product Description .................................................... 2-1
Calibre DRC / DRC-H / MT DRC-H / MTflex DRC-H ............... 2-1
Calibre LVS / LVS-H / MT LVS-H / MTflex LVS-H ............... 2-2
Calibre CB .............................................................. 2-2
Calibre Verification Utilities ......................................... 2-2
Audience .................................................................... 2-3
Syntax Conventions ..................................................... 2-3

## Chapter 3
Invocation ................................................................. 3-1
Before Invocation ....................................................... 3-1
Setting the MGC_HOME Environment Variable .................... 3-1
Required Input Files .................................................. 3-1
Rule File .................................................................. 3-2
Layout Database ....................................................... 3-3
Source Database ....................................................... 3-8
Invocation Procedures ................................................ 3-8
Calibre DRC/DRC-H .................................................. 3-9
Calibre LVS/LVS-H .................................................. 3-13
Examples ............................................................... 3-23
Calibre CB .............................................................. 3-25
Other Calibre Verification Tools ..................................... 3-26
Calibre Version Information ........................................ 3-26

## Chapter 4
DRC Concepts ............................................................ 4-1
Layers .................................................................... 4-2
Layer Types ............................................................. 4-2
Layer Type Summary .................................................. 4-4
Layer Definitions ....................................................... 4-5
Layer Operations ....................................................... 4-7
Layer Operation Classifications ..................................... 4-7
Layer of Origin ......................................................... 4-9
Rule Check Statements ............................................... 4-10
Rule Check Comments ............................................... 4-12
Check Text .............................................................. 4-12
Dimensional Check Operations ...................................... 4-13
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secondary Keywords</td>
<td>4-15</td>
</tr>
<tr>
<td>Edge Measurement</td>
<td>4-16</td>
</tr>
<tr>
<td>Measurement Region Construction</td>
<td>4-16</td>
</tr>
<tr>
<td>Metrics</td>
<td>4-18</td>
</tr>
<tr>
<td>Edge Cluster Generation</td>
<td>4-25</td>
</tr>
<tr>
<td>Interval Constraints for Output Suppression</td>
<td>4-29</td>
</tr>
<tr>
<td>Appropriateness Criteria</td>
<td>4-31</td>
</tr>
<tr>
<td>Intersection Criteria</td>
<td>4-33</td>
</tr>
<tr>
<td>Edge Breaking</td>
<td>4-33</td>
</tr>
<tr>
<td>Polygon Containment Criteria</td>
<td>4-35</td>
</tr>
<tr>
<td>Edge-Directed Output</td>
<td>4-36</td>
</tr>
<tr>
<td>Polygon-Directed Output</td>
<td>4-37</td>
</tr>
<tr>
<td>False Measurement Reduction</td>
<td>4-39</td>
</tr>
<tr>
<td>Measurement Tolerances</td>
<td>4-41</td>
</tr>
<tr>
<td>Error Tolerance Setting</td>
<td>4-41</td>
</tr>
<tr>
<td>Disk-Based Layers</td>
<td>4-41</td>
</tr>
<tr>
<td>Specialized DRC Applications</td>
<td>4-42</td>
</tr>
<tr>
<td>Datatypes and Texttypes in Calibre</td>
<td>4-42</td>
</tr>
<tr>
<td>Dual Database Capability</td>
<td>4-43</td>
</tr>
<tr>
<td>Mask DRC Results</td>
<td>4-46</td>
</tr>
<tr>
<td>Incremental Connectivity and Antenna Checks</td>
<td>4-49</td>
</tr>
<tr>
<td>Soft Connection Checks</td>
<td>4-55</td>
</tr>
<tr>
<td>Layout Input Control in Calibre</td>
<td>4-58</td>
</tr>
<tr>
<td>Area-Based Filtering</td>
<td>4-61</td>
</tr>
<tr>
<td>Flagging and Snapping Original Shapes</td>
<td>4-61</td>
</tr>
<tr>
<td>Input Layout Database Magnification</td>
<td>4-62</td>
</tr>
<tr>
<td><strong>Chapter 5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DRC Execution</strong></td>
<td>5-1</td>
</tr>
<tr>
<td>Data Flow in DRC</td>
<td>5-1</td>
</tr>
<tr>
<td>Invocation</td>
<td>5-1</td>
</tr>
<tr>
<td>Rule File Compilation</td>
<td>5-2</td>
</tr>
<tr>
<td>DRC Specification Statements</td>
<td>5-2</td>
</tr>
<tr>
<td>Rule Check Selection</td>
<td>5-2</td>
</tr>
<tr>
<td>DRC Rule Check Result Limits</td>
<td>5-2</td>
</tr>
<tr>
<td>Control of Empty Rule Checks</td>
<td>5-3</td>
</tr>
<tr>
<td>General Execution Characteristics</td>
<td>5-3</td>
</tr>
<tr>
<td>Concurrency</td>
<td>5-3</td>
</tr>
<tr>
<td>Redundancy Elimination</td>
<td>5-6</td>
</tr>
<tr>
<td>Layer Operation Scheduling</td>
<td>5-6</td>
</tr>
<tr>
<td>Polygon Segmentation</td>
<td>5-11</td>
</tr>
<tr>
<td>Layout Database End Segment Warning</td>
<td>5-12</td>
</tr>
<tr>
<td><strong>Chapter 6</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Hierarchical DRC</strong></td>
<td>6-1</td>
</tr>
<tr>
<td>Theory of Operation</td>
<td>6-1</td>
</tr>
<tr>
<td>DRC Data Storage</td>
<td>6-2</td>
</tr>
<tr>
<td>Flat Instantiations</td>
<td>6-4</td>
</tr>
</tbody>
</table>
# Table of Contents

Hierarchical Operation Efficiency ............................................. 6-4  
False Notch Error Suppression .................................................. 6-5  
Layer Area Printing ............................................................. 6-5  
Text Mapping ................................................................. 6-6  
Hierarchy-Specific Statements ................................................. 6-6  
DRC Use of Hcells ............................................................. 6-6

**Chapter 7**  
DRC Results .............................................................................. 7-1  
Session Transcript ............................................................... 7-1  
Rule File Compilation ........................................................... 7-1  
Layout Data Input ............................................................... 7-1  
Initialization Section .............................................................. 7-2  
Executive Process ............................................................... 7-3  
DRC Results Database .......................................................... 7-6  
ASCII and Binary DRC Results Databases ................................. 7-8  
GDSII DRC Results Database Format ....................................... 7-13  
OASIS DRC Results Database Format ..................................... 7-14  
Hierarchical DRC Results Database ....................................... 7-16  
DRC Summary Report ........................................................... 7-17

**Chapter 8**  
Connectivity Extraction .......................................................... 8-1  
Establishing and Verifying Connectivity .................................... 8-1  
Mask Connectivity Extraction .................................................. 8-2  
Connectivity and Rule File Compilation .................................... 8-2  
Establishing Connectivity for Layer Operations ......................... 8-3  
Recognizing Electrical Nets ..................................................... 8-3  
Shapes on a Single Layer ....................................................... 8-3  
Connect ............................................................................ 8-3  
Connect By .......................................................... 8-4  
Sconnect ................................................................. 8-4  
Stamp ................................................................. 8-5  
Ports and Pins ............................................................... 8-5  
Transferring Logical Information to Merged Layers ................... 8-7  
Attach Operation .............................................................. 8-8  
Net Name Specification ....................................................... 8-8  
Label Attachment ............................................................. 8-10  
Virtual Connect Statements .................................................. 8-14  
Short Isolation ................................................................. 8-15  
Short Isolation in Hierarchical Mode ....................................... 8-16  
Hierarchical Short Isolation Results Database ......................... 8-17  
Connectivity Extraction Errors and Warnings ............................ 8-18

**Chapter 9**  
Electrical Rule Checks ........................................................... 9-1  
ERC Statements and Operations .............................................. 9-1  
Execution of ERC Operations in LVS ......................................... 9-2
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERC PRINT Options</td>
<td>9-3</td>
</tr>
<tr>
<td>Rule Check Selection in LVS</td>
<td>9-3</td>
</tr>
<tr>
<td>ERC Operations in DRC</td>
<td>9-3</td>
</tr>
<tr>
<td>Rule Check Selection in DRC</td>
<td>9-4</td>
</tr>
<tr>
<td>ERC Licensing</td>
<td>9-4</td>
</tr>
<tr>
<td>ERC Output Files</td>
<td>9-4</td>
</tr>
<tr>
<td>ERC Results Database</td>
<td>9-4</td>
</tr>
<tr>
<td>ERC Auxiliary Files</td>
<td>9-5</td>
</tr>
<tr>
<td>ERC Examples</td>
<td>9-6</td>
</tr>
<tr>
<td>Chapter 10</td>
<td></td>
</tr>
<tr>
<td><strong>Device Recognition</strong></td>
<td>10-1</td>
</tr>
<tr>
<td>Concepts and Terminology</td>
<td>10-1</td>
</tr>
<tr>
<td>Recognition Logic</td>
<td>10-3</td>
</tr>
<tr>
<td>Layer Relations</td>
<td>10-3</td>
</tr>
<tr>
<td>Pin Relations</td>
<td>10-4</td>
</tr>
<tr>
<td>Fill-In Algorithm</td>
<td>10-5</td>
</tr>
<tr>
<td>Ill-Formed Devices</td>
<td>10-5</td>
</tr>
<tr>
<td>Recognition Example</td>
<td>10-6</td>
</tr>
<tr>
<td>Bad Device Reporting</td>
<td>10-6</td>
</tr>
<tr>
<td>Hierarchical Device Recognition</td>
<td>10-9</td>
</tr>
<tr>
<td>Property Computation</td>
<td>10-9</td>
</tr>
<tr>
<td>Default Property Computations</td>
<td>10-11</td>
</tr>
<tr>
<td>User-Defined Property Computation</td>
<td>10-14</td>
</tr>
<tr>
<td>Chapter 11</td>
<td></td>
</tr>
<tr>
<td><strong>LVS Circuit Comparison</strong></td>
<td>11-1</td>
</tr>
<tr>
<td>Data Flow in LVS</td>
<td>11-1</td>
</tr>
<tr>
<td>LVS Comparison</td>
<td>11-1</td>
</tr>
<tr>
<td>Component Types</td>
<td>11-3</td>
</tr>
<tr>
<td>Component Subtypes</td>
<td>11-3</td>
</tr>
<tr>
<td>Naming Conventions</td>
<td>11-4</td>
</tr>
<tr>
<td>Instance Pins and Pin Names</td>
<td>11-4</td>
</tr>
<tr>
<td>User-Given Names</td>
<td>11-5</td>
</tr>
<tr>
<td>Net and Instance Names</td>
<td>11-6</td>
</tr>
<tr>
<td>Ports and Port Names</td>
<td>11-6</td>
</tr>
<tr>
<td>Power and Ground Nets</td>
<td>11-7</td>
</tr>
<tr>
<td>Built-In Device Types</td>
<td>11-7</td>
</tr>
<tr>
<td>MOS Transistors</td>
<td>11-8</td>
</tr>
<tr>
<td>Capacitors</td>
<td>11-10</td>
</tr>
<tr>
<td>Resistors</td>
<td>11-10</td>
</tr>
<tr>
<td>Diodes</td>
<td>11-11</td>
</tr>
<tr>
<td>Bipolar Transistors</td>
<td>11-12</td>
</tr>
<tr>
<td>Jfet Transistors</td>
<td>11-12</td>
</tr>
<tr>
<td>Inductors</td>
<td>11-13</td>
</tr>
<tr>
<td>Voltage Sources</td>
<td>11-13</td>
</tr>
<tr>
<td>X+ Devices</td>
<td>11-14</td>
</tr>
<tr>
<td>MS and MF Schematic Devices</td>
<td>11-14</td>
</tr>
</tbody>
</table>
# Table of Contents

Matching of Circuit Elements ...................................................... 11-15
  Connectivity Comparison Results .............................................. 11-16
  Initial Correspondence Points .................................................. 11-16
  Resolving Ambiguities ............................................................. 11-16
Device Reduction ................................................................. 11-17
  Device Reduction Semantics .................................................... 11-17
  Device Reduction Programs ..................................................... 11-35
  Tolerance in Device Reduction .................................................. 11-36
Device Filtering ................................................................. 11-39
  Filtering Unused MOS Transistors ............................................. 11-39
  Filtering Unused Bipolar Transistors ....................................... 11-40
  Unused Device Filtering .......................................................... 11-40
Nets ....................................................................................... 11-40
  Usage of Power and Ground Nets .............................................. 11-41
  Floating Named Nets ............................................................... 11-41
  Isolated Nets ........................................................................... 11-41
  Pass-Through Nets ................................................................... 11-41
  Unattached Ports ....................................................................... 11-42
Logic Gate Recognition ............................................................ 11-42
  Recognition Processes ............................................................... 11-43
  Regular CMOS Gates ................................................................. 11-44
  Regular NMOS Gates ................................................................. 11-54
  LDD Gates ............................................................................... 11-59
  Excluding Transistors ............................................................... 11-63
  Overriding Of Pin Swapping In Logic Gates ............................... 11-63
  Overriding of Device Swapping In Logic Gates ......................... 11-64
  Pin Swapping ........................................................................... 11-64
  Tracing Properties ................................................................... 11-69
  Built-In Property Classification ................................................. 11-69
  Reading Built-In W/L Partner Properties .................................... 11-69
  SPICE-Like Property Syntax ....................................................... 11-72

Chapter 12
Hierarchical LVS ................................................................. 12-1
  Hierarchical Circuit Extraction ................................................. 12-1
  Hierarchical LVS Comparison ..................................................... 12-1
  Model Names ............................................................................. 12-2
  Connectivity Dependent Transformation .................................... 12-2
  Isolated Layout Nets ................................................................. 12-2
  Hierarchical Device Recognition ................................................. 12-2
  Hierarchical Layer Operations .................................................... 12-3
  Cell Pushdown .......................................................................... 12-4
  Hcells ......................................................................................... 12-5
    Many-Many Cell Correspondence ............................................. 12-7
  Hierarchical Pins ....................................................................... 12-8
    Matching Hcell Pins ............................................................... 12-8
    Trivial Pin Swappability .......................................................... 12-8
    SRAM Bit-Cell Recognition ....................................................... 12-9
Table of Contents

High-Short Resolution .................................................. 12-11
Parameterized Cells ..................................................... 12-13
Logic Injection ............................................................. 12-14
Injected Components and the LVS Report.............................. 12-15
Logic Injection and Gate Recognition................................. 12-18
Internal Net Matching .................................................. 12-18
Logic Injection and Pin Swappability ................................. 12-18
Changes in Calibre Transcript ........................................ 12-19
Injection Types .......................................................... 12-20
Hierarchical Cell Cycles ................................................ 12-42
Hierarchical SPICE ...................................................... 12-42
Dollar Signs in Cell Names ............................................ 12-42
Net Names ............................................................... 12-43
Ports ....................................................................... 12-43
String Property Values ................................................ 12-44
M Device Representation ................................................ 12-44
Cell Statistics ........................................................... 12-44
Case Preservation for Device Operations .............................. 12-45
$D Parameters in SPICE Netlists .................................... 12-45
Hierarchical Netlister Warnings ...................................... 12-45

Chapter 13
LVS Results ................................................................... 13-1
LVS Report ................................................................. 13-1
Overall Structure — Flat .................................................. 13-1
Overall Structure — Hierarchical ...................................... 13-6
Overall Structure — SPICE Syntax Check ........................... 13-13
Analyzing the LVS Report .............................................. 13-14
Errors and Warnings ..................................................... 13-16
LVS Report Listing Conventions ...................................... 13-17
Overall Comparison Results ............................................ 13-20
Errors in Names Given for Power/Ground Nets .................... 13-27
Absolute Trace Property Tolerances Out of Range .............. 13-27
Component Types with Non-Identical Signal Pins ................. 13-28
Input Errors ............................................................... 13-30
Hierarchical Cells Forming a Cycle .................................. 13-31
LVS Discrepancy Types ................................................ 13-32
Information and Warnings .............................................. 13-43
Detailed Instance Connections ....................................... 13-46
Unmatched Elements ..................................................... 13-46
Circuit Extraction Report ................................................ 13-48
Mask Results Database .................................................. 13-48
SVDB Cross-Reference Files .......................................... 13-48
Flat Instance Cross-Reference File .................................... 13-48
Flat Net Cross-Reference File .......................................... 13-50
Hierarchical Instance and Net Cross-Reference Files .......... 13-52
Source and Layout Placement Hierarchy Files .................... 13-53
SVDB Header .............................................................. 13-53
# Table of Contents

Binary Polygon File (BPF) Database. ................................................. 13-54

## Chapter 14

### SPICE Format ................................................................. 14-1

  - Introduction .................................................................. 14-1
  - General SPICE Syntax .................................................. 14-2
    - SPICE Notational Conventions .................................... 14-2
  - Control Statements ...................................................... 14-6
    - .END ........................................................................ 14-6
    - .ENDL ...................................................................... 14-7
    - .INCLUDE ................................................................ 14-7
    - .LIB ....................................................................... 14-8
    - .OPTION SCALE ..................................................... 14-9
    - *.BUSDELIMITER ..................................................... 14-11
    - *.CAPA ................................................................... 14-11
    - *.CONNECT ............................................................ 14-11
    - *.DIODE .................................................................. 14-12
    - *.EQUIV .................................................................. 14-13
    - *.LDD ..................................................................... 14-14
    - *.MEGA .................................................................... 14-14
    - *.PININFO ................................................................ 14-16
    - *.SEEDPROM .......................................................... 14-16
    - *.XPINS .................................................................. 14-16
  - Other Control Statements .................................................. 14-17
  - Element Statements ....................................................... 14-18
    - Resistor Element ...................................................... 14-18
    - Capacitor Element .................................................... 14-20
    - Inductor Element ..................................................... 14-22
    - Junction Diode Element ............................................. 14-24
    - BJT Element ........................................................... 14-26
    - JFET Element ........................................................... 14-28
    - MOSFET Element ..................................................... 14-29
    - Voltage Source Element ............................................. 14-32
  - Specification Statements for SPICE M Elements ................. 14-33
    - Subcircuits ................................................................ 14-33
    - .PARAM ................................................................... 14-43
    - String Parameters ..................................................... 14-44
    - .GLOBAL .................................................................. 14-45

## Chapter 15

### Utilities ................................................................. 15-1

  - Verilog-to-LVS ........................................................ 15-1
    - Description ............................................................ 15-1
    - Usage ....................................................................... 15-3
    - Arguments ............................................................... 15-3
    - Library Files ........................................................... 15-8
    - Supported Verilog Syntax ......................................... 15-8
    - Modules .................................................................. 15-8
# Table of Contents

- Declarations .................................................. 15-10
- Primitive Instances ........................................... 15-15
- Module Instantiations .......................................... 15-16
- UDP Declaration and Instantiation ........................... 15-21
- Behavioral Statements ......................................... 15-21
- Specify Section ................................................ 15-22
- Expressions .................................................... 15-22
- Other Language Features ...................................... 15-25
- Library Interface Files ........................................ 15-25
- Using the –e Switch to Create LVS Box Subcircuits .... 15-30
- Using –i to Generate Simulation Output ................... 15-31
- Generating a Calibre xRC Source Template File ......... 15-32
- V2LVS Errors and Warnings ................................... 15-33
- EDIF-to-LVS .................................................... 15-39
  - Description .................................................. 15-39
  - Usage ......................................................... 15-40
  - Arguments ................................................... 15-40
  - Examples .................................................... 15-42
  - Untranslated EDIF Syntax ................................ 15-43
  - EDIF Versus SPICE Syntax Considerations ............ 15-43
  - EDIF-to-SPICE Translation Issues ...................... 15-45
  - EDIF-to-SPICE Translations ............................... 15-47
  - Netlist Example ............................................. 15-53
- Compare Two GDSII Databases ................................. 15-56
- Create a Rule File for Generating Unused Layers ....... 15-57
- Create a Rule File for Comparing Two Layout Databases ........................... 15-59
- Other Utilities .................................................. 15-60

## Appendix A

### Related Publications ............................................ A-1
  - Configuration and Licensing ............................... A-1
  - Calibre RET and MDP ...................................... A-1
  - Calibre Verification ........................................ A-2
  - Calibre YieldAnalyzer and Calibre YieldEnhancer ... A-2
  - Calibre xL ................................................... A-3
  - Calibre xRC ................................................ A-3
  - Common User's Manuals .................................... A-3

### Index

- Open Source and Third-Party Information
- End-User License Agreement
**List of Figures**

<p>| Figure 4-1. Edge-Polygon Relationship | 4-1 |
| Figure 4-2. Layer Types | 4-2 |
| Figure 4-3. Layer Types and Data Flow in the DRC System | 4-5 |
| Figure 4-4. Coincident Edge Operation | 4-9 |
| Figure 4-5. Measured Edges in the Dimensional Check Operations | 4-15 |
| Figure 4-6. Generation of Output Edges | 4-16 |
| Figure 4-7. Measurement Region Formation | 4-17 |
| Figure 4-8. Metric Determination of Boundary Formation | 4-19 |
| Figure 4-9. Commutative Measurement | 4-20 |
| Figure 4-10. Non-Commutative Measurement | 4-20 |
| Figure 4-11. Output Adjustments for the Opposite Metric | 4-21 |
| Figure 4-12. Opposite Symmetric and Opposite FSymmetric | 4-23 |
| Figure 4-13. Square Orthogonal Measurement Region | 4-23 |
| Figure 4-14. SQUARE ORTHOGONAL Measurement Region (45-degree) | 4-24 |
| Figure 4-15. SQUARE ORTHOGONAL Measurement Region (Skew Edge) | 4-25 |
| Figure 4-16. Three-Edge Output Cluster | 4-26 |
| Figure 4-17. Four-Edge Output Cluster | 4-27 |
| Figure 4-18. Trivial Edge Generation | 4-28 |
| Figure 4-19. Point-to-Point Trivial Edge Generation | 4-29 |
| Figure 4-20. Suppressing Redundant Errors (part 1) | 4-30 |
| Figure 4-21. Suppressing Redundant Errors (part 2) | 4-31 |
| Figure 4-22. Edge Inside and Outside Planes | 4-32 |
| Figure 4-23. Appropriate Angles Between the Outsides of Edges | 4-33 |
| Figure 4-24. Edge Breaking in a Two-Layer Dimensional Check Operation | 4-34 |
| Figure 4-25. Polygon Containment | 4-35 |
| Figure 4-26. Error Reduction Using Polygon-Directed Output | 4-39 |
| Figure 4-27. False Notch Measurement (single polygon) | 4-39 |
| Figure 4-28. False Notch Measurement (two polygons) | 4-40 |
| Figure 5-1. DRC Data Flow | 5-1 |
| Figure 6-1. Hierarchical AND Operation | 6-2 |
| Figure 6-2. Hierarchical Error Suppression | 6-3 |
| Figure 7-1. ASCII DRC Results Database Example | 7-9 |
| Figure 8-1. Connected Shapes on a Single Layer | 8-3 |
| Figure 8-2. Polygons Connected Directly | 8-4 |
| Figure 8-3. Polygons Connected By Contact | 8-4 |
| Figure 8-4. Sconnect Operation | 8-4 |
| Figure 8-5. Example of Virtual Connect Box | 8-15 |
| Figure 10-1. Efficient Function Choice | 10-28 |
| Figure 11-1. LVS Data Flow | 11-1 |
| Figure 11-2. Parallel and Series Device Reduction | 11-18 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-3</td>
<td>Parallel MOS Transistor Reduction</td>
<td>11-19</td>
</tr>
<tr>
<td>11-4</td>
<td>Effective AS/AD Computation With Pin Swapping</td>
<td>11-20</td>
</tr>
<tr>
<td>11-5</td>
<td>Series MOS Transistor Reduction</td>
<td>11-21</td>
</tr>
<tr>
<td>11-6</td>
<td>Reduce Semi-Series MOS</td>
<td>11-23</td>
</tr>
<tr>
<td>11-7</td>
<td>Split Gate Reduction</td>
<td>11-25</td>
</tr>
<tr>
<td>11-8</td>
<td>Reduce Split Gates Example</td>
<td>11-26</td>
</tr>
<tr>
<td>11-9</td>
<td>SAME ORDER Option</td>
<td>11-27</td>
</tr>
<tr>
<td>11-10</td>
<td>Parallel Bipolar Transistor Reduction</td>
<td>11-28</td>
</tr>
<tr>
<td>11-11</td>
<td>Series Capacitor Reduction</td>
<td>11-29</td>
</tr>
<tr>
<td>11-12</td>
<td>Parallel Capacitor Reduction</td>
<td>11-30</td>
</tr>
<tr>
<td>11-13</td>
<td>Series Resistor Reduction</td>
<td>11-31</td>
</tr>
<tr>
<td>11-14</td>
<td>Parallel Resistor Reduction</td>
<td>11-32</td>
</tr>
<tr>
<td>11-15</td>
<td>Parallel Diode Reduction</td>
<td>11-33</td>
</tr>
<tr>
<td>11-16</td>
<td>Unused MOS Transistors</td>
<td>11-40</td>
</tr>
<tr>
<td>11-17</td>
<td>Unused Bipolar Transistor</td>
<td>11-40</td>
</tr>
<tr>
<td>11-18</td>
<td>LVS Logic Gate Selection</td>
<td>11-43</td>
</tr>
<tr>
<td>11-19</td>
<td>INV</td>
<td>11-44</td>
</tr>
<tr>
<td>11-20</td>
<td>NANDn</td>
<td>11-44</td>
</tr>
<tr>
<td>11-21</td>
<td>NORn</td>
<td>11-45</td>
</tr>
<tr>
<td>11-22</td>
<td>AOI_3_2</td>
<td>11-46</td>
</tr>
<tr>
<td>11-23</td>
<td>OAI_3_2</td>
<td>11-47</td>
</tr>
<tr>
<td>11-24</td>
<td>SUPn</td>
<td>11-48</td>
</tr>
<tr>
<td>11-25</td>
<td>SDWn</td>
<td>11-48</td>
</tr>
<tr>
<td>11-26</td>
<td>SPUP_3_2</td>
<td>11-49</td>
</tr>
<tr>
<td>11-27</td>
<td>SPDW_3_2</td>
<td>11-50</td>
</tr>
<tr>
<td>11-28</td>
<td>SMPn, SMNn, SMn, S(TTT)n Series</td>
<td>11-51</td>
</tr>
<tr>
<td>11-29</td>
<td>SPMP_3_2, SPMN_3_2, SPM_3_2, SP(TTT)_3_2</td>
<td>11-51</td>
</tr>
<tr>
<td>11-30</td>
<td>SPMP((2+1+1)*1) Structure</td>
<td>11-53</td>
</tr>
<tr>
<td>11-31</td>
<td>SPMN(((3<em>1)+2)</em>(2+2)) Structure</td>
<td>11-54</td>
</tr>
<tr>
<td>11-32</td>
<td>INV</td>
<td>11-54</td>
</tr>
<tr>
<td>11-33</td>
<td>NANDn</td>
<td>11-55</td>
</tr>
<tr>
<td>11-34</td>
<td>NORn</td>
<td>11-55</td>
</tr>
<tr>
<td>11-35</td>
<td>OAI_3_2</td>
<td>11-56</td>
</tr>
<tr>
<td>11-36</td>
<td>SDWn</td>
<td>11-56</td>
</tr>
<tr>
<td>11-37</td>
<td>SPDW_3_2</td>
<td>11-57</td>
</tr>
<tr>
<td>11-38</td>
<td>SMDn, SMEn, S(TTT)n</td>
<td>11-58</td>
</tr>
<tr>
<td>11-39</td>
<td>SPMD_3_2, SPME_3_2, SP(TTT)_3_2</td>
<td>11-58</td>
</tr>
<tr>
<td>11-40</td>
<td>LDD AOI_3_2</td>
<td>11-61</td>
</tr>
<tr>
<td>11-41</td>
<td>SLDDP3</td>
<td>11-62</td>
</tr>
<tr>
<td>11-42</td>
<td>SPMN-LDDN(D)_3_1</td>
<td>11-63</td>
</tr>
<tr>
<td>12-1</td>
<td>Trivial Pin Swappability</td>
<td>12-9</td>
</tr>
<tr>
<td>12-2</td>
<td>SRAM Bit-Cell</td>
<td>12-10</td>
</tr>
<tr>
<td>12-3</td>
<td>Carrying Pin Swappability Up the Hierarchy</td>
<td>12-11</td>
</tr>
<tr>
<td>12-4</td>
<td>._bitcorev Structure</td>
<td>12-20</td>
</tr>
<tr>
<td>12-5</td>
<td>._bitv Structure</td>
<td>12-22</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-6</td>
<td>_bitrow2v Structure</td>
<td>12-23</td>
</tr>
<tr>
<td>12-7</td>
<td>_inv Structures</td>
<td>12-25</td>
</tr>
<tr>
<td>12-8</td>
<td>_invx3v Structure</td>
<td>12-26</td>
</tr>
<tr>
<td>12-9</td>
<td>_smn2v, _smp2v, _sup2v, _sdw2v Gates</td>
<td>12-28</td>
</tr>
<tr>
<td>12-10</td>
<td>_pmn2v, _pmp2v, _pup2v, _pdw2v Gates</td>
<td>12-29</td>
</tr>
<tr>
<td>12-11</td>
<td>_nand2v and _nor2v Gates</td>
<td>12-31</td>
</tr>
<tr>
<td>12-12</td>
<td>_mx2v Gate Multiplexer</td>
<td>12-33</td>
</tr>
<tr>
<td>12-13</td>
<td>_xra2v Multiplexer</td>
<td>12-34</td>
</tr>
<tr>
<td>12-14</td>
<td>_xr2v Multiplexer</td>
<td>12-35</td>
</tr>
<tr>
<td>12-15</td>
<td>_xor2v and _xnor2v Gate</td>
<td>12-37</td>
</tr>
<tr>
<td>12-16</td>
<td>_xnor2v and xor2 Gates</td>
<td>12-37</td>
</tr>
<tr>
<td>12-17</td>
<td>_tgmb Gate Multiplexer</td>
<td>12-39</td>
</tr>
<tr>
<td>12-18</td>
<td>_bitrfv Structure</td>
<td>12-41</td>
</tr>
<tr>
<td>13-1</td>
<td>Split Gate Property Ratio Error</td>
<td>13-42</td>
</tr>
<tr>
<td>15-1</td>
<td>Typical V2LVS Inputs and Output</td>
<td>15-2</td>
</tr>
<tr>
<td>15-2</td>
<td>E2LVS Flow</td>
<td>15-39</td>
</tr>
</tbody>
</table>
List of Tables

Table 1-1. List of Products Discussed in this Manual ................................................. 1-1
Table 1-2. List of Related Products and Their Manuals ............................................. 1-2
Table 3-1. DRC/DRC-H — Required SVRF Rule File Statements .............................. 3-2
Table 3-2. LVS/LVS-H — Required SVRF Rule File Statements ................................. 3-3
Table 3-3. Layout Database Formats ............................................................................ 3-3
Table 3-4. Source Database Formats ............................................................................ 3-8
Table 5-1. Flat Layer Operation Run Time Rankings (typical) ........................................ 5-9
Table 9-1. ERC Specification Statements ..................................................................... 9-1
Table 9-2. ERC Operations .......................................................................................... 9-1
Table 10-1. Default Device Properties ......................................................................... 10-11
Table 10-2. Value Array Listing .................................................................................. 10-21
Table 10-3. Property Specification Error Messages ................................................... 10-36
Table 11-1. Built-in Device Types ................................................................................ 11-7
Table 11-2. MOS Transistor Required Pin Names ....................................................... 11-9
Table 11-3. Capacitor Required Pin Names ............................................................... 11-10
Table 11-4. Resistor Required Pin Names ................................................................... 11-10
Table 11-5. Diode Required Pin Names ...................................................................... 11-11
Table 11-6. Bipolar Transistor Required Pin Names ................................................... 11-12
Table 11-7. Jfet Transistor Required Pin Names ......................................................... 11-13
Table 11-8. Inductor Required Pin Names ................................................................... 11-13
Table 11-9. Voltage Source Required Pin Names ....................................................... 11-14
Table 12-1. High-shorted Pin Resolution Examples ..................................................... 12-12
Table 13-1. Primary Messages .................................................................................... 13-21
Table 13-2. Secondary Messages—Errors .................................................................... 13-22
Table 13-3. Secondary Messages—Warnings ............................................................... 13-24
Table 13-4. Power/Ground Net Errors ......................................................................... 13-27
Table 13-5. Input Errors ............................................................................................. 13-30
Table 13-6. Information and Warnings ........................................................................ 13-43
Table 14-1. LVS SPICE Netlist Notational Conventions .............................................. 14-2
Table 14-2. Resistor Element ...................................................................................... 14-19
Table 14-3. Capacitor Element .................................................................................... 14-21
Table 14-4. Inductor Element ..................................................................................... 14-23
Table 14-5. Junction Diode Element .......................................................................... 14-25
Table 14-6. BJT Element ............................................................................................ 14-26
Table 14-7. JFET Element ......................................................................................... 14-28
Table 14-8. MOSFET Element ................................................................................... 14-30
Table 14-9. Voltage Source Element ......................................................................... 14-32
Table 14-10. Specification Statements for SPICE M Elements ................................. 14-33
Table 14-11. Subckt Statement .................................................................................... 14-34
List of Tables

Table 14-12. Subcircuit Calls ................................................................. 14-36
Table 15-1. Power Signal Pin Translation ............................................ 15-12
Table 15-2. Gate-Level Primitives ......................................................... 15-16
Table 15-3. Behavioral Statements ....................................................... 15-22
Table 15-4. V2LVS Errors ................................................................. 15-33
Table 15-5. V2LVS Warnings ............................................................. 15-36
Chapter 1
Licensing

The following table shows the products discussed in this manual.

Table 1-1. List of Products Discussed in this Manual

<table>
<thead>
<tr>
<th>Product</th>
<th>Required Software License</th>
<th>Prerequisite Software License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibre® Advanced Device Properties</td>
<td>calibreadp</td>
<td>calibrelvs or calibrehlvs</td>
</tr>
<tr>
<td>Calibre® CB™</td>
<td>calibredrclvseve</td>
<td></td>
</tr>
<tr>
<td>Calibre® Connectivity Interface</td>
<td>calibreci</td>
<td>calibreqdb</td>
</tr>
<tr>
<td>Calibre® DESIGNrev™</td>
<td>caldesignrev</td>
<td></td>
</tr>
<tr>
<td>Calibre® DRC</td>
<td>calibredrc or calibredrclvseve</td>
<td></td>
</tr>
<tr>
<td>Calibre® DRC-H™</td>
<td>calibrehsrc</td>
<td></td>
</tr>
<tr>
<td>Calibre® Interactive™</td>
<td>calinteractive</td>
<td></td>
</tr>
<tr>
<td>Calibre® LVS</td>
<td>calibrelvs or calibredrclvseve</td>
<td></td>
</tr>
<tr>
<td>Calibre® LVS-H™</td>
<td>calibrehlvs</td>
<td></td>
</tr>
<tr>
<td>Calibre® MTflex™</td>
<td>varies by tool</td>
<td></td>
</tr>
<tr>
<td>Calibre® RVE™</td>
<td>calibrerve or calibreqdb or calibredrclvseve</td>
<td></td>
</tr>
<tr>
<td>Calibre® xRC™</td>
<td>calibrexrc</td>
<td></td>
</tr>
<tr>
<td>Calibre® YieldAnalyzer</td>
<td>calyieldanalyze</td>
<td>calibredrc or calibrehdlrc</td>
</tr>
<tr>
<td>Calibre® YieldEnhancer</td>
<td>calyieldenhance</td>
<td>calibredrc or calibrehdlrc</td>
</tr>
<tr>
<td>Query Server</td>
<td>calibreqdb</td>
<td></td>
</tr>
</tbody>
</table>

Refer to the *Configuring and Licensing Calibre Tools* manual for complete information.
The following product manuals contain information that is relevant to Calibre verification product users.

**Table 1-2. List of Related Products and Their Manuals**

<table>
<thead>
<tr>
<th>Related Products Documented</th>
<th>Link to Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Notes</td>
<td>Calibre Verification Release Notes</td>
</tr>
<tr>
<td>Calibre Interactive</td>
<td>Calibre Interactive User’s Manual</td>
</tr>
<tr>
<td>DFM tools</td>
<td>Calibre Design for Manufacturability (DFM) Guide</td>
</tr>
<tr>
<td>Query Server and</td>
<td></td>
</tr>
<tr>
<td>Calibre Connectivity</td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td></td>
</tr>
</tbody>
</table>
This manual contains functionality updates through version 2005.4 of the Calibre Verification applications. For release-specific information, see the 2006.1 Calibre Verification Release Notes.

Product Description

The Calibre Verification applications assist you in verifying the physical and electrical integrity of IC designs. The Calibre Verification tools discussed herein consist of Calibre DRC/DRC-H, Calibre LVS/LVS-H, Calibre RVE, and utilities.

When there is no possibility of confusion between tools, DRC is used instead of Calibre DRC/DRC-H. Similarly, LVS is used instead of Calibre LVS/LVS-H.

The Calibre Verification applications operate on rule files written in Standard Verification Rule Format (SVRF). Rule files consist of design rule (DRC) checks; electrical rule (ERC) checks; layout versus schematic (LVS) and connectivity checks; parasitic extraction (PEX) applications; lithography statements for the Resolution Enhancement Technologies (RET) applications; and FRACTURE statements for Mask Data Preparation (MDP) applications. For information about SVRF, refer to the Standard Verification Rule Format (SVRF) Manual.

Calibre DRC / DRC-H / MT DRC-H / MTflex DRC-H

The Calibre DRC/DRC-H tools perform physical verification of integrated circuit layout 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 geometry.
- **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** — The MT configuration of Calibre DRC-H allows you to take advantage of processing with multiple CPUs on the same machine.
- **Multi-threaded flex** — The MTflex configuration of DRC-H allows you to take advantage of distributed processing using multiple machines in either a homogeneous (all processors of the same type) or heterogeneous (processors of mixed types) environment.
Overview
Product Description

**Calibre LVS / LVS-H / MT LVS-H / MTflex LVS-H**

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

- **Flat** — Calibre LVS performs flat layout versus schematic netlist checking.
- **Hierarchical** — Calibre LVS-H performs hierarchical layout versus schematic netlist checking. 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** — The MT configuration of Calibre LVS-H allows you to take advantage of multiple CPUs on the same machine.
- **Multi-threaded flex** — The MTflex configuration of LVS-H allows you to take advantage of distributed processing using multiple machines in either a homogeneous (all processors of the same type) or heterogeneous (processors of mixed types) environment.

**Calibre CB**

Calibre Cell/Block is a license package consisting of flat Calibre DRC and LVS verification tools, Calibre Interactive, Calibre RVE, and the 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 Calibre:

- **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.
- **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.
- The `compare_gds` utility (refer to “Compare Two GDSII Databases“) allows you to compare two GDSII databases (flat). This utility produces an ASCII DRC results database based on a layer-by-layer analysis.
- The `create_layer_rules` utility (refer to “Create a Rule File for Generating Unused Layers“) allows you to generate a rule file for outputting the layers that are in an input layout database, but are not required in the runset of an input rule file.
- The `create_compare_rules` utility (refer to “Create a Rule File for Comparing Two Layout Databases“) allows you to generate a rule file for layer-by-layer XOR comparison, based upon an input layout database.
Audience

This manual addresses two audiences:

- **Users:** Typically, IC Layout Engineers who use existing rule files with the verification tools to check an IC layout design. Users may also write the rule files. CAD engineers also are in this category.

- **Programmers:** These are persons who *write* rule files. Often programmers are members of a process engineering team 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 find the information in this manual helpful.

Syntax Conventions

The notational elements for syntax and commands are as follows:

**Standard** Standard font indicates commands or statements to be entered in a GUI or rule file.

**Monospace** Monospace font indicates pathnames, command line arguments, and file text. It is also used for rule file excerpts.

**Bold** A bold font indicates a required argument. It also indicates a GUI menu item or menu path.

**[ ]** 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, unless they are actual syntactical elements.

**|** 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.
This chapter describes the file requirements and invocation procedures for selected Calibre Verification tools. For similar information on the utility programs, refer to Chapter 15, “Utilities.”

Before Invocation

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

Setting the MGC_HOME Environment Variable

1. Set the MGC_HOME environment variable:
   - 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. Verify the location of MGC_HOME:
   ```
   ls $MGC_HOME
   ```
   This lists the contents of the MGC_HOME directory.

Required Input Files

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

- Rule file.
- Layout database — typically a geometric database for DRC applications and LVS connectivity extraction, or a netlist for LVS comparison.
- Source database — used for LVS.
Rule File

All Calibre rule files are written in the Standard Verification Rule Format (SVRF) language and are compatible with all Calibre and ICverify™ tools. There is generally no need to have separate rule files for DRC, LVS, xRC, and xL. All verification rules can coexist in a single rule file, if desired.

Prior to its use by any verification application, an SVRF rule file is compiled. During compilation, the file is checked for syntax errors. SVRF is case-insensitive and whitespace insensitive by default.

Rule files contain two main categories of statements: specification statements and operations. Except for invocation arguments on the command line, rule file specification statements control the functional environment for Calibre, such as describing the layout and source databases, and specifying where to store the results of a Calibre run. Specification statements also guide internal heuristics.

Rule file operations control the manipulation of layers through such things as Boolean operations, measurement operations, topological property operations, and so forth. Layer operations generate layer data, which can be output to DRC-style results databases.

Rule file elements are discussed under “Rule File” in the SVRF Manual. A sample rule file is shown in the “Rule File Examples” chapter of the SVRF Manual.

Required Statements

The following tables show the rule file specification statements required for Calibre DRC/DRC-H and Calibre LVS/LVS-H. Each table shows the names of the required statements and a description of the statement.

Table 3-1 shows the required rule file statements for Calibre DRC applications.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout System</td>
<td>Specifies the format of the layout data.</td>
</tr>
<tr>
<td>Layout Path</td>
<td>Specifies the location of the layout data.</td>
</tr>
<tr>
<td>Layout Primary</td>
<td>Specifies the top-level cell within the layout data.</td>
</tr>
<tr>
<td>DRC Results Database</td>
<td>Specifies where to save the results.</td>
</tr>
</tbody>
</table>
Table 3-2 shows the required rule file statements for Calibre LVS applications.

**Table 3-2. LVS/LVS-H — Required SVRF Rule File Statements**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout System</td>
<td>Specifies the format of the layout data.</td>
</tr>
<tr>
<td>Layout Path</td>
<td>Specifies the location of the layout data.</td>
</tr>
<tr>
<td>Layout Primary&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Specifies the top-level cell within the layout data.</td>
</tr>
<tr>
<td>Source System</td>
<td>Specifies the format of the source data.</td>
</tr>
<tr>
<td>Source Path</td>
<td>Specifies the location of the source data.</td>
</tr>
<tr>
<td>Source Primary&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Specifies the top-level cell within the source data.</td>
</tr>
<tr>
<td>LVS Report</td>
<td>Specifies where to save the report.</td>
</tr>
</tbody>
</table>

<sup>a</sup>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 description of a circuit. Table 3-3 shows the allowed database formats. You specify these formats using the Layout System statement in your rule file. For a given toolset, the layout database must be one of the following system formats.

**Table 3-3. Layout Database Formats**

<table>
<thead>
<tr>
<th>System Format</th>
<th>DRC</th>
<th>DRC-H</th>
<th>LVS</th>
<th>LVS-H</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GDSII</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OASIS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPICE</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASCII</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNET database</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**CIF Layout Format**

When the Layout System is CIF (Caltech Intermediate Form), you must specify the Layout Path and Layout Primary specification statements in the rule file. Layout Path specifies the pathname to the CIF symbol file, and Layout Primary specifies the top-level layout cell name within the file—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 files. Multiple input files are treated as if all symbol definitions are embedded in the first file specified. Each input file must be syntactically complete.

The original Mead/Conway Bachus Naur Format (BNF) is followed except for the following extensions and limitations:

- The user extension command “9” immediately following a “DS” command defines 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 Input Exception Severity 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 are not added to unresolvable CIF layers. As an example, this is a rule file definition using an alias:

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

**GDSII Layout Database Format**

When the layout database format is GDSII (specified as Layout System GDSII), you must specify the pathname to the stream file in a Layout Path specification statement and you must identify the top-level cell name 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 must be syntactically complete.

Calibre processes GDSII version 6.0.

The following GDSII records are processed by Calibre:

HEADER        BGNLIB        LIBNAME        UNITS
Calibre can process BOX and BOXTYPE records as BOUNDARY and DATATYPE records, respectively. To do so, place the following statement in the rule file:

```
LAYOUT PROCESS BOX RECORD YES
```

The default is not to process BOX and BOXTYPE records.

Calibre can read GDS polygons of up to 8192 vertices. If a polygon has more than 8192 vertices, Calibre automatically partitions such polygons into smaller polygons having fewer vertices.

---

**Note**

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

---

Calibre outputs GDS polygons of up to 4096 vertices by default. However, there is no practical limit (other than the memory or disk capacity of the host platform) to the number of vertices a polygon generated by Calibre can have. The DRC Maximum Vertex rule file statement controls vertex counts for output.

The default maximum cell name length for a GDS file is 32 characters. Calibre can exceed this through the DRC Maximum Cell Name Length statement. The legal characters include A-Z, a-z, 1-9, underscore (_), question mark (?), and dollar sign ($).

The Layout Input Exception Severity rule file statement controls how Calibre handles layout input exceptions. The severities of most exceptions can be adjusted to your needs.

There are no practical limits (except for the memory or disk capacity of the host platform) for things such as database size, number of input databases, numbers of polygons, depth of hierarchy, and number of cells. Note that GDSII coordinate data is inherently limited to 32-bit precision, even on 64-bit platforms.

You can specify the layout depth for shapes with the optional rule file Layout Depth specification statement. The ALL keyword (the default) specifies that shapes are read from the top-level cell to the bottom of the hierarchy. PRIMARY specifies that shapes are read from the top-level cell only.
OASIS Format

When the layout database format is OASIS (specified as Layout System OASIS), you must specify the pathname to the file in a Layout Path specification statement and you must identify the top-level cell name 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 must be syntactically complete.

Calibre can read OASIS polygons of up to 8192 vertices. If a polygon has more than 8192 vertices, Calibre automatically partitions such polygons into smaller polygons having fewer vertices.

Calibre outputs OASIS polygons of up to 4096 vertices by default. However, there is no practical limit (other than the memory or disk capacity of the host platform) to the number of vertices a polygon generated by Calibre can have. The DRC Maximum Vertex rule file statement controls vertex counts for output.

The Layout Input Exception Severity rule file statement controls how Calibre handles layout input exceptions. The severities of most exceptions can be adjusted to your needs.

There are no practical limits (except for the memory or disk capacity of the host platform) for things such as database size, number of input databases, numbers of polygons, depth of hierarchy, and number of cells.

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

See the Calibre for the Open Artwork System Interchange Standard guide for more information about OASIS use in Calibre.

SPICE Format

When the layout format is SPICE or HSPICE (specified as Layout System SPICE), this is used for LVS-H netlist-to-netlist comparison. The path to the SPICE file must be in a Layout Path statement. See “SPICE Format” on page 14-1 for a description of the Calibre SPICE dialect.

ASCII Layout Format

When the layout database format is ASCII (specified as Layout System ASCII), it appears as a set of polygon files in the form icv_data_n, where n represents the corresponding drawn layer
The file is assumed to be in the current directory. This format is used only by flat Calibre applications and ICverify.

The ASCII format for a polygon file is 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 labels. Text is only read from Text specification statements in the rule file.

**Binary Layout Format**

When the layout format is binary (specified as 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. This format is used only by flat Calibre applications, and the current directory is assumed to have the binary file. It can also be used in ICverify.

The Binary layout database format allows you to compare two databases to ensure database integrity.

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>
```
CNET Database Format

When the layout database is an LVS CNET (Compiled NETlist: a proprietary Mentor Graphics format) database (specified by Layout System CNET), this is used in flat netlist-to-netlist comparison. The path to the CNET database directory must appear in a Layout Path statement. CNET can only be used in flat LVS.

Source Database

A source database contains the reference information of a circuit for LVS applications. This is also called a source schematic or source netlist. Table 3-4 shows the allowed database formats.

You must use a source database (schematic or netlist) when doing layout versus schematic (LVS) checks. The Source System statement identifies the reference to be compared.

<table>
<thead>
<tr>
<th>System Format</th>
<th>LVS</th>
<th>LVS-H</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPICE</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CNET database</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

You must also specify the Source Path, which is the path to the netlist or viewpoint.

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

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

Invocation Procedures

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

- Calibre DRC/DRC-H
- Calibre LVS/LVS-H
- Calibre CB
Calibre DRC/DRC-H

Usage

```bash
calibre { -drc [ -cb ]
    || -drc -hier
    {[ -turbo [ number_of_processors ]
        [ -turbo_litho [ number_of_processors ]
        [ -remote hostname [, hostname ... ]
        [ -remotefile filename ] ] ]
    [ -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/DRC-H is also used for performing RET, MDP, and DFM tasks.

Calibre DRC performs traditional design rule checking by reading the database flat and operating on the resultant flat shapes.

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 shapes overlapping cell placements, or overlaps of cell placements.

For accepted database formats, refer to the section “Layout Database” on page 3-3.

When you use Calibre DRC-H for mask preparation, you should output a GDSII or OASIS DRC results database (see 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 because Calibre DRC-H requires a large amount of internal overhead to generate a mask data DRC results database.

You can use DRC-RVE to analyze the DRC results database. If you use Mentor Graphics IC Station™, DESIGNrev, or any of the Calibre family of layout viewers, you can load an ASCII or mask data DRC results database for interactive debugging.

Both Calibre DRC and Calibre DRC-H use the same rule file. This means you typically do not need to add, remove, or modify any statements when switching between flat and hierarchical verification.

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 3-1 shows the required specification statements for Calibre DRC/DRC-H.

**Arguments**

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

```sh
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 DRC checking.

- **-cb**

  This option is discussed under “Calibre CB” on page 3-25.

- **-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 optional `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 number, Calibre DRC-H runs on the maximum number of CPUs available for which you have licenses.

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

  ```sh
calibre -drc -hier ... -turbo 3 ...
  ```

  operates on two processors for a 2-CPU machine.

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

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

- **-remote hostname [, hostname …]**

  This switch is part of the MTflex multi-threaded, parallel processing architecture. It must be specified in conjunction with the -turbo or -turbo_litho switches. It enables multi-threaded operation on remote hosts of a distributed network. You must specify at least one `hostname` parameter. A list of hostnames is comma-delimited and specifies that multiple hosts participate in multi-threaded operations. You must have the required number of licenses for your job.
This switch applies only to hierarchical applications on a homogeneous set of hosts. That is, all machines are the same supported platform type (HP-UX, Linux, or Solaris, respectively) and must have the same address mode (32- or 64-bit).

For more details, see the Using MTflex with the Calibre Toolset guide.

- remotefile filename

This switch is part of the MTflex multi-threaded, parallel processing architecture, which enables multi-threaded operation on remote hosts of a distributed network. It must be specified in conjunction with the -turbo or -turbo_litho switch, which specifies the number of processors you are using, including those on the remote hosts. The filename specifies the pathname of a configuration file containing information for the local and remote hosts. You must have the required number of licenses for your job.

This switch applies only to hierarchical applications. For more details, see the Using MTflex with the Calibre Toolset guide.

- 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 optional 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 number, Calibre DRC-H runs on the maximum number of CPUs available.

This switch is not for flat applications. For more information, refer to Configuring and Licensing Calibre Tools.

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

- turbo_all (Calibre DRC-H)

The -turbo_all switch is an optional switch you use in conjunction with the -turbo and -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.
Invocation

Invocation Procedures

See “-turbo_all Switch” in Configuring and Licensing Calibre Tools for licensing information.

- -nowait

This switch causes Calibre to queue only briefly (approximately 10 seconds) before attempting to acquire substitute licenses. This switch is equivalent to specifying “-wait 0”.

- -wait n

This switch sets the maximum amount of time (n) for Calibre to queue for a specific license. If the license is unavailable after queueing for n minutes, Calibre attempts to acquire any substitute licenses or exits if no suitable substitutions are defined. Refer to the section “Using License Substitution” in Configuring and Licensing Calibre Tools for more information on substituting licenses.

In the following example, the -wait switch queues on a calibredrc license for five minutes:

    calibre -drc -wait 5 rules

If a license does not become available within five minutes, the application exits with the following message:

    // Queue time specified by -wait switch has elapsed.

- -64

This switch invokes the 64-bit version of Calibre. The default is 32-bit mode. For information on supported 64-bit platforms, see Configuring and Licensing Calibre Tools.

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 running on the same data.

There is no limit (up to 64-bits) in the size of internal integer coordinate data (in database units) which DRC applications can process on 64-bit platforms. In reality, however, 32-bit precision (the default) and 64-bit precision are nominal values. The “safe” precisions are 30-bit and 62-bit, respectively, in order to prevent any overflow (with subsequent data corruption or application crashes) within the Calibre internal algorithms. Since integers are represented, internally, in sign-magnitude or two's-complement format, the safe precision ranges are, for a nominal 32-bit maximum, $-2^{29}$ to $+2^{29}$, and for a nominal 64-bit maximum, $-2^{61}$ to $+2^{61}$.

- 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

Calibre LVS/LVS-H

Usage

   calibre [ -lvs { [ { -tl | -ts } cnet_file_name ]
            [ -nonames ] [ -cell ]
            [ -dblayers "name1,..."]
            [ -bpf [no-extents]] [ -nl ] [ -cb ]
            || [ -hier [ -automatch ]] ]
            [ -ixf ] [ -nxf ]
            ]
   [ -spice spice_file_name
   [ -turbo [ number_of_processors ]
   [ -turbo_all ]
   [ -remote hostname [, hostname ... ] ||
   [ -remotefile filename ] ]
   ]
   [ -hcell cell_correspondence_file_name ]
   [ -siggen ]
   [ -nowait ] [ -wait n ] [ -64 ]
   rule_file_name

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

Description

Calibre LVS/LVS-H 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 shapes.

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

Calibre LVS/LVS-H directly read GDSII, OASIS, and CIF databases. Flat LVS compares the geometric layout to the source netlist directly. Hierarchical LVS compares an extracted SPICE layout netlist to a source netlist. The rule file **Source System** and **Layout System** specification statements determine which type of databases are being compared.
Invocation

Invocation Procedures

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. For more information, refer to Chapter 6, “Using the LVS-RVE Interface” in the *Calibre Interactive User’s Manual*.

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.

When using MTflex to run Calibre LVS-H on distributed processors, most layer derivation, Sconnect, and Connect operations run on distributed processors, while device extraction and short isolation run only on the master processor.

You can use Calibre LVS in one of two methods:

- LVS comparison
- CNET database translation

Table 3-2 shows the required rule file specification statements for Calibre LVS/LVS-H.

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 one of the following commands (without arguments):

```plaintext
    calibre -lvs
    calibre -lvs -hier
```

- **-lvs**
  
  When used alone, this switch specifies to run flat Calibre LVS.

  When you use `-lvs` with `-hier`, Calibre LVS-H compares a hierarchical source netlist with an extracted hierarchical layout netlist. The `-spice` option is used by LVS-H to extract a hierarchical SPICE netlist from the layout.

- **[-tl || -ts]**
  
  This switch determines whether 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:

Invocation

Invocation Procedures

-\textit{ts} — Selects source translation. You specify the source in the Source Path specification statement.

- \textit{cnet\_file\_name}
  The pathname of the file receiving the layout-data-to-CNET translation.

- \textit{-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.

- \textit{-cell (or -c)}
  This switch specifies that the CNET writer scan only the top-level cell (no hierarchical evaluation).

- \textit{-dblayers “name1, …” (or -db “name1, …”)}
  This switch controls the layer shapes 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 \texttt{Connect} and \texttt{Sconnect} operations, all \texttt{Device} seed and pin layers from the rule file, and all \texttt{Stamp} target layers. Possible exceptions are 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 select only layers that appear in \texttt{Connect} and \texttt{Sconnect} operations, serve as \texttt{Device} seed or pin layers, or serve as \texttt{Stamp} target layers.

- \textit{-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, -spice, or -hier switches; Calibre returns an error if this occurs.

  The optional \texttt{no-extents} option instructs the BPF writer to output actual coordinates for all shapes. Without this option, some polygons that have edges not orthogonal to the database axes are represented by their rectangular extents.

  You use BPF databases with third-party tools. The format is described in the section “Binary Layout Format” on page 3-7. You can use this switch in both normal flat operation and in translate operation (-tl argument). The files have names of the form \texttt{lvs\_report\_name.layer\_name.bpf}, where \texttt{layer\_name} is the rule file layer name and \texttt{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.

The header file, DrcBPFFReader.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, refer to the DrcBPFF_example.c provided with DrcBPFFReader.h.

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

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 layout 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. Specified 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

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 with third-party tools; it is not intended for general netlisting. To extract a hierarchical SPICE netlist.
from a layout, use the -spice switch. You cannot specify -nl with the -spice switch; Calibre returns an error if this occurs. The -nl option is often used with -bpf.

- **-cb**

This option is discussed under “Calibre CB” on page 3-25.

- **-hier**

This switch runs the LVS comparison hierarchically. It is used with the -lvs switch. Both layout and source systems must be SPICE when you use -hier, unless you also specify -spice, in which case the layout must be GDSII, OASIS, or CIF. You must have an LVS-H license to use this switch.

- **-automatch (-aut[o])**

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

This switch does not apply to the circuit extraction (calibre -spice) stage.

The -automatch should only be used if the layout cells actually contain the same devices as the source subckt of the same name.

Excellent LVS-H performance is generally obtained with a relatively brief, but carefully chosen list of hcells, and the use of the -hcell switch is recommended instead of the -automatch switch for most situations.

The -automatch switch allows the hierarchy to be handled as Calibre determines is best for performance. All the remaining cells that have been unaltered in the hierarchy, and have a like-named counterpart in the source netlist, are then compared.

Hcell names are treated as case-insensitive by default using -automatch, but become case-sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES. If hcell names are treated as case-sensitive, two cells having the same name in layout and source are not matched if their cases are different. Top-level cells always correspond, regardless of their names.

Cells having names of the form ICV_n do not participate in -automatch. Such cells are generated at artificial levels of hierarchy within Calibre and are unsuitable as hcells.

See “Hcells” on page 12-5 for details on hcell comparison.

- **-ixf**

**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.
This switch generates an instance cross-reference file. The filename is
\textit{lvs\_report\_name.ixf}, where \textit{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

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 does not use the
LVS Report name, but is in the form \textit{layout\_primary.ixf}, where \textit{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
\textit{layout\_primary}.

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

- \textbf{-nxf}

\textbf{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.

This switch generates a net cross-reference file. The filename is \textit{lvs\_report\_name.nxf},
where \textit{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 “SVDB Cross-Reference Files” on

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 does not use the LVS
Report name, but is in the form \textit{layout\_primary.nxf}, where \textit{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
\textit{layout\_primary}.

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

- \textbf{-spice} \textit{spice\_file\_name} (or -spi \textit{spice\_file\_name})

This switch extracts a hierarchical SPICE netlist from the layout system, which must be
GDSII, OASIS, or CIF, and directs output to \textit{spice\_file\_name}. When you specify this
option with -lvs, Calibre LVS -H 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 Calibre xRC, `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 and the Layout Primary specification statements in the rule file.

You can use the `-spice` switch when you run Calibre xRC (after running Calibre LVS-H) and specify the Mask SVDB Directory specification statement in the rule file using the keyword XRC. 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]`

  This switch instructs Calibre LVS-H to use multi-threaded parallel processing. The optional `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 number, 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 to be specified also.

  Calibre LVS-H is limited to running on the maximum number of CPUs available for which you have licenses. If you specify a number greater than the maximum available CPUs, Calibre LVS-H runs only on the maximum number. For example:

  ```
  calibre -spice -turbo 3 ...
  ```

  operates on two CPUs for a 2-CPU machine.

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

- **-turbo_all**

  The `-turbo_all` switch is an optional argument you use in conjunction with the `-turbo` switch. This switch halts Calibre tool invocation if the tool cannot obtain the exact number of CPUs you specified using `-turbo`.

  Specifying the `-turbo` switch without 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 Tools* for licensing information.

- **-remote hostname [, hostname …]**

This switch is part of the MTflex multi-threaded, parallel processing architecture, which enables multi-threaded operation on remote hosts of a distributed network. It must be specified in conjunction with the -turbo switch. You must specify at least one hostname parameter. A list of hostnames is comma-delimited and specifies that multiple hosts participate in multi-threaded operations. You must have the required number of licenses for your job.

This switch applies only to hierarchical applications. For more details, see the *Using MTflex with the Calibre Toolset* guide.

- **-remotefile filename**

This switch is part of the MTflex multi-threaded, parallel processing architecture, which enables multi-threaded operation on remote hosts of a distributed network. It must be specified in conjunction with the -turbo switch, which specifies the number of processors you are using, including those on the remote hosts. The filename specifies the pathname of a configuration file containing information for the local and remote hosts. You must have the required number of licenses for your job.

This switch applies only to hierarchical applications. For more details, see the *Using MTflex with the Calibre Toolset* guide.

- **-hcell cell_correspondence_file_name**

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.

Excellent LVS-H performance is generally obtained with a relatively brief, but carefully chosen list of hcells. See “Hcells” on page 12-5 and “Managing the Hcell List” in the *Calibre Query Server Manual* for more information.

Note that an exhaustive hcell list, containing every cell in the layout (or the same general idea), can actually lower performance in cases where Calibre internal heuristics would have flattened certain cells in order to streamline the hierarchy for its internal purposes. Once a cell is listed in the hcell list, its location in the hierarchy is maintained, even if it represents potential performance degradation.

You can also control cell correspondence 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 are concatenated.
Note

You must run Calibre LVS-H with the -hcell switch (or with Hcell specification statements in your rule file) before running Calibre xRC 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 generates source-to-layout cross-reference files (XREFs) suitable for hierarchical net extraction and places them in the SVDB directory.

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 previous 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, for example, this results in an error:

```
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.
Hcell names are treated as case-insensitive by default, but become case-sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES. If hcell names are treated as case-sensitive, two cells having the same name in layout and source are not matched if their cases are different.

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

- **-nowait**
  This switch causes Calibre to queue only briefly (approximately 10 seconds) before attempting to acquire substitute licenses. This switch is equivalent to specifying “-wait 0”.

- **-wait n**
  This switch sets the maximum amount of time (\(n\)) for Calibre to queue for a specific license. If the license is unavailable after queuing for \(n\) minutes, Calibre attempts to acquire any substitute licenses or exits if no suitable substitutions are defined. Refer to the section “Using License Substitution” in Configuring and Licensing Calibre Tools for more information on substituting licenses.

In the following example, the -wait switch queues on a calibredrc license for five minutes:

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

If a license does not become available within five minutes, the application exits with the following message:

```
// Queue time specified by -wait switch has elapsed.
```

- **-64**
  This switch invokes the 64-bit version of Calibre. The default is 32-bit mode. For information about supported 64-bit platforms, see Configuring and Licensing Calibre Tools.

  Coordinate values in the range +/-9,007,199,254,740,991 are supported for circuit extraction in both flat and hierarchical 64-bit Calibre LVS.

  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 running on the same data.

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

Invocation Procedures

This switch cannot be used with input systems other than SPICE netlists and cannot be used with other switches except -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.

LVS reserves a flat license when you use this switch, or both -cs and -cl.

- **-cl**

This switch instructs LVS to read and verify (through a syntax checker) the SPICE netlist specified in the Layout Path specification statement. LVS issues any applicable warnings or errors and writes them to the LVS report. 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 except -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.

LVS reserves a flat license when using this switch, or both -cl and -cs.

- **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
ocalibre -lvs -tl mycirc.cnet -nonames my_rules
```

Calibre LVS-H

- The following example extracts a hierarchical SPICE netlist from the geometric 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. The file called cells specifies cell correspondence.
Invocation Procedures

Invocation Procedures

`calibre -lvs -hier -hcell cells rules`

- The following example extracts a hierarchical SPICE netlist from a geometric layout system 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 a geometric layout specified in the rule file. The extraction preserves hcells as subcircuits in the extracted netlist, and the hcells are then available in for comparison.

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

- The following example extracts a hierarchical SPICE netlist from a geometric layout system, and then compares it to a SPICE source. Netlist extraction is hierarchical and comparison is flat: (This method is not recommended because it results 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`

**Note**

Using the `-automatch` switch is generally discouraged. Use it only if the layout cells actually contain the same devices as the source subckts of the same name.

- 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. 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 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`
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 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 CB

The Calibre Cell/Block license package provides 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.

You invoke the Calibre CB verification license through the -cb command line switch. The -cb switch causes Calibre to use a single caldrclvseve license. You use this license to run DRC, LVS, RVE, or the Query Server. You can only run DRC and LVS 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 you to do only one verification task at a time. Therefore you can do DRC, LVS, RVE, or access the Query Server at any one time with one license. For
example, if there is only one caldrclvseve license, and you are 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 (as in multi-threaded applications).

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 enables you to do interactive verification.

Other Calibre Verification Tools

Calibre Interactive is a front-end user interface for running Calibre tools in an interactive verification environment. Calibre RVE is a user interface that allows you to view Calibre results interactively with your layout editor. These are discussed in the Calibre Interactive User’s Manual.

Calibre DESIGNrev is a fast layout database viewer used for analyzing large layout databases in the final finishing stages before tapeout. Calibre DESIGNrev is discussed in the Calibre DESIGNrev User’s Manual.

Calibre YieldAnalyzer and Calibre YieldEnhancer applications are used for improving chip yield. These applications are discussed in the Calibre DFM Guide.

The Query Server is a command-line-driven database server that is used to access connectivity information in an SVDB database. The Query Server is discussed in the Calibre Query Server Manual.

Calibre Version Information

You can use the -version switch to view version information for Calibre. The MGC_HOME environment variable determines the location of your Mentor Graphics software tree (refer to page 3-1 for information on setting this variable) and the version of Calibre tools you are running.

% calibre -version
//
// Copyright Mentor Graphics Corporation 2005
// 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.
Invocation

Invocation Procedures

// Mentor Graphics software executing under Sun SPARC Solaris
//
// This software is in preproduction form and is considered to be
// beta code that is subject to the terms of section 3 of the
// End User License Agreement or your signed agreement with
// Mentor Graphics Corporation, whichever applies.
//
// Running on SunOS my_sun 5.8 Generic_108528-13 sun4u
//
// Starting time: Fri Oct 21 09:33:02 2005
//
Invocation

Invocation Procedures
This chapter covers fundamental concepts of how DRC applications operate. It is divided into these main topics:

- Layers
- Layer Operations
- Rule Check Statements
- Dimensional Check Operations
- Disk-Based Layers
- Specialized DRC Applications

Calibre DRC/DRC-H is an *edge-based* design rule checking (DRC) system. It works primarily with the edges of polygons rather than with the regions themselves; although it manipulates both. 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 when interior and exterior are important to distinguish.

Before reading this chapter, familiarize yourself with the “Rule File” section of the SVRF Manual. That section discusses many fundamental topics that are germane to this chapter.
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. This section discusses these major topics:

- Layer Types
- Layer Definitions

Layer Types

A rule file creates or uses data from four types of layers. Figure 4-2 shows examples:

- Original layers (drawn layers)
- Derived polygon layers
- Derived edge layers
- Derived error layers

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 in a Layer statement. The verification system also allows you to use layer sets, which consist of a layer name assigned to more than one layer number. In this case, the layer set behaves as if it were a simple layer with all shapes on the constituent layers combined. Like simple layers, layer sets are referred to in the rule file by their name or alias.

Here are examples of how original layers are declared in an SVRF rule file:

```
LAYER M1 2 // simple layer
LAYER MET 3 4 5 // layer set
LAYER ALL_MET M1 MET // another layer set
```
In most layer operations, DRC automatically merges the polygon data on an original layer before using that layer. In a merged-data representation, DRC 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.

Calibre is insensitive to LAYERTYPE and TEXTTYPE attributes of original layers. The Layer Map specification statement is useful for mapping these attributes to layer numbers for Calibre to process.

**Derived Polygon Layers**

Derived polygon layers represent merged polygons generated as the output of layer operations such as Boolean functions, topological polygon functions, and certain dimensional check operations that employ polygon output. An example of a derived polygon layer in a layer definition is:

```
gate = poly and oxide
```

This derivation could appear either outside of a rule check (a global derived layer) or within a rule check (a local derived layer). Whether used globally or locally (or both), DRC calculates the layer only once and uses it wherever needed.

Derived polygon output can be *error output* that is sent to the DRC results database. The following generates error output when placed in a rule check as a *free-standing* layer operation:

```
contact not interact metal
```

Note there is no assignment to a third layer using an equals (=) sign in this construct.

**Derived Edge Layers**

Derived edge layers represent edges or edge segments 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 produce error output if they occur in a free-standing operation (that is, no assignment to a derived layer name using an = sign). An example of a derived edge layer is:

```
long_metal_edge = length metal > 5
```

The operation:

```
length metal > 5
```

generates error output that is sent to the DRC results database, when this operation is placed in a rule check as a free-standing operation.
Derived Error Layers

Derived error layers represent clusters of one, two, three, or four edges from either one or two layers. Primarily, they hold output of error-directed dimensional check operations for instantiation into the DRC results database. The operations that can generate this type of data include the Enclosure, External Internal, DFM Measure, and TDDRC operations; and the Drawn Acute, Drawn Angled, Drawn Offgrid, and Drawn Skew operations.

For example, this syntax generates a derived error layer:

\[ x = \text{INTERNAL } m_1 < .08 \]

This derivation generates an edge cluster that cannot be passed to any layer operation, except DFM Analyze. DFM Analyze is the only operation that accepts derived error layers as input.

This syntax also generates a derived error layer:

\[ \text{INTERNAL } m_1 < .08 \]

This free-standing layer operation produces a derived error layer that is sent to the DRC results database, when this operation is placed inside a rule check.

This syntax generates a derived edge layer:

\[ y = \text{INTERNAL } [m_1] < .08 \]

Note the use of the brackets ( [ ] ) in this syntax. The derived edge layer y can be passed to any operation that accepts derived edge layer data as input.

Layer Type Summary

Derived polygon and derived edge layers can be used as output to the DRC results database, or in layer definitions that are used for further processing in other layer operations.

The ENClosure, EXTernal, INTernal, DFM Measure, TDDRC, Drawn Acute, Drawn Angled, Drawn Offgrid, and Drawn Skew operations produce derived error layers. Derived error layers are usually sent to the DRC results 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 data through the DRC system, in terms of the layer types supported by the system.
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:

\[ \text{layer\_name} = \text{layer\_operation} \]

The layer operation creates the derived layer and gives it the name \textit{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 (or in rule files that appear in an Include statement). 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 excerpt illustrating layer definitions:

```plaintext
// 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. In the latter case, they may be free-standing in that they do not send output to a derived layer used in another layer operation, but only to the DRC results database.

**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:

```plaintext
taps = ( pdiff AND ( bulk NOT nwell ) ) OR ( ndiff AND nwell )
```
In this example, an explicit layer definition defines the 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 converts 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. These are discussed later in this chapter.

Layer Operations

This section covers two main topics:

- **Layer Operation Classifications**
- **Layer of Origin**

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

- **Dimensional Check Operations** — Dimensional check operations provide the core design-rule checking capability of the DRC system. They are ENClosure, EXTernal, and INTernal. Rectangle Enclosure, [Not] With Neighbor, TDDRC, and DFM Measure also fit into this category. They measure distances between various types of edges and are discussed in detail in this chapter.

- **Auxiliary Operations** — Any layer operation that is not a dimensional check is an auxiliary operation. They may have edge output or polygon output. They generally use some type of Boolean or topological property to analyze edge or polygon relationships. The section “Auxiliary Layer Operations” in the SVRF Manual has a summary of all auxiliary layer operations.

Both types can be used to derive polygon or edge layers, depending on the function of the operation. Both types can be used to send data to the DRC results database. The “Summary of Rule File Elements” section in the SVRF Manual contains an overview of rule file basics and short descriptions of layer operations.

Note that the ENClosure, EXTernal, and INTernal operations have special edge- and polygon-directed syntaxes that produce derived layer data used by other operations. You can use derived layers created by polygon-directed and edge-directed layer operations as input to other operations, or as error output to the DRC results database. For more information about the results database, refer to “DRC Results Database” on page 7-6.

Layer Operation Classifications

Layer operations that generate derived edge or polygon layers are either layer constructors or layer selectors. The following sections describe these classifications. The section “Auxiliary
Layer Operations” in the SVRF Manual shows which operations are selectors and which are constructors by the use of superscript letters designating which operations fall into which category.

Layer Constructors

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

\[ \text{layer1 AND layer2} \]

is a layer constructor because it creates new polygons from the polygon data on both input layers.

Layer Selectors

Operations classified as layer selectors select existing polygon or edge data from the appropriate input layer. For example:

\[ \text{layer1 COINCIDENT EDGE layer2} \]

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.

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 x. For the operation:

\[ x = \text{COINCIDENT EDGE layer2 layer1} \]

the connectivity passes from the layer2 polygon (or edges) to the derived layer x. Although both definitions would generate the same geometric edge data, the connectivity attached to the output
is not likely the same in 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 understanding layer derivation.

Note

The layer of origin concept is not applicable to error-directed layer operations because you cannot 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. See “Layer Constructors” on page 4-8 for a description.

Calibre merges all original layers prior to using them in any layer operation. Because the merge operation is equivalent to the one-layer 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:

\[
X = \text{AND metal contact} \\
Y = \text{AREA X < 4} \\
Z = \text{RECTANGLE Y}
\]
The layer of origin for layer Z (and layer Y) is layer X, because it was the last layer generated by a layer constructor operation in the layer definition chain.

An important point regarding layer selector operations (see “Layer Selectors” on page 4-8): these operations select data from the appropriate input layer. It may appear, for example, that the layer definitions

\[
\begin{align*}
X &= A \text{ coincident edge } B \\
X &= B \text{ coincident edge } A
\end{align*}
\]

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 are not, in general, the same data because polygon and node references likely differ.

This has important implications for dimensional check operations. For example, the following Internal dimensional check sequence:

\[
\begin{align*}
X &= \text{metal coincident edge poly} \\
\text{ruleA} \{\text{internal X metal < 3}\}
\end{align*}
\]

has an entirely different definition from this sequence:

\[
\begin{align*}
X &= \text{poly coincident edge metal} \\
\text{ruleB} \{\text{internal X metal < 3}\}
\end{align*}
\]

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 (metal), while in the second case, the input layers have different layers of origin.

What this means in practice is, ruleA in the previous sequence measures distances between metal edges, while ruleB measures distances between poly and metal edges because of how the layer of origin for X is defined.

**Rule Check Statements**

Rule check statements are used in Calibre DRC applications (as well as any other Mentor Graphics tool that uses SVRF rule files). They are also used in specific electrical rule check (ERC) operations performed in LVS. These statements specify layer operations within the rule file that instantiate the resulting derived layers into the DRC (or ERC) results database. Refer to “DRC Results” on page 7-1 for information on the DRC results database.

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 stand-alone layer operation that outputs error data) 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 braces ({}). Oftentimes the braces are omitted in this manual when discussing layer operations that may occur in a rule check. You must include the braces for rule checks 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 {
    long_metal = metal LENGTH > 5 // Layer definition;
    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 (it has an = sign). Similarly, the third operation is a layer definition. The second and fourth operations are not layer definitions; these operations generate `error` data, which is output to the DRC results database under the name METAL_WIDTH. You must have at least one operation that generates output data in a rule check, or the rule check will not compile.

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

```
copy long_metal
```

into the METAL_WIDTH rule check would copy the long_metal layer to the DRC results database as output. You could then see what long_metal looks 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 three types of comments within 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 symbol (@). The third is a comment that spans multiple lines and is enclosed by a /* … */ sequence of characters.

The // characters begin a comment that terminates at the end of the line on which they occur. These comments serve to annotate the rule file and can be used inside or outside of a rule check.

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.
m2etal 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.
m2etal EXTERNAL < 3.5 notch /* Check spacing in the same polygon. */
}
```

In each of the previous rule checks, the three comments beginning with the @ symbols appear in the DRC results database. These comments also would appear in Calibre RVE when you load the DRC results database.

The C-style /* … */ comment can span multiple lines. You can use these anywhere in a rule file and they should be used with care. Commenting out entire sections of a rule file using these types of characters can lead to unexpected results.

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 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, title, and rule check comments.

**Dimensional Check Operations**

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

The major topics covered in this section are:

- Secondary Keywords
- Edge Measurement
- Measurement Region Construction
- Metrics
- Edge Cluster Generation
- Interval Constraints for Output Suppression
- Appropriateness Criteria
- Intersection Criteria
- Edge Breaking
- Polygon Containment Criteria
- Edge-Directed Output
- Polygon-Directed Output
- False Measurement Reduction
- Measurement Tolerances

The dimensional check operations function in three primary output modes:

- **Error-directed Dimensional Check Operations** — Error-directed dimensional check operations generate derived error layers consisting primarily of edge clusters whose members mutually meet the constraint of the operation.

  Error-directed edge clusters such as are generated by syntax like this:

  ```
  INT metal < .1 ABUT < 90 SINGULAR
  ```

  Output from this type of operation (as in a layer definition) cannot be passed to other layer operations, with one exception—DFM Analyze.
**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 “( )”. For example:

\[ x = \text{ENC} [\text{cont}] \text{ metal1} < .1 \]

sends the edge segments of cont which satisfy the constraint to layer x. Conversely, the statement:

\[ x = \text{ENC} (\text{cont}) \text{ metal1} < .1 \]

sends edges from layer cont, which do not satisfy the constraint to layer x.

**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, REGION EXTENTS, or REGION CENTERLINE keywords. For example:

\[ x = \text{ENC} \text{ cont metal1} < .1 \text{ REGION} \]

creates polygonal regions from the edge segments that satisfy the constraint and sends the regions to layer x.

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. These operations are discussed in detail in the *SVRF Manual*.

Figure 4-5 illustrates the edges that the dimensional check operations measure between. ENClosure has only a two-input-layer syntax, while External and Internal have single-layer and two-layer syntaxes. Note that full rule check syntax is not used in these examples.
Secondary Keywords

The dimensional check operations, as well as many other statements in the SVRF 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 *Standard Verification Rule Format (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 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 4-18), 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 of edge A; a similar region consists of all such points around edge B. (DRC assumes the user-unit of length is the micron.)

The output is an edge pair consisting of the segment of A that intersects the region around edge B and the segment 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 less than 3 microns but any other type of legal constraint or value works similarly.

**Figure 4-6. Generation of Output Edges**

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 shape of this boundary is determined, in part, by 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 forms that consists of all points in the half-plane in which the boundary is 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 \( x \leq a \), 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, \( a \leq x < b \), \( a < x \leq b \), and \( a \leq x \leq b \) 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 \( a < x \leq b \). 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 four basic forms of metrics you can use with dimensional check operations:

- **Euclidean** — This metric forms a region with quarter-circle boundaries that extend past the corners of the selected edges. This is the default metric and requires no keyword in a layer operation.

- **Square** — This metric forms a region with right-angle boundaries that extend past the corners of the selected edges.

- **Opposite** — This metric forms a region with right-angle boundaries that do not extend past the corners of the selected edges.

- **Opposite Extended** — This metric forms a region with right-angle boundaries that can extend past the corners of the selected edges, dependent upon an extension value you specify. This metric allows *non-commutative measurements*, where the measurement from edge A to edge B does not produce the same output as from edge B to edge A, to produce output. The value specified for this metric must be a positive number.

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.

The Opposite metrics (all the metrics with the word Opposite in them) treat the constraint $a \leq x < b$ as $a < x < b$, and the constraint $a \leq x \leq b$ as $a < x \leq b$, unless there is infinite intersection with the top portion of the inner curve of the design rule boundary.

Figure 4-8 shows measurement region construction for the four most common metric types.
Special Considerations for the Opposite Metric

Measurement output generation is slightly different from that discussed previously if the Opposite metric has been specified. The purpose of the Opposite metric is to reduce the number of non-orthogonal (with respect to the database axes) edges created when using polygon-directed (REGION) output.

First, the Opposite metric generates an output only when the measurement region from edge A to edge B intersects B at more than one point, as shown in Figure 4-9.
Edges that are not parallel to each other and not orthogonal to the database axes can result in non-commutative measurements, as shown in Figure 4-10. There is no output from the Opposite metric in this case, or in any other case where such measurements occur.

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. Given two edges A and B, with B being the non-orthogonal edge, edge B is intersected with the measurement region from edge A to produce output from B. Output from A is the projection of output from B onto edge A, as shown in Figure 4-11.
Advanced Metrics for Specialized Applications

The following forms are variations of the four basic metrics. These are specialized metrics mostly used for RET applications and mask data preparation:

- **Opposite Symmetric** — This metric applies special processing to the Opposite metric for edges that are not orthogonal to the database axes and are not parallel.

- **Opposite Extended Symmetric** — This metric applies the Opposite Extended measurement region to the Opposite Symmetric metric.

- **Opposite FSymmetric** — The Opposite FSymmetric metric behaves similarly to Opposite Symmetric; however, it has an additional feature of filling in disjoint edges that are output from an operation using Opposite Symmetric.

- **Opposite Extended FSymmetric** — This metric applies the Opposite Extended measurement region to the Opposite FSymmetric metric.

- **Square Orthogonal** — The Square Orthogonal metric simulates the effect of using a square of dimensions $V \times V$ to trace around the perimeter of each polygon on the input layer. At least one corner of the square is always in contact with the polygon as the...
square traces around the polygon. The square remains orthogonal to the database axes throughout the trace, regardless of how the input layer edges are oriented. The path the square traces out defines the measurement region.

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. Figure 4-12 illustrates the metric. Opposite Symmetric is defined by the following steps:

1. Given edges A and B, apply the Opposite metric if A and B are parallel, perpendicular, or intersecting.

2. Otherwise, measure A and B by using the Opposite metric; however, do not perform the special treatment for exactly one non-orthogonal edge, as is done with the Opposite metric. 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. Because of properties within the Opposite metric, there can be at most one output edge from each input edge (even for interval constraints).

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 segment PB. Project edge OB, if it exists, onto A, which forms a segment 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.

The Opposite FSymmetric metric modifies Step 6 so that any portion of edge A between OA and PA will be added to OA + PA, resulting in a single output edge from A. Likewise for B, OB, and PB. See the comment in Figure 4-12.

The Opposite Extended Symmetric metric replaces Opposite measurement in Steps 1, 2, and 3 of the definition by Opposite Extended value measurement.

The Opposite Extended FSymmetric metric replaces Opposite measurement in Steps 1, 2, and 3 of the definition by Opposite Extended value measurement and, in addition, performs the Step 6 modification of the Opposite FSymmetric metric.
**Square Orthogonal metric** — The Square Orthogonal metric for DRC dimensional check operations EXTernal, INTernal, and ENClosure simulates simultaneous mask misalignment in the x- and y-directions. To illustrate, consider the EXTernal operation with constraint value $V$:

$$\text{EXT layer} < V \ldots$$

The Square Orthogonal metric simulates the effect of using a square of dimensions $V \times V$ to trace around the perimeter of each polygon on the input layer. At least one corner of the square is always in contact with the polygon as the square traces around the polygon. The square remains orthogonal to the database axes throughout the trace, regardless of how the input layer edges are oriented. The path the square traces out defines the measurement region. See Figure 4-13.
This metric, like all others, must map to the measurement region construction methodology around an individual edge. For edges orthogonal to the database axes, Square Orthogonal is equivalent to the Square metric. For 45-degree edges, the measurement region is constructed as an isosceles trapezoid with an altitude of \( A = \sqrt{2} \times V \), where \( \sqrt{2} \) is the customary square root function. The longer of the base edges of the trapezoid is coincident with the 45-degree edge. The length of this longer base edge is the length of the 45-degree edge plus \( 2A \). See Figure 4-14.

**Figure 4-14. SQUARE ORTHOGONAL Measurement Region (45-degree)**

For polygons having edges that are non-orthogonal and non-45-degree with respect to the database axes (skew edges), Square Orthogonal measurement region construction is more difficult, involving polygonal chains of potentially numerous edge segments.

Calibre does not support measurement region constructions composed of such polygonal chains. It is also desirable to avoid a plethora of special cases for measurement region construction around skew edges. For this reason, the measurement region is constructed using the following principles:

- avoid polygonal chains of numerous edges
- have a single algorithm for all cases
- reasonably approximate the exact measurement region
- neatly approach the limiting cases of orthogonal and skew edges

These objectives lead to the following measurement region construction method for skew edges.

For any skew edge \( E \) (see Figure 4-15) having endpoints \((x_1, y_1)\) and \((x_2, y_2)\), define

\[
\begin{align*}
DX &= |x_2 - x_1| \\
DY &= |y_2 - y_1| \\
R &= \sqrt{DX \times DX + DY \times DY} \\
A &= V \times R / \text{max}(DX, DY) \\
B &= V \times |DX - DY| / R
\end{align*}
\]
Construct an isosceles trapezoid having the following dimensions:

base 1 = R + 2A
base 2 = R + 2B
altitude = A

The skew edge E is coincident with base 1.

**Figure 4-15. SQUARE ORTHOGONAL Measurement Region (Skew Edge)**

The Square Orthogonal metric produces no output when measuring two edges A and B if the measurement is non-commutative; that is, if

- Edge A intersects the measurement region around edge B, but not vice-versa, or
- Edge B intersects the measurement region around edge A but not vice-versa.

**Edge Cluster Generation**

The edge measurement methods described previously generate *edge clusters* in error-directed output (edge-directed or polygon-directed output does not have this feature). These are groups of edges or edge segments that are output by a rule check containing error-directed operations. Several different types of clusters can be formed. Figure 4-16 shows the generation of a three-edge output cluster.
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 keywords, or from a Drawn Angled or Drawn Skew operation. When Calibre sends an edge layer to the DRC results database, the layer becomes a one-edge cluster across that interface.

The concept of edge clustering applies only to derived error layers. DRC 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.

### Four-Edge Output Cluster

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

For example, Figure 4-17 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. The regions are based on using an interval constraint.
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, \( \leq a \)), and one of the edges intersects the region around the other edge at only a single point on the boundary.

DRC 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 the intersection of the measurement regions is not necessarily commutative. This is especially true for the non-default 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, DRC 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.

Figure 4-18 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 4-18. Trivial Edge Generation**

```plaintext
rule {
  external layerA layerB < 3 SQUARE
}
```

---

**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 4-19 would result in two output clusters consisting of trivial edges.
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”.

**Interval Constraints for Output Suppression**

In conjunctive design rule checking (rule checks where more than one operation appears), 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 4-20):

```plaintext
// 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.
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 4-20 and 4-21):

```plaintext
// 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 4-21. 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 4-21, with the left-most edge pair not being obvious.

The left-most edge pair occurs because the Euclidean and Square metric measurement regions contain area to the sides when you use interval constraints in the dimensional check operations. This area can cause unwanted intersections (and output) which were not intended 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 discussed previously, or else do not use interval constraints to suppress possible redundant errors in certain complicated conjunctive DRCs.

Note

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, DRC-H may use a large amount of CPU overhead to process skew edges.

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 4-22 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.

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.
Figure 4-23 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 4-23. Appropriate Angles Between the Outsides of Edges**

![Angles Diagram]

The appropriate angle is between the outsides of the dashed 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 such as ABUT and the INTERSECTING families of keywords.

In general, you always should use ABUT < 90 SINGULAR in your dimensional rule checks, unless you have good reasons not to.

**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 4-24 shows an example of edge breaking.
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

Given that edge breaking occurs in the two-layer dimensional check operations, we can define these operations in a more rigorous manner through the use of polygon containment criteria:

- An **Enclosure** check between layers X and Y, in that order, measures the separation between the exterior side of an edge A from X, and the interior side of an edge B from Y, *only if* A is inside of the polygon having edge B. Edge B is not inside of a polygon from layer X and not coincident inside with an edge from layer X.

- An **External** check between layers X and Y measures the separation between the exterior sides of edges A from X, and B from Y, *only if* A is not inside of a polygon from layer Y, and not coincident outside with an edge from layer Y. Edge B is not inside of a polygon from layer X, and not coincident outside with an edge from layer X.

- An **Internal** check between layers X and Y measures the separation between the interior sides of edges A from X, and B from Y, *only if* A is inside of the polygon having edge B. Edge B is inside of the polygon having edge A.

These criteria address the *looking through the boundary* problem of two-layer dimensional check operations. That is, these conditions prohibit the measurement between edges that are hidden from each other, even though the correct sides of the edges face each other. See Figure 4-25. The edges shown are *not* measured by default.

**Figure 4-25. Polygon Containment**

Note that the polygon containment criteria for the two-layer EXTernal check allow measured edges to be coincident inside with edges from the other input layer (but not coincident outside). In Figure 4-25, for the EXTernal example, if the upper edge of layer1 were coincident with the upper edge of layer2, then the measurement *would be* taken.

**Relaxing polygon containment** — There are cases when relaxing the polygon containment criteria is desirable. For INTernal and ENClosure, the secondary keyword MEASURE
COINcident allow coincident edges to be considered in the measurements. For all dimensional check operations, MEASURE ALL completely relaxes the containment criteria and permits all edges within the measurement region to be considered in the measurements. These keywords are fully discussed in the *SVRF Manual*.

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 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 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 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 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 conform only if the layer1 edge is not coincident with any layer2 edge.

### 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 square brackets 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 constraint.
Only one edge-directed output specification may appear in a single dimensional check operation. That is, no more than one set of brackets or parentheses may appear in a given operation. Edge-directed output specifications apply to ENClosure, EXTernal, INTernal, TDDRC, and DFM Measure operations.

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 }
```

has 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 that demonstrate this.

### 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 diff < 4}
```

would result in a derived-error layer containing edge clusters from poly and diff 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 diff REGION OPPOSITE
```

creates a derived polygon layer of regions between poly and diff violations. The “Rule File Examples” chapter of the *SVRF Manual* has numerous examples to study.

**Note**

If you are deriving polygon layers by using the REGION keyword, and the derived output is intended to be used in further rule checks, you should use REGION EXTENTS keyword or the OPPOSITE measurement metric. If you use OPPOSITE, be aware that this is a smaller measurement region than the default, and can result in missed errors if used improperly.

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 two database units, yielding a rectangle of width approximately four 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 four database units, yielding a rectangle of width approximately four 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 four database units, yielding a rectangle of width approximately four 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 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:
  
  o Two *true* trivial edges are the result of the measurement process.
  o The OPPOSITE or OPPOSITE SYMMETRIC metric was not specified.
  o The operation’s constraint is of the form “< $a$”.

Although primarily intended to construct intermediate layers in conjunctive processes, you may want 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 where the polygon-directed form has reduced the total error count:

**Figure 4-26. Error Reduction Using Polygon-Directed Output**

- error-directed output (6 errors)
- polygon-directed output (2 errors)

Although primarily used for final error output, the SINGULAR keyword has some interesting applications in conjunctive checks when used along with INTERSECTING ONLY and REGION. See the Poly-diffusion spacing at gate bend example in the SVRF Manual.

**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. Figures 4-27 and 4-28 show examples:

**Figure 4-27. False Notch Measurement (single polygon)**
Note that both of the previous measurements obey all previous semantics. However, because they are almost universally considered false, you should avoid them. In both cases, the measurement is considered false because of another edge completely blocking the line of sight between the two edges being measured.

Because eliminating false measurement in an edge-based system is time-consuming, DRC attempts to delete only the worst occurrences. You can instruct DRC to expend maximum effort to avoid false notch measurements by using the EXCLUDE FALSE NOTCH secondary keyword to specific single-layer EXTernal operations. The DRC Exclude False Notch YES specification statement applies globally to all single-layer EXTernal checks in a rule file.

Use of these statements is strongly discouraged in a production rule file as they are very time-consuming to execute in a DRC run. Therefore, you should use them strictly as debugging tools.

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 to the database axes and projecting onto each other.
- 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 previous examples 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.

Measurement Tolerances

Tolerances for dimensional check operations are controlled using the DRC Tolerance Factor specification statement.

Error Tolerance Setting

Calibre DRC can generate warnings that indicate a possible data integrity issue with the input database, such as when it excludes the objects that cause these warnings from processing. The Layout Input Exception Severity specification statement allows you to control how these situations are handled.

Disk-Based Layers

By default, geometric data on layers (original and derived) created during a DRC run is memory-based. For DRC applications, the majority of memory used during execution is consumed by geometric data. You may specify that geometric data on layers created during a DRC run is to be primarily disk-based rather than memory-based. In most cases, this can save substantial amounts of memory by transferring this resource to disk file(s).

Use of disk-based geometry storage in Calibre is controlled by the presence of the Layer Directory specification statement. If this statement is not specified, then geometry storage is memory-based. Otherwise, layers created during the run are placed in individual disk files in the specified directory. This specified directory is created if it does not exist.

A sub-directory is created in the specified directory to hold the layer files. This sub-directory 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 directory creation fails or the test open fails, then either a fatal error occurs and the program aborts, or a warning is issued and memory-based layers are used. You can control this severity, using the Layout Input Exception Severity LAYER_DIRECTORY setting. File I/O exceptions occurring from that point cause the program to stop.

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 memory-based layers. When execution is complete, all directories that were created are removed.
The amount of resource that is transferred from memory to disk files when disk-based layers are used is a function primarily of the number of large derived layers created during the run that exist at one time. Since the database is read in one scan at the start, all original layers are simultaneously present in memory prior to being written to disk files. In addition, each individual derived layer created during the run is first created in memory before being written to a disk file. In some cases, using disk-based layers saves no memory and increases the resource requirement by the amount of file space used.

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 the run completes, all directories that were created are removed.

Hierarchical Calibre applications must be used in the 64-bit mode to use this functionality.

Specialized DRC Applications

This section contains a compilation of specialized DRC topics that include:

- Datatypes and Texttypes in Calibre
- Dual Database Capability
- Mask DRC Results
- Incremental Connectivity and Antenna Checks
- Soft Connection Checks
- Layout Input Control in Calibre
- Cell Exclusion
- Area-Based Filtering
- Flagging and Snapping Original Shapes
- Input Layout Database Magnification

Datatypes and Texttypes in Calibre

If the layout database format for Calibre is GDSII or OASIS, then datatypes and texttypes are ignored by default. However, processing of drawn geometry and text based on datatypes and texttypes is possible through Layer Map specification statements. 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, then a Layer Map specification statement is called a datatype map. When the TEXTTYPE keyword is present, it is called a texttype map. Texttype maps work for texttypes and layers specified in Text Layer specification statements in the same
way that datatype maps work for 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
layout stream reader proceeds as follows:

1. 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 is
treated as if it were on target_layer.
2. 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.
3. If G is in no datatype map, then it is treated as being on layer L regardless of datatype.

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 chosen arbitrarily
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, per the mapping algorithm described previously. A simple solution is to map
objects on layer 100 to an unused layer. Because layer maps are not always carefully designed,
a warning is issued for any input layer that is:

1. required by the execution flow.
2. the target layer of a datatype map.
3. containing objects which themselves are not mapped by any datatype map.

**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.
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, OASIS, 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 (shapes 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):

```plaintext
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 would appear on the original layer metal1. This is because **Layer** specification statements apply equally to each of the two databases.

For dual databases, Calibre DRC 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, are 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 the input hierarchy instead of flattening it. DRC-H does this as follows:

1. Renames all cells in database one internally (such as A -> A$1$).
2. Renames all cells in database two internally (such as A -> A$2$), as in Step 1.
3. Creates a new top level cell and instantiates the top level cells T1$1$ and T2$2$ from databases one and two with identity transforms. The name of the new top-level cell is that of the top-level cell from database one.
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 A$1$ and A$2$ are in an exact overlap situation (equal extents, transforms, and array characteristics). Places all objects from A$1$ and A$2$ into cell A, and replaces the two placements of A$1$ and A$2$ 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 one 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.

**Mask DRC Results**

The **DRC Check Map** specification statement provides a way to designate the destination (layer,datatype) coordinate for DRC rule check output into mask data DRC result databases. The statement is much more complex than that. It allows an m->n mapping between DRC rule checks and DRC results databases, as well as AREF compaction capability for rectangles. It also allows 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. The output from these DRC rule checks 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 rule checks 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, GDSII, or OASIS.

**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, or OASIS). 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 DRC results databases. (OASIS supports array structures, but they are not recommended.)

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

- The check text mapping, as specified in the DRC Check Text specification statement.
- Whether to retain empty rule checks (those with no DRC results), as specified in the DRC Keep Empty specification statement.
- Whether to output cell names and transforms in ASCII DRC results databases, and leave coordinates untransformed, as specified in the DRC Cell Name specification statement.
• Whether pseudohierarchy is to be retained, as specified by the PSEUDO parameter in the DRC Results Database specification statement.

• The append string for cell names in GDSII or OASIS DRC results databases, as specified by the string parameter in the DRC Results Database specification statement.

• Transfer of input layout database text to mask data DRC results databases, as specified in the DRC Map Text specification statement.

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

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

• The maximum output cell and placement name length for cell names in GDSII or OASIS results databases, as specified in the DRC Maximum Cell Name Length specification statement.

• The LIBNAME attribute of GDSII databases as specified in the DRC Results Database Libname 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, \ldots, R_n\} to be executed in a Calibre DRC run, a set of unique DRC result databases \{D_1, \ldots, D_m\} is determined which encompasses the cumulative DRC output from the application. Each results database \(D_j\) has an associated (and unique) type (ASCII, binary, GDSII, or OASIS). The mapping is \(n \rightarrow m\). Any given DRC RuleCheck \(R_i\) may map to multiple members of \(\{D_1, \ldots, D_m\}\) and any given DRC results database \(D_j\) may be mapped into by multiple members of \(\{R_1, \ldots, R_n\}\).

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

• 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 OASIS) or to be globally specified in the DRC Results Database specification statement with a different type.

• If file_name is not specified in a DRC Check Map specification statement, then the type (ASCII, binary, GDSII, or OASIS) of the statement must match the global type specified (or default) in the DRC Results Database specification statement. This is because the default file_name parameter is the global DRC Results Database file name.

• Multiple specification of the (rule_name, file_name) coordinate in the set of DRC Check Map specification statements is a compilation error.
• 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 previously, also insures unambiguous determination of the MAXIMUM RESULTS values and AREF parameters for any Ri -> Dj map. A conceptual pseudocode of this is given as follows:

```plaintext
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**

AREF output is discussed in detail in the *SVRF Manual*.

You can use the SUBSTITUTE keyword to specify non-rectangular AREF structures.

**Using fill cells for AREF structures** — Specifying fill cells for AREF structures can be accomplished in one of two ways. Assume you have the following in your rule file:

```plaintext
LAYOUT PATH main_design.gds fill_cell.gds
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY TOP
```
Furthermore, assume that main_design.gds has the following cells properly instantiated in its hierarchy:

```
  TOP
    cell_A
    cell_B
    cell_C
```

The fill_cell.gds has one cell in its database that is not placed in the main design:

```
  cell_name
```

After you run Calibre DRC-H, the output.gds file contains the cells:

```
  TOP
    cell_A
    cell_B
    cell_C
```

It also will contain an AREF of the name cell_name, but it will have no cell by that name. If you open output.gds in a layout viewer, you will see that the cell definition for cell_name is missing. This is expected.

To obtain a cell of the name cell_name with the AREF structure of the same name within it, merged with the main layout, you can do one of two things:

- Rerun Calibre with the following rule file statements:

  ```
  LAYOUT PATH output.gds fill_cell.gds
  LAYOUT SYSTEM GDSII
  LAYOUT PRIMARY TOP
  DRC RESULTS DATABASE new_output.gds GDSII
  DRC CHECK MAP ... AREF cell_name
  ```

  The file new_output.gds will contain the AREF structure in a cell of the name cell_name.

- Use Calibre DESIGNrev to merge output.gds and fill_cell.gds.

### Incremental Connectivity and Antenna Checks

This section discusses some details involving incremental connectivity in the context of antenna checks. You can employ these methods in any problem involving incremental connectivity, however.

Antenna checks are a broad category of design checks intended for finding interconnect paths of sufficient surface area such that they 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, consider three layers of metal deposition.

Antenna checks for the metal $i$ deposition stage must ignore connectivity created by metal $j$, for $j > i$. Because Connect operations are executed as a single unit for non-incremental connectivity extraction (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
  also present in the rule file.
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 ensures that the layers
  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 because all layers in each set of Connect operations are
disjoint from all layers in any other set. The copying of layers gets around the default
characteristic that all Connect operations are executed together as one unit at the beginning of
the run, and effectively partitions the circuit into independent collections of nets, thus ensuring
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 because 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, six or more) cannot be checked in a single run and
you are forced to put each antenna level in a different rule file (and a different run).

The solution to the problem of efficient antenna checking is to support incremental connectivity.
This is the ability to execute a sequence like 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.
   ...
5. 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 previous 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
// 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
// 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 }

(The diode is not connected 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
views 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. Label Order and Sconnect operations are not
allowed. All errors resulting from this schema are flagged at compilation time.

Only the DRC applications support incremental connectivity. Other applications ignore the
DRC Incremental Connect specification statement (at run time, not compilation time) and treat
the rule file as order-independent (and Connect statements as global), as usual. You have the
responsibility of mitigating connectivity conflicts, if any, in rule files covering multiple
applications. 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, and Ornet, constrained polygon topological
layer operations with BY NET specified, and nodal dimensional check operations having the
CONNECTED or NOT CONNECTED keywords specified.

Connectivity of the appropriate parameters originates from their presence in a Connect or
Sconnect operation or their derivation through 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 that 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 Connect parameter must be defined prior to the Connect operation.
3. 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 an error 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 conditions 1 through 5 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); 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 allowed:

```
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 previous 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 rule file compiler also disables 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 do not cross connectivity zones in order to prevent inadvertent
creation of layers with incorrect connectivity.

The DRC execution sequence also changes 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 rule check output operations.

With specification of DRC Incremental Connect YES, the execution sequence changes to this
pseudocode:

For each connectivity zone from first to last {
    a. Produce layer parameters for all new Connect statements which defined the zone.
    b. 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 annotation. Perform net naming based upon existing
       connectivity if there are [Not] Net operations in the zone.
    c. Produce data for all DRC rule check output operations in the zone.
    d. 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 to 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 begins 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 careful copying of Connect layers, makes the Disconnect operation rarely useful. It is, however, indispensable in certain very advanced DRC checks where any existing connectivity must be completely discarded.

**Soft Connection Checks**

DRC applications 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, 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
}

**Layout Input Control in Calibre**

**GDSII input** — The following GDSII records are processed by Calibre applications for a GDSII-type input layout database:

```plaintext
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.

**OASIS input** — Details of how OASIS databases are handled are contained in *Calibre for the Open Artwork System Interchange Standard*.

**CIF input** — The original Mead/Conway Bachus Naur Format is adhered to except for the following extensions and limitations:

- The user extension command “9” immediately following a “DS” command defines 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 are not 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.
```

If the input layout database format is GDSII/CIF, the layout depth for shapes may be specified via the Layout Depth specification statement.

**Handling Duplicate Cells**

You can specify the Layout Path specification statement with multiple filename parameters and any number of times. This allows multiple input databases to be used. Multiple input databases are treated as if all the structure records (for GDSII or OASIS) 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). You 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 an input layout database. The recognition semantics are as follows:

1. If there is a literal match between a cell name and the Layout Primary parameter, then that cell becomes the top-level cell.
2. 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 is selected as the top-level cell; a warning is issued in this event.

3. If there is still no match, then a fatal error is issued.

Database Pre-Merging

A layout database that originated from a module generation program, for example, may often contain large numbers of overlapping shapes 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 shapes, which, if merged, would be M shapes where M << N. If cell C has P flat placements, then the total number of shapes 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 is then much more efficient and memory is saved. This is accomplished with the Layout Merge On Input YES specification statement.

The input layout database format must be GDSII, OASIS, or CIF for the merging to occur. In cases such as the previous one, 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 does not occur. YES should generally not be specified if the input layout database lacks the aforementioned characteristics.

Cell Renaming

You 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.

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 forth. Excluding a cell means that no objects from any placement of the cell is 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

Area filtering of layout database objects is supported through the following specification statements: Layout Window, Layout Window Layer, Layout Window Cell, Layout Windel, Layout Windel Layer, Layout Windel Cell, and Layout Window Clip. See the SVRF Manual for details.

Flagging and Snapping Original Shapes

Original geometry flagging causes warnings to be generated when original shapes which fail certain constraints are read from the layout database (the original shapes are still processed, however). Original geometry snapping aligns vertices of original shapes on specified grids.

Flagging of original shapes is controlled by the rule file specification statements Flag Acute, Flag Angled, Flag Skew, Flag Offgrid, Flag Nonsimple Polygon, and Flag Nonsimple Path. The default for all statements is NO.

The error-directed auxiliary operations Drawn Acute, Drawn Angled, Drawn Skew, and Drawn Offgrid provide an alternative method to flag original shapes than that provided by the corresponding Flag statements. Each method checks the same set of original database shapes and flags the same problems. The Drawn family of operations produce DRC results that can be sent only to the DRC results database. 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 using error-directed auxiliary operations or by Flag YES statements (or both), only checks shapes on original layers that 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 rule checks), not necessarily the entire set of original layers referenced in the rule file. Layers may be selectively excluded from error-directed auxiliary operations and Flag YES statements by specifying the relevant Exclude Acute, Exclude Angled, Exclude Skew, or Exclude Offgrid statement.

Note that Layer Resolution statements only apply if the layers they reference are actually read in during the DRC run.

If Snap Offgrid YES is specified in the rule file, then original layer shapes 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. Shapes 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 ensures that shapes 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 shapes 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 shapes and snapped in the same manner as shapes, as described previously. No other text objects are snapped.

**Input Layout Database Magnification**

The input layout database can be magnified as it is read into Calibre applications if you use 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, OASIS, and CIF—and array pitches for GDSII) by the given value, regardless of 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.
PRECISION 10000
// Was 1000. Better to increase that also.
```
This chapter describes how Calibre DRC functions upon execution.

Data Flow in DRC

The inputs to Calibre DRC are an SVRF rule file and a layout database. The outputs are a run transcript and a DRC Results Database, with an optional DRC Summary Report. Chapter 3 discusses the required inputs in detail. Chapter 7 discusses DRC results databases. Figure 5-1 shows the DRC data flow.

![Figure 5-1. DRC Data Flow](image)

The layout database can be GDSII, OASIS, CIF, Binary, or ASCII. The latter two are proprietary formats to Mentor Graphics Corporation and only work in flat DRC. The results database can be ASCII (default), GDSII, OASIS, or Binary.

Invocation

The following command lines invoke Calibre DRC and Calibre DRC-H, which operate using 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 3, “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 stop the program. Compilation errors are discussed in the “Error Messages” chapter of the SVRF Manual.

DRC Specification Statements

Specification statements control the overall behavior of a Calibre run. The section “DRC Specification Statements” in the SVRF Manual shows a summary. The section “Layout Database Specification Statements” in the same manual summarizes the statements that control the handling of layout databases. Some frequently used specification statements are discussed in this section.

Rule Check Selection

You can select one or more rule check statements from a rule file, which Calibre DRC runs 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.

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 by using the MAXIMUM RESULTS parameter to the DRC Check Map specification statement. Limiting results helps avoid generating large DRC results databases that can occur when you have a large DRC rule constraint value. By default, this maximum result limit is 1000 per DRC rule check.

Calibre issues a warning whenever Calibre reaches 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 stop the generation of DRC results when the results to be written only to the DRC results database (not used in a setting where more than one operation is used in a rule check) will exceed the result limit.

This optimization can save CPU time and memory by not generating and storing results. These results would later be discarded when writing the DRC results database because the maximum result limit had been reached. 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 result limiting 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 result limited operation.

**Control of Empty Rule Checks**

It is typical that rule checks may 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 performance of the rule check that is important, whether or not there actually are results.

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 are not instantiated into the DRC results database. If YES is specified, or the statement is not specified at all, then empty rule checks are instantiated.

**General Execution Characteristics**

This section discusses some of the general characteristics of Calibre DRC execution. Much of the discussion applies to all verification applications that use the DRC engine.

**Concurrency**

Calibre DRC performs many of the layer operations concurrently. This means that they run simultaneously, when present in a rule file. 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. You can optimize your rule file by taking advantage of this feature.

Calibre performs the following layer operations concurrently:

- All dimensional check (Enclosure, External, or Internal) operations with one input layer that have the same input layer.
DRC Execution

General Execution Characteristics

- 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) Inside
  - (Not) Outside
  - (Not) Enclose
  - (Not) Interact
  - (Not) Touch

- All constrained polygon topological operations having the same two input layers in a given order, such as:
  - (Not) Cut
  - (Not) Interact
  - (Not) Touch

- 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) Angle and (Not) Length operations having the same input layer.

- All (Not) Area operations having the same input layer.

- All Convex Edge operations having the same input layer.

- All Density operations that have the same input layer, boundary, window, and step parameters.

- All DFM Analyze having the same input layers and all other parameters identical.

- All DFM Transition operations having the same input layer and file parameter.

- All Expand Edge operations having the same input layer.

- All Holes operations having the same input layer.

- All Litho operations having the same input layer and file parameter.

- All MDPstat operations having the same input layer and file parameter.

- 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 **Path Length** operations having the same input layer.

• All (Not) **Rectangle** operations having the same input layer.

• All (Not) **With Edge** operations having the same two input layers in a given order.

• All (Not) **With Neighbor** operations having the same input layer.

• All (Not) **With Width** operations having the same input layer.

### 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:

```plaintext
regular_contact = contact outside pad
pad_contact = contact not regular_contact
```

This involves two potentially time-consuming operations. An equivalent method introduces concurrency and reduces the time required for one of the previous operations:

```plaintext
regular_contact = contact outside pad
pad_contact = contact not outside pad
```

These two operations are calculated concurrently in DRC. 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:

```plaintext
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:

```plaintext
rule_A { 
    external poly diff < 3
}
rule_B { 
    enclosure diff poly < 2
}
Redundancy Elimination

Calibre DRC combines all identical layer operations during run time. 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 or diff
    Y = X inside metal
    ...
}
```
```
DEF {
    P = diff or 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 duplicating 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 of 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 previously. 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.

Calibre DRC generally executes rule checks in the order they occur in your rule file. When the executive routine is finished with a particular layer (that is, no more rule checks are to be performed on a layer), that layer is deleted 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 detail next.
**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. Note that layers which appear in the rule file, but are not needed in the DRC run (that is, no operation in the check set requires such layers as input), are not generated.

2. If the check set requires connectivity extraction, perform the **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 previous 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 in the run. 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 should help you arrange 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 run time. Consider the following design rule:

All metal contacts must be inside of metal and, in addition, inside of either polysilicon or diffusion.
DRC Execution
General Execution Characteristics

This rule states that metal contacts connect metal to polysilicon or metal to diffusion. One way to check this rule is as follows:

```plaintext
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 by using conjunctive checks. The method shown next achieves this, and is as accurate as the previous check:

```plaintext
metal_contact_check {
  bad1 = contact not inside metal
  x = contact not inside diff
  bad2 = x not inside poly
  bad1 or bad2
}
```

The OR operation merges any duplicate objects in the derived layers bad1 and bad2. Given the objects in bad1 and bad2 are errors (these layers should be very small, or empty), the final OR operation is 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 Enclosure OUTSIDE ALSO option for these types of contact checks is much more efficient.)

The general principle to keep in mind when writing layer derivations is, try to reduce the polygon or edge count in a derived layer as much as possible, then any layer operation that uses that derived layer will perform faster. Whenever possible, choose the fastest operation in terms of runtime that provides the data you need. See Table 5-1 for a list of operations ordered by runtime.

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:

```plaintext
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 causes 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:

```
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 an approximate order for comparing the run time of most of the layer operations for flat layout verification. The order is slightly different for hierarchical verification, as discussed in the section “Hierarchical Operation Efficiency” in Chapter 6.

This ranking is dependent on layout configurations, but it does give a general idea of the relative speed of these operations. This is not a comprehensive list, but includes operations frequently used:

<table>
<thead>
<tr>
<th>Layer operations are listed alphabetically in each performance group. Performance groups are shown beginning with the slowest group and ending with the fastest.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENClosure, EXTernal, INTernal</td>
</tr>
<tr>
<td>TDDRC</td>
</tr>
</tbody>
</table>
### Table 5-1. Flat Layer Operation Run Time Rankings (typical)

Layer operations are listed alphabetically in each performance group. Performance groups are shown beginning with the slowest group and ending with the fastest.

<table>
<thead>
<tr>
<th>AND (two layer)</th>
<th>(Not) Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Not) Cut (unconstrained)</td>
<td>Density</td>
</tr>
<tr>
<td>(Not) Inside</td>
<td>Expand Edge</td>
</tr>
<tr>
<td>NOT</td>
<td>Extents</td>
</tr>
<tr>
<td>OR (two layer)</td>
<td>Holes</td>
</tr>
<tr>
<td>(Not) Outside</td>
<td>(Not) Length</td>
</tr>
<tr>
<td>Stamp</td>
<td>OR (one layer)</td>
</tr>
<tr>
<td>(Not) With Text</td>
<td>Snap</td>
</tr>
<tr>
<td>XOR (two layer)</td>
<td>Size</td>
</tr>
<tr>
<td>(Not) Area</td>
<td>XOR (one layer)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XOR (two layer)</th>
<th>AND (one layer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Not) Area</td>
<td>(Not) Donut</td>
</tr>
<tr>
<td>Net Area</td>
<td>Net Area Ratio</td>
</tr>
<tr>
<td>Path Length</td>
<td>Perimeter</td>
</tr>
<tr>
<td>(Not) Rectangle</td>
<td>Vertex</td>
</tr>
<tr>
<td>(Not) With Edge (same layer of origin)</td>
<td>(Not) With Edge (different layer of origin)</td>
</tr>
</tbody>
</table>

(Not) Touch Edge, (Not) Touch Inside Edge, (Not) Touch Outside Edge

(Not) Coincident Edge, (Not) Coincident Inside Edge, (Not) Coincident Outside Edge
Convex Edge
(Not) Cut (constrained)
(Not) Enclose
(Not) Inside Edge
(Not) Interact
(Not) Outside Edge
(Not) Touch
(Not) With Edge (different layer of origin)
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 DRC Maximum Vertex 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.

As stated before, Calibre DRC segments any DRC polygon result whose vertex count exceeds 4096. The specification statement DRC Maximum Vertex ALL denotes a very large number (2147483647 to be exact) of vertices.
For example, if the DRC results database type is GDSII, then you may want to use DRC Maximum Vertex 199 because 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 are not segmented simply on the basis of the results database format being GDSII). You can segment DRC polygon results into quadrilaterals by using DRC Maximum Vertex 4.

Note that if the value specified in DRC Maximum Vertex exceeds 4096, Calibre DRC uses this value even though some DRC results databases (ASCII, for example) are specified to contain no polygons with more than 4096 vertices. This would present a problem if such a DRC results database were later loaded into IC Station (using REStore DRc Results).

Whenever a polygon result is segmented, a warning is issued since this can produce potentially confusing results for certain operations; however, in general, segmentation is the desirable thing to do.

In an area-based DRC check, result polygons are filtered out by the specified area before any segmentation is applied. However, DRC result counts that appear in the DRC results database and that govern the maximum number of results are computed after any polygon segmentation is applied.

**Layout Database End Segment Warning**

The Calibre layout data input module issues 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, OASIS, and CIF databases and is issued in both Calibre DRC and DRC-H. This behavior is controlled through the Layout Input Exception Severity PATH_ENDSEGMENT_SHORT rule file setting.
Calibre DRC-H is a *fully-hierarchical* DRC application. This exploits the hierarchy in a design to reduce processing time, memory use, and DRC result counts.

**Note**

Calibre LVS and Calibre xRC also have hierarchical versions. Much of the information in this section applies to those applications also.

Calibre DRC-H can use the same rule file as its flat counterpart, Calibre DRC-F. Calibre DRC-F and Calibre DRC-H are nearly identical, except for hierarchical processing algorithms and the reduction of design rule errors through hierarchical error suppression. There are some SVRF specification statements that apply only to hierarchical DRC. These statements may appear in a rule file during a flat DRC run and are ignored in that case.

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, OASIS, and CIF input, Calibre DRC-F flattens the layout database at the start of a run 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) 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 run.

For each cell, every operation determines which portion of the data Calibre can analyze independently 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.
Cell B contains two placements of cell A, and a shape that overlaps one placement of cell A. You can combine the two right-hand shapes 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 becomes a shape in cell B.

The performance improvement 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 by using DRC-H. This ranges from $2\times$ for many MPU designs, to orders of magnitude for memory designs. Hierarchical processing can similarly reduce memory use requirements.

All hierarchical layer operations can promote geometry out of the cells in which that geometry occurs when internal algorithms determine it is most efficient to do so.

### 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 shapes, this means that Calibre stores the data once with the cell (just as in the original user design).
Layer operations that generate derived layer shapes attempt to analyze them within each cell, promoting only when necessary to examine data in context. Calibre then creates the derived layer shapes at the lowest hierarchical level and, as with original layer shapes, stores shapes 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](image)

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 shapes in the top-level cell, and three separate errors. In this example, Calibre DRC-H implicitly recognizes that two errors are 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. Error suppression occurs as follows:

- Assume that DRC result R is a derived polygon 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 are normally fewer DRC results.
Hierarchical DRC
Flat Instantiations

If you want to see results on a per-cell basis, you must specify **DRC Cell Name** YES CELL SPACE XFORM in your rule file. This presents DRC results in cell space instead of top-cell coordinates.

DRC results databases are discussed in the chapter “DRC Results” on page 7-1. You can find information about viewing DRC results in Chapter 5, “Using the DRC-RVE Interface” of the *Calibre Interactive User’s Guide*.

Flat Instantiations

Calibre DRC-H supports layers in both hierarchical and flattened form.

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 that 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 it to a dual instantiation. Another possibility is that Calibre DRC-H may create the input layer as 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 flattens a layer with the Flatten operation.

Hierarchical Operation Efficiency

The following differences between flat and hierarchical performance exist in layer operations (refer to Table 5-1 on page 5-9):

- Dimensional check operations **Enclosure**, **External**, and **Internal** are slower when using a constrained PROJECTING filter, an interval constraint without using the OPPOSITE keyword, a NOT PROJECTING filter, a NOTCH or SPACE filter, or a CONNECTED or NOT CONNECTED filter.
- Two-layer Boolean operations are generally faster than polygon topological operations.
• Calibre DRC-H performs the one-layer Boolean operations \textbf{AND} (with a constraint other than $\geq 1$ or $> 1$) and \textbf{XOR}, in addition to \textbf{Magnify}, \textbf{Ornet}, \textbf{Polynet}, and \textbf{Rotate}, in a flat manner. This results in an exclusive flat instantiation of the output layer.

• Calibre DRC-H performs the \textbf{Angle} operation flat whenever the measurement constraint includes zero (0) but not 90, and vice-versa.

• You can perform the \textbf{Shift}, \textbf{Grow}, and \textbf{Shrink} operations in a cell if all of the placements of that cell have a consistent rotational/reflectional transformation component in the flat view of the design. Otherwise, input geometry must be promoted up to the lowest hierarchical level having the aforementioned characteristics. This ensures that the result of the operation is correct from the flat view.

• Some of the more rarely-used operations including (Not) \textbf{Donut}, (Not) \textbf{Enclose Rectangle}, \textbf{Extents}, \textbf{Holes}, \textbf{Perimeter}, \textbf{Size} with OVERLAP ONLY specified, \textbf{Vertex}, and the (Not) \textbf{With Edge} operation, 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.

• \textbf{Net Area Ratio} accumulation layers are flattened prior to printing.

• The \textbf{Rectangles} operation is more complex in hierarchical mode as compared to flat mode and is one of slower 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. Calibre DRC-F uses post-processing to find such false notch errors. In DRC-H, this post-processing 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 \textbf{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 run time to the operation. You should use this instruction \textit{only} to suppress false notch errors that Calibre actually generates, and \textit{you should not use this keyword in a production rule file.}

Because the possibility of false notch errors is low, you should not increase run time unless required. In fact, a minimal number of false notch errors can be acceptable in certain cases.

**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 \textbf{DRC Print Area} specification statement.
Hierarchical DRC

Text Mapping

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.

**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 mask data DRC results database. The text objects have the same hierarchy in both the input and results databases, unless Calibre DRC-H expands or flattens a placement during hierarchical processing. In that case, Calibre DRC-H moves the text up the hierarchy, as appropriate.

Calibre applications retain TEXTTYPE properties from GDSII input layout databases. If Calibre maps text with the Layer Map specification statements, then text in the DRC results database is on the target layer(s) of the mapping.

Hierarchy-Specific Statements

The specification statements: Expand Cell, Flatten Cell, Layout Base Layer, Layout Top Layer, Push Cell, and Layout Base Cell are used exclusively in hierarchical applications. By using one or more of these statements, you can potentially improve the performance of the hierarchical Calibre application. The improvement you may experience is design-dependent.

Layout Base Layer (or, alternatively, Layout Top Layer) is highly recommended. It assists Calibre in determining the design style. It has the benefit of being process-specific, not design-specific. Layout Base Layer is generally the more convenient of the two statements to use, but they both accomplish the same thing.

Layout Base Layer enumerates a set of device-level original layer names. You should include all device-forming layers. For example:

```
LAYOUT BASE LAYER substrate poly diff nwell nplus pplus contact
//do not use via layers
```

If you use Layout Top Layer, you specify high-level interconnect layers (non-device-forming layers such as metal, via, solder bump, pad, and cell boundary layers). When appropriate, include any new original layers defined for dual database capability in a Layout Top Layer statement.

**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 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 introduce problems for Calibre LVS-H. Therefore, the hierarchical database construction phase for Calibre LVS-H does 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 goal 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 that may later be designated in the LVS portion as hcells. Therefore, DRC-H also inhibits expansion of any layout cell in a rule file Hcell specification statement, regardless of the performance cost. The Hcell specification statement should only be in a rule file for DRC-H 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.
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

**Session Transcript**

DRC and LVS applications produce a transcript showing statistics for the following:

- Rule file compilation
- Layout data input
- Results database 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 compilation.

Errors terminate the run and are reported below the line identifying the pathname of the rule file:

```
--- RULE FILE = drc.db/gds/brules_drc
```

Refer to the chapter “Error Messages” in the *Standard Verification Rule Format (SVRF) Manual* for a description of 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:

- Layout Stream Summary Information
DRC Results
Session Transcript

- Layout Stream Data for Individual Cells
- Text Objects for Connectivity Extraction
- Text Objects for With Text Operations
- Ports (Calibre LVS / LVS-H only)
- Layer Read Summary (Shapes)
- 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 reports as notes in the transcript all (layer,datatype) pairs that contain geometric data which is not required in the run.

Limiting Transcript Output

In the transcript, Calibre applications print all connectivity extraction text objects in the run. Because 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 parentheses. The first number is the hierarchical count and the second is the (estimated) flat count. For example:

--- TOTAL GEOMETRIES WRITTEN TO ORIGINAL LAYERS = 866804 (6508994)

or

DRC RuleCheck 5.2.1.1 COMPLETED. Number of Results = 20 (134567)

The estimated number of flat objects depends upon the number of overlapping and touching objects in cell placements. This can differ from the actual flat count of objects.

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), connect node number placement, and device extraction (LVS). The executive module performs gate reduction, recognition, 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 Flat DRC

The following example shows layer statistics for a derived polygon layer:

nplus = diff NOT plus
---------------------
nplus (TYP=1 CFG=1 ECT=504382 OCT=381317 SRT=1 CMP=T MPN=14231
CPU TIME = 140 REAL TIME = 142 LVHEAP = 40/55/56 OPS COMPLETE = 34 OF 589

The operation has generated layer nplus. It required 140 CPU seconds and 142 real seconds. It was the 34th layer operation completed out of a total flow requiring 589 layer operations. The LVHEAP is explained under LVHEAP Statistics. These are the other components of this report format:

TYP — The nplus layer is type 1. These are the type definitions:

1: derived polygon layer
2: derived edge layer
3: derived error layer

CFG — The nplus layer is configuration 1. These are the configuration definitions:

0: neither polygon nor node
1: polygon
2: node
3: polygon and node
ECT — The edge count on nplus is 504382. This does not count vertical edges, which are not stored for TYP 1 layers (but are stored for TYP 2 layers).

OCT — The object count on this layer is 381317.

SRT and CMP — Nplus is sorted (SRT=1) and compressed (CMP=T).

OCT, SRT, and CMP are internal statistics.

MPN — The maximum polygon number on nplus. It is reported for TYP 1 layers with CFG 1 or 3. For layer selectors, it may not correspond to the actual number of polygons on the layer but generally does for layer constructors.

For error layers (TYP=3), the report is somewhat different, for example:

```
Rule23a::<1> = EXT nplus < 1.4 SINGULAR

Rule23a::<1> (TYP=3 ECT=4 CCT=2)
CPU TIME = 51 REAL TIME = 51 LVHEAP = 49/59/59 OPS COMPLETE = 34 OF 589
```

The elements in the third line of the report are similar to what is explained previously in this section.

Rule23a::<1> indicates the first output operation in DRC rule check Rule23a, whereas Rule34::X would identify locally-defined layer X in DRC rule check Rule34. The :: in this example signifies the layer is of local scope to the rule check in which it appears. Any layer name of the form TMP<number>, with possible local scope, is an internal name used for unwinding implicit layer definitions. These are the other elements in the report:

CCT — The number of edge clusters. This is only reported for TYP 3 objects.

ECT — The total number of individual edges that comprise the clusters.

It is important to note that multiple operations may appear in a single block. This indicates concurrency. For example:

```
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)
5.4.1::<1> (TYP=3 ECT=0 CCT=0)
CPU TIME = 383 REAL TIME = 388 LVHEAP = 58/61/63 OPS COMPLETE = 34 OF 589
```

The three operations in the previous example have been executed concurrently.

Layer Statistics in Calibre DRC-H

For Calibre DRC-H, sample statistics for a layer are quite different. As an example:

```
nplus = sdm NOT ppm
```
The first line shows the layer derivation. The second line begins with the derived layer name. The HIER indicates that the layer has a hierarchical instantiation. If nplus had an exclusive flat instantiation, the statistics would be the same as in Calibre DRC-F. If the layer had a dual instantiation, both flat and hierarchical statistics would be reported.

There are several variations of the HIER form:

- **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.

These are the other elements of the report:

- **TYP** — The nplus layer is type 1. These are the type definitions:
  - 1: derived polygon layer
  - 2: derived edge layer
  - 3: derived error layer

- **CFG** — The nplus layer is configuration 1. These are the configuration definitions:
  - 0: neither polygon nor node
  - 1: polygon
  - 2: node
  - 3: polygon and node

- **HGC** — The number of objects on the layer counted hierarchically, that is, once per cell.

- **FGC** — The *estimated* number of flat objects on the layer, computed by multiplying, for each cell, the number of objects in the cell by the total number of flat placements of the cell in the hierarchy, and then adding this together for all the cells. It is an estimate in the sense that no inter-cell merging of geometry occurs when making this calculation. The objects are polygons of TYP 0 and 1, edges of TYP 2, and edge clusters of TYP 3.

The estimated number of flat objects depends upon the number of overlapping and touching objects in cell placements. This can differ from the actual flat count of objects.

- **HEC** — For original and derived polygon layers, this is the total number of edges on the polygons counted by HGC. For derived edge layers, HEC is identical to HGC. For
derived error layers, HEC is the total number of individual edges on the edge clusters counted by HGC.

For polygon layers, HEC can be a better measure of the actual geometric size or density of a layer than HGC, since a polygon with N edges is only counted once in HGC, regardless of the size of N.

**FEC** — The *estimated* number of flat edges on the layer, computed by multiplying, for each cell, the number of edges in the cell by the total number of flat placements of the cell in the hierarchy, and then adding this together for all the cells. It is an estimate in the sense that no inter-cell merging of geometry occurs when making this calculation. Edges of TYP 2 and edge clusters of TYP 3 are included.

The estimated number of flat objects edges upon the number of overlapping and touching edges in cell placements. This can differ from the actual flat count of edges.

**VHC** and **VPC** — Identify the connectivity status of the layer. They are internal statistics.

Note that the flat statistic MPN is not reported.

The third line shows the CPU and real time required (in seconds) to generate the layer, the LVHEAP statistics, and the number of layer operations completed out of a total number required for the run.

**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:

$$\text{LVHEAP} = 28/47/49$$

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, GDSII, or OASIS format in DRC applications. (ERC and short isolation results generated by LVS applications also use the ASCII DRC results database format.)
The DRC results database is a collection of geometric objects grouped by rule check. Each rule check in the DRC results database contains a list of objects that 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 “Layers” on page 4-2 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-edge groups) 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, GDSII, or OASIS DRC results database. The ASCII database is in user-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 ICrudules into ICgraph. As such, a Calibre DRC results database can be scanned by the ICrudules DRC results presentation commands. An ICrudules DRC results database can also be written out as an ASCII DRC results database. These databases can be read by Calibre RVE and viewed interactively with supported layout editors.

Calibre DRC/DRC-H allows you to output as many DRC results databases as needed during one DRC/DRC-H run. 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 completes the writing (including file close) to a DRC results database as soon as all of the DRC rule checks that contribute to that database have completed. A message is transcribed when output to a specific DRC results database is completed. This allows you 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 corresponds 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 that contains rule check statements but no results.

The precision of a DRC results database matches the rule file Precision statement by default. You can change the results database precision by using the DRC Results Database Precision statement.

**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.

**ASCII DRC Results Database Format**

Calibre DRC can generate an ASCII DRC results database if the ASCII keyword appears in the DRC Results Database specification statement. This format is used by Calibre RVE and ICrules for viewing geometric results.

No blank lines appear in the ASCII DRC results database, and data always start at the beginning of the line. Figure 7-1 shows an example ASCII DRC results database.
Cell Name and Database Precision

The first line in the database 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.
DRC Results Database

- The date/time stamp shows when the rule check was executed. The date/time format is as follows (blanks are significant):
  
  \[ \text{mmm dd hh:mm:ss yyyy} \]

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:

- Pathname of the rule file.
- Title of the rule file, if any.
- Any rule check comments.

You can remove this information from the header, or you can add more information 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. These are 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 by 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 does 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.
DRC Cell Name Results

If you use the DRC Cell Name or ERC Cell Name specification statements, the results database has additional lines like this:

```
CN NAND034 c 0 1 -1 0 323446 345646
```

The CN stands for “Cell Name”, followed by the cell name, followed by a c character (if result is in cell space coordinates), followed by the transformation matrix to top-level space. This is described in detail in the SVRF Manual.

Results presentation in cell space is discussed in Chapter 5, “Using the DRC-RVE Interface” of the Calibre Interactive User’s Manual.

Properties in ASCII DRC Results Databases

ASCII DRC results databases, including those generated with the RDB keyword, allow properties to be attached to individual results. An ASCII DRC results database property is a string (“ID”) followed by context-dependent information (which may be empty). The property appears following these syntactical elements:

```
p <number> <vertex-count>
or
e <number> <edge-count>
```

and before any coordinate information. A property ID may not be numeric, and an individual property must be on a single line. A DRC result may have any number of attached properties. For example, this comes from a Density operation:

```
TOP 1000
rule_A
15918 15918 1 May 12 08:32:00 2003
DENSITY M1M PGM >= 0.25 WINDOW 100
p 1 4
DV 520
DG 0.866
DA 10000
DA M1M 520
DA PGM 0
-8936900 -4262700
-8935900 -4262700
-8935900 -4261700
-8936900 -4261700
...
```

In this example, the lines starting with DV, DG, and DA all specify property IDs.

Such properties appear in results databases generated by Net Area Ratio and Density, the DFM family of operations, as well as for DRC Cell Name results. Properties available for mapping can be displayed in color maps and histograms in RVE. Refer to Chapter 5, “Using the DRC-RVE Interface” of the Calibre Interactive User’s Manual for more information.
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. This format is not readable by the ICrules REStore DRc Results command.

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 2× smaller than ASCII format.

The Bachus Naur Format for a binary DRC results database is similar to that of its ASCII counterpart, 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>
```


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. Refer to section “Hierarchical DRC Results Database” on page 7-16 for potentially different semantics in the hierarchical case.

The GDSII format is a standard GDSII representation of the DRC results database. A GDSII DRC results database has the following BNF (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 is 6.0.
- The modification and last access times in the BGNLIB and BGNSTR records are the date/time of database creation. Years are relative to 0 BCE and January is month 1.
- The default library name in the LIBNAME structure is drc.db.
- The UNITS are drawn from the rule file Precision and Unit Length specification statements or their defaults. (Note that the Unit Length statement is not used by DRC applications.)

For flat DRC, 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, OASIS,
or CIF), then the cell name is drc. DRC applications 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 case of clusters. Clustering is lost.

### OASIS DRC Results Database Format

Calibre DRC generates an OASIS DRC results database if the OASIS keyword is specified in the required DRC Results Database specification statement (or in a DRC Check Map specification statement for a particular DRC rule check). This format is not readable by the ICrules RESTORE DRC RESULTS command.

The OASIS format is a standard OASIS representation of the DRC results database. An OASIS DRC results database generated by Calibre has the following BNF (refer to the OASIS specification for more information):

```plaintext
<magic-bytes> START <name>* <cell>* END
<name> -> CELLNAME | TEXTSTRING | LAYERNAME
<cell> -> CELL <element>*
<element> -> <geometry> | PLACEMENT | TEXT
<geometry> -> RECTANGLE | POLYGON | PATH | CTRAPEZOID
```

where

- The version number in the START record is 1.0.
- The precision value in the START record is drawn from the rule file Precision specification statement, or its default. It is either a type 4 or 7 real (the only real types ever written).
- There is no <name> record en tabulation in the START or END record.
- The END record is the hex byte 0x02 (the record ID), followed by 253 hex bytes 0x80 then one zero byte (a b-string of length 0) followed by a zero byte (the validation signature). The physical end of file follows.

Observe that there are no PAD, PROPERTY, PROPNAME, PROPSTRING, XNAME, XYRELATIVE, XYABSOLUTE, XELEMENT, TRAPEZOID, CIRCLE, or XGEOMETRY records present. There are no forward references to <name> records.

For Calibre DRC-F, 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; this optional string is that following the OASIS keyword in the DRC Results Database specification statement. If there is
no Layout Primary specification statement (meaning that the layout system is not GDSII, OASIS, or CIF) then the cell name is drc.

For Calibre DRC-H, the database hierarchy is preserved in an OASIS-type DRC results database, and there is no suppression or top-level transformation of DRC results (as with ASCII- and binary-type DRC result databases). Cell names are maintained with an optional string appended; this optional string is that following the OASIS keyword in the DRC Results Database specification statement.

DRC rule check output to OASIS-type DRC results databases is not partitioned by cell, but by layer and datatype only. The output (layer,datatype) coordinate for a DRC rule check must be specified in a DRC Check Map specification statement. This statement specifies that output from the given rule check is on layer $<layer-number>$ with datatype $<datatype>$, or layer 0 if $<layer-number>$ is not specified, and datatype 0 if $<datatype>$ is not specified. If output for a DRC rule check is to an OASIS-type DRC results database and there is no DRC Check Map specification statement corresponding to the rule check, then Calibre DRC issues a warning and layer 0, datatype 0 are used.

A LAYERNAME record is written to an OASIS-type DRC results database for each DRC rule check that is output to that database. The record consists of the name of the DRC rule check and the (layer,datatype) coordinate specified in the appropriate DRC Check Map statement (or (0,0) if none).

Edges and edge clusters are each written out to an OASIS DRC results database as 0-width paths, one path per edge in the latter case. Note that clustering is lost.

Other features of OASIS DRC results databases:

- Modal variables placement-cell, text-string, xy-mode, polygon-point-list, path-point-list, path-start-extension, path-end-extension, circle-radius, last-property-name, and last-value-list are not utilized.
- Character set restrictions in a-strings and n-strings are not enforced if it is necessary to output strings which already violate these restrictions.
- Repetitions of type 8 are never output.
- Point lists of type 0, 1, and 5 are never output.
- TEXT and PATH repetitions are not deployed.
- OASIS LAYERNAME records are output to an OASIS DRC Results Database for all DRC RuleChecks which map to that database. The record consists of the name of the DRC RuleCheck and the layer/datatype specified in the appropriate DRC Check Map statement (or the default 0/0).
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 an upper bound value, whenever DRC generates results equal to the upper bound for any single rule check, it adds no further results into the DRC results database for that rule check and issues a warning message.

To limit the DRC result count, use the 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 for 32-bit machines and 4294967294 for 64-bit machines. This value defaults to 1000 if the statement is omitted.

A warning is generated if any DRC rule check outputs to a GDSII or OASIS DRC results database with a maximum result count less than ALL. The purpose is to warn of inadvertent use of the default value of 1000 when writing mask data versus DRC errors. Setting the environment variable CALIBRE_EXIT_MAXIMUM_RESULTS to a non-null value causes Calibre DRC to immediately exit after any such warnings are generated.

Hierarchical DRC Results Database

The database hierarchy is completely preserved in a mask data 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 appended string. This optional string is determined by the DRC Results Database specification statement. Recall that the flat Calibre DRC applications output a geometric DRC results database with exactly one cell. 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 the hierarchical algorithms. These new internal cells are called “pseudocells.” They are named ICV\_n, where n is incremental, so that the cell names are unique. By default, DRC results that end up in pseudocells are transformed up the hierarchy to the first true user cell, prior to being instantiated in the DRC results database. Note that pseudocells of this type cannot be used as hcells in LVS.

For a mask data DRC results database, no pseudohierarchy is instantiated. You can reverse this suppression of pseudohierarchy 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 DRC results databases
used for mask data 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 geometric 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 geometric 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 run. The following is an example:

```
======================================================================
=== CALIBRE::DRC-F SUMMARY REPORT                                      ===
======================================================================
Execution Date/Time:      Wed Apr 28 15:04:33 2003
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/john/drc_example
User Name:                john
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 a DRC run. See “**Runtime Messages**” in the *SVRF Manual* for a listing.

- **Original Layer Statistics** — This section lists the original layers and the number of original shapes 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 performed.

  Note that in DRC-H, rule check results are shown both hierarchically and flat. Here is an example:
DRC Results
DRC Summary Report

RULECHECK 10 ............... TOTAL Result Count = 14 (2366)

The first result count is the number of results counted once per cell. The second number (in parentheses) is the *estimated* flat count of results from all placements of each cell. This is only an estimate obtained by multiplying the first number by the number of placements of each cell. This estimate does not take into account hierarchical cell transformations such as rotation and reflection, so the true flat count may differ.

- Summary information — This section shows the total run time, the number of original shapes processed, the number of rule checks executed, and the number of results generated.

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 is 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.
This chapter discusses how connectivity extraction is performed and the factors that influence this extraction. Main topics include:

- Establishing and Verifying Connectivity
- Connectivity and Rule File Compilation
- Recognizing Electrical Nets
- Transferring Logical Information to Merged Layers
- Short Isolation
- Connectivity Extraction Errors and Warnings

Connectivity extraction recognizes electrically-connected regions in the layout called nets. Nets are recognized from layout shapes through analysis of the relations between layout shapes and other objects on various layers. The analysis is driven by statements you specify in the rule file. Connectivity extraction results are then used internally by various Calibre products.

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.

Establishing and Verifying Connectivity

Many layer operations require that you establish the circuit connectivity of the design beforehand so that the correct connectivity information is present on the input layers.

For example, in DRC, nodal dimensional check operations (that is, ENClosure, EXTernal, or INTernal operations 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. Parasitic extraction operations and device recognition also require circuit connectivity to be available or computable.
There are two types of connectivity extraction methods that you can specify in the SVRF language, Mask and Direct. Direct extraction applies only to ICverify, not to Calibre.

**Mask Connectivity Extraction**

Mask connectivity extraction analyzes a layout hierarchy and extracts connectivity from mask-level shapes. It is only invoked as an internal subsystem by other components of the layout verification system, namely LVS and DRC. LVS-extracted connectivity stores its own database on disk, thus allowing graphic highlighting of nets.

Connectivity extraction happens automatically when performing flat LVS. In LVS-H, connectivity extraction is triggered by the `-spice` command line argument.

DRC only uses connectivity information internally for connectivity-dependent rule checks. ERC, which is performed as a part of LVS, behaves like DRC with regard to connectivity extraction.

Mask connectivity extraction internally merges overlapping shapes and paths on any single database layer. References to the original database objects are not maintained. LVS, PEX, ERC, and DRC analyze the layout hierarchy. Connectivity is then extracted from completely flat shapes. Only external shapes (typically pins) of cell instances are processed. Instance pins contribute to connectivity extraction as explained in the section “Ports and Pins” on page 8-5. Ports of the top-level cell also participate in Mask connectivity extraction.

**Connectivity and Rule File Compilation**

The rule file compilation verifies that any operation requiring connectivity has 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**

- **Note**

  Calibre functions only in Mask mode for connectivity extraction, while ICverify uses both Mask and Direct modes. In Calibre, you need not be concerned with this distinction between connectivity modes.

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.
- 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.
Connectivity Extraction

Recognizing Electrical Nets

- Be derived (by a sequence of net-preserving operations) from a layer appearing directly in a Connect or Sconnect operation.

- Be derived (by 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 (by 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.

Establishing Connectivity for Layer Operations

The (Not) Net, Net Area, Net Area Ratio, Ornet, Polynet, and Stamp operations; the nodal dimensional check operations Enclosure, External, and Internal (containing the CONNECTED or NOT CONNECTED parameters); and the (Not) Cut, (Not) Enclose, (Not) Interact, (Not) Touch, operations that use the BY NET keyword are the only layer operations that require connectivity on their input layers.

Recognizing Electrical Nets

The statements that connectivity extraction uses 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 8-1.) 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 8-1. 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. Figure 8-2 shows an example.
Connectivity Extraction
Recognizing Electrical Nets

**Figure 8-2. Polygons Connected Directly**

```plaintext
connect polya polyb
```

**Figure 8-3. Polygons Connected By Contact**

```plaintext
connect metal poly by cont
```

**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. Figure 8-3 shows an example.

**Sconnect**

**Sconnect** establishes soft connections from an upper layer to a lower layer. You can 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. Figure 8-4 shows an example.

**Figure 8-4. Sconnect Operation**

```plaintext
sconnect metal1 poly by contact
```

You can specify polygons from lower layers that 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. LVS
Softchk operations are executed after ERC checks and ERC Pathchk operations to prevent LVS Softchk from negatively affecting the hierarchy for ERC operations. In particular, the additional promotion caused by LVS Softchk could interfere with unused device filtering.

To view Sconnect conflicts, LVS Report Option S turns on detailed reporting of Sconnect conflicts in the session transcript, as well as in the LVS Report (for flat LVS), and in the circuit extraction report (for LVS-H). The format of the warning is shown under “Connectivity Extraction Errors and Warnings” on page 8-18. This option is ignored in DRC applications.

You can also cause Calibre LVS to stop after detecting any Sconnect conflicts, regardless if LVS Softchk statements are present or not, by using the LVS Abort On Softchk YES specification statement in your rule file.

**Stamp**

Stamp is a layer operation used in Calibre DRC and Calibre LVS to create a derived layer that receives its connectivity information from a stamping layer. The derived layer is used to pass logical connectivity between overlapping polygons of two different layers.

In the following example, the Stamp operation selects all polyc polygons, overlapped by metall polygons, that can receive valid connectivity information from metall polygons. The connectivity information is assigned to the derived layer named x.

```
x = stamp polyc by metall
// x is a derived layer with metall connectivity information
```

This type of connectivity allows LVS to assign net numbers to intentional devices based upon these overlaps. By knowing which intentional devices reside on which nets, a comparison can be made against the schematic.

For most applications involving soft connection (or nwell jumper) checks, using the Sconnect statement is preferable to using Stamp. There are some specialized DRC applications in which Stamp may be useful.

In Calibre xRC, the Stamp operation is basically treated like a Sconnect statement. However, the layer specified in the BY argument of the Stamp operation is frequently a contact layer and in Calibre xRC it is illegal to directly connect a contact layer to any other layer. As a result, all layers necessary to connect parasitic layers to device layers should be identified in a Connect (or Sconnect) statement, which provides all of the connectivity necessary to produce a valid point of connection for the device pins.

**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, a pin) when the cell is placed at a lower level of the layout hierarchy.
In Calibre, you can specify ports for the top-level cell using Port Layer Text and Port Layer Polygon specification statements in the rule file. Port layer specification statements allow you to specify ports in GDSII, OASIS, and CIF layout databases. In Calibre LVS and Calibre xRC 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 shapes 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 shapes defined in the rule file (with the Text and Polygon specification statements) cannot become ports.

The depth from which port objects are read is controlled by the Port Depth statement.

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 shapes whose layers appear in Connect or Connect By operations, or as first parameters in Attach operations in the rule file. Other shapes do not participate as ports in connectivity extraction. Information on ports and pins can be transferred from original layers to derived polygon layers by means of the Attach operation (see “Attach Operation” on page 8-8). Other geometry does not participate as ports in connectivity extraction.

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.

The effects of ports and pins on connectivity extraction are described in the following sections.

---

**Note**

The hierarchical connectivity extractor strips off leading and trailing whitespace (including newlines) and non-printable characters from port names. If a port name consists entirely of whitespace or non-printable characters, then the port is unnamed.

---

**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 can nevertheless be used by Calibre xRC—see the Performing Gate-Level Parasitic Extraction manual for more information.

Port text and polygon objects at any level of hierarchy can overlap shapes at lower levels of the hierarchy and can be attached to those lower-level shapes. The hierarchical connectivity
extractor forms any pins in lower level cells that are necessary to connect to port objects higher up in the hierarchy. This is similar to how layout 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. The .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 can specify the hierarchical depth for reading 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).

Also see “Ports” on page 12-43 for netlist port naming details.

**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, and pin information from shapes and paths of the corresponding original layers, namely:
Connectivity Extraction

Transferring Logical Information to Merged Layers

- 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.

After merging has occurred, connectivity information is established through various rule file elements.

Attach Operation

Attach is a connectivity extraction operation that transfers connectivity information (net names, port locations, and pin locations) from one layer to another layer.

- For more information on net names, refer to “Net Name Specification” on page 8-8.
- Port locations are transferred from port shapes on layer1 to overlapping polygons on layer2. Port names 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.

In the previous 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 run.

You can 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

These are the ways to specify net names in connectivity extraction:

- **Text** and **Layout Text** specification statements in the rule file.
- Layout database text objects (GDSII, OASIS, 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.

Connectivity extraction is case-insensitive by default. For example, net names “abc” and “ABC” are not netlisted separately. In such a case, the connectivity extractor selects one label and discards the other. The net for which the label is discarded receives a numeric ID. A warning is issued for open circuits.
Text Specification Statements

Text objects can be specified directly in the rule file with the following statements:

- **Text specification statements** — The Text specification statement allows you to specify free-standing text directly in the rule file. It also allows you to edit text objects read from the layout database.

- **Layout Text specification statements** — This statement is applicable only when the layout system is GDSII, OASIS, or CIF. The Layout Text specification statement allows you to specify free-standing cell-based text 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” on page 8-10 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.

The following 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 override 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), whereas 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, geometric 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.

**GDSII, OASIS, and CIF text objects** — Only GDSII, OASIS, or CIF layout text on layers specified in Text Layer specification statements is used by connectivity extraction. Therefore, if there are no Text Layer statements in the rule file, then no layout text is used by connectivity extraction. In addition, if the Text Depth ALL statement is specified, then text for connectivity extraction is read from throughout the hierarchy. If the Text Depth PRIMARY statement is specified, then only text from the top-level cell is used by connectivity extraction. If this statement is not specified in the rule file, then Text Depth PRIMARY is the default.

The Text Layer and Text Depth statements do not apply to text objects entered with Text specification statements.

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 previously, the connectivity extractor generates internal label objects to represent text data from various sources: rule file Text specification statements, and geometric 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 following order:

1. **Explicitly** — 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 the label location may intersect 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, …, Bn 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** — 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** — If the rule file contains the following operation:

```
LABEL ORDER ... B1 B2 ... Bn
```

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 are attached to metal if metal is present at the location of the label. Otherwise, text is attached to poly if poly is present.

Additionally, if two or more different names are found on a single net, Calibre chooses the first name according to alphabetical order and discards the other names. 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.

- **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. If you include the statement:

```
ATTACH diffusion src_drn
```

in your rule file, src_drn receives the nodal information from diffusion. To assign a net name directly to a source or drain region, place a text object on the diffusion layer over this region.
• **Implicit attachment** — The original database layers poly and metal appear in Connect operations in your rule file. Poly and metal shapes that touch share connectivity information.

• **Free attachment** — Suppose you use the database layers metal1, metal2, and poly for interconnect and you want to use database layer text to specify net names. To do this, include the following 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 is chosen.

See also the Layout Property Text and Layout Rename Text specification statements in the SVRF Manual.

**Label Stripping**

The hierarchical connectivity extractor strips off leading and trailing whitespace (including newlines) and non-printable characters from names of net labels. If a label name consists entirely of whitespace or non-printable characters then the label is discarded.

**Hierarchical Treatment of Net Labels**

This section describes how net labels are treated in hierarchical connectivity extraction.

• Labels from a cell are used locally in the cell to name nets in the cell.

• Labels in a cell can be attached to polygons in the cell or in the sub-hierarchy of the cell. When labels in a cell (the owner cell) are attached to polygons at a lower level of hierarchy, *virtual nets* are created in the owner cell to represent this, and the label names are assigned to those virtual nets in the owner cell. Pins are created through the hierarchy to properly represent the connections between virtual nets in the owner cell and original polygons lower down. Label names from higher levels of hierarchy are not assigned to actual nets at lower levels of hierarchy.

• Labels from each cell are used locally in the cell regardless of the Text Depth setting. Text Depth controls only which labels are used in the top level cell. Text Depth ALL causes labels from lower levels of hierarchy to be used in the top-level cell, in addition to being used at the lower level (not instead of being used there).

• When cells are expanded by hierarchical database heuristics or by rule file directives, net labels from those cells are discarded and are not used for naming nets. However, the Expand Cell Text specification statement can be used to *promote* cell text to a higher level of hierarchy.
Virtual Connect Statements

The virtual connect technique in layout verification is using the capability of the layout connectivity extractor to form a single net from two or more disjoint nets by virtue of net segments that share the same name, after special parsing. Virtual connectivity 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

You can specify Virtual Connect Colon once in a rule file. If you specify YES, 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).

The Virtual Connect Semicolon As Colon specification statement controls whether semicolons (;) are handled in the same way an colons (default is YES).

You can enter labels with rule file Text statements, or geometric database text as described in section “Layout Database Text Objects” on page 8-9. 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.

In hierarchical connectivity extraction, stripping of colon suffixes from net names occurs at all levels of hierarchy. Actual virtual connections, however, are normally performed in the top-level cell only; this can be changed with the Virtual Connect Depth specification statement.

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.

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
```

If you specify NO, 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.
Virtual Connect Name

You can specify Virtual Connect Name any number of times in a rule file. 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:

1. VIRTUAL CONNECT BOX NAME “?”
   This connects net segments with identical names in LVS Box cells. It also connects together respective pins. In Figure 8-5, if A, B, and C are LVS Box cells, then the points marked 1, 2, 3, and 4 all belong to the same net.

Figure 8-5. Example of Virtual Connect Box

2. 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 essentially detects 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 statement isolates the short by finding the shortest path between two or more pieces of conflicting text. The results are sent to an ASCII DRC database format file, which is viewable using a layout editor and Calibre RVE. Refer to the section “ASCII DRC Results Database Format” on page 7-8 for information on the results database format. For more information on the hierarchical short isolation database, refer to the section “Hierarchical Short Isolation Results Database” on page 8-17.
Connectivity Extraction

Short Isolation

The following commands execute short isolation in flat LVS or layout-to-Cnet translation:

```
calibre -lvs rules
calibre -lvs -tl lay.cnet rules
```

In flat and hierarchical systems, all short isolation results are returned in the coordinate space of the top-level cell. Short isolation in the top-level cell operates on the text at that level only, subject to the 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 before the run terminates, 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
```

If the run produces no shorts and short isolation is enabled in the rule file with LVS Isolate Shorts YES, Calibre automatically removes an existing short isolation results database. If LVS Isolate Shorts YES is not specified and the run produces no shorts, the existing short isolation results database is not removed.

**Short Isolation in Hierarchical Mode**

In hierarchical mode, circuit extraction is performed hierarchically, the short isolation algorithm operates hierarchically, and shapes are no longer flattened. 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. To execute short isolation in hierarchical mode:

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

When more than one path exists between a pair of conflicting text points, the hierarchical short isolation algorithm prefers paths that consist of shapes 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 as follows:

1. Hierarchical connectivity extraction is performed.
2. If shorts exist, short isolation algorithm operates hierarchically and shapes 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 textured shapes that are involved in a short. Therefore, if you use Connect statements to form connections to the substrate, you can replace them with Sconnect statements to increase efficiency.

Hierarchical Short Isolation Results Database

The DRC ASCII results database produced by the hierarchical short isolation contains sequence numbers of polygons in the short. The sequence numbers indicate the order of polygons in the path connecting the two shorted texts. These numbers increase monotonically along the path of the short.

The first shorted text always has sequence number 1, the polygon with the smallest sequence number greater than 1 connects to this text, the polygon with the next smallest sequence number connects to the last polygon, and so on. The second shorted text always has the largest sequence number. Sequence numbers in general are not consecutive, and the polygons in the results database are not sorted by sequence numbers.

The sequence numbers help in tracing the short path and can also be used for pruning the short path. For example, if the path connects texts VCC and VSS and has sequence numbers from 1 (VCC) to 100 (VSS), and you assert that the polygon with the sequence number 50 belongs to net VCC, you can then ignore all polygons with sequence numbers below 50 since they form the part of the path from the text VCC to the asserted VCC polygon. The polygon which causes the short has to be somewhere in the path from VSS to the asserted VCC, which means that its sequence number is between 50 and 100.

Sequence numbers are printed out in the DRC result listing section of the ASCII DRC results database using the SN property. For example:

```
p 1 4
SN 14
1000 2000
3000 2000
3000 2500
1000 2500
```
Connectivity Extraction
Connectivity Extraction Errors and Warnings

For the shorted texts, which are endpoints of the short, the SN property is printed in the check text after the text information as shown in the following example:

```
  TOP 1000
  SHORT 1.  VCC - VSS in TOP
  7 7 3 Oct 12 18:54:40 2005
  2 Shorted texts:
  "VCC" at (1275, 510) on layer "MET1" SN 1
  "VSS" at (-205, 510) on layer "MET1" SN 17
```

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 and in the circuit extraction report file. 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 that 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 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 favoring the following items, in order: (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 cannot 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: Hierarchical Calibre LVS warns about unattached ports, but does not send them to the comparison phase. Flat Calibre LVS reads unattached ports into the comparison phase. They are then filtered out, unless they serve as initial correspondence points. (A port serves as initial correspondence point if the source has a port with the same name). The LVS end result (CORRECT or INCORRECT) is usually the same either hierarchical or flat, except in some rare cases; for example, the end result may be different when the layout has both a good, attached port text “A” and a false, unattached port text “A”.

- WARNING: Stamping conflict in SCONNECT - Multiple source nets stamp one target net.

Indicates that several different nets in an Sconnect operation tried to form a connection to a single, lower-layer target polygon (similar to what the Stamp operation does). The net selected for stamping is reported; this is the net to which the target polygon gets connected. The rejected nets are reported as well. In hierarchical operation, all net names and net numbers are in the context of the cell being discussed. In hierarchical operation, the net name or net number of the target polygon may also be reported. This is done when the target polygon retains its own distinct net number within the cell and the connection is made instead at a higher level of hierarchy. For example:

```
WARNING: Stamping conflict in SCONNECT - Multiple source nets stamp one target net.
Target net: 10.
Net VSSQ is selected for stamping.
Rejected nets: 13 14
```

This example is from a hierarchical run. Net VSSQ was selected for stamping. Nets 13 and 14 were rejected. The target polygon retained its own net number - net 10 - in the cell, but nets 10 and VSSQ were connected at a higher level of hierarchy.

This type of warning is issued when LVS Report Option S is specified.

- WARNING: Stamping conflict in SCONNECT - Multiple source nets stamp one target net. Use LVS REPORT OPTION S or LVS SOFTCHK statement to obtain detailed information.

Indicates that several different nets in an Sconnect operation tried to form a connection to a single, lower-layer target polygon (similar to what the Stamp operation does). This type of warning is issued when LVS Report Option S is not specified.

- NOTE: Virtually connected <description>

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)
Electrical rule check (ERC) operations in Calibre perform tasks related to electrical rule checking. In many respects, ERC is similar to DRC. However, ERC is essentially a separate task and (unlike DRC) is performed as part of an LVS run. That means ERC has a separate results database and a dedicated set of specification statements to control operation and result generation. Many of these statements 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. Note that a DRC and LVS license is required to run ERC checks.

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, you can use ERC operations 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 9-1 lists the specification statements related to ERC.

<table>
<thead>
<tr>
<th>ERC Cell Name</th>
<th>ERC Check Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERC Keep Empty</td>
<td>ERC Maximum Results</td>
</tr>
<tr>
<td>ERC Maximum Vertex</td>
<td>ERC Path Also</td>
</tr>
<tr>
<td>ERC Pathchk</td>
<td>ERC Results Database</td>
</tr>
<tr>
<td>ERC Select Check</td>
<td>ERC Summary Report</td>
</tr>
<tr>
<td>ERC Unselect Check</td>
<td>LVS Execute ERC</td>
</tr>
</tbody>
</table>

Table 9-2 shows the ERC operations.

<table>
<thead>
<tr>
<th>Pathchk</th>
<th>Device Layer</th>
</tr>
</thead>
</table>
ERC operations may contain PRINT secondary keywords that are responsible for the production of auxiliary files. See “ERC Output Files” for more information.

For detailed information on each statement, refer to the Standard Verification Rule Format (SVRF) Manual.

Execution of ERC Operations in LVS

ERC within Calibre LVS (including PRINT options) is controlled by the ERC Select Check and LVS Execute ERC specification statements.

Note

If any rule checks are selected, DRC licenses are acquired in addition to LVS licenses. See “ERC Licensing” on page 9-4. For further information about licensing, refer to Configuring and Licensing Calibre Tools.

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 “Rule Check Statements” on page 4-10.) ERC is performed in the LVS connectivity extraction stage.

You can suppress execution of all selected rule checks in Calibre LVS by using the LVS Execute ERC NO specification statement. DRC licenses are not used 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 performed.

Note

Some operations (for example DRC measurement operations) may run slower and consume more memory when executed by ERC within Calibre LVS when compared to Calibre DRC.

The following commands perform ERC in Calibre LVS:

- calibre -spice … rules
  For this command line syntax, ERC runs hierarchically.

- calibre -lvs rules
  For this command line syntax, ERC runs flat, and the Layout System statement must specify a format that consists of shapes, such as GDSII, OASIS, 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 geometry, such as GDSII, OASIS, and CIF.
All these commands use DRC licenses, in addition to LVS licenses, if ERC rule checks are selected and ERC is not disabled.

ERC operations are not performed in Calibre xRC.

**ERC PRINT Options**

LVS applications also perform side-effects of ERC operations, such as the PRINT POLYGONS and PRINT NETS secondary keywords, 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:

```plaintext
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):

```plaintext
X {
    (PATHCHK !POWER PRINT POLYGONS "file1") AND 999
    //layer 999 is empty
    ERC SELECT CHECK X // triggers execution of X
}
```

**Rule Check Selection in LVS**

LVS uses the following process for rule check selection:

1. Unselect all rule checks.
2. Select rule checks from ERC Select Check specification statements.
3. Unselect rule checks from ERC Unselect Check specification statements.

**ERC Operations in DRC**

Calibre DRC performs all selected rule check statements subject to the DRC rule check selection mechanism described next. 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

DRC applications use the following process 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. DRC unselects all rule checks.
2. DRC selects rule checks from DRC Select Check specification statements.

otherwise

1. DRC selects all rule checks.
2. DRC unselects rule checks from ERC Select Check specification statements.
3. DRC unselects rule checks from DRC Unselect Check specifications statements.

ERC Licensing

When ERC is used in flat LVS, flat DRC and flat LVS licenses are used. When used in hierarchical LVS (calibre -spice), four licenses are used: LVS, LVS-H, DRC, DRC-H. In -turbo mode, Calibre LVS uses additional DRC and DRC-H licenses depending on the number of threads. All four licenses are always used in equal numbers. For example, if the requested number of threads requires four LVS and LVS-H licenses, and Calibre is run as calibre -spice -turbo, it attempts to use four DRC and DRC-H licenses as well. If less than four are available, then the number of threads is reduced, and the remaining LVS licenses are released. If Calibre is invoked as calibre -spice ... -lvs -hier, all DRC and DRC-H licenses are released before LVS comparison begins. DRC licenses are used only if ERC rule checks are selected for execution (as controlled by ERC [Un]Select Check) and ERC is not disabled by LVS Execute ERC NO.

Alternatively, when Calibre LVS is invoked with the -cb command line option, it may execute ERC rule checks without acquiring additional licenses beyond the single caldrclvseve license used by Calibre LVS in that mode. Note that the -cb option can only be used in flat mode.

ERC Output Files

In addition to derived layers, ERC 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 can generate a results database similar to the type generated during a DRC run, which is discussed in section “ASCII DRC Results Database Format” on page 7-8.
ERC Auxiliary Files

ERC operations contain secondary keywords that instruct Calibre to produce auxiliary files during ERC.

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**
  - 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) ]
  ```

  - `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**
  - 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, ...
    ```

    - `layout_primary` — specifies the top-level cell as defined in the **Layout Primary** specification statement.
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** — shows creation of a layer with the Pathchk operation, and how different invocations on the same rule file affect the output.

Assume the following rule file excerpt exists:

```plaintext
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 assume the following invocation command:

```
calibre -spice z.net rules
```

Calibre executes rule checks E1 and E2, and sends the results to an ERC results database named “ercdb”. Rule check E1 requires the prior 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 perform any rule checks or the Pathchk operation.

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** — demonstrates 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 assume 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 operation output layer is written to rule check X in the ERC results database ercdb.

- **Example 3** — generate 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 assume 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. You can assume that there are no shapes on layer 999.
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.

Connectivity extraction precedes device recognition in the chain of LVS processes. Connectivity extraction recognizes the electrical nets of the layout and annotates layer shapes with net numbers. For further information, refer to the “Connectivity Extraction” chapter. Device recognition uses these net numbers to associate the pins of recognized devices with the nets to which they connect. This 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 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.

This chapter discusses the following major topics:

- Concepts and Terminology
- Recognition Logic
- Property Computation

Concepts and Terminology

Device recognition attempts to match device instances based upon Device statements you provide in your rule file. In this sense, Device statements are patterns, or templates, which the
Device recognition algorithms use to attempt to classify your device instances. It is important to recognize the distinction between a device and an instance:

- A device, as defined in a Device statement, is an abstract template that describes how a collection of shapes can be used to match instances of the device in your layout.
- 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 that 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. Within each swap group the connections to the pins are considered interchangeable. For example, the source and drain pins of a MOS transistor are usually interchangeable and would belong to the same swap group. Swap group information is used by LVS when comparing two devices.
- **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 that aid in the recognition of device instances.
- **Text model layer** — specifies a text layer in the layout used to determine 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.

A device instance also has many aspects. It inherits the element and model names from the device definition of which it is an instance. It also inherits pin names and properties. Since the pins have physical realizations, they each have a pin location and an associated electrical net. The property rules of the device can be applied to compute property values for the instance.
A net is said to be isolated, if it contains no port or instance pins. Information about the location of such nets is also made available to LVS.

### 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.
- For each seed shape, identify an initial set of auxiliary shapes and pins to which the seed shape is connected.
- Classify the device if the resulting pattern of auxiliary shapes and instance pins matches 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 (bad) 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.

Consider these Device statements. The three statements compete to classify each shape on layer dev_lay.

```plaintext
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)
```

Because 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 previous example, 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 fails to match any of the three operations and is 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, in the absence of the D3 operation, only pin layers pin_lay_1 and pin_lay_2 are 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 does not consider the third dimension; therefore, shapes 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 that 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 bad.

## 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 as follows:

- 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 and with 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, which are connected to the same net, 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 a device if the set of initial pins and layers exactly matches 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 rule file 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**

The fill-in algorithm tries to find a unique match to a Device 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 are true:

- The pins and layers of the initial set can be matched with a subset of the pins and layers in the Device operation.
- The initial pin set contains at least one pin on each pin layer of the Device operation.
- The initial pin set contains exactly one pin on each pin layer of the Device 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.

An example of where the fill-in algorithm would be successful follows. Assume you have this five-pin device defined in the rule file:

```
DEV Q base coll(C) base(B) emitt(E) sub(S) emitt(E2)
```

A four-pin device instance with a pin on layer coll, a pin on layer emitt, a pin on layer base, and a pin on layer sub is classified as a Q device because it satisfies the three conditions for fill-in (this assumes there are no other device statements that could provide an exact match). The pins E and E2 in the Device statement are considered shorted together for this instance.

**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.
Recognition Logic

- 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 the example, the fill-in algorithm does not find a provisional match because of the combinations covered by the four statements.

Bad Device Reporting

Bad devices in hierarchical LVS are reported in the circuit extraction report file. Bad devices in flat LVS are reported in the LVS report.

In addition to the seed layer, location, and cell name, the following information is provided for bad devices:

- The reason why the device was classified as bad.
- The number of relevant objects found to interact with the seed shape.
Relevant objects are pins, auxiliary shapes, and TEXT MODEL LAYER text objects. Pin shapes on the same net are counted and reported as one pin unless BY SHAPE is specified. Multiple auxiliary shapes on the same layer are counted and reported as one. When too many pins or TEXT MODEL LAYER text objects are present, only the first such extra item is counted and reported. For TEXT MODEL LAYER text objects, if Layout Preserve Case No is specified or the Layout Preserve Case statement is omitted, then text strings which differ only by case are treated as identical.

- A list of the relevant objects found to interact with the seed shape.
  
  For each such object, pertinent information is provided, such as layer name, net name or number, location, and text string for text objects. Objects determined to be extra are indicated as such.

- Possible element names and optional model names for the device.
  
  This is a list of *element_name* and optional *model_name* values from all Device operations in the rule file that use that device recognition layer.

There are five possible reasons for classifying a device as bad:

1. (No matching Device operation).
   
   This usually means that some pins or auxiliary shapes are missing.

2. (Too many pins).
   
   Too many pins were found.

3. (Too many matching Device operations).
   
   This usually means that pins on one or more pin layers were present in insufficient numbers, and more than one distinct Device operation could be matched by the pin fill-in algorithm.

4. (Too many TEXT MODEL LAYER texts).
   
   More than one TEXT MODEL LAYER text object intersects the seed shape.

5. (Too many TEXT PROPERTY LAYER texts on same layer).
   
   More than one TEXT PROPERTY LAYER text objects on the same layer intersect the seed shape.

**Calibre LVS-H** — Hierarchical device recognition provides detailed information about bad devices when they are present in the design. Extracted SPICE netlists do not contain this information, but an annotation to the circuit extraction report (*lvs_report_name*.ext, see “Circuit Extraction Report” on page 13-48) is added.

Examples:

*WARNING: BAD DEVICE on layer Pseed at location (3,-8.5) in cell D*
Flat LVS — Bad devices are reported in the “Bad Devices” section of the LVS report. The formatting is the same as for the circuit extraction report in hierarchical LVS, discussed previously. See the following examples.

(Each of the following is a shape that appeared as the device layer of one or more Device rules. However, the set of pins that it touched did not correspond to a Device rule.)

BAD DEVICE on layer Pseed at location (-1.5,-7.5)  
(No matching DEVICE operation).  
Found 3 interaction(s):  
Pin on layer poly, net 61 at location (-1.5,-7.25)  
Pin on layer SD, net 137 at location (-1,-7.25)  
Auxiliary shape on layer C_aux1 at location (-1.35,-7.5)  
Possible Element Names: C MP C(dual_auxes)

BAD DEVICE on layer Pseed at location (-1,-5)  
(No matching DEVICE operation).  
Found 1 interaction(s):  
Pin on layer poly, net 63 at location (-1,-4.5)  
Possible Element Names: C MP C(dual_auxes)

BAD DEVICE on layer Pseed at location (4,-7.5)  
(Too many TEXT MODEL LAYER texts).  
Found 4 interaction(s):  
Pin on layer poly, net 72 at location (4,-7.25)  
Pin on layer SD, net 148 at location (4.5,-7.25)  
Text "m2" on layer metal at location (4.123,-7.375)  
Text "m3" on layer metal at location (4.123,-7.125)  
(extra model text)  
Possible Element Names: C MP C(dual_auxes)
BAD DEVICE on layer Pseed at location (3,-5.75)
   (Too many pins).
   Found 4 interaction(s):
   Pin on layer poly, net 69 at location (3,-5.625)
   Pin on layer SD, net 18 at location (3,-5.625)
   Pin on layer SD, net 142 at location (3,-5.125)
   Pin on layer SD, net 143 at location (3,-4.5) (extra pin)
   Possible Element Names: C MP C(dual_auxes)

BAD DEVICE on layer Pseed at location (6,-7.5)
   (No matching DEVICE operation).
   Found 3 interaction(s):
   Pin on layer poly, net 77 at location (6,-7.25)
   Pin on layer SD, net 158 at location (6.5,-7.25)
   Auxiliary shape on layer C_aux1 at location (6.15,-7.5)
   Possible Element Names: C MP C(dual_auxes)

BAD DEVICE on layer badseed at location (6,-6.5)
   (No matching DEVICE operation).
   Found 0 interaction(s):
   No interacting pins, auxiliary shapes or texts were observed.
   Possible Element Names: always_bad_dev

Hierarchical Device Recognition

Considerations for hierarchical device recognition are discussed under “Hierarchical Device Recognition” on page 12-2.

Property Computation

This section introduces and distinguishes some of the ways in which the computation of properties can be specified. It covers these major topics:

- Default Property Computations
- User-Defined Property Computation
- Efficiency Considerations
- Debugging Property Computations

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 computed by Calibre xRC.

After LVS recognizes a device instance, it 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, if specified.
For each Device operation there are three choices for the property specification:

- No specification given.
- One or two numbers (used by built-in for recognition devices only).
- A short program written in the property computation language (see “User-Defined Property Computation” on page 10-14).

For certain reserved element names, some of these choices are invalid.

The element names D, C, R, MN, MP, MD, and ME represent known (built-in) 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:

- 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 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 results if you use a list of floating point numbers as a property specification with any element name other than those identified previously.

- 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 following section, “Default Property Computations,” discusses the default property computations available for element names C, R, MN, MP, MD, and ME. The sections “User-Defined Property Computation” 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 Table 10-1. The default traceable properties and the corresponding property names are shown in the third and fifth columns, respectively.

You select the default computation by either omitting the property specification and its square brackets completely in the DEVice statement, or by supplying the appropriate number of numeric parameters between brackets [ ].

<table>
<thead>
<tr>
<th>name</th>
<th>default traceable properties</th>
<th>property parameters</th>
<th>property names</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN</td>
<td>transistor width, length</td>
<td>effective_width_factor</td>
<td>W, L</td>
</tr>
<tr>
<td>MP</td>
<td>transistor width, length</td>
<td>effective_width_factor</td>
<td>W, L</td>
</tr>
<tr>
<td>MD</td>
<td>transistor width, length</td>
<td>effective_width_factor</td>
<td>W, L</td>
</tr>
<tr>
<td>ME</td>
<td>transistor width, length</td>
<td>effective_width_factor</td>
<td>W, L</td>
</tr>
<tr>
<td>D</td>
<td>diode area, perimeter</td>
<td>(none)</td>
<td>A, P</td>
</tr>
<tr>
<td>C</td>
<td>capacitor capacitance</td>
<td>area_cap, perim_cap</td>
<td>C</td>
</tr>
<tr>
<td>R</td>
<td>resistor resistance</td>
<td>resistivity</td>
<td>R</td>
</tr>
</tbody>
</table>

For each Device operation containing a reserved element name from Table 10-1, you can supply appropriate property parameters. An exception is the diode, which requires no parameters. If the property parameters are omitted, they default to zero. The effect of a zero parameter value is discussed in the following subsections.

The default internal property computation code for these devices is given under “Examples” of the Device section in the SVRF Manual.

The following examples illustrate the four possible property_parameter setups for reserved element names. Namely, the cases of diode, resistor, capacitor, and MOS transistor. The following subsections cover the meaning of the parameters for each of these classes in turn.

- **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. Because no parameters are required for this computation, no property parameters (in square brackets) are specified in the operation.
**Resistors** — Resistance is computed for resistors. One parameter is needed— the resistivity in units of 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 you can set it 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:

\[
\text{resistance} = r \times \left( \frac{L}{W} \right)
\]

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, you may have to use appropriate layer operations to alter the geometry prior to computation and then to specify the computation using the built-in property 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 in the Device operation, 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 be non-negative. Specifying either value as zero has a small effect on execution time, because it eliminates 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 you can set it 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:

\[
\text{capacitance} = (ca \times \text{area} + cp \times \text{perimeter})
\]

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. You can specify one parameter—the effective width (weffect) constant—in the Device statement. 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 you specify a non-zero effective width (weffect) parameter in the Device operation.

- The weffect is zero: W and L are computed as discussed previously and are delivered as the width and length properties of the device.
- The weffect 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 simple rectangle has I = 0 and a right-angle L-shaped polygon has I = 1. More formally, I can be computed using the following formula:

  \[
  I = \frac{S}{90}
  \]

  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 of the shape turns clockwise, add the number of degrees of turn to S. At each vertex where the boundary turns counter-clockwise, 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:

$$W - \text{weffect} \times I \times L$$

If $W \leq L$, then $W$ is left unadjusted, and $L$ is set to the value:

$$L - \text{weffect} \times I \times W$$

The resulting values of $W$ and $L$ are returned as the width and length properties of the device.

**User-Defined Property Computation**

The **DEVice** operation allows you to define your own property computation for any devices. For any device, including built-in devices, if you choose to compute properties using your own algorithm, you assume responsibility for calculating *all* properties for that device.

The optional property computation section of the Device operation uses a built-in language. This language is described under “Device Property Computation Built-In Language” in the *SVRF Manual*.

The remainder of this section shows some examples of user-defined device property calculations and important things to consider when writing your own property calculations.

**Built-In Language Examples**

This section contains rule-file excerpts of property computations.

```verbatim
// Example rule file for Computing Device Parameters
// NRS, NRD, AS, AD, PD, PS, L, W

layer ipoly 4
layer diff 5
layer contact 6
layer metal1 8
layer pwell 10
ngate = ipoly AND diff
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, contact Mentor Graphics
//    Customer Support.

// Device Description Example 1

// Note, the auxiliary layers "diff" and "nsd_rs" are added
// to this first device description; use diff
// in the AS, AD, PS, PD property computations below to
// account for shared source/drain; use nsd_rs in the
// NRS, NRD property computations.

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
      // that 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, because 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 = \text{perimeter}(D) \times W / \text{perimeter}_\text{inside}(D, \text{diff})
\begin{verbatim}
//-----------------------------------------------------------
//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 = \text{average distance from gate to contact’s nearest edges} / \text{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 that have no edges that face the gate edge are not involved in the calculation.
//3. Calculations assume all contacts are equally sized.
\end{verbatim}

\begin{verbatim}
SUM\_S\_LENGTH = \text{perimeter}_\text{inside}(\text{nsd}_\text{rs}, S) - \text{perimeter}_\text{coincide}(\text{nsd}_\text{rs}, S)
COUNT\_S = \text{trunc}((\text{count}(\text{nsd}_\text{rs}) \times \text{perimeter}_\text{coincide}(\text{nsd}_\text{rs}, S) / \text{perimeter}_\text{coincide}(\text{nsd}_\text{rs}, G)) + 0.5)
IF\ (\text{COUNT\_S} \neq 0)
{\n   NRS = SUM\_S\_LENGTH / COUNT\_S / W / 2
}
ELSE
{\n   NRS = AS / (W \times W)
}
SUM\_D\_LENGTH = \text{perimeter}_\text{inside}(\text{nsd}_\text{rs}, D) - \text{perimeter}_\text{coincide}(\text{nsd}_\text{rs}, D)
COUNT\_D = \text{count}(\text{nsd}_\text{rs}) - COUNT\_S
IF\ (\text{COUNT\_D} \neq 0)
{\n   NRD = SUM\_D\_LENGTH / COUNT\_D / W / 2
}
ELSE
{\n   NRD = AD / (W \times W)
}
\end{verbatim}

\\begin{verbatim}
/////////////////////////////////////////////////////////////
//                 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, the auxiliary layer diff is added to this
//device description; 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>
[\n\begin{verbatim}
// property W, L, AD, AS, NRD, NRS, PD, PS
\end{verbatim}
\end{verbatim}
Device Recognition
Property Computation

//The line above is commented out until NRD and NRS computations are added
//to prevent syntax error upon compiling 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(D, diff)

//-----------------------------------------------------------
//This section of the property computations measures number of
//resistance squares in source and number of resistance
//squares in drain. Note, in the case where neither the source
//nor drain has any contacts,
//NRS = area of source / width / width
//That is, average length of AS = AS / W
//number of 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 illustrates two MOS transistors that share
//a common pin. The source pin of the top transistor is also

Calibre Verification User’s Manual 10-17
//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, perhaps to a netlist, then 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 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 as 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.
//Here is an example of a rule file that computes
//the device source/drain areas when the junction is
//electrically connected by multiple devices. It uses an
//auxiliary layer, "diff", to compute the total width
//dimensions of all devices that associate with a pin.
//A similar procedure can be used to compute source/drain
//perimeters.
layer poly 1 // polysilicon
layer diff 2 // diffusion
layer well 3 // well
layer metl 4 // metal
layer nimp 5 // n implant
layer subc 6 // substrate contact
layer cont 7 // contact cut

srcdrn = diff not poly
tran = diff and poly
ptran = tran not nimp
ntran = tran and nimp

connect metl poly srcdrn by cont
connect srcdrn nimp by subc
connect srcdrn well by subc

device mp ptran poly(g) srcdrn(s) srcdrn(d) well(b) <diff> [ 
  property w, l, as, ad, nrs, nrd
  w = perimeter_coincide( ptran, srcdrn ) / 2
  l = perimeter_outside( ptran, srcdrn) / 2
  as = area(s) * ( w / perimeter_inside( s, diff ) )
  ad = area(d) * ( w / perimeter_inside( d, diff ) )
  nrs = as / (w * w)
  nrd = ad / (w * w)
]

device mn ntran poly(g) srcdrn(s) srcdrn(d) nimp(b) <diff> [ 
  property w, l, as, ad
  w = perimeter_coincide( ntran, srcdrn ) / 2
  l = perimeter_outside( ntran, srcdrn) / 2
  as = area(s) * ( w / perimeter_inside( s, diff ) )
  ad = area(d) * ( w / perimeter_inside( d, diff ) )
]

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 this:

\[ C = \text{perim\_factor} \times \text{perim(device)} + \text{area\_factor} \times \text{area(device)} \]

To compute a proper value for C, the following seven questions must be answered:

1. In what units is \text{perim(device)} delivered?
2. In what units is \text{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 previous questions are answered as follows:

1. The value returned by perim(device) is 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 you may encounter a property such as “C = 5p”, where the capacitance is represented as 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. The “p” provides the appearance of picofarads while the schematic actually uses farads. Consequently, the property evaluation formula must use farads to compute the result.

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 meters 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/um. You must re-express this value in farads/meter before using it in the formula. For example:

   \[0.05 \text{ pf/um} = 0.05e+6 \text{ pf/m} = 0.05e-6 \text{ f/m} = 5e-8 \text{ f/m}\]

   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:

- 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 10-2 shows a sample listing of a value array.

<table>
<thead>
<tr>
<th>Index</th>
<th>Type</th>
<th>Name (as shown in debugging output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>property value</td>
<td>w</td>
</tr>
<tr>
<td>1</td>
<td>property value</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>property value</td>
<td>as</td>
</tr>
<tr>
<td>3</td>
<td>property value</td>
<td>ad</td>
</tr>
<tr>
<td>4</td>
<td>local variable</td>
<td>weffect</td>
</tr>
<tr>
<td>5</td>
<td>constant</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>data value</td>
<td>area(S)</td>
</tr>
<tr>
<td>7</td>
<td>data value</td>
<td>area(D)</td>
</tr>
<tr>
<td>8</td>
<td>data value</td>
<td>perimeter_coincide(G, diff)</td>
</tr>
<tr>
<td>9</td>
<td>data value</td>
<td>perimeter_inside(G, diff)</td>
</tr>
<tr>
<td>10</td>
<td>constant</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>data value</td>
<td>perimeter_outside(G, diff)</td>
</tr>
<tr>
<td>12</td>
<td>data value</td>
<td>bends(G)</td>
</tr>
</tbody>
</table>
Device Recognition

Property Computation

Table 10-2. Value Array Listing (cont.)

<table>
<thead>
<tr>
<th>Index</th>
<th>Type</th>
<th>Name (as shown in debugging output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>constant</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>temporary variable</td>
<td>temp1</td>
</tr>
<tr>
<td>15</td>
<td>temporary variable</td>
<td>temp2</td>
</tr>
</tbody>
</table>

After loading of the rule file, the application creates the array, sets all entries to zero, 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

Because you can generate property computations 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. For more
information on the value array and on the initiation and evaluation phases of computation, refer
to “Property Computation Structure” on page 10-21.

How the Compiler Optimizes

The rule file compiler attempts 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 the rule compiler can move to the initiation phase are those
that are instance-independent. That is, computations that do not depend on data from a
recognized instance get moved. Because 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: \texttt{X\_LOCATION()}
- All PERIMETER functions: \texttt{Y\_LOCATION()}
- COUNT(): \texttt{PIN\_NET()}
- BENDS(): \texttt{INSTANCE()}

All other functions, including NAMED\_NET(), deliver instance-independent data.

The compiler uses two methods to move 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 moves fewer lines to initiation to avoid splitting an IF/ELSE computation between phases.

Once the compiler has moved the initial instance-independent portion of the program to the initiation phase, the compiler 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, the compiler is currently not capable of determining if a reference to a local variable or a property variable is instance-independent, so all such references are treated as instance-dependent.

The following example illustrates some of the above concepts. In this example, you should assume that \texttt{P} and \texttt{Q} represent pin names, and that \texttt{normal\_adjustment} is a process variable.

```plaintext
[1] property ap, aq, inst
  2  power\_adjustment = 0.04 + normal\_adjustment
  3  power\_net = named\_net(“VDD”)
  4  if (pin\_net(\texttt{P}) == power\_net)
  5    pin\_adjustment = power\_adjustment
  6  else
  7    pin\_adjustment = normal\_adjustment
  8  ap = area(\texttt{P}) + pin\_adjustment
  9  aq = area(\texttt{Q}) + 2 * normal\_adjustment
 10  inst = instance()
11 ]
```

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 obtains these variables directly from the value array. Also, the statement:

\[
\text{subexpression } 2 \times \text{normal调整ment}
\]

on line 10 is instance-independent. A variable “init1” is created in the value array to store the value of this subexpression. Line 11 is then treated as if it read

\[
aq = \text{area}(Q) + \text{init1}
\]

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

\[
\text{inst} = \text{instance()}
\]

had been moved from line 11 to just ahead of line 3, there would have been no instance-independent initial segment of the program. The only optimizations the compiler could perform would be on the two instance-independent subexpressions

\[
0.04 + \text{normal调整ment}
\]

and

\[
2 \times \text{normal调整ment}
\]

However, if you follow the information in the next section, you can ensure that 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 can ignore.

- **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 is performed only once in the initiation phase. Instead, if you intermix independent and dependent portions, the compiler still attempts to move independent subexpressions to the initiation phase, but it is not currently capable of moving entire assignment statements.

- **Use variables for constants** — You can assign constants to mnemonic variables in the initial instance independent section of your program without loss of performance. 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 \(\text{bend\_effect} = 0.5\) during initiation. Then during each evaluation, it accesses the \(\text{bend\_effect}\) 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 parentheses to get the compiler to notice certain instance-independent expressions. For example, consider the following assignment statement where var is a process variable:

\[ a = \text{area}(P) + 0.5 + \text{var} \]

The compiler does not find the instance independent subexpression \(0.5 + \text{var}\) in the above statement because it does additions from left to right by default. Thus, the expression is treated as if were parenthesized as follows:

\[ a = (\text{area}(P) + 0.5) + \text{var} \]

To optimize the original statement, you can introduce your own parentheses as follows:

\[ a = \text{area}(P) + (0.5 + \text{var}) \]

The compiler uses this to compute “\(0.5 + \text{var}\)” before adding in \(\text{area}(P)\). In this case, it recognizes \((0.5 + \text{var})\) as being instance-independent and computes it only once during initiation.

A second way to handle this situation is to use the “variables for constants” method of the previous tip by computing \(0.5 + \text{var}\) 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. This occurs before the property computation is performed 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 in a local variable. Actually, the assignment to a local variable would be an extra step and would slow the evaluation.

For example, the first of the following two programs takes more time to evaluate since it contains an extra assignment statement, which must be executed at evaluation time.
• **Avoid time-consuming geometry functions (if possible)** — Some of the geometric instance data functions are more time-consuming to compute than others. In some cases, you can rewrite an expression involving more time-consuming functions to use more efficient functions.

  o Functions that request information about geometry on the boundary of, or within the interior of the seed shape are the least time-consuming. For example, AREA(device_layer) or AREA_COMMON(device_layer, pin_layer) fall in this class.

  o Functions that require information about a single pin or auxiliary layer or about geometry lying strictly outside the seed shape are more time-consuming. For example, AREA(pin_layer) or PERIMETER_OUTSIDE(pin_name, device_layer) are in this class.

  o The most time-consuming functions are those that request 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.

Next is a list of the functions classified according to speed. In the list, dev_lay represents the device layer or the device as a pin, and pin_lay, pin_lay_1, pin_layN represent other pins or layers including auxiliary layers.

  o Fastest 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)

- Slower 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)

- Slowest 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 example illustrates the preceding classifications.

Suppose the devices to be recognized have a pin “P”, which is known to always lie strictly within the area of the seed shape on layer dev_lay. Therefore, the two functions shown next 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 faster to compute.

\begin{align*}
\text{AREA(P)} \\
\text{AREA_COMMON(P, dev_lay)}
\end{align*}

As another example, in Figure 10-1 A and B are two rectangular shapes that intersect to form a third rectangular shape D. In the device statement, D is the device shape, and A and B are pin shapes. In the property computation, the goal is to find the length of the vertical boundary at the left of the D shape. Either of the two expressions shown can do this, but the first involves the interaction of two pins and the second involves a pin and the device shape. Thus, the second is more efficient.
In some situations, you might have a choice of which layer to use for the seed layer. If so, consider using a seed layer whose shapes contain most of the other shapes involved so that the geometry on or within the seed shape applies 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. For more information on how the property computations are structured, refer to the section “Property Computation Structure” on page 10-21 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 and that there are some mistakes within the specification.

The first mistakes are likely to be errors detected by the rule file compiler. The rule file compiler generates 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. Errors can remain that cause a PROPERTY ERRORS section to appear in the LVS report. Some of these may be apparent, which you can find by studying the property specification according to the numbers produced. Others might not be so apparent. At this point, 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 consists of 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 not part of

```
L = perim_in(B,A)   //slower
L = perim_in(D,A)   //faster
```
the Device operation or the file itself. The line numbers are used in debugging output to identify which line of the program is responsible for each step in the analysis.

```plaintext
1 device mp(pmos) gate gate(G) diff(S) diff(D) base(B)
2 [  
3 property W, L, AS, AD
4 bend_effect = 0.5
5 AS = area(S)
6 AD = area(D)
7 W = (perimeter_coincide(G, diff) + perimeter_inside(G, diff)) / 2
8 L = perimeter_outside(G, diff) / 2
9 if (bends(G) > 0)
10 {  
11 if (W > L)
12 W = W - bend_effect * bends(G) * L
13 else
14 L = L - bend_effect * bends(G) * W
15 }  
16 ]
```

**Property Error Report**

Suppose the rule file containing the Device statement shown 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: 30u4%
```

Note that the property values for AD, AS, and W do not compare properly. If you cannot resolve the discrepancy in any other way, you turn attention to the computation and decide to use the DEBUG statement to get more information about the computation for this instance.

The first thing you 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 run of flat device recognition, the device instances are numbered consecutively beginning with zero. This numbering applies to all device types and subtypes. Thus, the “5” uniquely identifies the device instance you must debug.

**Debug Statement Placement**

The next step is to determine which Device operation in the rule file was responsible for generating instance 5. In this case, you know 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. Because the DEBUG statement specifically identifies instance 5, and because instance 5 is unique to only one of the rules, the evaluation computation is only traced for that one instance. However, the initiation computation is 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, add the DEBUG statement to the end of line 24. Recall that the DEBUG statement must precede the PROPERTY statement. The altered Device operation now appears as follows:

```plaintext
1 device   mp(pmos)   gate   gate(G)   diff(S)   diff(D)   base(B)
2          [debug 5
3          property W, L, AS, AD
4          bend_effect = 0.5
5          AS = area(S)
6          AD = area(D)
7          W = (perimeter_coincide(G, diff) +
                  perimeter_inside(G, diff)) / 2
8          L = perimeter_outside(G, diff) / 2
9          if (bends(G) > 0)
10          {
11              if (W > L)
12                  W = W - bend_effect * bends(G) * L
13              else
14                  L = L - bend_effect * bends(G) * W
15          }
16        ]
```

**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 operations that would cause a change in the number of devices recognized. This could alter the instance number of the device instance you 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 in the following example, interspersed with 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 are all 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 do not affect the course of future computations because they are 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 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” on page 10-22 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 number and (x,y) coordinates. Next are 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 previously. Note that most of the values in the array are no longer zero. This is because they contain values left over 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) = 7.199999999999975e-11
28: \( ad \) = area(D) = 5.599999999999978e-11
29: \( \text{temp1} \)
   = perimeter_coincide(G, Diff)
   + perimeter_inside(G, diff)
   = 6.399999999999976e-05 + 0
   = 6.399999999999976e-05
29: \( w \)
   = temp1 / 2
   = 6.399999999999976e-05 / 2
   = 3.199999999999998e-05
30: \( l \)
   = perimeter_outside(G, diff) / 2
   = 3.999999999999973e-06 / 2
   = 1.999999999999998e-06
31: \( w > l \)
   \( w \) = 3.199999999999998e-05
   \( l \) = 1.999999999999998e-06
   TRUE
33: \( w > l \)
   \( w \) = 3.199999999999998e-05
   \( l \) = 1.999999999999998e-06
   TRUE
34: \( \text{temp2} \)
   = bend_effect * bends(G)
   = 0.5 * 1
   = 0.5
34: \( \text{temp1} \)
   = temp2 * 1
   = 0.5 * 1.999999999999998e-06
   = 9.9999999999999928e-07
34: \( w \)
   = w - temp1
   = 3.1999999999999996e-05 - 9.9999999999999928e-07
   = 3.0999999999999996e-05

Values after:
0: \( w \) 3.0999999999999996e-05
1: \( l \) 1.999999999999998e-06
2: \( as \) 7.199999999999975e-11
3: \( ad \) 5.599999999999978e-11
4: \( \text{bend_effect} \) 0.5
5: 0.5 0.5
6: area(S) 7.199999999999975e-11
7: area(D) 5.599999999999978e-11
8: perimeter_coincide(G, Diff) 6.399999999999976e-05
9: perimeter_inside(G, diff) 0
10: 2 2
11: perimeter_outside(G, diff) 3.999999999999973e-06
12: bends(G) 1
13: 0 0
14: \( \text{temp1} \) 9.9999999999999928e-07
15: \( \text{temp2} \) 0.5
Device Recognition

Property Computation

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 that numerical errors due to roundoff or to the inexact representation of decimal fractions on a binary machine are more apparent. This often results in many trailing 9s in the output, but these do have value. For example, with only a few digits, it would not be apparent how the test “1.3 < 1.3” could return TRUE. However, with 17 digits, it becomes clear that “1.29999999999999996 < 1.29999999999999998” is 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 this example, 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 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\textit{n}, where \textit{n} is a small integer. In the example above, the statement on line 34 of the rules is:

\begin{equation*}
34 \ W = \ W - \text{bend\_effect} \times \text{bends}(G) \times L
\end{equation*}

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 does not show it, there is a second kind of temporary variable whose name is of the form init\textit{n}, where \textit{n} is a small integer. These variables 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.

The end of the debugging output is noted by the following line:

\begin{verbatim}
// END EXTRACTED DEVICE PROPERTY COMPUTATION DEBUG OUTPUT
\end{verbatim}
### 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. You can check these values against the layout to see if the proper data functions have been used.

In the example covered previously, 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, and so forth.

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 the operations used the same device layer, then both initiations would appear, followed by a mixture of evaluations for both types of device. This occurs because 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. Because each initiation or evaluation output begins by identifying the device by type, subtype, and rule file line number, it should be easy to identify the output. 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 because 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 a statement 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 the more elegant solution.

2. Rerun the design (or a portion of the design) in flat LVS.
## Property Specification Error Messages

### Table 10-3. Property Specification Error Messages

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPR1</td>
<td>A definition for this device name may not have a numeric parameter set specification: &lt;element name&gt;</td>
<td>The only element names that 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.</td>
</tr>
<tr>
<td>DPR2</td>
<td>The numeric parameter set specification for this device definition must contain a single number: &lt;element name&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR3</td>
<td>The numeric parameter set specification for this device definition must contain one or two numbers: &lt;element name&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR4</td>
<td>A statement was expected at this point (did you forget a comma?): &lt;syntax element&gt;</td>
<td>The syntax element displayed was found where a valid assignment statement, IF statement, or compound statement beginning with a left brace ( { ) was expected.</td>
</tr>
<tr>
<td>DPR5</td>
<td>This was found where the second number of a debug range was expected: &lt;syntax element&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR6</td>
<td>This was found where the first number of a debug range was expected: &lt;syntax element&gt;</td>
<td>A number must appear immediately following the DEBUG keyword and immediately following each comma in the DEBUG statement.</td>
</tr>
<tr>
<td>DPR7</td>
<td>This was found where a right brace, “}” was expected: &lt;syntax element&gt;</td>
<td>The syntax element shown appears where a closing brace was expected to terminate a compound statement.</td>
</tr>
</tbody>
</table>
Table 10-3. Property Specification Error Messages (cont.)

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPR8</td>
<td>This was found where a left parenthesis, “(” was expected:</td>
<td>A left parenthesis must follow the keyword IF and all function name keywords.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR9</td>
<td>This was found where a right parenthesis, “)” was expected:</td>
<td>The syntax element displayed could not be parsed in the current context, but a right parenthesis would be valid at this point.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR10</td>
<td>This was found where a relational operator such as “&lt;” was expected:</td>
<td>The expressions used in an IF test must contain relational tests possibly combined with logical operators to form compound tests.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR12</td>
<td>This was found where a comma was expected:</td>
<td>Commas must be used to separate items in lists and the arguments of functions.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR13</td>
<td>This was found where the keyword PROPERTY was expected:</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR14</td>
<td>This was found where a property identifier was expected:</td>
<td>A property identifier must immediately follow the keyword PROPERTY and each comma in the PROPERTY statement.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR15</td>
<td>This property identifier is declared twice:</td>
<td>Each property identifier may appear only once in the list of the PROPERTY statement.</td>
</tr>
<tr>
<td></td>
<td>&lt;identifier&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR16</td>
<td>This was found where a property or variable name was expected:</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
<tr>
<td>DPR17</td>
<td>This was found where a layer or pin name was expected:</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>&lt;syntax element&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Table 10-3. Property Specification Error Messages (cont.)

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPR18</td>
<td>This was found where a right bracket, “[” was expected: &lt;syntax element&gt;</td>
<td>The property specification appears to end just prior to the syntax element shown, but no closing right bracket was found there.</td>
</tr>
<tr>
<td>DPR19</td>
<td>this was found where an expression representing a numeric value was expected: &lt;syntax element&gt;</td>
<td>The syntax element shown was found where a constant, local variable, process variable, or numeric-valued function was expected.</td>
</tr>
<tr>
<td>DPR20</td>
<td>This process variable does not have a numeric value: &lt;variable&gt;</td>
<td>The process variable referenced at this point must have a numeric value, but this one does not.</td>
</tr>
</tbody>
</table>
| 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:  
  \[
  \text{if ( b == 0 )} \\
  \text{X = 1} \\
  \text{A = X}
  \]                                                      |
<p>| DPR22        | This was found where a pin name was expected: &lt;syntax element&gt;           | 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: &lt;variable&gt; | 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. |</p>
<table>
<thead>
<tr>
<th>Error Number</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPR24</td>
<td>Assignment to this process variable is not allowed: &lt;variable&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR25</td>
<td>This was found where a net name was expected: &lt;syntax element&gt;</td>
<td>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 (“ ”).</td>
</tr>
<tr>
<td>DPR35</td>
<td>This property was never assigned a value: &lt;property name&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR36</td>
<td>This property was not assigned a value in all cases: &lt;property name&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>DPR37</td>
<td>This function cannot have identical arguments: &lt;function name&gt;</td>
<td>The function shown requires two arguments, but they cannot be identical.</td>
</tr>
</tbody>
</table>
Table 10-3. Property Specification Error Messages (cont.)

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPR38</td>
<td>Both arguments of this function cannot be associated with the same layer: &lt;function name&gt;</td>
<td>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 that can be expressed in another way. For example, if A and B are pins on layer L, then PERIM_CO(A, B) would always be 0 (zero) since A and B would have to be disjoint, and PERIM_CO(A, L) would be the same as PERIM(A) since pin A lies on layer L.</td>
</tr>
</tbody>
</table>
Chapter 11
LVS Circuit Comparison

This chapter discusses various LVS concepts pertaining to circuit comparison. LVS comparison occurs after device recognition in LVS applications.

Data Flow in LVS

The inputs to Calibre LVS are an SVRF rule file, a layout netlist or database, and a source netlist or database. The LVS outputs that always occur include a run transcript and an LVS Report. If you run LVS-H with the -spice option, a hierarchical layout netlist is output. If you run flat LVS with the -nl option, a flat layout netlist is output. You can also output a Mask Results Database, a Mask SVDB Directory, and a short isolation database (LVS Isolate Shorts). Chapter 3 discusses the required inputs in detail, as well as the various ways of running LVS. Chapter 13 discusses LVS reports and results databases. Chapter 14 discusses SPICE syntax. Figure 11-1 shows the basic LVS data flow.

Figure 11-1. LVS Data Flow

The layout database can be GDSII, OASIS, CIF, SPICE, Cnet, binary, or ASCII. The latter three only work in flat LVS. The source database can be SPICE or Cnet.

LVS Comparison

The major sections of this chapter include:

- Component Types
Calibre LVS Circuit Comparison

LVS Comparison

- Naming Conventions
- Built-In Device Types
- Matching of Circuit Elements
- Device Reduction
- Device Filtering
- Nets
- Logic Gate Recognition
- Tracing Properties
- SPICE-Like Property Syntax

Calibre LVS applications compare electrical circuits from the specified source netlist and layout geometry. You can also do netlist-to-netlist comparisons in Calibre LVS-H. When the compared circuits are equivalent, 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, 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. You do not need to supply initial correspondence points (see “Initial Correspondence Points” on page 11-16 or the LVS Cpoint statement) between circuit elements, but if you do, the run time decreases.

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. The correspondence between elements can be used for cross-probing in Calibre RVE.

To ensure that the matching of different elements does not generate misleading results, internal techniques are applied. These internal techniques select the best solution among those derived from various methods at different stages of the program.

Discrepancies reported in the LVS report show the differences between the two circuits, which Calibre LVS treats 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 compare all possible views of the data. See “LVS Results” on page 13-1 for a description of results files.

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.
Note

In general, it is recommended that you run LVS in its default configuration for most comparison runs. This means you usually do not need to specify any LVS specification statements other than LVS Report, unless you have particular needs can only be met by changing the LVS defaults. A list of the LVS specification statement settings used in a run appears in the LVS report, including the defaults if you leave them unchanged.

A summary of all rule file specification statements that guide LVS is in the “LVS Specification Statements” section of the SVRF Manual.

Component Types

An 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.

- **SPICE netlist** — The component types of SPICE elements are described in section “General SPICE Syntax” on page 14-2.

The LVS Compare Case statement controls case sensitivity of LVS comparison of element names, model names, and pin names.

Component Subtypes

An 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.
LVS Circuit Comparison

Naming Conventions

- **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, this 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.

- **SPICE netlist** — The component subtypes of SPICE elements are described in the section “General SPICE Syntax” in Chapter 14.

See LVS Compare Case in the SVRF Manual for details about the handling of case-sensitivity of subtypes during circuit comparison.

Naming Conventions

This section discusses naming conventions for various circuit elements.

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 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 can specify user-defined device pin names. By user-defined here, it means for the purpose of device recognition. Pin ordering in the extracted netlist follows the order specified in the corresponding Device statement.

    Device types J, L, LDD, LDDD, LDDE, LDDN, LDDP, M, and V are treated as user-defined if you do not conform to the pin names specified in “Built-In Device Types” on page 11-7.

  - **Built-in devices** — These have default pin naming and ordering conventions and receive special processing by LVS.

    Device types C, D, MD, ME, MN, MP, Q, and R are built-in for both recognition and LVS comparison. They have a set of hard-coded pin names that must be used.

    Device types J, L, LDD, LDDD, LDDE, LDDN, LDDP, M and V are treated as built-in if you conform to the pin names specified in “Built-In Device Types” on page 11-7.
Arbitrarily named devices can be treated as built-in if you use the LVS Device Type specification statement and if you conform to standard pin naming and ordering conventions.

- **SPICE netlist** — Pin names of SPICE elements are described in section “General SPICE Syntax” in Chapter 14.

## 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.

- **SPICE netlist** — 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 equals (=) 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. Note that when the original SPICE netlist hierarchy is expanded, the names of nodes and elements that originate from lower levels of hierarchy can never qualify as user given names. This is because those names become hierarchical pathnames that contain ‘/’ characters.

The prefixes n$ and i$ by convention denote system-generated net and instance names in Mentor Graphics schematic databases. The / is used as delimiter to form hierarchical pathnames. 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.

- **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.

- **Mask layout** — Calibre LVS specifies a geometric layout port with the [Port Layer Text](#) and [Port Layer Polygon](#) specification statements in the rule file.

- **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, unless you specify LVS Globals Are Ports NO in the rule file.

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 either of the following conditions are met:

- 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.

Port names on cells that match the LVS Power Name and LVS Ground Name settings can propagate power and ground net names up and down the hierarchy into nets that are attached to these ports using the LVS Cell Supply statement.

### Built-In Device Types

Table 11-1 lists the built-in device types and their corresponding component type values. The section “Component Types” on page 11-3 shows how component types are determined in the source circuit and in the layout.

<table>
<thead>
<tr>
<th>Device</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOS N transistor</td>
<td>MN&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>CMOS P transistor</td>
<td>MP&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>NMOS enhancement transistor</td>
<td>ME&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>NMOS depletion transistor</td>
<td>MD&lt;sup&gt;a&lt;/sup&gt;</td>
</tr>
<tr>
<td>MOS generic transistor</td>
<td>M</td>
</tr>
<tr>
<td>CMOS LDD N transistor</td>
<td>LDDN</td>
</tr>
<tr>
<td>CMOS LDD P transistor</td>
<td>LDDP</td>
</tr>
<tr>
<td>NMOS LDD enhancement transistor</td>
<td>LDDE</td>
</tr>
<tr>
<td>NMOS LDD depletion transistor</td>
<td>LDDD</td>
</tr>
</tbody>
</table>
Calibre Verification User's Manual

LVS Circuit Comparison

**Built-In Device Types**

**Table 11-1. Built-in Device Types (cont.)**

<table>
<thead>
<tr>
<th>Device</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS LDD generic transistor</td>
<td>LDD</td>
</tr>
<tr>
<td>Resistor</td>
<td>Ra</td>
</tr>
<tr>
<td>Capacitor</td>
<td>Ca</td>
</tr>
<tr>
<td>Diode</td>
<td>Da</td>
</tr>
<tr>
<td>Bipolar transistor</td>
<td>QA</td>
</tr>
<tr>
<td>Jfet transistor</td>
<td>J</td>
</tr>
<tr>
<td>Inductor</td>
<td>L</td>
</tr>
<tr>
<td>Voltage source</td>
<td>V</td>
</tr>
</tbody>
</table>

*a built-in devices for both recognition and LVS comparison, having hard-coded pin names

Note that these device types are built-in with respect to the LVS circuit comparison subsystem. Some, but not all, of these device types are also built-in with respect to the device recognition subsystem and specifically the Device operation. The devices that are built-in for recognition are indicated with a superscript a in Table 11-1. Devices that are not built-in with respect to device recognition may be entered as user-defined devices in the rule file.

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.

To perform the special processing described in the following sections, LVS requires that the built-in devices conform to certain conventions regarding the number of pins and pin names. None of the special processing is performed for built-in devices that do not follow these conventions. See “Instance Pins and Pin Names” on page 11-4 for how pin names are determined in the source circuit and in the layout.

You can designate devices with arbitrary names as built-in devices by using the LVS Device Type specification statement. Such devices must conform to the pin naming conventions discussed in this section in order for these devices to participate in LVS processes like reduction, filtering, pin swapping, and so forth.

**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 11-2 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, and equivalent types specified in an LVS Device Type specification statement) 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. (The rule file Device definition syntax enforces this convention.) The optional pins may represent one or more bulk connections or they may be used for any other purpose. Table 11-2 lists the three required pin names:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS transistor gate</td>
<td>G or GATE</td>
</tr>
<tr>
<td>MOS transistor source</td>
<td>S or SOURCE</td>
</tr>
<tr>
<td>MOS transistor drain</td>
<td>D or DRAIN</td>
</tr>
<tr>
<td>optional pins</td>
<td>any names</td>
</tr>
</tbody>
</table>

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.

MOS devices M, LDDN, LDDP, LDDE, LDDD, and LDD that do not have the required pin names shown in Table 11-2 receive no special LVS comparison processing and are considered user-defined. Remember, pin ordering for user-defined devices is taken from corresponding Device statements in the rule file.

The following specification statements control special processing functions used by Calibre LVS to recognize, reduce, and filter MOS transistors:

- LVS Device Type
- 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 and equivalent types specified in an LVS Device Type specification statement) 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 11-3 lists the two required pin names:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>capacitor positive pin</td>
<td>POS or P</td>
</tr>
<tr>
<td>capacitor negative pin</td>
<td>NEG or N</td>
</tr>
</tbody>
</table>

Optional pins: any names

The following specification statements control special processing functions used by Calibre LVS to reduce capacitors:

- LVS Device Type
- 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 and equivalent types specified in an LVS Device Type specification statement) 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 11-4 lists the two required pin names:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>resistor positive pin</td>
<td>POS or P</td>
</tr>
<tr>
<td>resistor negative pin</td>
<td>NEG or N</td>
</tr>
</tbody>
</table>

Table 11-3. Capacitor Required Pin Names

Table 11-4. Resistor Required Pin Names
The following specification statements control special processing functions used by Calibre LVS to reduce resistors:

- LVS Device Type
- 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 and equivalent types specified in an LVS Device Type specification statement) 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 11-5 lists the two required pin names:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>diode positive pin</td>
<td>POS or P</td>
</tr>
<tr>
<td>diode negative pin</td>
<td>NEG or N</td>
</tr>
<tr>
<td>optional pins</td>
<td>any names</td>
</tr>
</tbody>
</table>

The following specification statements control special processing functions used by Calibre LVS to reduce diodes:

- LVS Device Type
- 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 and equivalent types specified in an LVS Device Type specification statement) 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. (The rule file Device definition syntax enforces this convention.) The optional pins can represent one or more substrate connections or they can be used for any other purpose. Table 11-6 lists the three required pin names:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q transistor collector</td>
<td>C</td>
</tr>
<tr>
<td>Q transistor base</td>
<td>B</td>
</tr>
<tr>
<td>Q transistor emitter</td>
<td>E</td>
</tr>
<tr>
<td>other pins</td>
<td>any names (fourth pin must be S to pass syntax check)</td>
</tr>
</tbody>
</table>

The following specification statements control special processing functions used by Calibre LVS to reduce and filter bipolar transistors:

- LVS Device Type
- LVS Reduce Parallel Bipolar
- LVS Filter Unused Bipolar
- LVS Filter
- LVS Reduce

Jfet Transistors

LVS performs the following for Jfets that conform to Table 11-7: 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:

Table 11-7. Jfet Transistor Required Pin Names

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>J transistor gate</td>
<td>G or GATE</td>
</tr>
<tr>
<td>J transistor source</td>
<td>S or SOURCE</td>
</tr>
<tr>
<td>J transistor drain</td>
<td>D or DRAIN</td>
</tr>
<tr>
<td>other pins</td>
<td>any names</td>
</tr>
</tbody>
</table>

If you specify J devices differently from what Table 11-7 shows, these devices are considered user-defined and receive no special comparison processing by LVS. Pin ordering for user-defined devices is taken from corresponding Device statements in the rule file.

**Inductors**

LVS performs the following for inductors that conform to Table 11-8: 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:

Table 11-8. Inductor Required Pin Names

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>inductor positive pin</td>
<td>POS or P</td>
</tr>
<tr>
<td>inductor negative pin</td>
<td>NEG or N</td>
</tr>
<tr>
<td>optional pins</td>
<td>any names</td>
</tr>
</tbody>
</table>

If you specify L devices differently from what Table 11-8 shows, these devices are considered user-defined and receive no special comparison processing by LVS. Pin ordering for user-defined devices is taken from corresponding Device statements in the rule file.

**Voltage Sources**

LVS performs the following for voltage sources that conform to Table 11-9: 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:

Table 11-9. Voltage Source Required Pin Names

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage source positive pin</td>
<td>POS or P</td>
</tr>
<tr>
<td>voltage source negative pin</td>
<td>NEG or N</td>
</tr>
<tr>
<td>optional pins</td>
<td>any names</td>
</tr>
</tbody>
</table>

If you specify V devices differently than what Table 11-9 shows, these devices are considered user-defined and receive no special comparison processing by LVS. Pin ordering for user-defined devices is taken from corresponding Device statements in the rule file.

X+ Devices

X+ devices are MOS transistors with component subtype that begins with the letter X or x, followed by at least one other character. Note that if the subtype consists of only one character, X, then the transistor is not an X+ device.

X+ devices may prevent the formation of logic gates, and they have a limiting effect on split-gate reduction (see “Split Gate Reduction” on page 11-24). In other respects, they behave as regular MOS transistors; for example, they are subject to LVS Reduce Parallel MOS, LVS Reduce Series MOS, and similar statements.

Rule file example:

```
DEVICE M(XP) gate poly sd sd well // X+ device.
DEVICE M(XABC) gate poly sd sd well // X+ device.
DEVICE M(X) gate poly sd sd well // Regular MOS device.
```

SPICE netlist example:

```
M1 1 2 3 4 XP $ X+ device.
M2 1 2 3 4 XABC $ X+ device.
M3 1 2 3 4 X $ Regular MOS device.
```

MS and MF Schematic Devices

MS and MF are special device types that are supported 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” on page 11-7. This means that you must use some property other than element or SPICE model, respectively, to specify the LVS component type for these instances. The phy_comp property is a suggested choice. See the section “Component Types” on page 11-3 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 internally tries to correct 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 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 matching pairs of nets, pairs of instances, and pairs of ports, which have identical User-Given Names in both the source and layout circuits, as initial correspondence points. 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. LVS trusts initial correspondence points and matches elements that form initial correspondence points, even if they differ from each other. You can also specify correspondence points in the rule file by using the LVS Cpoint specification statement.

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, LVS uses component subtypes to resolve ambiguities and 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, determine 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 that 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, you should name nets on interchangeable parts of the circuit. It is acceptable to name one element on every interchangeable part. You should always 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 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 and Series Device Reduction

For device configurations that allow both series and parallel reduction, you can control the type of device reduction that is performed using the LVS Reduction Priority specification statement. This statement allows you to specify whether parallel or series reduction is preferred. For example, a device configuration that includes two resistors connected in parallel with one pin floating can be reduced as parallel, between nets 1 and 2 as shown in Figure 11-2, or as series with an internal net 2.

In general, two parallel devices can also be reduced in series when the following conditions are met:

- The two pins which form the parallel connection are swappable with each other.
- The two pins which form the parallel connection are not swappable with any other pins.
- At least one of the nets connected to the parallel devices has no other connections.

**Note**

Depending on the options controlling the filtering of unused devices, the device pairs may be filtered as unused if they are reduced by one method but not by the other. In the previous example, if LVS Filter Unused Option RB is specified, the resistors are considered unused when they are reduced in parallel. However, if LVS Filter Unused Option RC is specified, the resistors are considered unused when they are reduced in series.

Parallel MOS Transistor Reduction

Calibre LVS can reduce a group of parallel MOS transistors to a single transistor. MOS transistors are component types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and any equivalent types indicated with the LVS Device Type specification statement. All transistors in the group must have the same component type, the same optional subtype, the same number of pins and the same pin names. All the gate, source and drain pins, as well as any optional pins, must be connected to the same nets respectively. The source and drain connections of MN, MP, ME, MD, M and equivalent devices may be swapped. 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 11-2.
Figure 11-3 shows an example of parallel MOS transistor reduction.

**Figure 11-3. Parallel MOS Transistor Reduction**

By default, the effective width and length values of the reduced transistor are computed as follows:

\[
W = \sqrt{P \times Q} \\
L = \sqrt{P / Q}
\]

where \(\sqrt{\cdot}\) is the square root function and

\[
P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \\
Q = W_1 / L_1 + W_2 / L_2 + \ldots + W_n / L_n
\]

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:

\[
AS = AS_1 + AS_2 + \ldots + AS_n \\
AD = AD_1 + AD_2 + \ldots + AD_n \\
PS = PS_1 + PS_2 + \ldots + PS_n \\
PD = PD_1 + PD_2 + \ldots + PD_n
\]

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. Figure 11-4 shows an example.
The **LVS Reduce Parallel MOS** specification statement controls parallel MOS transistor reduction.

Effective property values are computed in both layout and source. If you use default effective property computation, then to compute width, length, area of source, area of drain, perimeter of source, and perimeter of drain values, you must use the built-in property names “w”, “l”, “as”, “ad”, “ps”, or “pd”, 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 is traced for MP devices. If L is present in the input database, L is also available automatically for property computations. When parallel MOS device reduction occurs, effective W and L values are calculated for reduced MP devices and the Trace Property statement has valid W values to trace.

You can override the default effective property computation and specify other formulas, as described in the section “**Device Reduction Programs**” on page 11-35.

**Series MOS Transistor Reduction**

Calibre LVS can reduce a group of series MOS transistors to a single transistor. MOS transistors are component types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and any equivalent types indicated with the **LVS Device Type** specification statement. 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, M and equivalent devices, source and drain are equivalent and their polarity is immaterial. In LDDN, LDDP, LDDE, LDDD, LDD and equivalent devices, source and drain must alternate within the series; a source-to-source or drain-to-drain connection breaks 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” on page 11-64 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 11-2.

Figure 11-5 shows an example of series MOS transistor reduction.

![Figure 11-5. Series MOS Transistor Reduction](image)

By default, the effective width and length values of the reduced transistor are computed as follows:

\[
W = \sqrt{P / Q} \\
L = \sqrt{P \times Q}
\]

where \(\sqrt{\text{sqrt}}\) is the square root function and

\[
P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \\
Q = L_1 / W_1 + L_2 / W_2 + \ldots + L_n / W_n
\]

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 you use default effective property computation, then to compute width and length values, you must use the built-in property names “w” and “l”, respectively. You can use the Trace Property specification statement to specify otherwise.

You can override the default effective property computation and specify other formulas, as described in the section “Device Reduction Programs” on page 11-35.

**Semi-Series MOS Transistor Reduction**

Calibre LVS can reduce a group of semi-series MOS transistors to a single transistor. MOS transistors are component types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and any equivalent types indicated with the LVS Device Type specification statement.
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 11-6. 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, M, and equivalent devices, source and drain are equivalent and their polarity is immaterial.

In LDDN, LDDP, LDDE, LDDD, LDD, and equivalent devices, source and drain must alternate within the series; a source-to-source or drain-to-drain connection breaks the chain at that point. Bulk and optional pins, for all transistor types, are interchangeable if they are specified as logically equivalent. The section “Logically Equivalent Pins” on page 11-64 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 11-2.

Figure 11-6 illustrates this. The bypass net must not be connected to any devices outside of the respective “row” of the series-parallel structure.
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:

\[ W = \sqrt{P / Q} \]
\[ L = \sqrt{P \times Q} \]

Where \( \sqrt{\text{sqrt}} \) is the square root function and

\[ P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \]
\[ Q = L_1 / W_1 + L_2 / W_2 + \ldots + L_n / W_n \]

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 you use default effective property computation, then to compute width and length values, you must use the built-in property names “w” and “l”, respectively.

You can override the default effective property computation and specify other formulas, as described in “Device Reduction Programs” on page 11-35.

### Split Gate Reduction

Calibre LVS can reduce a split-gate structure to a single gate structure. Split gate reduction is controlled by the LVS Reduce Split Gates specification statement.

**Note**

Split gate reduction implies parallel MOS transistor reduction, even when Series MOS Transistor Reduction is not requested.

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 types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and any equivalent types specified with the LVS Device Type specification statement). 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 11-7. 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 (subject to the LVS Reduce Split Gates SAME ORDER option. See page 11-26). When logic gate recognition is disabled, then the order must be the same in all strings.

Normally, all the transistors in the split gate structure must have the same component type, the same number of pins, and the same pin names, and the same pin swap conditions. This can be changed with the LVS Reduce Split Gates MIX TYPES option (see page 11-27). Subtypes must be the same in each group of transistors that are collapsed together.

For example, in Figure 11-7, subtypes must be the same in each row; they can be different in different rows. If there are more than three pins, then all optional pins must be connected in parallel. That is, they must all be connected to the same nets, respectively.

In MN, MP, ME, MD, M, and equivalent devices, source and drain pins may be swapped. In LDD-type and equivalent 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 “Logically Equivalent Pins” on page 11-64.
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” on page 11-7.

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 11-7. Split Gate Reduction](image)

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 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:

\[
L = \sqrt{\frac{P}{Q}} \\
W = \sqrt{P \times Q}
\]

where \(\sqrt{\text{q}}\) is the square root function and

\[
P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \\
Q = W_1 / L_1 + W_2 / L_2 + \ldots + W_n / L_n
\]

where \(W_i\), \(L_i\) are the width and length of the \(i\)th transistor, respectively.

Effective width and length values are computed in both layout and source. If you use default effective property computation, then to compute width and length values, you must use the built-in property names “w” and “l”, respectively. You can use Trace Property specification statements to specify otherwise.

You can override the default effective property computation and specify other formulas as described in the section “Device Reduction Programs” on page 11-35.
Semi-split gate reduction — You can perform semi-split gate reduction by using the SEMI ALSO option to the LVS Reduce Split Gates statement.

Calibre LVS can reduce a semi-split gate structure in addition to 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 11-8 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 11-8. Reduce Split Gates Example**

Input order considerations — 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 11-9 shows the effect of the SAME ORDER option.

**Figure 11-9. SAME ORDER Option**

Error tolerances — The WITHIN TOLERANCE option of the LVS Reduce Split Gates specification statement indicates that split structures should not be reduced if they violate the tolerances in any pertinent LVS Split Gate Ratio specification statements. When WITHIN TOLERANCE is specified, split gate reduction does not occur if it would have resulted in a LVS Split Gate Ratio discrepancy.

Mixing component types — The MIX TYPES option of the LVS Reduce Split Gates specification statement specifies that a split gate structure may contain transistors with different component types, different numbers of pins, different pin names or different pin swappability. However, component types, numbers of pins, pin names and pin swappability, as well as component subtypes, must still be the same in each group of transistors that are collapsed together (that is, in each “row” of the split gate).

The presence of devices with different component types, numbers of pins, pin names or pin swappability anywhere in a split gate structure forces the SAME ORDER option for the entire structure.

Restrictions related to logic gate recognition — Split gate reduction obeys the following restrictions:

- Disabling logic gate recognition (LVS Recognize Gates NONE) forces the SAME ORDER option for split gate reduction, even when that option is not explicitly specified.

- The presence of devices with different subtypes anywhere in a split gate structure forces the SAME ORDER option for the entire structure, unless LVS Recognize Gates MIX SUBTYPES is specified.

- The presence of X+ (see “X+ Devices” on page 11-14) devices anywhere in a split gate structure forces the SAME ORDER option for the entire structure, unless LVS Recognize Gates XALSO is specified.
These restrictions are all intended to prevent arbitrary results. Without the SAME ORDER option, the order of inputs in the final reduced structure may be inherited from any one of the original transistor strings that form the split gate. This is acceptable if the transistors forming the reduced structure subsequently form logic gates (because inputs to logic gates are logically equivalent). However, this may lead to arbitrary results when logic gates are not formed, hence, the restrictions.

**Parallel Bipolar Transistor Reduction**

Calibre LVS can reduce a group of parallel bipolar transistors (device type Q and any equivalent types indicated with the LVS Device Type specification statement) 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. The section “Logically Equivalent Pins” on page 11-64 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 11-6. Figure 11-10 shows an example of parallel bipolar transistor reduction.

![Figure 11-10. Parallel Bipolar Transistor Reduction](image)

By default, the area of the reduced transistor is computed as follows:

\[ A = A_1 + A_2 + \ldots + A_n \]

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:

\[ W = \sqrt{P \times Q} \]
\[ L = \sqrt{P / Q} \]

where \( \sqrt{ } \) is the square root function and

\[ P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \]
\[ Q = W_1 / L_1 + W_2 / L_2 + \ldots + W_n / L_n \]

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 you use default effective property computation, then to compute width and length values, you must use the built-in property names “a”, “w”, and “l”, respectively. You can use Trace Property specification statements to specify otherwise.

The default effective property computation can be overridden, and other formulas can be specified as described in the section “Device Reduction Programs” on page 11-35.

Series Capacitor Reduction

Calibre LVS can reduce a group of serially connected capacitor devices (device type C and any equivalent types indicated with the LVS Device Type specification statement) 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 breaks the chain at that point, unless they are specified as logically equivalent. The section “Logically Equivalent Pins” on page 11-64 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 11-3.

Figure 11-11 shows an example of series capacitor reduction.

![Figure 11-11. Series Capacitor Reduction](image)

By default, the capacitance of the resulting device is computed as follows:

\[ C = \frac{1}{1/C_1 + 1/C_2 + \ldots + 1/C_n} \]

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 you use default effective property computation, then to compute the capacitance value, you must use the built-in property name “c.”

You can use Trace Property to override the default effective property computation and specify other formulas, as described in the section “Device Reduction Programs” on page 11-35.
Parallel Capacitor Reduction

Calibre LVS can reduce a group of parallel capacitors (device type C and any equivalent types indicated with the LVS Device Type specification statement) 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. The section “Logically Equivalent Pins” on page 11-64 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 11-3.

Figure 11-12 shows an example of series capacitor reduction.

Figure 11-12. Parallel Capacitor Reduction

By default, the effective capacitance, area, and perimeter values of the resulting device are computed as follows:

\[
C = C_1 + C_2 + \ldots + C_n \\
A = A_1 + A_2 + \ldots + A_n \\
P = P_1 + P_2 + \ldots + P_n
\]

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 you use default effective property computation, then to compute the capacitance, area, and perimeter values, you must use the built-in property names “c”, “a”, and “p.” You can use Trace Property specification statements to specify otherwise.

You can override the default effective property computation and specify other formulas, as described in the section “Device Reduction Programs” on page 11-35.
Series Resistor Reduction

Calibre LVS can reduce a group of serial resistors (device type R and any equivalent types indicated with the LVS Device Type specification statement) 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 breaks the chain at that point, unless they are specified as logically equivalent. The section “Logically Equivalent Pins” on page 11-64 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 11-4.

Figure 11-13 shows an example of series resistor reduction.

By default, the resistance value of the reduced transistor is computed as follows:

\[ R = R_1 + R_2 + \ldots + R_n \]

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:

\[
\begin{align*}
W &= \sqrt{P / Q} \\
L &= \sqrt{P * Q}
\end{align*}
\]

where \( \sqrt{ } \) is the square root function and

\[
\begin{align*}
P &= W_1 * L_1 + W_2 * L_2 + \ldots + W_n * L_n \\
Q &= L_1 / W_1 + L_2 / W_2 + \ldots + L_n / W_n
\end{align*}
\]

where \( W_i, L_i \) are the width and length of the \( i \)th resistor respectively.

The LVS Reduce Series Resistors specification statement controls series resistor reduction.

Effective resistance, width, and length values are computed in both layout and source. If you use default effective property computation, then to compute the resistance, width, and length values, you must use the built-in property names “r”, “w”, and “l.” You can use Trace Property specification statements to specify otherwise.

You can override the default effective property computation and specify other formulas, as described in the section “Device Reduction Programs” on page 11-35.
Parallel Resistor Reduction

Calibre LVS can reduce a group of parallel resistors (device type R and any equivalent types indicated with the LVS Device Type specification statement) 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. The section “Logically Equivalent Pins” on page 11-64 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 11-4.

Figure 11-14 shows an example of parallel resistor reduction.

Figure 11-14. Parallel Resistor Reduction

By default, the resistance value of the reduced transistor is computed as follows:

\[ R = \frac{1}{1/R_1 + 1/R_2 + \ldots + 1/R_n} \]

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:

\[ W = \sqrt{P \times Q} \]
\[ L = \sqrt{P / Q} \]

where \( \sqrt{\text{sqrt}} \) is the square root function and

\[ P = W_1 \times L_1 + W_2 \times L_2 + \ldots + W_n \times L_n \]
\[ Q = W_1 / L_1 + W_2 / L_2 + \ldots + W_n / L_n \]

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 you use default effective property computation, then to compute the resistance, width, and length
values, you must use the built-in property names “r”, “w”, and “l.” You can use Trace Property
specification statements to specify otherwise.

You can override the default effective property computation and specify other formulas, as
described in the section “Device Reduction Programs” on page 11-35.

**Parallel Diode Reduction**

Calibre LVS can reduce a group of parallel diodes (device type D and any equivalent types
indicated with the LVS Device Type specification statement) 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” on page 11-64 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 11-5.

Figure 11-15 shows an example of parallel diode reduction.

![Figure 11-15. Parallel Diode Reduction](image)

By default, the area and perimeter values of the reduced diode are computed as follows:

\[
A = A_1 + A_2 + \ldots + A_n \\
P = P_1 + P_2 + \ldots + P_n
\]

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 diode reduction.

Effective area and perimeter values are computed in both layout and source. If you use default
effective property computation, then to compute area and perimeter values, you must use the
built-in property names “a” and “p”. You can use the Trace Property specification statements to
specify otherwise.

The default effective property computation can be overridden, and other formulas can be
specified as described in the section “Device Reduction Programs” on page 11-35.
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, LDD, or equivalent devices specified in an LVS Device Type statement that have at least three pins with the standard names as specified in Table 11-4.

Placing groups of parallel MOS transistors in the source ensures that the layout consists of a specified number of transistors. If this is not a requirement, then you should use single transistors in the source.

You can disable this check with LVS Report Option F.

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 are 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 question mark (?) 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 because they are necessarily caused by missing values on original input devices.

Device Reduction Programs

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 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 )
]
```

The device reduction programming language is discussed under “Device Reduction Effective Property Computation Language” in the SVRF Manual.
Tolerance in Device Reduction

The device reduction statements listed 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
- LVS Reduce

It consists of two optional statements called TOLerance and TOLERance STRing, one for numeric properties and the other for string properties. The syntax is:

```
[TOLERance numeric_property_name tolerance_number
 [... numeric_property_name tolerance_number]]
[TOLERance STRing string_property_name
 [... string_property_name]]
```

For example:

```
LVS REDUCE PARALLEL MOS YES [  
   TOLERANCE L 0 W 0  
   TOLERANCE STRING str1 str2 str3  
]
```

The TOLERANCE and TOLERANCE STRING parameters may appear in any order.

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.

The TOLERANCE STRING keywords is followed by one or more `string_property_name` parameters, indicating string property names. No tolerance values are specified for string properties.

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 reduction tolerance section may be specified per device reduction program.

For a TOLERANCE specification, devices are not reduced together if they own a numeric property name, the property has different numeric 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 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 you specify several numeric_property_name tolerance_number pairs, LVS does the check for each property separately. LVS does not reduce devices if the difference in at least one property exceeds the tolerance specified for that property.

For TOLERANCE STRING specifications, devices are not reduced together if they own a string property name with different string values. Case sensitivity for comparisons is determined by the VALUES setting in the LVS Compare Case specification statement.

When more than one string_property_name is specified, the check is done for each string property separately; devices are not reduced if at least one string value is not equal to the others.

A property name cannot be declared in both a TOLERANCE and TOLERANCE STRING statement within the same device reduction program. For example, the following generates a compile error:

```
LVS REDUCE ... [
    TOLERANCE x 0
    TOLERANCE STRING x   // Error, duplicate 'x' declaration.
]
```

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 forth). In the first iteration, processing of TOLERANCE and TOLERANCE STRING statements is based on original property values specified in the input database. In subsequent iterations, processing of TOLERANCE and TOLERANCE STRING 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” on page 11-34. The treatment of the latter is described later in this section.

If a property used in a reduction TOLERANCE or TOLERANCE STRING statement is missing on a device, then the device participates in the reduction as if the TOLERANCE or TOLERANCE STRING statement were not present. In other words, with respect to the reduction TOLERANCE or TOLERANCE STRING statement, a device with a missing property value behaves as if the property value were 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 name is specified in a (numeric) TOLERANCE statement, but a string property with that name is found on a particular device instead, then the property is treated as missing for
the purpose of the TOLERANCE statement. Similarly, if a property name is specified in a TOLERANCE STRING statement, but a numeric property with that name is found on a particular device instead, then the property is treated as missing for the purpose of the TOLERANCE STRING statement.

If a property used in a reduction TOLERANCE or TOLERANCE STRING 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. Example:

- **Layout Instances With Undetermined Reduction TOLERANCE Properties:**

  Listed 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.

<table>
<thead>
<tr>
<th>instance</th>
<th>property</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2001 (MN)</td>
<td>(reduced instance) w: ?</td>
</tr>
<tr>
<td>M2014 (MN)</td>
<td>(reduced instance) w: ?</td>
</tr>
<tr>
<td>M2021 (MN)</td>
<td>(reduced instance) w: ?</td>
</tr>
</tbody>
</table>

**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]
```

The following example reduces parallel capacitors only when capacitance values are equal and values of the string property COLOR are equal as well.

```
LVS REDUCE PARALLEL CAPACITORS YES [
    TOLERANCE C 0
    TOLERANCE STRING COLOR
]```
Device Filtering

Calibre LVS allows you to filter out unused transistors during the LVS run. You could do this 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 the LVS Filter Unused Option specification statement.

Filtering Unused MOS Transistors

Calibre LVS can internally filter unused MOS transistors from source and layout circuits. Filtering is performed only for properly formed MOS transistors. Specifically, these are instances with component type MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD or equivalent devices specified in an LVS Device Type statement that have at least three pins with the standard names, as shown in Table 11-4.

The following configurations of transistors are filtered out:

1. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and equivalent transistors with floating source or drain pin.
2. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDD, and equivalent transistors with source, drain, and gate pins tied together.
3. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and equivalent 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, LDD, and equivalent 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, LDD, and equivalent transistors with source shorted to drain, and gate pin connected to a power net.
6. MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, and equivalent transistors with source shorted to drain, and gate connected to a ground net.
7. MP, LDDP, and equivalent transistors with gate pin connected to a power net.
8. MN, LDDN, and equivalent transistors with gate pin connected to a ground net.

Figure 11-16 shows examples of the previous filters, where x denotes a floating pin and o denotes a pin connected to other instances and/or ports.
Filtering Unused Bipolar Transistors

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 or equivalent device types that have at least three pins with the standard names as specified in Table 11-6.

Q transistors with base and emitter tied together are filtered out.

Unused Device Filtering

The LVS Filter Unused Bipolar specification statement controls unused bipolar transistor filtering.

Nets

This section discusses how various nets are handled during the LVS comparison phase.
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.

Floating Named Nets

Circuit comparison of Calibre LVS preserves, in most cases, floating nets with user-given names. These nets become initial correspondence points, just like non-floating nets with user-given names. In contrast, unnamed floating nets are removed during circuit transformation.

Named floating nets can be introduced in several ways, including device filtering and trivial pin removal. In addition, such nets can be preserved in the layout by setting the LVS Preserve Floating Top Nets statement to YES. For information related to the handling of trivial ports, refer to the LVS Ignore Trivial Named Ports statement in the Standard Verification Rule Format (SVRF) Manual.

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 the ports have user-generated names that are present in both circuits. This filtering is done because pass-through nets are often present in the layout, but they are rarely present in a schematic design. The LVS report lists pass-through layout nets and their ports.
Unattached Ports

If there are unattached ports in the layout, the hierarchical connectivity extractor indicates this in the run transcript and the circuit extraction report. See page 8-19 for more information. Such unattached ports are not processed by LVS-H and may not cause an INCORRECT result.

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 performs comparison 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), or equivalent device types.
- LVS forms LDD CMOS gates from transistors with component type LDDP (pullup) and LDDN (pulldown), or equivalent device types.
- LVS forms regular NMOS gates from transistors with component type MD (pullup) and ME (pulldown), or equivalent device types.
- LVS forms LDD NMOS gates from transistors with component type LDDD (pullup) and LDDE (pulldown), or equivalent device types.

Other series-parallel gates are formed from the previous types as well as component types M and LDD, or equivalent types. The LVS Device Type statement allows you to specify device types equivalent to the ones mentioned previously. To see how component types are determined refer to the section “Component Types” on page 11-3.

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” on page 11-7. 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. Examples of half-gates are pullup or pulldown sections of logic gates, serial up and serial down structures, series-parallel up and series-parallel down structures, and Sm* and SPm* structures. 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 of type SPxxx(expression) where xxx is a string of characters. Note that internal nets are matched in all gate types that are formed when the secondary keyword SIMPLE is specified.

The following are exceptions to the logic gate recognition process.

- 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 that serves as an initial correspondence point 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 be in the gate. LVS forms only half-gates in these cases to prevent false discrepancies involving subtypes and property values.

In Figure 11-18, LVS must make a choice between the mp(x) and mp(y) transistors. LVS does not form a complete NAND2 gate, but forms an SDW2 structure.

![Figure 11-18. LVS Logic Gate Selection](image)
Regular CMOS Gates

- **INV**
  
  CMOS inverter.

  ![Figure 11-19. INV](image)

- **NANDn**
  
  $n$-input CMOS NAND. LVS considers all input pins of a NANDn gate logically equivalent. In Figure 11-20, signals IN1, IN2, INn are all interchangeable.

  ![Figure 11-20. NANDn](image)

- **NORn**
  
  $n$-input CMOS NOR. LVS considers all input pins of a NORn gate logically equivalent. In Figure 11-21, signals IN1, IN2, INn are all interchangeable.
• **AOI\_n1\_n2\_\ldots\_nm**

CMOS and-or-invert consisting of $m$ And structures with $n1$, $n2$, $nm$ inputs each, respectively, leading to an Or-Invert structure.

LVS considers the input pins of each one of the And structures in an AOI gate logically equivalent. In Figure 11-22, signals A, B, and C may be interchanged. Signals D and E are also interchangeable.
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$_{n_1 n_2 \ldots n_m}$ always has $n_1 \geq n_2 \geq \ldots \geq n_m$. 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$_{n_1 n_2 \ldots n_m}$**

  CMOS or-and-invert consisting of $m$ OR structures with $n_1$, $n_2$, $n_m$ inputs each, respectively, leading to an AND-invert structure.

  LVS considers the input pins of each one of the OR structures in an OAI gate logically equivalent. In Figure 11-23, signals A, B, and C are all interchangeable. Signals D and E are also interchangeable.
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 \texttt{OAI\_n1\_n2\_ \ldots \_nm} always has \( n1 \geq n2 \geq nm \). 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 \texttt{LVS} \texttt{Recognize Gates SIMPLE} statement in the rule file. In which case, LVS instead forms separate structures of type \texttt{SUP} and \texttt{SPDW}.

- **SUPn** (serial up)

  \( n \)-input CMOS serial pullup. LVS considers all input pins of a SUPn gate logically equivalent. In Figure 11-24, signals IN1, IN2, INn are all interchangeable.
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. LVS considers all input pins of a SDWn gate logically equivalent. In Figure 11-25, signals IN1, IN2, INn are all interchangeable.

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\_\ldots\_nm** (serial-parallel up)

CMOS serial-parallel pullup. This is a series of \(m\) parallel groups consisting of \(n_1, n_2, \ldots, n_m\) transistors, respectively, leading to a power net.

LVS considers the inputs to the transistors in each parallel group logically equivalent. In Figure 11-26, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

![Figure 11-26. SPUP\_3\_2](image)

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\_\ldots\_nm** always has \(n_1 \geq n_2 \geq \ldots \geq n_m\). 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\_\ldots\_nm** (serial-parallel down)

CMOS serial-parallel pulldown. This is a series of \(m\) parallel groups consisting of \(n_1, n_2, \ldots, n_m\) transistors, respectively, leading to a ground net.

LVS considers the inputs to the transistors in each parallel group logically equivalent. In Figure 11-27, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.
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 $\text{SPDW}_n_{12...n}$ always has $n_1 \geq n_2 \geq n_m$. In Figure 11-27, they could have been placed below the group of two parallel MP transistors connected to D and E, respectively.

- **SMP$_n$, SMN$_n$, SM$_n$, S(TTT)$_n$**

  Series of $n$ MP, MN, M, or equivalent devices TTT, respectively.

  The string TTT stands for any device type classified as PMOS, NMOS or MOS with an LVS Device Type specification statement.

  LVS considers all input pins of a SMP$_n$, SMN$_n$, SM$_n$, or S(TTT)$_n$ gate logically equivalent, and the two output pins logically equivalent. In Figure 11-28, signals IN1, IN2, INN are all interchangeable, and signals OUT1 and OUT2 are also interchangeable.
Figure 11-28. SMPn, SMNn, SMn, S(TTT)n Series

- **SPMP\_n1\_n2\_...\_nm, SPMN\_n1\_n2\_...\_nm, SPM\_n1\_n2\_...\_nm, SP(TTT)\_n1\_n2\_...\_nm**

Serial-parallel structures of MP, MN, M, or equivalent TTT devices, respectively.

The string TTT stands for any device type classified as PMOS, NMOS or MOS with an LVS Device Type specification statement. Each structure is a series of \( m \) parallel groups consisting of \( n1, n2, nm \) transistors, respectively, connected between any two nets (other than power in the case of MP, or ground in the case of MN).

LVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In Figure 11-29, signals A, B, and C are all interchangeable, signals D and E are interchangeable, and signals OUT1 and OUT2 are interchangeable.

Figure 11-29. SPMP\_3\_2, SPMN\_3\_2, SPM\_3\_2, SP(TTT)\_3\_2
The order of the parallel groups in a SPMP, SPMN, SPM, or SP(TTT) gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The name suffixes \(_n1\_n2\_\ldots\_nm\) always have \(n1 \geq n2 \geq \ldots \geq nm\). 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), SP(TTT)(expression)**

High level serial-parallel structures consisting of MP, MN, M, or equivalent devices, respectively.

The string TTT stands for any device type classified as PMOS, NMOS, or MOS with an LVS Device Type specification statement. 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.
- \(expression \ast n\) 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.

In Figure 11-30, signals A and B are interchangeable, signals C and D are interchangeable, and signals OUT1 and OUT2 are interchangeable. The single transistor connected to E could be moved from the bottom to the top of the structure.
In Figure 11-31, 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.
LVS Circuit Comparison
Logic Gate Recognition

Figure 11-31. SPMN((3*1)+2)*(2+2)) Structure

Regular NMOS Gates

- **INV** — NMOS INVerter.

Figure 11-32. INV

- **NANDn** — n-input NMOS NAND.

LVS considers all input pins of a NANDn gate logically equivalent. In Figure 11-33, signals IN1, IN2, INn are all interchangeable.
**Figure 11-33. NANDn**

- **NORn** — \( n \)-input NMOS NOR.

  LVS considers all input pins of a NOR\( n \) gate logically equivalent. In Figure 11-34, signals IN1, IN2, IN\( n \) are all interchangeable.

**Figure 11-34. NORn**

- **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.

  LVS considers the input pins of each one of the Or structures in an OAI gate logically equivalent. In Figure 11-35, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.
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\(_{n1\_n2\_\ldots\_nm}\) always has \(n1 >= n2 >= nm\). In Figure 11-35, 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.

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.

- **SDW\(_n\)** — \(n\)-input NMOS serial pulldown.

  LVS considers all input pins of a SDW\(_n\) gate logically equivalent. In Figure 11-36, signals IN1, IN2, IN\(_n\) are all interchangeable.

LVS represents NMOS serial pulldowns as standalone gates when they are not contained in complete gates.
• **SPDW\_n1\_n2\_…\_nm** — 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.

LVS considers the inputs to the transistors in each parallel group logically equivalent. In Figure 11-37, signals A, B, and C are all interchangeable, and signals D and E are also interchangeable.

![Figure 11-37. SPDW\_3\_2](image)

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 an 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 11-37, 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, SME\(n\), S(TTT)\(n\)** — Series of \( n \) MD, ME, or equivalent TTT type devices.

The string TTT stands for any device type classified as DEPL or ENH with an LVS Device Type specification statement.

LVS considers all input pins of an SMD\(n\), SME\(n\), or S(TTT)\(n\) gate logically equivalent, and the two output pins logically equivalent. In Figure 11-38, signals IN1, IN2, IN\(n\) are all interchangeable, and signals OUT1 and OUT2 are also interchangeable.
• **SPMD\(_{n1\_n2\_…\_nm}\), SPME\(_{n1\_n2\_…\_nm}\), SP(TTT)\(_{n\_n1\_n2\_…\_nm}\)**

Serial-parallel structures of MD, ME, or equivalent TTT devices, respectively.

The string TTT stands for any device type classified as DEPL, or ENH with an LVS Device Type specification statement. Each structure is a series of \(m\) parallel groups consisting of \(n1, n2, nm\) transistors, respectively, connected between any two nets (other than power in the case of MD, or ground in the case of ME, or equivalent device types).

LVS considers the inputs to the transistors in each parallel group logically equivalent and the two outputs logically equivalent. In Figure 11-39, signals A, B, and C are all interchangeable, signals D and E are interchangeable, and signals OUT1 and OUT2 are interchangeable.
The order of the parallel groups in a SPMD, SPME, or SP(TTT) gate is also interchangeable. Gates that differ only in the order of their parallel groups are considered equivalent. The names SPMD_{n1 \_n2 \_\ldots \_nm} and SPME_{n1 \_n2 \_\ldots \_nm} always have \( n1 \geq n2 \geq \ldots \geq nm \). In Figure 11-39, 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), SP(TTT)(expression)**

High level serial-parallel structure consisting of ME, MD, or equivalent TTT devices, respectively.

The string TTT stands for any device type classified as DEPL, or ENH with an LVS Device Type specification statement. The series and parallel groups can be nested to an unlimited number of levels. For more information, refer to the description of SPMP(expression), SPMN(expression), SPM(expression), SP(TTT)(expression).

### 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.

When LVS Builtin Device Pin Swap NO is specified, it is possible for regular MOS devices (component types MN, MP, ME, MD and M) to have non-swappable source/drain pins. For the purpose of logic gate recognition, such devices are treated as LDD-type devices (component types LDDN, LDDP, LDDE, LDDD and LDD respectively) and they form logic gates according to the rules that govern LDD-type devices. Logic gate naming conventions are generally the same as for LDD gates, except that for non-voltage gates the generic naming convention is used; for example, \( S(TTT)n \).

LDD transistors form logic gates in the way the corresponding regular MOS transistors form gates with some extra conditions listed in the following paragraphs. CMOS-style gates are formed with LDDN, LDDP, and equivalent types, which replace MN and MP in regular gates. NMOS style gates are formed with LDDE, LDDD, and equivalent types, which replace ME and MD in regular gates. Other series-parallel gates are formed from the previously-mentioned types as well as component type LDD and equivalent types. Equivalent device types are specified with the LVS Device Type statement. 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 \_\ldots \_nm}
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, LDDD, or equivalent transistors.

4. The ground net must be connected to the source pins of the LDDN, LDDE, or equivalent transistors.

5. The output of an LDD voltage gate is called \( \text{output}(d) \), while the output of a regular MOS gate is called \( \text{output} \). The \( (d) \) means that the output net is connected to drain pins of LDD transistors.

Figure 11-40 shows a CMOS-style AOI\(_3\_2\) gate made with LDD-type transistors.
LDD Non-Voltage Gates

Non-voltage gates are gates that do not require power or ground nets. These gates include:

- SLDDP\_n
- SLDD\_n
- SPLDDN\_n1\_n2\_\ldots\_nm
- SPLDD\_{expression}
- SPLDD\_{expression}
- SLDDD\_n
- SPLDE\_n
- SPLDDE\_n1\_n2\_\ldots\_nm
- SPLDDE\_{expression}
- S(TTT)\_n
- SP(TTT)\_{expression}

The string TTT stands for any device type classified a LDD-type MOS device with an LVS Device Type statement. It can also stand for a regular MOS device type when LVS Buildin
Device Pin Swap NO is specified. For more information on the \textit{(expression)} notation, refer to the description of SPMP\textit{(expression)} and SPMN\textit{(expression)} on page 11-52.

The following conditions apply when forming LDD non-voltage gates.

1. Each parallel group of LDD-type transistors must have its 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 to 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\textit{(expression)} corresponds to SPME\textit{(expression)}.

4. An LDD-type non-voltage gate has two non-swappable outputs. One is connected only to source pins and is called \textit{output\textit{(s)}}. The other is connected only to drain pins and is called \textit{output\textit{(d)}}. Recall that the outputs of a regular MOS gate are both called \textit{output}.

Figure 11-41 shows a SLDDP3 gate.

![Figure 11-41. SLDDP3](image)

**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 an \textit{S} or \textit{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 \textit{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 \textit{output}. The other is called \textit{output\textit{(d)}} if it is connected to drain pins of LDD-type transistors or \textit{output\textit{(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. Figure 11-42 shows this gate.

**Figure 11-42. SPMN-LDDN(D)_3_1**

Excluding Transistors

X+ transistors (see “X+ Devices” on page 11-14) normally do not 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 most other respects, these transistors behave as regular MOS devices.

The optional keyword XALSO in the LVS Recognize Gates specification statement disables the special treatment of X+ transistors in logic gate recognition. It instructs LVS to include X+ transistors in the formation of logic gates.

Overriding Of Pin Swapping in Logic Gates

You can check the physical order of connections to logic gate inputs, which are normally considered logically equivalent, 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 section “Excluding Transistors.”
Overriding of Device Swapping In Logic Gates

You can check the physical order of parallel device groups, which are normally interchangeable in logic gates, 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 “Excluding Transistors” on page 11-63.

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” on page 11-42 for a special case of this capability. For other component types, there are several methods for designating logically equivalent pins.

LVS always considers pins that have identical names to be logically equivalent.

Default Pin Swapping for Devices

LVS considers source (S) and drain (D) pins of regular MOS transistors (component types MN, MP, ME, MD, M, and equivalent devices specified in an LVS Device Type statement) logically equivalent.

LVS considers pins of all resistor devices (component type `R` and equivalent devices specified in an LVS Device Type statement) logically equivalent.

POS and NEG pins of all capacitor devices are considered logically equivalent if LVS All Capacitor Pins Swappable YES is specified. Otherwise, other rules described in this section apply. Capacitor devices are component type `C` and any equivalent types indicated with LVS Device Type specification statements.

Default pin swap conditions (except for capacitors) can be disabled through the LVS Builtin Device Pin Swap statement.

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).

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 geometric database systems (for example, GDSII, OASIS, CIF, ACSII, and binary)
- SPICE
- CNET databases derived from SPICE

CNET databases derived from Calibre geometric 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 geometric database systems), and the source is a database system with no pin swappability information (specifically, SPICE or CNET derived from SPICE), 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 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 14-33 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. Note that, in hierarchical circuit comparison, both layout and source are represented in the form of SPICE netlists.

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 previous steps are performed after any pins have been discarded using the LVS Discard Pins By Device specification statement, if present.
- In the previous steps, all name comparisons are case-insensitive. In all following examples, 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 is applied to instances of mos5pin in both layout and source. LVS determines that pins S and D of mos5pin are swappable (because they are on the same layer) and uses this information in circuit comparison.

Example 2

Rule file:

```
DEVICE C(A) capl 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), previously. Its pins are swappable. C2 receives pin swappability from the Device C(B) operation according to step (1), previously. 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), previously. 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), previously. 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), previously. Its pins are not swappable (unless indicated swappable by other means). C2 receives pin swappability from the DEVICE C operation according to step (2), previously. Its pins are swappable. C3 receives pin swappability from the DEVICE C operation according to step (1), previously. 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), previously. 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 as 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 12-8) and pin swapping in memory cells and containing blocks (see “SRAM Bit-Cell Recognition” on page 12-9).
## 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” on page 11-17.

Properties are classified based on their names as specified, either by default or by a user-defined property calculation, in a rule file `Device` statement. Specifically, the properties 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, are calculated by default for built-in devices. This convention is used in the layout as well as in the source. These properties are made available for LVS comparison using the `Trace Property` statement.

---

**Note**

In general, you do not want to use the extracted SPICE netlist to choose the parameter names to trace. The device for which this is particularly important in the diode (element D). The SPICE netlister shows area as “AREA=n” and perimeter as “PJ=m”. In flat LVS with a geometric input database, you may be able to trace “AREA” and “PJ”; however, using these parameters for hierarchical LVS where the comparison is SPICE-to-SPICE causes problems. The reason is, the device recognition algorithm assigns the parameters “A” and “P” for the area and perimeter properties for tracing. So, you want to use A and P in your Trace Property statement.

`Trace Property` specification statements may override the built-in naming convention and may specify different property names in the source. For example, the statement:

```plaintext
TRACE PROPERTY MP(X) WIDTH W 0
```

implies that for elements of type MP(X), the width property in the source is WIDTH, while the layout.

### 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, LVS automatically reads L properties from
the input database for a particular device type and optional subtype 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 required to calculate effective values for the other. For example, if you specify your own formula for calculation of effective W that 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 are 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 next:

```plaintext
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 layout to SPICE using the previous set of statements. Consider the following cases:

- **Layout:** 3 MP devices in parallel
  - **Source netlist:** M1 1 2 3 4 P M=3
  - **Result:** Correct

- **Layout:** 3 MP devices in parallel
  - **Source netlist:** M1 1 2 3 4 P
    - **Source netlist:** M2 1 2 3 4 P
    - **Source netlist:** M3 1 2 3 4 P
  - **Result:** Correct (M values in SPICE default to 1)

- **Layout:** 3 MP devices in parallel
  - **Source netlist:** M1 1 2 3 4 P M=2
    - **Source netlist:** M2 1 2 3 4 P
  - **Result:** Correct (M=2 value in M1 is added to the default value M=1 in M2)

- **Layout:** 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:** 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 that the Device operation is not necessary if you compare SPICE to SPICE.

See also **LVS Spice Replicate Devices** in the *SVRF Manual*. 
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 parentheses. LVS uses this format 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 lowercase):

- **w**: MOS width
- **l**: MOS length
- **r**: Resistor resistance
- **c**: Capacitor capacitance
- **a**: Diode area
- **p**: Diode perimeter

The “(parameter)” 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 parentheses 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.

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).
Calibre LVS-H is a fully-hierarchical LVS application. Calibre LVS-H maintains the database hierarchy and exploits this hierarchy to reduce processing time, memory use, and LVS discrepancy counts.

You can invoke Calibre LVS-H from the shell command line with the -spice and -hier command line options. You can also invoke it from Calibre Interactive. 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. The command line options are discussed under “Calibre LVS/LVS-H” on page 3-13.

LVS-H uses the same rule file as its flat counterparts. Generally there are no statements which need to be added or removed from your rule file. (Some limitations do exist, however, regarding the use of certain operations; those are described in the SVRF Manual, where applicable. 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

In general, there are no special caveats to be aware of regarding circuit extraction in LVS-H, with one exception: 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 fall into this category. You can circumvent this slowdown by specifying the Layout Base Layer specification statement in the rule file, which expands cells by one layer. Layout Base Layer is usually a useful and recommended statement to include in your rule file for all hierarchical applications.

Hierarchical LVS Comparison

This section describes the differences between flat and hierarchical LVS circuit comparison.

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.
Hierarchical LVS

Hierarchical LVS Comparison

There are two exceptions to this rule:

- You can specify device element names MP, MN, ME, and MD without model names in the rule file if the corresponding model name used in the source netlist is P, N, E, and D, respectively.

- You can specify device element names Q and D 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 command line switches, which contains two or more nets shorted together at a higher level of hierarchy. These nets are 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, that is, nets that are not connected to any other layout objects are not reported. LVS-H may report isolated nets that were originally connected to other layout objects but those other objects were removed or ignored during analysis.

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 limitation of these types of recognition, as well as the computation of two hierarchical properties.

- **BY NET device recognition** — Hierarchical device recognition does not promote devices based upon connectivity. 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) treats such 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 shapes 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. The named_net() function 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, shapes on derived layers that would otherwise be part of a lower level cell can be promoted to higher levels of hierarchy. If those shapes serve as seed or pin shapes in device recognition, or if they, in turn, derive seed or pin shapes, 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
- Polygon measurement: (Not) Area
- Sizing: Size without 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 you must use them, 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 is then done hierarchically and comparison is done flat.

- Two-layer Booleans: Xor
- Holes: Holes
- Polygon measurement: Perimeter, Vertex, (Not) Donut
- Sizing: Size with OVERLAP ONLY
- Shifting: Shift
- Extent: Extents (as opposed to Extent)
- Edge expansion: 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),
Hierarchical LVS

Cell Pushdown

The following operations produce flat or empty layers by definition. You cannot use them in hierarchical LVS.

- **Merging:** Merge
- **Dimensional check:** External, Internal, Enclosure

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 (see the SVRF Manual for details about hierarchical applications of Density)

Methodology: Pins, Ports, Topex

The following operations can generate only DRC error layers. They are not applicable to LVS.

- **Error-directed:** Drawn Acute, Drawn Angled, 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 you use an illegal type of Angle or Not Angle operation, Calibre LVS-H issues a message and quits.

**Cell Pushdown**

Cell pushdown is an optimization in the hierarchical database constructor that 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 Base Layer** (or Layout Top Layer) is specified in the rule file, then only top-layer cell placements are candidates for pushdown, and cell placements below the top layer are never pushed down. If Layout Base 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 Base Layer is
properly specified, the hierarchical database constructor in Calibre LVS-H does not push cells that contain devices down into other cells. This prevents undesired pushdown of devices 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 never pushes layout LVS Box cells down into other cells. This prevents undesired pushdown of LVS Box cells into hcells.

Top-layer cell placements are placements of cells that do not contain any layers other than metal routing layers and associated vias (or layers that are not specified in a Layout Base Layer statement).

These rules apply to Calibre LVS-H as well as to certain other hierarchical Calibre applications, including Calibre xRC (but they do not apply to Calibre DRC-H).

**Hcells**

The term *hcell* stands for *hierarchically corresponding cell*. Hcells in Calibre LVS-H are cells that exist in both the layout and source designs and that correspond (or are expected to exist and correspond). You specify hcells to improve LVS-H performance. *Note that hcells are not intended to enforce design hierarchy.*

Other Calibre applications may behave differently with respect to hcells than what is described in this section. For information on how Calibre DRC-H handles hcells, see “DRC Use of Hcells” on page 6-6.

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 and used as a single hcell list. All cell name pairs are treated equally, regardless of how they were established.

The LVS Exclude Hcell specification statement in the rule file prevents specified cells from serving as hcells under any circumstances.

Hcell specification is optional. If you do not specify hcells with any of the methods described, then the hcell list is empty. For information about using the Query Server to determine useful hcell candidates, see “Managing the Hcell List” in the Calibre Query Server Manual.

Be aware 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).
Hierarchical LVS
Hcells

Note
Use of the -automatch command line switch is generally discouraged. In most cases, a well-chosen hcell list delivers better performance.

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 one-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-one 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).

In general, an hcell list should be a relatively brief list of cells that correspond in layout and source, such that these cells are placed numerous times in the hierarchy. Providing an exhaustive list of all cells in the hierarchy is not needed, and is often counter-productive. This is because LVS-H can selectively flatten the hierarchy to improve performance in some cases like dense overlap removal. Cells that appear in hcell lists are not flattened, and therefore could impede the performance-improvement heuristics in LVS-H.

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 often runs faster.

1. Showing one-to-one and many-to-one correspondence of hcells.

   BUFF BUFF
   BITL BIT
   BITR BIT
   TOP TOP
   // ordinarily, you do not need to specify top-level cells as
   // corresponding

2. This flattens the design to the transistor level.

   TOP TOP

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.

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 later in this section. 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 one-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 (an n-to-one 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.

For information on expanding seed promotion cells, see LVS Expand Seed Promotions.

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 never expands 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). For example, you can specify cell correspondence with hcell lists 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 previous example, 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 flat 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. This may cause specific matching of pins, nets, and instances within cells at lower levels of hierarchy. Nevertheless, naming 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, logic gate, or injected component 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, logic gate, or injected component 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
Hierarchical LVS
Hierarchical Pins

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. Property values must be exactly equal and equality within tolerance values is not sufficient. Note that usually only the source is required to have equal values.

Complex swappability of injected component pins, such as multiple-level swappability or coupling of swappable pins, is not propagated to the hcell level.

Trivial pin swappability is not carried up from injected component pins where the properties or subtypes of respective transistors within the injected component are not symmetric. Only traced properties are considered.

Given a pair of corresponding hcells, pins in the pair are considered trivially swappable if they are trivially swappable in the source, even if the respective layout pins were not found to be swappable—with this exception: in the case of many-to-one hcell relations, if the “one” side is layout, then swappability is determined by the layout side. In any event, note that this is acceptable in the sense that no errors are missed; for example, if pins are found to be trivially swappable in the source but not in the layout, then the differences between source and layout are reported as discrepancies in the cell (except for differences in properties that are within tolerance values). See Figure 12-1.

**Figure 12-1. Trivial Pin Swappability**

In the figure, pins A, B, and C are swappable because they are connected directly to a NAND gate and nothing else. Pins E and F are swappable because they are connected to a resistor.

**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 next (hcells are corresponding cells used in hierarchical circuit comparison).
In Figure 12-2, the names POWER and GROUND were replaced with SUP1 and SUP2, respectively.

**Figure 12-2. SRAM Bit-Cell**

The names B, BN, W, SUP1, and SUP2 are for reference only; LVS does not require text. Pins B and BN must serve as pins of the cell that contains the structure. They cannot connect to any other devices in the cell. Net 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 than that, SUP1 and SUP2 may be any nets and do not have to be designated as power or ground supplies. Nets 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, and equivalent types specified in an LVS Device Type statement) as long as symmetry is observed.

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:

- the transistors designated m1 and m2.
- the transistors designated m3 and m5.
- 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. Property values must be exactly equal and equality within tolerance values is not sufficient. Note that, usually, only the source is required to have equal values, as described in the next paragraph. 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. Given a pair of corresponding hcells, pins B and BN in the pair are considered swappable if they are swappable in the source, even if the respective layout pins were not found to be swappable—*with this exception*: in the case of many-to-one hcell relation, if the “one” side is layout, then swappability is determined by the layout side. Note that this is acceptable in the sense that no errors are missed; for example, if the pins are found to be swappable in the source, but not in the layout, then the differences between source and layout are reported as discrepancies in the cell (except for differences in properties that are within tolerance values).

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 12-3, given that pins B and BN of cell X are swappable, then in cell Y pin B1 is swappable with BN1, B2 is swappable with BN2, and B3 is swappable with BN3.

**Figure 12-3. Carrying Pin Swappability Up the Hierarchy**

![Figure 12-3](image)

**High-Short Resolution**

High-short resolution is a process used in Calibre LVS-H 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 next.
• **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 12-1 shows some possibilities of how high-shorted pins are resolved.

### Table 12-1. High-shorted Pin Resolution Examples

<table>
<thead>
<tr>
<th>Original High-shorted Pins</th>
<th>Pins After High-short resolution</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>A B</td>
<td>Different names remain separate—even if unnamed pins are present.</td>
</tr>
<tr>
<td>A B 1</td>
<td>A B 1</td>
<td></td>
</tr>
<tr>
<td>A A A B B</td>
<td>A B</td>
<td>Pins are merged by name.</td>
</tr>
<tr>
<td>A A A B B 3 4 5</td>
<td>A B 3</td>
<td>Pins are merged by name, and unnamed pins are merged.</td>
</tr>
<tr>
<td>A 1</td>
<td>A</td>
<td>Only one name, therefore all pins are merged.</td>
</tr>
<tr>
<td>A A 1 2 3</td>
<td>A</td>
<td>Only one name, therefore all pins are merged.</td>
</tr>
</tbody>
</table>

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. Calibre may merge pins with different names if there is a difference between pin names in layout and source.
High-short resolution eliminates additional pins created by Sconnect operations, even if the high short involves texted pins. If an Sconnect operation causes an extra pin to be added to a cell, high-short resolution removes this pin in the case where more than one named pin is high-shorted to the stamped pin.

- **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 does not combine high-shorted pins with different user-given names if:
  - All names involved in the high-short are user-given.
  - All names involved in the corresponding high-short in the other design are also user-given
  - 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:

- Names that appear in both layout and source.
- Power names, in rule file order.
- Ground names, in rule file order.
- Names that begin with the letter ‘V’ or ‘v’ (‘V’ is commonly used in power supplies.)
- 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, 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
Hierarchical LVS

Logic Injection

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:

```verbatim
.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 rule file with the LVS Preserve Parameterized Cells specification statement.

Logic Injection

Logic injection is an algorithm in hierarchical circuit comparison that is designed to reduce memory consumption by replacing common logic circuits with new, primitive elements. For example, an SRAM bit structure consisting of six MOS devices is replaced by a single, primitive element. The original devices and related data structures are then removed from memory. The new, primitive elements are called “injected components.” In addition to memory consumption, execution time is often reduced.

The injected components retain all necessary information about the original devices: instance names, types, subtypes, and properties. When an injected component, or one of its original devices, is involved in a discrepancy, this information is included in the LVS report (see the section on the LVS report on page 12-15).

If cross-reference information is requested, corresponding original devices are written out.

Logic injection is done in every hcell (including the top-level cell) after instances in the cell are flattened and transformed, but before logic gates are recognized. Injection is done separately in layout and source. This means that LVS has to first create an internal representation of a cell with all its devices before it can inject logic and remove some of those devices. This fact has an important implication for memory savings provided by logic injection: logic injection is more efficient when the design contains several large blocks that serve as hcells. If there are no hcells at all, logic injection can reduce memory consumption by no more than half, because each half of the design—layout and source—must be represented in memory fully before injection takes
place. If the design is partitioned into several large blocks, memory savings from logic injection can be significantly higher.

When recognizing the circuits to inject, logic injection ignores user-given names inside hcells. Only user-given names in the top-level cell are preserved by logic injection and prevent injection if they appear on internal nets of injected circuits. Automatic detection of swappable hcell pins is more efficient in Calibre LVS since pin swappability, once detected by logic injection, is propagated to the pins of an hcell as appropriate.

The best-case candidates to benefit from logic injection, in its current form, are designs with significant embedded-memory content, where the embedded-memory blocks have similar sizes, serve as hcells, but are, in turn, processed mostly at the transistor level (that is, they contain no significant hcells inside). Note that if the design already has an adequate hcell list, then logic injection may provide no benefit; in some cases, there may even be a slight penalty in run time or memory consumption.

The maximum possible impact of logic injection on memory consumption is listed for every type of injected structure. The current theoretical limit is $4.7 \times$ savings (by injecting SRAM bit structures).

Logic injection operates only in hierarchical LVS.

Logic injection is controlled by the specification statement:

```
LVS INJECT LOGIC { NO | YES }
```

This statement may appear at most once. The default value is currently NO, and logic injection must be explicitly enabled by the user.

### Injected Components and the LVS Report

From the user’s perspective, components created by logic injection behave largely like logic gates. Injected components are reported in the LVS report in each cell where injection takes place. They are listed similarly to logic gates in the **NUMBERS OF OBJECTS AFTER TRANSFORMATION** section, and also in the **INFORMATION AND WARNINGS** section where counts of matched and unmatched instances are given. All injected components have predefined names that start with an underscore (\_).

Names of injected components are listed in this chapter for each type of injected logic. For every type of logic there may be several configurations of injected components, which differ by number of pins and configuration of substrate pins. These configurations will have different injected component names. Detailed descriptions of each type of injected logic contain names and descriptions of all such configurations. See page 12-20 for details on these configurations.

Unlike built-in or user-defined primitive devices, injected components with different names can be matched to each other, as long as they represent the same type of injected logic. When this happens, discrepancies are reported for individual transistors, as appropriate (for example,
substrate connection discrepancies). The source name is used for both injected components when reporting object counts in such cases, for example, in the NUMBERS OF OBJECTS AFTER TRANSFORMATION section.

A design may contain an hcell or a user-defined primitive device with the same name as an injected component.

**Injected Component Identification**

An injected-component instance is identified in the LVS report by its type, in parentheses ( ), followed by a list of individual device instances forming the injected component. Component types and optional subtypes of the individual devices are indicated as well. For example:

```
(_bitv)
```

**Devices:**
```
  x2/x2/m1  MP (P)
x2/x2/m2  MN (N)
x2/x1/m1  MP (P)
x2/x1/m2  MN (N)
x2/m4    MP (P)
x2/m3    MP (P)
```

This describes a _bitv structure formed by the six transistors shown.

**Injected Component Instance Pin Identification**

An injected-component instance pin is identified in the LVS report by the injected-component type, in parentheses ( ), followed by a colon (:), followed by the injected-component pin name. This is followed by a list of the individual device instances connected to that pin within the injected component. Each individual device instance is identified by the device instance name, followed by a colon (:), followed by the name of the device pin that leads to that injected-component pin. In MOS transistors, the string “s/d” may be shown in the pin name field. This stands for “either source or drain pin.” For example:

```
(_bitv):bl
x2/m4:s/d
```

This describes pin bl of an injected _bitv structure. The bl pin is formed by the either the s or the d pin of transistor x2/m4.

**Missing Injected Instance Discrepancy**

This type of discrepancy indicates a missing injected-component instance in the layout or source circuit. This is similar to the “missing instance” discrepancy, except that it is reported for instances of injected components. This discrepancy happens when all instances of a particular injected-component type in one of the circuits have been matched, and there are some unmatched instances of the same injected-component type left in the other circuit. The unmatched instances are reported as missing. When encountering this discrepancy, check the
numbers of instances after transformation reported in the overall comparison results or cell comparison results section of the report.

Report format: The injected component type is indicated, in parentheses, followed by a list of devices forming the injected component.

Graphic representation: The devices forming the injected component are highlighted.

Example:

2  (_bitv)  

** missing injected instance **

Devices:
  x1/x2/m1  MP(P)
  x1/x2/m2  MN(N)
  x1/x1/m1  MP(P)
  x1/x1/m2  MN(N)
  x1/m4   MP(P)
  x1/m3   MP(P)

The _bitv structure formed by the layout transistors shown is missing in the source. MP(P) and MN(N) are the component types and subtypes of the respective transistors.

Unmatched Injected Instance

An unmatched injected instance is an injected-component instance in the layout or source that cannot be matched to a corresponding injected component in the other circuit and cannot be classified as any of the available discrepancy types.

Report format: The injected component type is indicated, in parentheses, followed by a list of devices forming the injected component.

Graphic representation: The devices forming the injected component are highlighted.

Example:

(_bitv)  

** unmatched injected instance **

Devices:
  x1/x2/m1  MP(P)
  x1/x2/m2  MN(N)
  x1/x1/m1  MP(P)
  x1/x1/m2  MN(N)
  x1/m4   MP(P)
  x1/m3   MP(P)

The _bitv structure formed by the layout transistors shown could not be matched. MP(P) and MN(N) are the component types and subtypes of the respective transistors.
Logic Injection and Gate Recognition

Logic injection and gate recognition, in general, are unrelated transformations. They can be enabled and disabled separately, and serve different purposes: gate recognition allows for swappability of signals and structures that are logically equivalent, but topologically different; gate recognition does not save any memory. Logic injection, on the other hand, saves memory, but does not allow for any swappability that would not exist otherwise. However, logic injection interacts with gate recognition in the following ways:

- When logic injection replaces groups of original devices by new, injected components, the original devices are removed and are not involved in gate recognition. For example, if inverters are injected, INV gates will not be formed.

- When logic injection creates components having pins that are logically swappable, but topologically not swappable, such as input pins of a NAND gate, the swappability of the pins of the injected component depends on the state of gate recognition. If both logic injection and gate recognition are enabled, pins will be swappable. If logic injection is enabled but gate recognition is disabled, pins will not be swappable. Therefore, logic injection never allows swapping of any signals that otherwise would not be swappable.

- When logic injection is followed by gate recognition, injection of “voltage gates,” such as inverters, requires that power and ground nets be present. Groups of MOS devices that form an inverter, but do not connect to power and ground, are not replaced by injected components. This is done to prevent logic injection from interfering with recognition of complex gates. However, if gate recognition is disabled, logic injection injects any configuration of devices with correct topology, for example, inverters not connected to power and ground. In some cases, such injection could be ambiguous; these cases are detected, and logic injection disabled for such device configurations to avoid false errors.

Internal Net Matching

Internal nets in injected circuits are nets that are not connected to the pins of the injected component. These nets are removed from the design, along with the original devices. However, unlike the original devices, the internal nets are not matched, and are not reported in cross-reference databases.

If an internal net has a user given name that appears in both layout and source, logic injection of the corresponding instance is suppressed and the net name is preserved.

Logic Injection and Pin Swappability

Some of the circuits injected by logic injection have intrinsic topological symmetry, for example, SRAM bit structures. For such circuits, components created by logic injection have swappable pins. As a result, primitive devices with different component types, subtypes, and properties can be matched to each other, if such matching is required by the connectivity. That
is, asymmetry in types, subtypes, and properties does not prevent pin swappability of injected components.

Once matching of circuit elements is complete, LVS verifies types, subtypes, and properties of the original devices, and reports discrepancies if they differ; thus, logic injection causes no loss of information.

When matching of swappable pins of an injected component is topologically ambiguous, LVS attempts to resolve the ambiguity using information about device subtypes and properties of original devices that form the component. This is consistent with the behavior of LVS without logic injection.

Detailed descriptions of every type of injected logic, beginning on page 12-20, contain information on pin swappability for each type.

**Trivial Pin Swappability**

In certain cases, information about logical equivalence of pins can be carried up from an injected-circuit level to respective pins of a containing hcell. The respective hcell pins are then logically equivalent, or “swappable,” as well.

Specifically, hcell pins that are connected directly to swappable pins of an injected component within the hcell are swappable, provided that they are not connected to anything else within the hcell. Exception: complex swappability of injected component pins, such as multiple-level swappability, or coupling of swappable pins, is not propagated to the hcell level. Trivial pin swappability is not carried up from injected component pins where the properties or subtypes of respective transistors within the injected component are not symmetric.

---

**Note**

Trivial pin swappability from relevant injected components, such as bit and bit core structures, is carried up.

**Changes in Calibre Transcript**

When logic injection is enabled, the transcript changes in several ways. First, for each cell, the injection step is reported as it is done, as follows:

```
Logic Injection. CPU TIME = 0 REAL TIME = 0
```

Second, after the last cell undergoes logic injection, an injection summary is printed out in the transcript, which reports how many devices of different kind were injected.

**Device Types**

In the following description, CMOS P-type transistors are component type MP and equivalent types indicated with LVS Device Type specification statements. CMOS N-type transistors are
component type MN and equivalent types indicated with LVS Device Type specification statements.

**Naming Convention**

Injected component names follow this general naming convention:

- **_xxxv** — Voltage. Injected component of type xxx consisting of 3-pin or 4-pin MOS devices with substrate pins connected to the main power/ground supply nets in the component.

- **_xxxb** — Bulk. Injected component of type xxx consisting of 4-pin MOS devices with substrate pins not connected to the main power/ground supply nets of the component.

**Injection Types**

This section discusses injected logic structures.

**Bit Structure Core**

A two-inverter core of an SRAM bit structure is recognized and injected, but only if it did not form part of a full bit structure:

![Figure 12-4. _bitcorev Structure](image_url)

Requirements for successful logic injection:

- In **Figure 12-4**, the names BL, BR, SUP1, and SUP2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram.

- In each bit core, there must be only one “power” net (marked SUP1 in the diagram) and one “ground” net (marked SUP2 in the diagram). Configurations with different power or ground nets used by the two inverters comprising the bit structure core are rejected. The power and ground nets SUP1 and SUP2 may be any nets, and do not have to be named or designated as power or ground supplies.
- Devices M1 and M3 must be CMOS P-type transistors; devices M2 and M4 must be CMOS N-type transistors. Devices M1 and M3 may have different component types as long as they are CMOS P-type transistors. Devices M2 and M4 may have different component types as long as they are CMOS N-type transistors.

- All four MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, all four must have one. Substrate pins of all P-type transistors must be connected to one net, and the same is true for substrate pins of all N-type transistors. Substrate pins may be connected to the power and ground nets marked SUP1 and SUP2, but do not have to be.

Pin swappability:

Pins BL and BR are always swappable. Asymmetries, such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to BL and BR are ambiguous, ambiguity resolution examines types, subtypes, and properties of devices inside the injected bit core structure.

Possible configurations:

_bitcorev — 4-pin bit core (BL, BR, SUP1, SUP2), 3-pin or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively. (This configuration is shown in Figure 12-4.)

_bitcoreb — 6-pin bit core (BL, BR, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked, but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only bit cores) memory consumption is reduced by 2.5×. For comparison, if the bit core could be made into an hcell, memory consumption is reduced by 2.8×.

**Bit Structure**

A standard SRAM bit structure is recognized and injected.
Requirements for successful logic injection:

- In Figure 12-5, the names BL, BR, W, SUP1, and SUP2 are for reference only; no text is actually required by LVS. The internal nets marked 1 and 2 must not be connected to any other devices. The nets marked W, BL, BR, SUP1, and SUP2 may be connected to other devices.

- In each bit structure, there must be only one “power” net (marked SUP1 in the diagram) and one “ground” net (marked SUP2 in the diagram). Configurations with different power or ground nets used by the two inverters comprising the bit structure are rejected. The power and ground nets SUP1 and SUP2 may be any nets, and do not have to be named or designated as power or ground supplies.

- Devices M1 and M3 must be CMOS P-type transistors, devices M2 and M4 must be CMOS N-type transistors, devices M5 and M6 can be either CMOS P-type or CMOS N-type transistors (and do not have to be of the same type). Devices M1 and M3 may have different component types as long as they are CMOS P-type transistors. Devices M2 and M4 may have different component types as long as they are CMOS N-type transistors.

- All six MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, all six must have one. Substrate pins of all P-type transistors must be connected to one net, and the same is true for substrate pins of all N-type transistors. Substrate pins may be connected to the power and ground nets marked SUP1 and SUP2, but do not have to be.

- In some cases, grouping of MOS devices into bit structures is topologically ambiguous or may appear so to the logic injection algorithm. If logic injection were to take place, and an ambiguous bit structure chosen differently in layout and source, false errors would be reported. To prevent this, potentially ambiguous configurations are
recognized, and logic injection is disabled for them. As a result, certain bit structures may not be injected even though they fit the other criteria described previously.

Pin swappability:

Pins BL and BR are always swappable. Asymmetries, such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to BL and BR are ambiguous, ambiguity resolution examines types, subtypes, and properties of devices inside the injected bit core structure.

Possible configurations:

_bitv — 5-pin bit structure (BL, BR, W, SUP1, SUP2), 3-pin or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively. (This configuration is drawn in Figure 12-5.)

_bitb — 7-pin bit structure (BL, BR, W, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only bit structures) memory consumption is reduced by $4.7 \times$. For comparison, if the bit structure could be made into an hcell, memory consumption would be reduced by $6 \times$.

Bit Rows

Rows of SRAM bit cells with shared BL and BR pins are recognized and injected. This is known as a *metainjected* structure because it contains the bit structure, which is itself injected.

Figure 12-6. _bitrow2v Structure

Requirements for successful logic injection:
Hierarchical LVS
Logic Injection

- In Figure 12-6, the names BL, BR, W0, and W1 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram.

- In the entire structure there must be only one “power” net and one “ground” net for all bit cells. Configurations with different power or ground nets used by different bit cells comprising the row are rejected. The power and ground nets may be any nets and do not have to be named or designated as power or ground supplies.

- In the entire structure there must be at most one substrate net for each type of MOS device (PMOS and NMOS).

- Pin swapping of individual bits in the row is allowed; BL and BR pins of bit cells will be swapped as necessary when the row is recognized. Errors are reported for structures chosen differently in layout and source. To prevent this, potentially ambiguous configurations are recognized, and logic injection is disabled for them.

- Currently, bit row injection is limited to identical bit cells: the corresponding MOS devices in each bit cell (after swapping BL and BR pins if necessary) must have identical device type, subtype, and properties.

Pin swappability:

Pins BL and BR are always swappable. Word pins W0, W1, and so forth are always swappable. Asymmetries, such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to BL and BR or W0, W1, and so forth are ambiguous, ambiguity resolution will examine types, subtypes, and properties of devices inside the injected structure.

Possible configurations:

_bitrowNv — (N+4)-pin row of N bit cells (BL, BR, W0, W1, … W(N-1), SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_bitrowNb — (N+6)-pin row of N bit cells (BL, BR, W0, W1, … W(N-1), SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked, but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only bit structures) memory consumption is reduced by about $16\times$, depending on the length of the row.
Inverters

An inverter is recognized and injected, but only if it does not form part of a more complex structure, such as a bit structure:

![Figure 12-7. _inv Structures](image)

Requirements for successful logic injection:

- In Figure 12-7, the names IN, OUT, SUP1, SUP2, SUB1, and SUB2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices, not shown in the diagram.

- Nets labeled SUP1 and SUP2 in the diagram above must be power and ground nets as specified by LVS Power Name and LVS Ground Name statements, unless LVS Recognize Gates NONE is also specified.

- Device M1 must be CMOS P-type transistor, device M2 must be CMOS N-type transistor.

- Both MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, the other one must have one as well. Substrate pins may be connected to the “power” and “ground” nets marked SUP1 and SUP2, but do not have to be.

- If LVS Recognize Gates NONE is specified, it is possible that grouping of MOS devices into inverters is topologically ambiguous or may appear so to the logic injection algorithm. If logic injection were to take place, and an ambiguous inverter chosen differently in layout and source, false errors would be reported. To prevent this, potentially ambiguous configurations are recognized, and logic injection is disabled for them. As a result, certain inverter structures may not be injected even though they fit the other criteria described previously.
Pin swappability:

None.

Possible configurations:

_invv — 4-pin inverter (IN, OUT, SUP1, SUP2), 3-pin or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_invb — 6-pin inverter (IN, OUT, SUP1, SUP2, SUB1, SUB2), 3-pin or 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pin of the P device is connected to net SUB1 and substrate pin of the N device is connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only inverter structures) memory consumption is reduced by 1.5\times. For comparison, if the inverter structure could be made into an hcell, memory consumption is reduced by 1.6\times.

Inverter Chains

Chains of inverters are recognized and injected. This is known as a metainjected structure because it contains the inverter structure, which is itself injected.

Figure 12-8. _invx3v Structure

Requirements for successful logic injection:

- In Figure 12-8, the names IN and OUT are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram. Internal nets of the chain, marked 1, 2, and so forth cannot connect to any other devices.
- In the entire structure there must be only one “power” net and one “ground” net for all bit cells. Configurations with different power or ground nets used by different bit cells
comprising the row are rejected. The power and ground nets may be any nets and do not have to be named or designated as power or ground supplies.

- In the entire structure there must be at most one substrate net for each type of MOS device (PMOS and NMOS).
- Currently inverter chain injection is limited to identical inverters: the corresponding MOS devices in each inverter must have identical device type, subtype, and properties.

Pin swappability:

None.

Possible configurations:

_invxNv — 4-pin chain of N inverters (IN, OUT, SUP1, SUP2), 3-pin or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_invxNb — 6-pin chain of N inverters (IN, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1, and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked, but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only bit structures), memory consumption is reduced by about 20\(\times\) for chains of 32 inverters.

**Series Gates**

Series gates are recognized and injected, unless they form a part of some other injected structure, such as NAND or NOR gates, or can potentially form a part of a complex gate, and gate recognition is enabled through the LVS Recognize Gates statement in your rule file. Gates with an arbitrary number of input pins are recognized and injected. In the diagram below, there is a 2-input series gate with no substrate pins.
Requirements for successful logic injection:

- In Figure 12-9, the names IN1, IN2, OUT1, and OUT2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram. The internal nets of the gate must not be connected to any other devices.

- All devices forming the gate must be CMOS transistors of the same type (P-type or N-type). In addition, if gate recognition is enabled, all devices must have the same component type and subtype.

- All MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, the others must have it as well. Substrate pins of all transistors must be connected to one net. Substrate pins may be connected to one of the output nets marked OUT1 and OUT2, but do not have to be.

Pin swappability:

- If gate recognition is disabled, the entire series gate can be flipped upside-down, that is, pins OUT1 and OUT2 can be swapped, but at the same time input pins must also be swapped pair-wise: IN1 with INn, IN2 with INn-1, and so forth.

- If gate recognition is enabled, all input pins IN1 … INn are swappable. Output pins OUT1 and OUT2 are also swappable, and can be swapped independently from input pins.

Asymmetries such as different component types, subtypes, properties, or substrate connections do not prevent pin swappability (but are checked after matching is completed). If nets connected to input pins are ambiguous, the ambiguity resolution step examines component types, subtypes, properties, and substrate connections of devices inside the injected gate.

Possible configurations:

- _smn2v — series gate with n input pins IN1 … INn, and two output pins OUT1 and OUT2 (n+2 pins altogether) formed from 3-pin and 4-pin MOS devices with substrate pins of all transistors connected to one net, which is either OUT1 or OUT2. All devices are CMOS P-type transistors.

- _smn2b — series gate with n input pins IN1 … INn, two output pins OUT1 and OUT2, and a substrate pin SUB (n+3 pins altogether) formed from 4-pin MOS devices with
substrate pins of all devices connected to net SUB. All devices are CMOS P-type transistors.

_smnnv — similar to _smpnv, but all devices are CMOS N-type transistors.

_smnnb — similar to _smpnb, but all devices are CMOS N-type transistors.

_supnv — _smpnv with OUT1 or OUT2 pin connected to a POWER net.

_supnb — _smpnb with OUT1 or OUT2 pin connected to a POWER net.

_sdwnv — _smnnv with OUT1 or OUT2 pin connected to a GROUND net.

_sdwnb — _smnnb with OUT1 or OUT2 pin connected to a GROUND net.

Note: SUB is a “second class” pin and normally does not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Memory savings are a function of the size of series gates found. Under optimal conditions (many large hcells of approximately equal size containing only series gates) memory consumption is reduced by 1.6× for 2-transistor series gates, and 2.1× for 4-transistor series gates. For comparison, if the series gates could be made into hcells, memory consumption would be reduced by 1.7× for 2-transistor series gates, and 2.2× for 4-transistor series gates.

Parallel Gates

Parallel gates are recognized and injected, unless they form a part of some other injected structure, such as NAND or NOR gates, or can potentially form a part of a complex gate, and gate recognition is enabled through the LVS Recognize Gates statement in your rule file. Gates with an arbitrary number of input pins are recognized and injected. In the diagram below, there is a 2-input parallel gate with no substrate pins.

**Figure 12-10. _pmn2v, _pmp2v, _pup2v, _pdw2v Gates**

Requirements for successful logic injection:

- In Figure 12-10, the names IN1, IN2, OUT1, and OUT2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram.
• All devices forming the gate must be CMOS transistors of the same type (P-type or N-type).

• All MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, the others must have it as well. Substrate pins of all transistors must be connected to one net. Substrate pins may be connected to one of the output nets marked OUT1 and OUT2, but do not have to be.

• LVS may decide not to inject certain parallel gates if it determines that injecting them would not reduce overall memory consumption. For example, when an hcell contains only a small number of parallel gates of a particular size, LVS may decide to avoid injecting those gates in that hcell.

Pin swappability:

• All input pins IN1 … INn are swappable. Output pins OUT1 and OUT2 are also swappable, and can be swapped independently from input pins.

• Asymmetries such as different component types, subtypes, properties, or substrate connections do not prevent pin swappability (but are checked after matching is completed). If nets connected to input pins are ambiguous, the ambiguity resolution step examines component types, subtypes, properties, and substrate connections of devices inside the injected gate.

Possible configurations:

_pmpnv — parallel gate with n input pins IN1 … INn, and two output pins OUT1 and OUT2 (n+2 pins altogether) formed from 3-pin and 4-pin MOS devices with substrate pins of all transistors connected to one net, which is either OUT1 or OUT2. All devices are CMOS P-type transistors.

_pmpnb — parallel gate with n input pins IN1 … INn, two output pins OUT1 and OUT2, and a substrate pin SUB (n+3 pins altogether) formed from 4-pin MOS devices with substrate pins of all devices connected to net SUB. All devices are CMOS P-type transistors.

_pmnnv — similar to _pmpnv, but all devices are CMOS N-type transistors.

_pmnnb — similar to _pmpnb, but all devices are CMOS N-type transistors.

_pupnv — _pmpnv with OUT1 or OUT2 pin connected to a POWER net.

_pupnb — _pmpnb with OUT1 or OUT2 pin connected to a POWER net.

_pdwnv — _pmnnv with OUT1 or OUT2 pin connected to a GROUND net.

_pdwnb — _pmnnb with OUT1 or OUT2 pin connected to a GROUND net.

Note: SUB is a “second class” pin and normally does not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.
Memory savings:

Memory savings are a function of the size of parallel gates found. Under optimal conditions (many large hcells of approximately equal size containing only parallel gates) memory consumption is reduced by $1.4 \times$ for 2-transistor parallel gates, and $1.7 \times$ for 4-transistor parallel gates. For comparison, if the parallel gates could be made into hcells, memory consumption would be reduced by $1.5 \times$ for 2-transistor parallel gates, and $1.9 \times$ for 4-transistor parallel gates.

**Simple Gates**

Simple gates (NAND and NOR gates) are recognized and injected, unless they can potentially form a part of a complex gate, and gate recognition is enabled. The LVS Recognize Gates statement controls gate recognition and is independent of logic injection. Gates with arbitrary number of input pins are recognized and injected. In the diagram below, there is a two-input NAND gate and a two-input NOR gate, both with no substrate pins.

**Figure 12-11. _nand2v and _nor2v Gates**

Requirements for successful logic injection:

- In Figure 12-11, the names IN1, IN2, OUT, SUP1, and SUP2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram. The internal nets of the series half of the gate must not be connected to any other devices.

- When gate recognition is enabled, SUP1 in Figure 12-11 must be a power net and SUP2 in the diagram must a ground net, as specified by LVS Power Name and LVS Ground Name statements, respectively. When gate recognition is disabled, SUP1 and SUP2 can be arbitrary nets.

- Pullup devices (MP in the diagram) must be CMOS P-type transistors, and pulldown devices (MN in the diagram) must be CMOS N-type transistors.

- All MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, the others must have it as well.
Substrate pins of all P-type transistors must be connected to one net, and substrate pins of all N-type transistors must be connected to one net. Substrate pins may be connected to the power and ground nets marked SUP1 and SUP2, but do not have to be.

- When gate recognition is disabled, it is possible that grouping of MOS devices into gates is topologically ambiguous, or may appear so to the logic injection algorithm. If logic injection were to take place, and an ambiguous gate chosen differently in layout and source, false errors would be reported. To prevent this, potentially ambiguous configurations are recognized, and logic injection is disabled for them. As a result, certain gate structures may not be injected, even though they fit the other criteria described previously.

Pin swappability:

- When gate recognition is disabled, pins are not swappable.
- When gate recognition is enabled, all input pins IN1 … INn are swappable.

Asymmetries such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to input pins are ambiguous, the ambiguity resolution step examines types, subtypes, and properties of devices inside the injected gate.

Possible configurations:

_nandnv — NAND gate with \( n \) input pins IN1 … INn, and pins OUT, SUP1, and SUP2 (n+3 pins altogether) formed from 3-pin and 4-pin MOS devices with substrate pins of P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_nandnb — NAND gate with \( n \) input pins IN1 … INn, and pins OUT, SUP1, SUP2, B1, and B2 (n+5 pins altogether) formed from 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of P-type devices are connected to net B1 and substrate pins of N-type devices are connected to net B2.

_norrn — NOR gate with \( n \) input pins IN1 … INn, and pins OUT, SUP1, and SUP2 (n+3 pins altogether) formed from 3-pin and 4-pin MOS devices with substrate pins of P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_norrb — NOR gate with \( n \) input pins IN1 … INn, and pins OUT, SUP1, SUP2, B1, and B2 (n+5 pins altogether) formed from 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of P-type devices are connected to net B1 and substrate pins of N-type devices are connected to net B2.

Note: B1 and B2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Memory savings are a function of the size of NAND and NOR gates found. Under optimal conditions (many large hcells of approximately equal size containing only
NAND and NOR gates) memory consumption is reduced by $2.2 \times$ for 2-input NAND and NOR gates, and $3 \times$ for 4-input NAND and NOR gates. For comparison, if the NAND and NOR gates could be made into hcells, memory consumption would be reduced by $2.5 \times$ for 2-input NAND and NOR gates, and $3.4 \times$ for 4-input NAND and NOR gates.

**Multiplexer Structure**

Multiplexers based on a TGM structure are recognized and injected. Figure 12-12 shows the multiplexer structure:

![Figure 12-12. _mx2v Gate Multiplexer](image)

All nets labelled SEL and !SEL, respectively, are connected even when the connections are not shown. !SEL is an internal net, not a pin.

Requirements for successful logic injection:

- In Figure 12-12, the names IN1, IN2, SEL, !SEL, SUP1, and SUP2 are used for reference only; no text is actually required by Calibre LVS. These nets may be connected to other devices, not shown in the diagram. The internal net marked !SEL must not be connected to any other devices.

- Devices MP1, MP2, and MP5 must be CMOS P-type transistors, devices MN3, MN4, and MN6 must be CMOS N-type transistors. All devices may have different component types as long as they are CMOS P-type or N-type transistors.

- All six MOS devices can have no more than one substrate pin. It is not required that all MOS devices have the same number of substrate pins. However, all substrate pins of all P-type transistors must be connected to one net. This same condition applies for substrate pins of all N-type transistors. Substrate pins may be connected to the “power” and “ground” nets marked SUP1 and SUP2, but it is not required.

Pin swappability:

- None.
Hierarchical LVS
Logic Injection

Possible configurations:

_mx2v — 6-pin multiplexer structure (IN1, IN2, SEL, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_mx2b — 8-pin multiplexer structure (IN1, IN2, SEL, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports. Substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only multiplexers) memory consumption is reduced by 4 times. For comparison, if the multiplexer could be made into an hcell, memory consumption would be reduced by 4.9 times.

**XOR and XNOR Gates**

XOR gates based on a TGM structure are recognized and injected. Figure 12-13 and Figure 12-14 show two XOR gates based on a multiplexer _mx2 structure:

![Figure 12-13. _xra2v Multiplexer](image)
All nets labelled A, B, !A, !B, and !!B, respectively, are connected even when the connections are not shown. !A, !B, and !!B are internal nets, not pins.

Requirements for successful logic injection:

- In Figure 12-13 and Figure 12-14, the names A, B, OUT, SUP1, and SUP2 are used for reference only. No text is actually required by Calibre LVS. These nets may be connected to other devices, not shown in the figures. The internal nets marked !A, !B, and !!B must not be connected to any other devices.

- Devices MP1, MP2, MP5, MP7, and MP9 must be CMOS P-type transistors. Devices MN3, MN4, MN6, MN8, and MN10 must be CMOS N-type transistors. All devices may have different component types as long as they are CMOS P-type or N-type transistors.

- All eight or ten MOS devices can have no more than one substrate pin. It is not required that all MOS devices have the same number of substrate pins. However, all substrate pins of all P-type transistors must be connected to one net. The same condition applies for substrate pins of all N-type transistors. Substrate pins may be connected to the “power” and “ground” nets marked SUP1 and SUP2, but this is not a requirement.

Pin swappability:

- When gate recognition is disabled, pins are not swappable.

- When gate recognition is enabled, input pins A and B are swappable. Note that the internal structure of these XOR gates is highly asymmetric, which prevents any sensible relation between component instances and input ports. As a result, if gate recognition is enabled, input pins can be swapped independently of component instances and their possible asymmetries in subtypes or properties. If nets connected to input pins are ambiguous, ambiguity resolution will not examine devices inside the injected gate.
Possible configurations:

_xr2v — 5-pin XOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xr2b — 7-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

_xra2v — 5-pin XOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xra2b — 7-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports. Substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only XOR gates) memory consumption is reduced by 5.8 times for _xra2 structures and by 7 times for _xr2 structures. For comparison, if the XOR could be made into an hcell, memory consumption would be reduced by 7 or 9 times, respectively.

XOR and XNOR gates based on a series-parallel structure are recognized and injected. Figure 12-15 shows an XOR gate, _xor2v. With a trailing inverter connected to the output net, this structure is recognized instead as a XNOR gate, xnori2v.
An alternate configuration of inputs results in a different XNOR gate which does not require a trailing inverter. Figure 12-16 shows an XNOR gate, _xnor2v. With a trailing inverter connected to the output net, this structure is recognized as an XOR gate, _xori2v.

All nets labelled A, B, !A, and !B are connected even when the connections are not shown. !A and !B are internal nets, not pins. The trailing inverter consisting of M13 and M14 is optional. When it is not present, OUT is the output pin. When it is present, OUT is an internal net and !OUT is the output pin.
Requirements for successful logic injection:

- In Figure 12-16, the names A, B, OUT, SUP1, and SUP2 are used for reference only. No text is actually required by Calibre LVS. These nets may be connected to other devices, not shown in the figure. The internal nets marked !A, !B, and !!B must not be connected to any other devices.
- Devices M1, M2, M3, M4, M9, M11, and M13 must be CMOS P-type transistors. Devices M5, M6, M7, M8, M10, M12, and MN14 must be CMOS N-type transistors.
- When gate recognition is enabled, SUP1 in the figure must be a power net and SUP2 must be a ground net, as specified by LVS Power Name and LVS Ground Name statements, respectively. When gate recognition is disabled, SUP1 and SUP2 can be arbitrary nets.
- All MOS devices can have no more than one substrate pin. It is not required that all MOS devices have the same number of substrate pins. However, all substrate pins of all P-type transistors must be connected to one net. The same condition applies for substrate pins of all N-type transistors. Substrate pins may be connected to the “power” and “ground” nets marked SUP1 and SUP2, but this is not a requirement.

Pin swappability:

- When gate recognition is disabled, pins are not swappable.
- When gate recognition is enabled, input pins A and B are swappable. Asymmetries such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to input pins are ambiguous, ambiguity resolution will examine types, subtypes, and properties of devices inside the injected gate.

Possible configurations:

_\texttt{\_xor2v} — 5-pin XOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_\texttt{\_xor2b} — 7-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

_\texttt{\_xori2v} — 5-pin XOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_\texttt{\_xori2b} — 7-pin XOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the
P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

_xnor2v — 5-pin XNOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xnor2b — 7-pin XNOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

_xnori2v — 5-pin XNOR gate (A, B, OUT, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_xnori2b — 7-pin XNOR gate (A, B, OUT, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only XOR gates) memory consumption is reduced by 8.5 times for _xor2 and _xnor2 structures and by 9.5 times for _xori2 and _xnori2 structures. For comparison, if the XOR gate could be made into an hcell, memory consumption would be reduced by 12.6 or 14.7 times, respectively.

Transmission Gate Multiplexers

A 4-MOS Transmission Gate Multiplexer (TGM) is recognized and injected.

Figure 12-17. _tgmb Gate Multiplexer

Requirements for successful logic injection:
Hierarchical LVS
Logic Injection

- In Figure 12-17, the names SEL1, SEL2, IN1, IN2, OUT, B1, and B2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram, except for the output net OUT, which cannot be connected to any source/drain pins of MOS devices.

- MP devices must be CMOS P-type transistors, MN devices must be CMOS N-type transistors. The MP devices may have different component types, and the MN devices may have different component types.

- All four MOS devices can have no more than one substrate pin. Furthermore, if at least one MOS device has a substrate pin, all four must have it. Substrate pins of all P-type transistors must be connected to one net, and substrate pins of all N-type transistors must be connected to one net.

- It is possible that grouping of MOS devices into TGM logic is topologically ambiguous or may appear so to the logic injection algorithm. If logic injection were to take place, and an ambiguous TGM chosen differently in layout and source, false errors would be reported. To prevent this, potentially ambiguous configurations are recognized, and logic injection is disabled for them. As a result, certain TGM structures may not be injected, even though they fit the other criteria described previously.

Pin swappability:

- Pins IN1 and IN2 are always swappable, pins SEL1 and SEL2 are also swappable. However, both pairs of pins can only be swapped simultaneously. Swapping of only one pair of pins is not allowed.

- Asymmetries, such as different component types, subtypes, or properties, do not prevent pin swappability (but are checked after matching is completed). If nets connected to swappable pins are ambiguous, the ambiguity resolution step examines types, subtypes, and properties of devices inside the injected structure.

Possible configurations:

\_tgmb — 7-pin transmission gate multiplexer (IN1, IN2, SEL1, SEL2, OUT, B1, and B2), formed from 3-pin and 4-pin MOS devices with substrate pins of the P devices connected to net B1 and substrate pins of the N devices connected to net B2.

Note: B1 and B2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only TGM logic) memory consumption is reduced by 1.8\times. For comparison, if the TGM logic could be made into an hcell, memory consumption would be reduced by 2.0\times.
Register File Bit

The Register File Bit (RFB) structure is recognized and injected.

**Figure 12-18. _bitrfv Structure**

Requirements for successful logic injection:

- In *Figure 12-18*, the names B, BN, W, RWL, RBL, SUP1, and SUP2 are for reference only; no text is actually required by LVS. These nets may be connected to other devices not shown in the diagram. The internal nets marked 1 and 2 must not be connected to any other devices.

- In each bit structure, there must be only one “power” net (marked SUP1 in the diagram) and one “ground” net (marked SUP2 in the diagram). Configurations with different power or ground nets used by the two inverters comprising the bit structure are rejected. The power and ground nets SUP1 and SUP2 may be any nets and do not have to be named or designated as power or ground supplies.

- Devices M1 and M3 must be CMOS P-type transistors, devices M2 and M4 must be CMOS N-type transistors, devices M5, M6, M7, and M8 can be either CMOS P-type or CMOS N-type transistors (and do not have to be of the same type). Devices M1 and M3 may have different component types as long as they are CMOS P-type transistors. Devices M2 and M4 may have different component types as long as they are CMOS N-type transistors.

- All eight MOS devices can have no more than 1 substrate pin. Furthermore, if at least one MOS device has a substrate pin, all eight must have it. Substrate pins of all P-type transistors must be connected to one net; the same holds for substrate pins of all N-type transistors. Substrate pins may be connected to the power and ground nets marked SUP1 and SUP2, but do not have to be.
Hierarchical LVS

Hierarchical Cell Cycles

- In some cases, grouping of MOS devices into bit structures is topologically ambiguous. If logic injection were to take place, and an ambiguous bit is structure chosen differently in layout and source, false errors would be reported. To prevent this, potentially ambiguous configurations are recognized, and logic injection is disabled for them.

Pin swappability:

None.

Possible configurations:

_bitrfv — 7-pin bit structure (B, BN, W, RWL, RBL, SUP1, SUP2), 3-pin MOS devices or 4-pin MOS devices with substrate pins of CMOS P-type and N-type transistors connected to nets SUP1 and SUP2, respectively.

_bitrff — 9-pin bit structure (B, BN, W, RWL, RBL, SUP1, SUP2, SUB1, SUB2), 4-pin MOS devices with substrate pins not connected to nets SUP1 and SUP2. Substrate pins of the P devices are connected to net SUB1 and substrate pins of the N devices are connected to net SUB2.

Note: SUB1 and SUB2 are “second class” pins and normally do not appear in LVS reports; substrate connections are checked, but discrepancies are reported at individual transistor level.

Memory savings:

Under optimal conditions (many large hcells of approximately equal size containing only bit structures) memory consumption is reduced by $5 \times$. For comparison, if the bit structure could be made into an hcell, memory consumption would be reduced by $6.3 \times$.

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 13-31.

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 paragraphs 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, the netlister adds two leading underscores and may add additional underscores to avoid conflicts with user cell names. Specifically, the
number of underscores added is one larger than 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 $xyz appears as __$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**

Port objects created with Port Layer Text and Port Layer Polygon specification statements in the top-level cell are output in the hierarchical SPICE netlist as pins in the top-level .SUBCKT line. In addition, port objects in LVS Box cells include pins in those cells. In other cells, port objects are not represented in the netlist.

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:

- In the top cell, if port objects are present, then net names are determined from net or port names, whichever is texted.
- If both a net and its port are texted, then the net name has precedence.
- If a port name already appears on a different net, then the port name is ignored and a warning is issued.
- If two port names appear on the same net, then one of the port names is chosen arbitrarily and the other port name is silently discarded.
- Ports and nets are not considered texted if their names are not valid for netlisting.

To be valid for netlisting, a port name must obey the same rules as for nets.
String Property Values
String property values appear in double-quotes (“ ”) in the hierarchical SPICE layout netlist. Example:

```plaintext
M0 sd2 g sd1 p a=5 b="foo" $X=0 $Y=0 $D=0
```

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 can be treated as built-in for comparison if they conform to “MOS Transistor Required Pin Names” on page 11-9. 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:

```plaintext
DEVICE m(h) gate gate(g) sd(s) sd(d) bulk(b)
[     property w, l ...
]
```
generates something like this in the layout netlist:

```plaintext
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 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, like this:

```plaintext
.SUBCKT m g s d b
.ENDS
X1 1 2 3 4 m $[h]
```

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:

```plaintext
.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.

**Case Preservation for Device Operations**

This is discussed under LVS Compare Case in the SVRF Manual.

**$D Parameters in SPICE Netlists**

The hierarchical SPICE layout netlister annotates each extracted device instance in the netlist with a number that identifies the associated rule file Device operation. These identifiers are coded as comments in the form $D=<number>. For example:

```
M0 3 1 2 pa L=1e-05 W=1e-05 $X=0 $Y=0 $D=0
```

This information is intended for use by certain downstream applications, for example, Calibre xRC. It is not used in LVS. Rule file Device operations are normally numbered sequentially from zero in the order in which they appear in the rule file; however, this is not guaranteed.

**Hierarchical Netlister Warnings**

The hierarchical SPICE netlister, used in calibre -spice, may issue the warnings described next. Unless indicated otherwise, these warnings appear in the Calibre transcript and in the circuit extraction report file.

- **Port not valid for netlisting**

  WARNING: Top level port name “<name>” at location (<x>, <y>) on net <net-number> not valid for netlisting; net id used instead.

  Indicates a top-level port name not valid for netlisting. For example a port name that contains illegal characters. Example:

  ```
  WARNING: Top level port name "po/rt" at location (23,-8) on net 6 not valid for netlisting; net id used instead.
  ```

- **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.
```

- **Net and port names not valid for netlisting**

  WARNING: {Net | Top level port} name <name> {on net <net> in cell in cell <cell> | at location (<x>, <y>) on net <net>} not valid for netlisting; net id used instead.

  Examples are names that contain slash (/) characters or other illegal characters. These warnings appear in the Calibre transcript and also in the circuit extraction report. For example:

  ```
  WARNING: Net name "ne/t1" on net 1 in cell "badtext_1" not valid for netlisting; net id used instead.
  WARNING: Top level port name "po/rt" at location (23,-8) on net 6 not valid for netlisting; net id used instead.
  ```

- **Bad device**

  WARNING: BAD DEVICE on layer <layer> at location (<x>,<y>) in cell <name>

  This warning indicates a bad device as defined by the Device statement. 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
  ```
This chapter discusses the results files generated by LVS comparison. These include:

- LVS Report
- Circuit Extraction Report
- Mask Results Database
- SVDB Cross-Reference Files

The run transcript format is discussed under “Session Transcript” on page 7-1. ERC and LVS short isolation, which occur during the circuit extraction phase of LVS, generate results that are in ASCII DRC results format. This format is discussed under “ASCII DRC Results Database Format” on page 7-8.

**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 that were found.
- LVS netlist compiler errors and warnings that were found.
- An LVS header section, specifying the report file name, the layout and source design names (top-level cell names are indicated in parentheses when applicable), the rule file name, the rule file title (if a Title statement is 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 Results
LVS Report

- LVS parameters section, showing the LVS settings used. Settings are expressed in SVRF syntax indicating the respective specification statements. Commented lines are used, when appropriate, to show statements that have no default setting and for which you do not provide a setting.

- Optional information and warnings section.

- Optional detailed instance connections section.

- Optional list of unmatched elements.

The next example shows a flat LVS report.

```
#   #         #####################
# #          #                   #
#           #     INCORRECT     #
# #          #                   #
#   #         #####################

Error: Different numbers of nets (see below).
Error: Different numbers of instances (see below).

INITIAL NUMBERS OF OBJECTS

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets</td>
<td>8</td>
<td>12 *</td>
</tr>
</tbody>
</table>
```
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
---------------------------------------

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets:</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>INV (2 pins)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>NAND2 (3 pins)</td>
</tr>
</tbody>
</table>
|        | ------ | -----
| 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
*************************************************************

<table>
<thead>
<tr>
<th>DISC#</th>
<th>LAYOUT NAME</th>
<th>SOURCE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>** missing net **</td>
<td>0/1</td>
</tr>
<tr>
<td>2</td>
<td>** missing net **</td>
<td>0/2</td>
</tr>
<tr>
<td>3</td>
<td>** missing net **</td>
<td>1/1</td>
</tr>
<tr>
<td>4</td>
<td>** missing net **</td>
<td>1/2</td>
</tr>
</tbody>
</table>

*************************************************************
INCORRECT INSTANCES
*************************************************************
LVS Results

LVS Report

<table>
<thead>
<tr>
<th>DISC#</th>
<th>LAYOUT NAME</th>
<th>SOURCE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>** missing instance **</td>
<td>0/C1 C</td>
</tr>
<tr>
<td>6</td>
<td>** missing instance **</td>
<td>1/C1 C</td>
</tr>
</tbody>
</table>

LVS PARAMETERS

- LVS Setup:
  - LVS POWER NAME: "VCC"
  - LVS GROUND NAME: "GROUND"
  - LVS RECOGNIZE GATES: ALL MIX SUBTYPES
  - LVS IGNORE PORTS: NO
  - LVS CHECK PORT NAMES: NO
  - LVS ALL CAPACITOR PINS SWAPPABLE: NO
  - LVS DISCARD PINS BY DEVICE: NO
  - LVS SOFT SUBSTRATE PINS: NO
  - LVS EXPAND UNBALANCED CELLS: YES
  - LVS PRESERVE PARAMETERIZED CELLS: NO
  - LVS GLOBALS ARE PORTS: YES
  - LVS REVERSE WL: NO
  - LVS SPICE PREFER PINS: NO
  - LVS SPICE SLASH IS SPACE: YES
  - LVS SPICE ALLOW FLOATING PINS: YES
  - LVS SPICE CONDITIONAL LDD: NO
  - LVS SPICE OVERRIDEGLOBALS: NO
  - LVS SPICE REDEFINE PARAM: NO
  - LVS SPICE REPLICATE DEVICES: NO
  - LVS STRICT SUBTYPES: NO
  - LAYOUT CASE: NO
  - SOURCE CASE: NO
  - LVS COMPARE CASE: NO
  - LVS REPORT MAXIMUM: 50
  - LVS PROPERTY RESOLUTION MAXIMUM: 32
  - LVS SIGNATURE MAXIMUM: ALL
  - LVS FILTER UNUSED OPTION: AB RC RE RG YC

// LVS REPORT OPTION
// LVS NON USER NAME PORT
// LVS NON USER NAME NET
// LVS NON USER NAME INSTANCE

// Reduction
LVS REDUCE SERIES MOS NO
LVS REDUCE PARALLEL MOS YES [ TOLERANCE L 0 ]
LVS REDUCE SEMI SERIES MOS NO
LVS REDUCE SPLIT GATES YES
LVS REDUCE PARALLEL BIPOLAR YES
LVS REDUCE SERIES CAPACITORS YES
LVS REDUCE PARALLEL CAPACITORS YES
LVS REDUCE SERIES RESISTORS N
LVS REDUCE PARALLEL RESISTORS YES
LVS REDUCE PARALLEL DIODES YES

LVS REDUCE r(a) SERIES pos neg [ TOLERANCE L 0 W 0 ]
LVS REDUCE r(b) SERIES pos neg [ TOLERANCE L 1 W 2 ]

// Filter
LVS FILTER abc OPEN
LVS FILTER r(a) r == 0 SHORT SOURCE

// Trace Property

TRACE PROPERTY mp w w 2
TRACE PROPERTY mp l l 1
TRACE PROPERTY mn w w 2
TRACE PROPERTY mn l l 1

************************************************************
INFORMATION AND WARNINGS
************************************************************

Matched Layout Matched Source Unmatched Layout Unmatched Source Component 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
LVS Results
LVS Report

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 parentheses when applicable), the rule file name, the rule file title (if a Title statement was specified), the external hcell file name (if specified on the command line), 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 \textbf{CORRECT} if all individual cells are correct, \textbf{INCORRECT} if at least one cell is incorrect, and, otherwise, \textbf{NOT COMPARED} if at least one cell is not compared. The usual graphics are provided as well (check mark and smiling face, or X respectively). Refer to section “Overall Comparison Results” on page 13-20 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” on page 13-16 for the meaning of individual status messages.
  - A Cell Summary, listing the primary comparison status for each individual cell (\textbf{CORRECT}, \textbf{INCORRECT} or \textbf{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 \texttt{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 here:
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 2003
CURRENT DIRECTORY: /user/johns/hlvs/test
USER NAME: johns
CALIBRE VERSION: v9.3_4.1 Fri Jun 16 12:47:12 PDT 2003

OVERALL COMPARISON RESULTS

Error: Different numbers of nets (see below).
Error: Different numbers of instances (see below).

CELL SUMMARY

<table>
<thead>
<tr>
<th>Result</th>
<th>Layout</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCORRECT</td>
<td>inv</td>
<td>inv</td>
</tr>
<tr>
<td>CORRECT</td>
<td>nand</td>
<td>nand</td>
</tr>
<tr>
<td>CORRECT</td>
<td>mix</td>
<td>mix</td>
</tr>
</tbody>
</table>

LVS PARAMETERS

o LVS Setup:
LVS COMPONENT TYPE PROPERTY  element
// LVS COMPONENT SUBTYPE PROPERTY
// LVS PIN NAME PROPERTY
LVS POWER NAME  "VCC"
LVS GROUND NAME  "GROUND"
LVS RECOGNIZE GATES  ALL MIX SUBTYPES
LVS IGNORE PORTS  NO
LVS CHECK PORT NAMES  NO
LVS ALL CAPACITOR PINS SWAPPABLE  N
LVS DISCARD PINS BY DEVICE  NO
LVS SOFT SUBSTRATE PINS  NO
LVS EXPAND UNBALANCED CELLS  YES
LVS PRESERVE PARAMETERIZED CELLS  NO
LVS GLOBALS ARE PORTS  YES
LVS REVERSE WL  NO
LVS SPICE PREFER PINS  NO
LVS SPICE SLASH IS SPACE  YES
LVS SPICE ALLOW FLOATING PINS  YES
LVS SPICE CONDITIONAL LDD  NO
LVS SPICE OVERRIDE GLOBALS  NO
LVS SPICE REDEFINE PARAM  NO
LVS SPICE REPLICATE DEVICES  NO
LVS STRICT SUBTYPES  NO
LAYOUT CASE  NO
SOURCE CASE  NO
LVS COMPARE CASE  NO
LVS REPORT MAXIMUM  50
LVS PROPERTY RESOLUTION MAXIMUM  32
LVS SIGNATURE MAXIMUM  ALL
LVS FILTER UNUSED OPTION  AB RC RE RG YC
// LVS REPORT OPTION
// LVS NON USER NAME PORT
// LVS NON USER NAME NET
// LVS NON USER NAME INSTANCE

// Reduction
LVS REDUCE SERIES MOS  NO
LVS REDUCE PARALLEL MOS  YES [ TOLERANCE L 0 ]
LVS REDUCE SEMI SERIES MOS  NO
LVS REDUCE SPLIT GATES  YES
LVS REDUCE PARALLEL BIPOLAR  YES
LVS REDUCE SERIES CAPACITORS  YES
LVS REDUCE PARALLEL CAPACITORS  YES
LVS REDUCE SERIES RESISTORS  NO
LVS REDUCE PARALLEL RESISTORS  YES
LVS REDUCE PARALLEL DIODES  YES
LVS REDUCE  r(a)  SERIES pos neg [ TOLERANCE L 0 W 0 ]
LVS REDUCE  r(b)  SERIES pos neg [ TOLERANCE L 1 W 2 ]

// Filter
LVS FILTER  abc  OPEN
LVS FILTER  r(a)  r == 0 SHORT SOURCE

// Trace Property
TRACE PROPERTY  mp  w w 2
TRACE PROPERTY  mp  l l 1
TRACE PROPERTY  mn  w w 2
TRACE PROPERTY  mn  l l 1

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

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports:</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Nets:</td>
<td>4</td>
<td>6 *</td>
</tr>
<tr>
<td>Instances:</td>
<td>1 1</td>
<td>MN (4 pins)</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>C (2 pins)</td>
</tr>
<tr>
<td>Total Inst:</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

NUMBERS OF OBJECTS AFTER TRANSFORMATION

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports:</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Nets:</td>
<td>4</td>
<td>6 *</td>
</tr>
<tr>
<td>Instances:</td>
<td>0 1</td>
<td>C (2 pins)</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>INV (2 pins: output input)</td>
</tr>
<tr>
<td>Total Inst:</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

*=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 SOURCE NAME
*****************************************************
1 ** missing net ** 1
---
2 ** missing net ** 2

*****************************************************
INCORRECT INSTANCES
*****************************************************
DISC# LAYOUT NAME 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          1          C
               1        1          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
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

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets: 4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Instances:</td>
<td></td>
<td>inv (3 pins): ground vcc out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nand (4 pins)</td>
</tr>
<tr>
<td>Total Inst:</td>
<td>3 3</td>
<td></td>
</tr>
</tbody>
</table>

INFORMATION AND WARNINGS

<table>
<thead>
<tr>
<th>Matched Layout</th>
<th>Matched Source</th>
<th>Unmatched Layout</th>
<th>Unmatched Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets: 4 4 0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Calibre Verification User's Manual
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 parentheses when applicable), the rule file name, the rule file title (if a Title specification 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>"

##################################################
##                                              ##
REPORT FILE NAME:         lvs.rep
LAYOUT NAME:     
SOURCE NAME:              z2.net ('top')
RULE FILE:                rules
CURRENT DIRECTORY:        /scratch1/john/play
USER NAME:                john
CALIBRE VERSION:          v9.1_5.1    Tue Apr 16 22:14:21 PDT 2003

*************************************************************************
OVERALL SYNTAX CHECK RESULTS
*************************************************************************

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 reports 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.

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 are redundant between the two sections.

- Skim the Information and Warnings section before examining the problem in detail.

The Information and Warnings section contains statistics and information about the comparison run, including an exact set of numbers regarding 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 Calibre RVE when the Mask SVDB Directory specification statement is specified in the rule file. For more information, refer to Chapter 5, “Using the LVS-RVE Interface” in the Calibre Interactive User’s Manual.

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 \( Y \). For example:

    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. For example:

    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.

- **Naming Errors**
  
  The “ne” marker, which stands for naming error, appears in the LVS report with individual naming error discrepancies, as well as in the LEGEND section. Naming errors occur when a layout name appears in the source, but the object was matched by other means. Note that naming error discrepancies are rare.

  The affected report sections are:
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

- **SPICE netlist source** — 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 hierarchical LVS, if the SPICE netlist contains appropriate layout location information, then pathnames that identify instances are followed by \((x,y)\) layout locations in parentheses. The format is:

```
subckt_call_name_1/.../subckt_call_name_n/element_name(x,y)
subckt_call_name_1/.../subckt_call_name_n/
subckt_call_name(x,y)
```

Layout location information in SPICE netlists is provided with comment-coded $X, $Y, and $T parameters in element statements and subcircuit calls. This is described in the “SPICE Format” chapter. The hierarchical SPICE netlister (calibre -spice) writes this information to the output netlist.

In flat LVS, \( subckt_call_name_1 \) through \( subckt_call_name_n \) appear without the preceding X.
Flat:

```
NETA
1/netb
fred/4/7/R7
fred/Xabc
```

Hierarchical:

```
NETA
X1/netb
Xfred/X4/X7/R7
Xfred/Xabc
Xfred/X4/X7/R7(-7754.800,-5406.700)
Xfred/Xabc(-7000.000,-8000.000)
```

A design port in a SPICE netlist is identified by its name. 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.

A layout port in Mask LVS is identified by its name, or, in the absence of name, by its numerical handle.

### Logic Gate Identification

LVS forms logic gates internally by the logic gate recognition feature. See “Logic Gate Recognition” on page 11-42. 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.

### Injected Component Identification

Injected components are formed internally by the logic injection feature of LVS. An injected-component instance is identified by its type in parentheses, followed by a list of individual device instances forming the injected component. Component types and optional subtypes of the individual devices are indicated as well. Example:
This describes a \textit{\_bitv} structure formed by the six transistors shown.

\textbf{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:

\begin{verbatim}
I\$702:G
\end{verbatim}

identifies pin G of schematic instance I\$702, and

\begin{verbatim}
27(230,540):B
\end{verbatim}

identifies pin B of an extracted mask device with an ID of 27, and a location of X=230, Y=540.

\textbf{Logic Gate Pin Identification}

Logic gates are formed internally by the logic gate recognition feature. See \textit{``Logic Gate Recognition``} on page 11-42. LVS identifies a pin of an internally-generated logic gate with the following format:

\begin{verbatim}
(gate_type): {INPUT | OUTPUT}
\end{verbatim}

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:

\begin{verbatim}
(INV):OUTPUT
 /I\$767/MP/MP/I\$702:D
 /I\$767/MN/MN/I\$786:S
\end{verbatim}

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.
Injected Component Instance Pin Identification

Injected components are formed internally by the logic injection feature of LVS. An injected-component instance pin is identified in the LVS report by the injected-component type in parentheses, followed by a colon (:), followed by the injected-component pin name. This is followed by a list of the individual device instances connected to that pin within the injected component. Each individual device instance is identified by the device instance name, followed by a “:”, followed by the name of the device pin that leads to that injected-component pin. Example:

```
(_bitv):bl
  x2/m4:s
```

This describes pin “bl” of an injected “_bitv” structure. The “bl” pin is formed by the either the “s” pin of transistor x2/m4.

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 entitled 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

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets:</td>
<td>966</td>
<td>967</td>
</tr>
<tr>
<td></td>
<td>1182</td>
<td>1184 * MP (4 pins)</td>
</tr>
<tr>
<td>Inst:</td>
<td>1182</td>
<td>1184 * MN (4 pins)</td>
</tr>
<tr>
<td></td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Total Inst:</td>
<td>2364</td>
<td>2368</td>
</tr>
</tbody>
</table>

### NUMBERS OF ELEMENTS AFTER TRANSFORMATION

<table>
<thead>
<tr>
<th>Layout</th>
<th>Source</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nets:</td>
<td>784</td>
<td>785</td>
</tr>
<tr>
<td></td>
<td>471</td>
<td>472 * MP (4 pins)</td>
</tr>
<tr>
<td></td>
<td>471</td>
<td>472 * MN (4 pins)</td>
</tr>
<tr>
<td></td>
<td>347</td>
<td>348 * INV (2 pins)</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>59 NOR2 (3 pins)</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>123 NAND2 (3 pins)</td>
</tr>
<tr>
<td>Inst:</td>
<td>1471</td>
<td>1474</td>
</tr>
<tr>
<td></td>
<td>------</td>
<td>------</td>
</tr>
</tbody>
</table>

* = Number of elements in layout different from number in source.

### Primary Messages

Table 13-1 lists the primary comparison status messages from the overall results section.

**Table 13-1. Primary Messages**

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORRECT</td>
<td>The layout connectivity is equivalent to the source connectivity.</td>
</tr>
<tr>
<td>CORRECT except for naming or swap-override errors</td>
<td>The layout connectivity is equivalent to the source connectivity, except for differences in element names or violations of restrictions on swapping of pins.</td>
</tr>
<tr>
<td>INCORRECT</td>
<td>Discrepancies were detected between the two circuits.</td>
</tr>
<tr>
<td>NOT COMPARED</td>
<td>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.</td>
</tr>
</tbody>
</table>
Secondary Messages

Secondary messages, if any, follow the primary comparison status message. Table 13-2 lists these messages that 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 13-2. Secondary Messages—Errors**

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: Connectivity errors.</td>
<td>Connectivity errors were found, for example, incorrect nets, incorrect instances, unmatched elements.</td>
</tr>
<tr>
<td>Error: Incorrect names for power/ground nets.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Errors in layout.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Errors in source.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Power net missing in layout. Power net missing in source. Ground net missing in layout. Ground net missing in source.</td>
<td>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 appear together on one line.</td>
</tr>
<tr>
<td>Error: Power or ground net missing.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Components with non-identical signal pins were found.</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 13-2. Secondary Messages—Errors (cont.)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: Different numbers of ports (see next).</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Different numbers of nets (see next).</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Different numbers of instances (see next).</td>
<td>The number of instances where one or more component type in the layout differs 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.</td>
</tr>
<tr>
<td>Error: Instances of different types or subtypes were matched.</td>
<td>Some layout instances were matched to some schematic instances which had different component types or subtypes. These instances are listed as discrepancies.</td>
</tr>
<tr>
<td>Error: Cells with non-floating extra pins.</td>
<td>A cell contains instances of other cells that have extra pins in the source or layout. 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.</td>
</tr>
<tr>
<td>Error: Property errors.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Property tolerances out of range of actual values.</td>
<td>Means that ABSOLUTE tolerance values in some Trace Property specification statements were found to be much larger than the actual property values found. This usually means that tolerance values were specified with the wrong scale or units. See “Tolerance in Device Reduction” on page 11-36 for details. The respective Trace Property statements are listed in the ABSOLUTE TRACE PROPERTY TOLERANCES OUT OF RANGE section of the report.</td>
</tr>
<tr>
<td>Error: Substrate pin errors.</td>
<td>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.</td>
</tr>
<tr>
<td>Error: Components with non-identical power or ground pins were found.</td>
<td>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.</td>
</tr>
</tbody>
</table>
The following are LVS Warnings.

### Table 13-3. Secondary Messages—Warnings

<table>
<thead>
<tr>
<th>Warning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning: Bad devices in layout.</td>
<td>Issued only in flat LVS. Badly formed devices were found while recognizing devices from layout shapes. The bad devices are listed in the “Information and Warnings” section of the report.</td>
</tr>
<tr>
<td>Warning: Components with non-identical power or ground pins were found.</td>
<td>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.</td>
</tr>
<tr>
<td>Warning: Unbalanced smashed mosfets were matched.</td>
<td>The source contains at least one group of MOS transistors connected in parallel. That is, they are 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.</td>
</tr>
</tbody>
</table>
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 the source and drain nets of some transistor devices. Such devices 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.

The source and layout, as seen by the LVS circuit comparison module, refer to the same data. For example, in Calibre LVS-H, 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.

Warning: Corresponding hcells have incompatible pin swap sets (see transcript).

One or more pairs of component types that are specified as corresponding in the hcell list have different pin swappability. For example, the hcell list contains the pair “AAA BBB” but pin swappability of AAA is different from BBB. Detailed information appears in the transcript. The correspondence in such cases is ignored.

Warning: Corresponding hcells are of different kinds (see transcript).

One or more pairs of component types that are specified as corresponding in the hcell list are of different kinds. For example, the hcell list contains the pair “AAA BBB” but AAA is a cell and BBB is a primitive device; or, AAA is defined to be a CMOS N transistor and BBB is defined to be a CMOS P transistor. Detailed information appears in the transcript. The correspondence in such cases is ignored.

<table>
<thead>
<tr>
<th>Warning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning: User-names were overridden.</td>
<td>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.</td>
</tr>
<tr>
<td>Warning: Ambiguity points were found and resolved arbitrarily.</td>
<td>The compared circuits contain interchangeable parts. The “Information and Warnings” section lists the elements which were matched arbitrarily.</td>
</tr>
<tr>
<td>Warning: Extra ports in layout.</td>
<td>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.</td>
</tr>
<tr>
<td>Warning: Extra ports in source.</td>
<td>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.</td>
</tr>
<tr>
<td>Warning: FY, GY, M, and N filters did not connect some s and d pins.</td>
<td>Unused device filter options FY, GY, M, or N did not connect the source and drain nets of some transistor devices. Such devices 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.</td>
</tr>
<tr>
<td>Warning: Source and layout refer to the same data.</td>
<td>The source and layout, as seen by the LVS circuit comparison module, refer to the same data. For example, in Calibre LVS-H, 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.</td>
</tr>
<tr>
<td>Warning: Corresponding hcells have incompatible pin swap sets (see transcript).</td>
<td>One or more pairs of component types that are specified as corresponding in the hcell list have different pin swappability. For example, the hcell list contains the pair “AAA BBB” but pin swappability of AAA is different from BBB. Detailed information appears in the transcript. The correspondence in such cases is ignored.</td>
</tr>
<tr>
<td>Warning: Corresponding hcells are of different kinds (see transcript).</td>
<td>One or more pairs of component types that are specified as corresponding in the hcell list are of different kinds. For example, the hcell list contains the pair “AAA BBB” but AAA is a cell and BBB is a primitive device; or, AAA is defined to be a CMOS N transistor and BBB is defined to be a CMOS P transistor. Detailed information appears in the transcript. The correspondence in such cases is ignored.</td>
</tr>
</tbody>
</table>
LVS Results
LVS Report

Table 13-3. Secondary Messages—Warnings (cont.)

<table>
<thead>
<tr>
<th>Warning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning: Corresponding hcells have different property rules (see transcript).</td>
<td>One or more pairs of component types that are specified as corresponding in the hcell list have different Trace Property rules specified. For example, the hcell list contains the pair “AAA BBB” and the rule file contains a Trace Property rule for AAA but no Trace Property rule for BBB. Detailed information appears in the Calibre transcript. The correspondence in such cases is ignored.</td>
</tr>
</tbody>
</table>

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, such as 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 previous 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 **.

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 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 13-4 explains the two sub-sections.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Badly Formed Power/Ground Net Names</td>
<td>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.</td>
</tr>
<tr>
<td>Contradictory Power/Ground Net Names</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**Absolute Trace Property Tolerances Out of Range**

This section of the report lists Trace Property specification statements where ABSOLUTE tolerance values were found to be much larger than the actual property values found. This
usually means that the tolerance values were specified with the wrong scale or units. See “Tolerance in Device Reduction” on page 11-36 for details. Example:

***************************************************************************************************
ABSOLUTE TRACE PROPERTY TOLERANCES OUT OF RANGE
***************************************************************************************************

(In the following TRACE PROPERTY rules, most values found for the respective property in the source were significantly smaller than the ABSOLUTE tolerance value specified in the rule. The tolerance value was probably specified with the wrong scale or units. Note: only matched instances are counted).

TRACE PROPERTY  mp  l  l
TRACE PROPERTY  mp  w  w

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. Reporting formats differ somewhat between flat and hierarchical LVS.

Flat LVS example:

***************************************************************************************************
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.)

<table>
<thead>
<tr>
<th>Layout Pin Name</th>
<th>Source Pin Name</th>
<th>Component Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>** missing pin **</td>
<td>B</td>
<td>MN</td>
</tr>
<tr>
<td>RESET</td>
<td>** missing pin **</td>
<td>REG5</td>
</tr>
</tbody>
</table>

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

In this example, component type MN has a pin B in the source but not in the layout. Component type REG5 has a pin RESET in the layout but not in the source. Note that MN and REG5 are component types, not instance names (there may be many instances of MN and REG5).
Hierarchical LVS example:

In hierarchical LVS, every primitive component type with differences in signal pins has its own section in the report, similar to the one shown here:

```
****************************************
COMPONENT TYPES WITH NON-IDENTICAL SIGNAL PINS
****************************************

(Cells with the same (or corresponding) name that have different signal pin names are listed below. Pins that do not appear in all corresponding cells in both source and layout are ignored by the comparison algorithm.)

Layout Component Type:  C (3 pins): p n sub
Layout Extra Pins:      sub

Source Component Type:  C (2 pins): p n
No Extra Pins.

Source Component Type:  C (3 pins): p n sub
Source Extra Pins:      sub

Note that only primitive component types are reported in this way (for example, primitive devices or LVS Box cells); pin differences in hcells are reported in other ways.

When LVS Report Option X is specified, hierarchical LVS also prints a representative list of instances for each configuration of pins being reported. Each list of instances is preceded by the parent cell name. In addition, the final pin configuration (after discarding of extra pins) is shown as well. For example, with option X specified, the above report might look like this:

```
Layout Component Type:  C (3 pins): p n sub
Layout Extra Pins:      sub
Layout Instances:
    Cell:     aaa
    Instances:  c1 c2 c3
    Cell:     bbb
    Instances:  c10 c11

Source Component Type:  C (2 pins): p n
No Extra Pins.
Source Instances:
    Cell:     bbb
    Instances:  c1

Source Component Type:  C (3 pins): p n sub
Source Extra Pins:      sub
Source Instances:
    Cell:     aaa
    Instances:  c30 c40

Result Component Type:  c (2 pins): p n
```
In this example, in the layout, the 3-pin configuration is represented by instances c1, c2, and c3 in layout cell aaa and instances c10 and c11 in layout cell bbb. In the source, the 2-pin configuration is represented by instance c1 in source cell bbb and the 3-pin configuration is represented by instances c30 and c40 in source cell aaa. The Result Component Type line shows the final pin configuration, which consists of 2 pins.

Up to LVS Report Maximum (but no more than 100) representative instances are reported for each configuration of pins.

Note that LVS Report Option X does not operate in flat LVS.

**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>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISSING COMPONENT TYPES</td>
<td>A list of instances whose component types cannot be determined.</td>
</tr>
<tr>
<td>MISSING PIN NAMES</td>
<td>A list of instances, grouped by component type, that have pins whose names cannot be determined. Component subtypes are indicated in parentheses, if specified.</td>
</tr>
<tr>
<td>BAD INSTANCES</td>
<td>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” on page 11-7.</td>
</tr>
<tr>
<td>CONFLICTING INSTANCES</td>
<td>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” on page 11-4.</td>
</tr>
<tr>
<td>SERIES PIN NAMES NOT FOUND</td>
<td>This is a list of instances that are subject to a specification statement of the form LVS Reduce &lt;component&gt; 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).</td>
</tr>
</tbody>
</table>
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
******************************************************************************

<table>
<thead>
<tr>
<th>Layout Cells</th>
<th>Source Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>lay3</td>
<td>src4</td>
</tr>
<tr>
<td>lay2</td>
<td>src3</td>
</tr>
<tr>
<td>lay1</td>
<td>src2</td>
</tr>
<tr>
<td>lay4</td>
<td>src1</td>
</tr>
<tr>
<td>lay3</td>
<td></td>
</tr>
</tbody>
</table>
```

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

Table 13-5. Input Errors (cont.)

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBIGUOUS SERIES PIN NAMES SPECIFIED</td>
<td>This is a list of instances that are subject to a specification statement of the form LVS Reduce <code>&lt;component&gt; SERIES pin-name-1 pin-name-2</code> but have more than 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.</td>
</tr>
<tr>
<td>INCORRECT SERIES PIN SWAP SETS</td>
<td>This is a list of instances that are subject to a specification statement of the form LVS Reduce <code>&lt;component&gt; SERIES pin-name-1 pin-name-2</code> but in which pins pin-name-1 and pin-name-2 of the instance are swappable with other pins (that is, pins not named in the particular LVS Reduce <code>&lt;component&gt; SERIES</code> statement). For each instance, LVS indicates the component type, optional component subtype, instance name and offending pin names (one pin per line).</td>
</tr>
</tbody>
</table>
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 previous example. (Cell correspondence is specified explicitly in the hcell list or implicitly when you use the \texttt{--automatch} command line switch).

Here is another example:

\begin{tabular}{|l|}
\hline
\textbf{Layout Cells} & \textbf{Source Cells} \\
\hline
lay3 & src3 \\
lay4 & src1 \\
lay3 & src3 \\
\hline
\end{tabular}

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 \texttt{--> src3 --> lay3} instead of src1 \texttt{--> lay4 --> lay3}. In cases like this, you need to examine the layout, source netlist, and/or the hcell list to determine how the cycle is formed.

Similar information appears in the transcript of the Calibre LVS-H circuit comparison module.

\section*{LVS Discrepancy Types}

The LVS discrepancy types are described next. For each discrepancy type, the report format and a graphic representation are specified.

\begin{itemize}
\item \textbf{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:

\begin{verbatim}
---------------------------------------------
5    Net VDD //VDD
      /N$87
---------------------------------------------
\end{verbatim}
For the previous example, source nets //VDD and /N$87 are connected 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 previous example, layout nets N4 and N87 should be connected 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 are missing in the source net. The indicated transistors are a pair of MP and MN transistors that 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 connections of a net in circuit A are different from the connections of the corresponding net in circuit B, and LVS 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.

- **Missing Injected Instance** — This type of discrepancy indicates a missing injected-component instance in the layout or source circuit. This is similar to the missing instance discrepancy, except that it is reported for injected components formed internally by the logic injection feature of LVS. This discrepancy happens when all instances of a particular injected-component type in one of the circuits have been matched, and there are some unmatched instances of the same injected-component type left in the other circuit. The unmatched instances are reported as missing. When encountering this discrepancy, check the numbers of instances after transformation reported in the Overall Comparison Results or Cell Comparison Results section of the report.
The injected component type is indicated in parentheses followed by a list of devices forming the injected component. The devices forming the injected component are highlighted. Example:

```
2  (_bitv) ** missing injected instance **

Devices:
  x1/x2/m1  MP (P)
  x1/x2/m2  MN (N)
  x1/x1/m1  MP (P)
  x1/x1/m2  MN (N)
  x1/m4   MP (P)
  x1/m3   MP (P)
```

The _bitv structure formed by the layout transistors shown is missing in the source. MP(P) and MN(N) are the component types and subtypes of the respective transistors.

- **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 LVS cannot match the net and classify it 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 LVS cannot match the port or the corresponding net and cannot classify the port 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
bad component type: MP
--------------------------------------------------------
/I$34 MN
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 appear on the left. If the layout net is matched, then the corresponding source net name appears 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 appear on the right. If the source net is matched, then the corresponding layout net name appears 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. 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 reported only 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 appears on the left. If the layout net is matched, then the corresponding source net name appears 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 appears on the right. If the source net is matched, then the corresponding layout net name appears 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:

```plaintext
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    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 of the LVS report. 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 cannot be computed for the device; see “Missing and Unknown Property Values” on page 11-34 for more information. A unknown property value is indicated with a question mark and the words “reduced instance.” For example:

\[ p: ? \text{ (reduced instance)} \]

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:

```
+----------------------------------+
<table>
<thead>
<tr>
<th>DISC#</th>
<th>LAYOUT</th>
<th>SOURCE</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27(230,540) MD</td>
<td>/I$867 MD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>w: 5.2u</td>
<td>w = 5u</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td>l: 2.2u</td>
<td>l = 2u</td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>35(120,70) ME</td>
<td>/I$302 ME</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L = 12.2u</td>
<td>L = 12u</td>
<td>2%</td>
</tr>
</tbody>
</table>
+----------------------------------+
```

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 microns, 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 statements 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 13-1. 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 previous 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 statements such as Trace Property or device reduction Tolerance. The property may also be required because it is used in effective property calculation for other properties that appear in such statements. 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 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 out of the ordinary but are not considered errors, and about additional information that can be useful in verifying a design. Table 13-6 lists these statements.

**Table 13-6. Information and Warnings**

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers of Matched And Unmatched Elements</td>
<td>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.</td>
</tr>
<tr>
<td>Statistics</td>
<td>Provides various statistical information about the LVS run.</td>
</tr>
<tr>
<td>Component Types With Non-Identical Power Or Ground Pins</td>
<td>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.” While LVS treats differences in signal pins as errors, differences in power or ground pins are treated as warnings.</td>
</tr>
<tr>
<td>Bad Devices</td>
<td>Provides a list of badly formed layout devices. A bad device occurs because the combination of pin shapes that it touches does not match any of the Device statements for this layer. See “Bad Device Reporting” on page 10-6.</td>
</tr>
</tbody>
</table>
## Table 13-6. Information and Warnings

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matched Mosfets Which Have Been Unequally Reduced</td>
<td>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 “Unequally Reduced Devices” on page 11-34 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.</td>
</tr>
<tr>
<td>Isolated Layout Nets</td>
<td>Provides a list of layout nets that 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.</td>
</tr>
<tr>
<td>Passthrough Layout Nets And Their Ports</td>
<td>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.</td>
</tr>
<tr>
<td>Layout Names That Are Missing In The Source</td>
<td>Provides a list of user-given net, instance, and port names in the layout, which are not present in the source.</td>
</tr>
<tr>
<td>Layout Names That Appear On More Than One Element</td>
<td>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.</td>
</tr>
<tr>
<td>Source Names That Appear On More Than One Element</td>
<td>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.</td>
</tr>
<tr>
<td>Conflicting Layout Names</td>
<td>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.</td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Conflicting Source Names</td>
<td>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.</td>
</tr>
<tr>
<td>Initial Correspondence Points</td>
<td>Provides a list containing pairs of identically named nets, ports, and instances used as initial correspondence points. See the section “Initial Correspondence Points” on page 11-16, for more information.</td>
</tr>
<tr>
<td>Cpoints</td>
<td>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.</td>
</tr>
<tr>
<td>Failed Cpoints</td>
<td>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.</td>
</tr>
<tr>
<td>Ambiguity Resolution Points</td>
<td>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” on page 11-16 for more information.</td>
</tr>
<tr>
<td>Layout/Source FY, GY, M, and N Filtered Devices That Did Not Connect S And D Pins</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 cannot 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 LVS cannot 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 while ignoring this section, and run LVS again. The unmatched elements 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 cannot 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
```

- **Unmatched Injected Instance** — An unmatched injected instance is an injected-component instance in the layout or source that cannot be matched to a corresponding injected component in the other circuit and cannot be classified as any of the available discrepancy types. This error is reported for injected components formed internally by the logic injection feature of LVS. The injected component type is indicated in parentheses followed by a list of devices forming the injected component. The devices forming the injected component are highlighted. Example:

```
(_bitv) ** unmatched injected instance **
---------------------
Devices:
  x1/x2/m1  MP (P)
  x1/x2/m2  MN (N)
  x1/x1/m1  MP (P)
  x1/x1/m2  MN (N)
  x1/m4     MP (P)
  x1/m3     MP (P)
```

The injected component type is indicated in parentheses followed by a list of devices forming the injected component. The devices forming the injected component are highlighted.
The _bitv structure formed by the layout transistors shown could not be matched. MP(P) and MN(N) are the component types and subtypes of the respective transistors.

## Circuit Extraction Report

The circuit extraction report is written during the circuit extraction (calibre -spice) phase of Calibre LVS-H. This report contains a summary of circuit extraction warnings and errors that 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.

This database is not related to the Standard Verification Database (SVDB).

## SVDB Cross-Reference Files

Calibre LVS / LVS-H applications can generate SVDB instance and net cross-reference files. These are discussed in this section.

## Flat Instance Cross-Reference File

You can use the Calibre -ixf command line switch to instruct the application to generate flat instance cross-reference files. This is discussed on page 3-17.
The IXF secondary keyword of 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. If you do not specify a Layout Primary statement in the rule file, layout_primary defaults to ICV_UNNAMED_TOP.

The instance cross-reference file is a text file that contains matched instances. The file contains one line per instance in the following form:

\[ \text{layout_id} \quad \text{layout_name} \quad \text{source_id} \quad \text{source_name} \quad [\text{SL} \mid \text{SS}] \quad [\text{X}] \]

where

- \text{layout_id} is a number that represents the instance (ID) in the layout.
- \text{layout_name} is a user-given name that represents the layout instance. The value of \text{layout_id} is used if no user-given name was specified.
- \text{source_id} is a number that represents the instance (ID) in the source.
- \text{source_name} is a user-given name that represents the source instance. The value of \text{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:

\[ 1 \quad 1(-12.000,-1.000) \quad 4 \quad R1 \]

**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.
**LVS Results**

**SVDB Cross-Reference Files**

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, then additional information about logic gates is provided. Specifically, transistor pairs that belong to logic gates are marked with the letters G (gate start) or GC (gate continuation). 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.
```

**Flat Net Cross-Reference File**

The Calibre -nxf command line switch instructs LVS to generate a flat net cross-reference file. This is discussed on page 3-18.
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. If you do not specify a Layout Primary statement in the rule file, layout_primary defaults to ICV_UNNAMED_TOP.

The net cross-reference file is a text file that contains matched nets. 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 leave nets unmatched and still be successful. 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.


**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. These files establish layout-to-source correspondence in Calibre RVE and Calibre xRC 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” on page 13-53. After the SVDB header, the file contains one section for each LVS correspondence cells (or hcells). There is one section for each hcell. Each section begins with a % line specifying the layout cell name and pin count and the source cell name and pin count. This is followed by lines of corresponding layout-source elements in the cell. 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
...
...
```

The format of individual instance or net lines is discussed in sections “Flat Instance Cross-Reference File” on page 13-48 and “Flat Net Cross-Reference File” on page 13-50.
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, Query Server, and Calibre xRC 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 the 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:

```plaintext
# 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:

```plaintext
# 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 file, and source netlist 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
```

Binary Polygon File (BPF) Database

The BPF database provides an interface from flat Calibre LVS to external tools. It provides access to shapes on layers involved in connectivity extraction and device recognition, and to some other related information. You create the BPF database with the -bpf command line option in flat Calibre LVS. You cannot create the BPF database in Calibre LVS-H. The database consists of a set of files, described next.

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 reported. You can use the Calibre LVS -dblayers command line option to explicitly select layers for generation.

Polygons in BPF files are annotated with node numbers (for Connect layers, Device pin layers, and Stamp layers) 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:

\[
\text{ngate1} = \text{COPY ngate} \\
\text{DEVICE MN ngate1 ngate(G) nact(S) nact(D) dpsub(B)}
\]

Situations where the same layer is used in a Connect operation and as a Device seed layer are reported with warnings in the Calibre transcript. For example:

```
WARNING: BPF file for DEVICE seed layer ngate contains NET IDs, not DEVICE IDs.
```

**BPF Layout Cross Reference File** — The layout cross-reference file, lvs_report.lxf, is a text file that provides a cross reference between internal net numbers and layout text names.

**BPF Ports File** — The lvs_report.ports file is a text file that contains information about top-level ports. This file contains one line for each top-level port (unattached ports are not reported). Each line has the following fields:

\[
\text{port_name node_number node_name port_location port_layer_attached}
\]

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 of a vertex on the port polygon marker when using Port Layer Polygon.

- **port_layer_attached** — layer of the polygon to which the port was 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
```

**Warning for Connect Layers** — The BPF interface provides a warning for situations where a layer is used in a Connect operation that is also a Device seed layer. The BPF interface was designed with the assumption that Device seed layers would not be used in Connect operations. However, in some cases, rule files may use Device seed layers in Connect operations. As a result, the BPF output file has net IDs rather than the expected device IDs. This can break
downstream flows that expect device IDs. The warning helps to detect and correct these situations.

Note that, in general, it is not necessary for Device seed layers to appear in Connect operations. In certain rare situations, layer derivations may require that connectivity be established through Device seed layers. In these cases, the layer intended for use as a Device seed layer can be copied for that purpose.

For example if a Device seed layer “nfet” also appears in a Connect operation and the -bpf switch is used:

```
CONNECT nfet poly
DEVICE MN nfet poly sd sd nwell
```

Then the following warning appears in the transcript:

```
WARNING: BPF file for DEVICE seed layer nfet contains NET IDs, not DEVICE IDs.
```

This situation may be rectified in one of two ways:

- If the Connect operation is not adding required connectivity to the extraction, remove it from the rule file. This saves processing time.

- If the Connect operation is adding required connectivity to the extraction, make a copy of the “nfet” layer for use in the device operation:

```
nfet:1 = COPY nfet
CONNECT nfet poly
DEVICE MN nfet:1 poly sd sd nwell
```
Chapter 14
SPICE Format

Introduction

This chapter discusses the SPICE syntax used for LVS. Knowledge of the SPICE language is assumed.

A hierarchical SPICE netlist may serve as a source of connectivity for hierarchical or flat LVS. LVS parses the netlist and compares it to the layout. The SPICE dialect 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 Cadence 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 you specify a top-level subcircuit name 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 have arbitrary parameters.

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.
General SPICE Syntax

SPICE Notational Conventions

Table 14-1 shows the notational conventions used to describe the SPICE syntax in the following sections.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; &gt;</td>
<td>Indicates an optional argument.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td>Indicates repetition (zero or more times).</td>
</tr>
<tr>
<td>UPPER</td>
<td>Upper case letters indicate literal keywords.</td>
</tr>
<tr>
<td>lower</td>
<td>Lower case letters indicate arguments to be substituted by other values.</td>
</tr>
<tr>
<td>+</td>
<td>SPICE continuation character used when the syntax description spans across several lines.</td>
</tr>
<tr>
<td>bold</td>
<td>LVS uses bold arguments.</td>
</tr>
<tr>
<td>italic</td>
<td>LVS does not use italicized arguments but does check their syntax.</td>
</tr>
<tr>
<td>&lt;ARG=val&gt;</td>
<td>Unless indicated otherwise, optional arguments preceded by literal names can appear in any order.</td>
</tr>
<tr>
<td>&lt;val&gt;</td>
<td>Arguments not preceded by literal names must appear exactly in the order shown.</td>
</tr>
</tbody>
</table>

Case Sensitivity

Unless indicated otherwise, all names, identifiers and keywords are case-insensitive. This includes node names, element names, subcircuit names, parameter names, and scaling factors. You may enable case-sensitive parsing with the Layout Case and Source Case specification statements.

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.
• **Characters used** — Alphanumeric characters are allowed for nets and devices.

• **Characters restricted** — The following characters are problematic for use in node names, element names, subcircuit names, subcircuit call names and other names in LVS SPICE netlists:
  
  o White space (space, tab, carriage-return, newline, vertical-tab, form-feed, and so forth) cannot be used.
  
  o Comma (,) and equals sign (=) should be avoided.
    
    These may actually be accepted in instance names and subcircuit call names, but they are best avoided.
  
  o Dollar sign ($) cannot appear at the beginning of a name but may be used elsewhere.
  
  o Slash (/) may be used, but is not recommended, because it has special meaning and normally splits up names (unless overridden with LVS Spice Slash Is Space NO).
  
  o Control characters are not recommended.

Other characters not listed in this set may be used.

• **White space** — White space characters include: space, tab, cr, If, vt, ff, “,”

• **Numeric value types** —

  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   A = 1E-18

The following are syntactical considerations for specific device types:

• **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 L= and W= prefixes; in this case, the values should appear as the first two tokens in the string. You can specify L= and W= in upper or lower case. The values for length and width are numeric values with units of meters, and you can use them 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
  ```
Arithmetic Expression

Generally, arithmetic expressions can appear anywhere a number can appear. Valid operators are: +, -, *, and /. You can use parentheses ( ) 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. 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. The * extensions are used to add entire statements to the language. The $ extensions are used to add new fields to existing SPICE statements. For example:

```
*.CONNECT 10 20
  C1 1 2 100 $A=100 $P=40
```

Because 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

Syntax:

```
.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.

.ENDOR

Syntax:

.ENDL <comment>

Indicates the end of a .LIB section. Any statements after a .ENDL are ignored, but a valid comment may appear after .ENDL.

.INCLUDE

Syntax:

.INCLUDE <pathname>

-or-

.INC <pathname>

where pathname specifies a file. You can enclose pathname in single or double quotation marks. 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
```

When a filename is given in a .INCLUDE statement, it is searched for according to these criteria:

1. Literally, as an absolute path.
2. As a relative path from the current directory, that is, the one from which LVS was invoked.
3. As a relative path from the location of the file containing the .INCLUDE statement.

The last search option allows a library to contain internal relative paths, and only the first reference (.INCLUDE) to the library from the user’s description needs to have the full path to the library. The library can thus be relocated without internal changes, or even continue to operate if different users see it through different nfs mount points. On the other hand, the user can override specific library files with files in the current directory, because it is searched first.

You can specify environment variables as components of file name (or path name) parameters in .INCLUDE statements. Environment variables are signified by preceding them with the
dollar sign ($) character. They are evaluated and substituted into the file name string. For example, these are all valid:

```
.INCLUDE $FOO
.INCLUDE "$BAR"
.INCLUDE "$ENV1/foo/$ENV2/bar"
```

If $ENV1 is defined as “/dir1” and $ENV2 is defined as “dir2”, then the last statement in the example is evaluated as:

```
.INCLUDE "./dir1/foo/dir2/bar"
```

To make the notion of *filename component* precise, a file name parameter is defined by the following BNF:

```
filename -> [“/”] path_str
path_str -> [“$”] string [ “/” path_str ]
string -> (any sequence of allowed characters except “/”)
```

where $ designates that the string immediately following is interpreted as an environment variable.

It is a compilation error for an environment variable required by file name evaluation to be undefined, or be defined with only a nil value, except as described next.

To ensure compatibility with earlier versions, in the unlikely event that a file exists with a literal name equal to the name of the environment variable, including the $ sign, then that name takes precedence and the environment variable is not evaluated. For example, if there is a file called $FOO, then the statement:

```
.INCLUDE $FOO
```

includes that file, regardless of the value of the $FOO environment variable, and regardless of whether $FOO is defined or not.

**.LIB**

Syntax:

```
.LIB <section_name>
-or-
.LIB <lib_name> <section_name>
```

The .LIB statement comes in two forms, a definition form and a reference form.

The definition form has only a *section_name* parameter, where *section_name* is any single-word SPICE character string. This format defines a library section with the given name, which
includes all lines of the current source file up to the next occurrence of a .ENDL statement, or the end of the file. Note that the nesting of definitions is not allowed. If a nested definition is attempted, the previously-open section is terminated and a new section is opened, rather than generating an error. This is compatible with some dialects of SPICE, but not others.

The reference form of the .LIB statement has `lib_name` and `section_name` parameters. This statement causes the file `lib_name` to be opened and the contents of the section `section_name` within that file to be included. All other parts of the included file are ignored, regardless of syntax, as long as there are no syntax errors in .LIB or .ENDL statements within the file, or in the referenced section. This applies to .INCLUDE statements as well, which means that .LIB definitions must appear in the file `lib_name` and not in any files included from that file.

If the file `lib_name` contains multiple definitions of the same section `section_name`, the contents of all sections are concatenated and included. If file `lib_name` does not contain a section with `section_name`, LVS generates a warning.

Note that library definitions can reference other libraries, and while a particular `lib_name` - `section_name` pair cannot reference itself, it can reference other sections within the same file `lib_name` or the same `section_name` within another file. It is possible to generate a multi-file or multi-section loop. This error is detected as soon as a reference is made to an already open `lib_name` - `section_name` pair.

When a filename is given in a .LIB statement, it is searched for according to these criteria:

1. Literally, as an absolute path.
2. As a relative path from the current directory, that is, the one from which LVS was invoked.
3. As a relative path from the location of the file containing the .LIB statement.

The last search option allows a library to contain internal relative paths, and only the first reference (.LIB) to the library from the user’s description needs to have the full path to the library. The library can thus be relocated without internal changes, or even continue to operate if different users see it through different nfs mount points. On the other hand, the user can override specific library files with files in the current directory, because it is searched first.

**.OPTION SCALE**

Syntax:

```
.OPTIONS <option> …
-or-
.OPTION <option> …
-or-
.PC <option> …
```
-or-

`.CONTROL <option> ...`

You can specify any option, but SPICE ignores all options except SCALE.

Options can have these formats:

\[
\begin{align*}
\text{opt} \\
\text{opt}=x
\end{align*}
\]

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 that is read is 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:

\[\text{SCALE}=x\]

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:

```plaintext
.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
```
*.BUSDELMITER

Syntax:

*.BUSDELMITER symbol

where symbol is one of the characters [, {, <, or (.

This statement is used exclusively in v2lvs (the Calibre LVS Verilog reader). It specifies SPICE bus delimiter characters for v2lvs. This is explained under “Verilog-to-LVS” on page 15-1.

This statement is coded as comment. The Calibre LVS SPICE parser checks this statement for syntax but otherwise ignores it.

*.CAPA

Syntax:

*.CAPA

This statement instructs LVS to ignore capacitor (C) elements in the netlist. It is coded as a comment and has no arguments.

*.CONNECT

Syntax:

*.CONNECT -or- *

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-defined 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: power and ground names, global nets, user-given names, and subcircuit pin names.
Nets specified in *.CONNECT statements are added to the circuit description seen by LVS, even if they do not appear in any other statements in the netlist. For example, in the following netlist:

```
C1 1 3
*.CONNECT 1 2
*.CONNECT 2 3
```

nets 1 and 3 get connected through net 2 by the pair of *.CONNECT statements.

The *.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.

The *.CONNECT and *.J statements that appear in a SPICE netlist inside of a subcircuit definition apply to all global nets, as usual, but they do not apply to non-global nets outside of the subcircuit definition in which these statements occur.

In the following example, global nets VCC1 and VCC2 are connected together. Local nets A and B in SUB1 are also connected together, but local nets A and B in SUB2 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
```

You must use the *.CONNECT or *.J statements to join together different nets in a netlist. The *.EQUIV statement specifies correlation only between different source and layout names.

### *.DIODE

Syntax:

```
*.DIODE
```

This statement instructs LVS to ignore diode (D) elements in the netlist. It is coded as a comment and has no arguments.
**.EQUIV**

Syntax:

```
*.EQUIV < new_name = old_name > …
```

This statement is coded as a comment. It typically specifies equivalence between different source and layout names. It appears 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 for 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 for 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
```
The *.EQUIV statement 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

Syntax:

*.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 YES specification statement in the rule file, the SPICE reader processes SLDD designators in M elements only if a *.LDD statement is present somewhere in the netlist. It ignores $LDD designators otherwise. When you specify the secondary keyword NO, or when the LVS Spice Conditional LDD statement does not appear in the rule file, the *.LDD statement has no effect on LVS. The following example shows how to use the *.LDD statement in SPICE syntax:

* .LDD
M2 4 5 6 7 P $LDD[PPP]

*.MEGA

Syntax:

*.MEGA

This statement tells the LVS SPICE parser to interpret uppercase “M” scale factors as “Mega” (1E+6) instead of the usual “milli” (1E-3). Lower case “m” scale factors are interpreted as milli. The default in the absence of *.MEGA statements is to interpret both uppercase M and lower case m as milli (1E-3). This statement is coded as comment and has no arguments. It may appear in the netlist any number of times.

For example, the normal semantics are:

R1 1 2 7M $ 7E-3
R2 3 4 7m $ 7E-3

With *.MEGA, the semantics are:

* .MEGA
R1 1 2 7M $ 7E+6
R2 3 4 7m $ 7E-3

Generally, the *.MEGA statement has global scope; it applies to numeric expressions that appear anywhere in the netlist, before or after the *.MEGA statement. The *.MEGA statement may appear within or outside of subcircuit definitions; in both cases, it applies globally.
However, there is one exception to this rule: when applied to other statements with global scope, specifically `.PARAM` and `.OPTIONS` (or equivalent syntax), the application of `*.MEGA` is order-dependent. `*.MEGA` is not applied to numeric expressions within global-scope `.PARAM` or `.OPTIONS` statements that appear prior to it in the netlist. (Note however, that when `LVS Spice Redefine Param YES` is specified, `.PARAM` statements contained within subcircuit definitions have local scope and `*.MEGA` applies to them regardless of their relative position in the netlist).

Consider this example:

```
.SUBCKT SSS
R1 1 2 R = 9M
.ENDS

.OPTIONS SCALE = 1M
.PARAM AA = 4M
*.MEGA
.PARAM BB = 5M

.SUBCKT TOP
C1 1 2 100 W=7 NA=AA NB=BB NC=8M
X2 SSS
.ENDS
```

In the example, `*.MEGA` applies to the `.PARAM BB` statement but not to the `.OPTIONS SCALE` statement or the `.PARAM AA` statement. Thus, the values are as follows:

- `SCALE = 1E-3` (not `1E+6`)
- `AA = 4E-3` (not `4E+6`)
- `BB = 5E+6` (`*.MEGA` applied)

For capacitor C1 in TOP:

- `W = 7E-3` (SCALE applied)
- `NA = 4E-3` (from AA parameter)
- `NB = 5E+6` (from BB parameter)
- `NC = 8E+6` (`*.MEGA` applied globally)

For resistor R1 in SSS:

- `R = 9E+6` (`*.MEGA` applied globally).

The SPICE parser issues warnings when it finds global-scope `.PARAM` or `.OPTIONS` (or equivalent) statements that appear prior to `*.MEGA`. For the previous example:

```
Warning: *.MEGA at line 7 in file "z2" not applied to earlier global-scope .PARAM statements
Warning: *.MEGA at line 7 in file "z2" not applied to earlier global-scope .OPTION (or equivalent) statements
```
**.PININFO**

Syntax:

```
*.PININFO <text>
```

The comment-coded statement *.PININFO is silently ignored. The keyword *.PININFO may be followed by arbitrary text, which is ignored as well. Any associated “+” continuation lines are interpreted as a continuation of the *.PININFO statement and are ignored.

**.SEEDPROM**

Syntax:

```
*.SEEDPROM
```

This statement is designed for use with extracted layout netlists. When this statement appears inside a subcircuit, it indicates that seed promotion occurred during hierarchical device recognition in the layout cell represented by the subcircuit. Seed promotion occurs when hierarchical device recognition is unable to completely process devices within a cell, so device recognition promotes those devices out of the cell and places them at a higher level of hierarchy.

The LVS hierarchical SPICE netlister places *.SEEDPROM statements in the extracted layout netlist in order to communicate seed promotion events to the LVS circuit comparison module. Hierarchical circuit comparison can be instructed to expand cells containing *.SEEDPROM statements by means of the LVS Expand Seed Promotions specification statement.

This statement is coded as comment and takes no arguments. It may appear any number of times in the netlist, and any number of times in a given subcircuit. It may appear anywhere in a subcircuit. This statement has no meaning when it appears outside of subcircuit definitions. This statement is ignored when it appears inside LVS Box cells. This statement is treated like an element statement for the purpose of determining primitive subcircuits; in other words, when placed inside an otherwise empty subcircuit, it renders the subcircuit non-primitive.

**.XPINS**

Syntax:

```
*.XPINS
```

This statement supports the e2lvsp 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.
*X.PINS $ Subcircuit has explicit pin connections.
*.J 1 ==B $ Join net 1 with pin 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. In the following example, capacitor C1 is connected to nets A and B in SS2 but not to the respective pins A and B:

```
.SUBCKT SS2 A B
*X.PINS
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, there are two different nets named A in SS2. One connects to the subcircuit pin and one connects to capacitor C1. Similarly, there are two nets named 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 transcribed.

```
.AC  .IC  .NOISE  .PROTECT  .TF
.DC  .MEASURE  .OP  .PZ  .TITLE
.DCVOLT  .MODEL  .PLOT  .SAMPLE  .TRAN
.DISTO  .NET  .PRINT  .SENS  .UNPROTECT
```
Element Statements

This section contains descriptions of SPICE elements used by LVS. For each element there is a
syntactical representation, followed by examples, followed by a table describing the syntactical
elements of the statement.

SPICE arguments and Trace Property names — In each table in this section, the first
column shows the netlist arguments that appear in the syntax description. The last column
shows Trace Property arguments that correspond to various netlist arguments. When tracing
properties of built-in devices during LVS, you should use the parameters shown in the Trace
Property Name column, exactly as they are shown, not the parameter name in the SPICE netlist.

Resistor Element

\textbf{Rxxx n1 n2 <mname> <r tcl tc2 <scale m <ac>>>><}
+ <L=l> <W=w> <parnam=pval> ... <$SUB=ns>
+ <$[mname]$ | $.MODEL=mname> <$W=w> <$L=l> <$X=x> <$Y=y>
+ <$D=d>

\textbf{Rxxx n1 n2 <mname> <r TC=tcl tc2 <scale>>> <M=m> <AC=ac>
+ <L=l> <W=w> <parnam=pval> ... <$SUB=ns>
+ <$[mname]$ | $.MODEL=mname> <$W=w> <$L=l> <$X=x> <$Y=y>
+ <$D=d>

\textbf{Rxxx n1 n2 <R=r> <TC1=tcl1> <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> <$Y=y>
+ <$D=d>

Table 14-2 shows the arguments. The following are some examples:

\begin{verbatim}
R1 1 2 4 $comment
R2 n5 n6 4 TC=2 3 M=3 W=10U L=20U AAA=5 BBB="zz" $[x]
R3 na nb 4 1 1 4 M=3 $SUB=3 $.MODEL=x $W=10U $L=20U
\end{verbatim}

To enter resistor devices with more than one substrate pin, define a primitive subcircuit as
described in the section “.SUBCKT, .SUBCIRCUIT, .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 a previously defined parameter), then LVS interprets that token as model name.

### Table 14-2. Resistor Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rxxx</td>
<td>Resistor element name. Must begin with an R followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n1</td>
<td>Positive terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n2</td>
<td>Negative terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>mname</td>
<td>Optional model name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Resistance in ohms.</td>
<td>r</td>
</tr>
<tr>
<td>tc1</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>tc2</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>scale</td>
<td>Optional scale factor. Multiplies resistance (r).</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>Optional multiplier factor used to simulate multiple parallel resistors. Normally, divides resistance (r), multiplies width (w). If \texttt{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. Alternative multiplier names can be specified with the \texttt{LVS Spice Multiplier Name} rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>ac</td>
<td>Optional AC resistance for AC analysis.</td>
<td>ac</td>
</tr>
<tr>
<td>l</td>
<td>Optional length.</td>
<td>l</td>
</tr>
<tr>
<td>w</td>
<td>Optional width.</td>
<td>w</td>
</tr>
<tr>
<td>parnam=pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>ns</td>
<td>Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>${mname}$ or $.MODEL =mname</td>
<td>Optional model name, coded as a comment. String of any number of alphanumeric characters. Overrides the regular optional mname parameter.</td>
<td></td>
</tr>
</tbody>
</table>
Capacitor Element

\[ C_{xxx \; n1 \; n2} \; mname <c <tc1 <tc2 <scale <ic <m>>>>> <IC=ic> <M=m> + <L=l> <W=w> <A=a> <P=p> <parnam=pval> … <$SUB=ns> + <$[mname] | $.MODEL=mname> <$A=a> <$P=p> <$X=x> <$Y=y> + <$D=d> \\]

Table 14-3 shows the arguments. The following are some examples:

\[
\begin{align*}
C1 & \; 2 \; 10P \\
C3 & \; n1 \; n2 \; 10P \; M=4 \; W=10U \; L=20U \; AAA=5 \; BBB=\text{"zz"} \; \$[mc] \; \$comment \\
C4 & \; n1 \; n2 \; 10P \; 1 \; 1 \; 4 \; \$SUB=3 \; \$.MODEL=mc \; \$A=10P \; \$P=40U \\
\end{align*}
\]

To enter capacitor devices with more than one substrate pin, define a primitive subcircuit as described in section “.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO.”
LVS first tries to interpret the fourth token as a capacitance 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 14-3. Capacitor Element**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cxxx</td>
<td>Capacitor element name. Must begin with a C followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n1</td>
<td>Positive terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n2</td>
<td>Negative terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>mname</td>
<td>Optional model name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>Capacitance in farads.</td>
<td>c</td>
</tr>
<tr>
<td>tc1</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>tc2</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>scale</td>
<td>Optional scale factor. Multiplies capacitance (c).</td>
<td></td>
</tr>
<tr>
<td>ic</td>
<td>Optional initial voltage across the capacitor in volts.</td>
<td>ic</td>
</tr>
<tr>
<td>m</td>
<td>Optional multiplier factor used to simulate multiple parallel capacitors. Normally, multiplies 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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>l</td>
<td>Optional length.</td>
<td>l</td>
</tr>
<tr>
<td>w</td>
<td>Optional width.</td>
<td>w</td>
</tr>
<tr>
<td>a</td>
<td>Optional area.</td>
<td>a</td>
</tr>
<tr>
<td>p</td>
<td>Optional perimeter.</td>
<td>p</td>
</tr>
<tr>
<td>parnam=</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>pval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ns</td>
<td>Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
</tbody>
</table>
**Inductor Element**

\[
\text{Lxxx n1 n2} <l <tc1 <tc2>> <SCALE=scale> <M=m> <R=r> \\
+ <parnam=pval> … <$SUB=ns> <$[mname] | $.MODEL=mname> \\
+ <$X=x> <$Y=y> <$D=d>
\]

\[
\text{Lxxx n1 n2} <L=l> <TC1=tc1> <TC2=tc2> <SCALE=scale> <M=m> <R=r> \\
+ <parnam=pval> … <$SUB=ns> <$[mname] | $.MODEL=mname> \\
+ <$X=x> <$Y=y> <$D=d>
\]

Table 14-4 shows the arguments. The following are some examples:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[mname]$ or $.MODEL=mname</td>
<td>Optional model name, coded as a comment. String of any number of alphanumeric characters. Overrides the regular optional mname parameter</td>
</tr>
<tr>
<td>$A=a</td>
<td>Optional area. Coded as comment. Overrides A, if present.</td>
</tr>
<tr>
<td>$P=p</td>
<td>Optional perimeter. Coded as comment. Overrides P, if present.</td>
</tr>
<tr>
<td>$X=x $Y=y</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.</td>
</tr>
<tr>
<td>$D=d</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device. Parsed but not used in LVS.</td>
</tr>
</tbody>
</table>

The Dracula CDL statement *.CAPA is supported. It instructs LVS to ignore capacitor elements in the netlist.
To enter inductor devices with more than one substrate pin, define a primitive subcircuit as described in the section “.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO.”

### Table 14-4. Inductor Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lxxx</td>
<td>Inductor element name. Must begin with an L followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n1</td>
<td>Positive terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>n2</td>
<td>Negative terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Optional inductance in henrys.</td>
<td>l</td>
</tr>
<tr>
<td>tc1</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>tc2</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>scale</td>
<td>Optional scale factor. Multiplies inductance (l) and resistance (r).</td>
<td>m</td>
</tr>
<tr>
<td>m</td>
<td>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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>r</td>
<td>Optional resistance in ohms.</td>
<td>r</td>
</tr>
<tr>
<td>w</td>
<td>Optional width.</td>
<td>w</td>
</tr>
<tr>
<td>parnam= pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>ns</td>
<td>Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>$[mname]$ or $.MODEL= mname</td>
<td>Optional model name, coded as a comment. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
</tbody>
</table>
| $X=x$  
| $Y=y$     | Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only. |                     |
SPICE Format
Element Statements

Table 14-4. Inductor Element (cont.)

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D=d</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device. Parsed but not used in LVS.</td>
<td></td>
</tr>
</tbody>
</table>

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> <$Y=y> <$D=d>
```

```
Dxxx nplus nminus mname <a <pj>> <M=m> <OFF>
+ <parnam=pval> … <$SUB=ns> <$X=x> <$Y=y> <$D=d>
```

Table 14-5 shows the arguments. The following are some examples:

```
D1 1 2 mdio
D2 a b mdio 2P 3U M=3 AAA=5 BBB="zz"
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, .SUBCIRCUIT, .SUB, or .MACRO.”

Table 14-5. Junction Diode Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dxxx</td>
<td>Diode element name. Must begin with a D followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nplus</td>
<td>Positive (anode) terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nminus</td>
<td>Negative (cathode) terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>mname</td>
<td>Model name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Optional diode area.</td>
<td>a</td>
</tr>
<tr>
<td>pj</td>
<td>Optional periphery of junction.</td>
<td>p</td>
</tr>
<tr>
<td>m</td>
<td>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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>OFF</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>parnam=pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>ns</td>
<td>Optional substrate terminal node name coded as a comment. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>$X=x$</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.</td>
<td></td>
</tr>
<tr>
<td>$Y=y$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D=d$</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device.Parsed but not used in LVS.</td>
<td></td>
</tr>
</tbody>
</table>

**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> <W=w> <L=l> <M=m> <OFF>
+ <parnam=pval> ... <$SUB=ns> <$EA=a> <$W=w> <$L=l>
+ <$X=x> <$Y=y> <$D=d>
```

```
Qxxx nc nb ne <[ns]> <ns> mname <a> <W=w> <L=l> <M=m> <OFF>
+ <IC=vbe, vce> <parnam=pval> ... <$SUB=ns> <$EA=a>
+ <$W=w> <$L=l> <$X=x> <$Y=y> <$D=d>
```

Table 14-6 shows the arguments. 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="zz"
+ $L=2U $EA=4P $W=6U $comment
```

To enter BJT devices with more than one substrate pin, define a primitive subcircuit as described in the section “.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO.”

### Table 14-6. BJT Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qxxx</td>
<td>BJT element name. Must begin with a Q followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nc</td>
<td>Collector terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nb</td>
<td>Base terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>ne</td>
<td>Emitter terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>[ns]</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>ns</td>
<td>Optional substrate terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>mname</td>
<td>Model name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Optional emitter area.</td>
<td>a</td>
</tr>
<tr>
<td>w</td>
<td>Optional width.</td>
<td>w</td>
</tr>
</tbody>
</table>
### Table 14-6. BJT Element (cont.)

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Optional length.</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>OFF</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>parnam=pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>vbe</td>
<td>Used (Optional).</td>
<td></td>
</tr>
<tr>
<td>vce</td>
<td>Used (Optional).</td>
<td></td>
</tr>
<tr>
<td>$SUB=ns</td>
<td>Optional substrate terminal node name, coded as a comment. Overrides regular SPICE substrate terminal field, if present.</td>
<td></td>
</tr>
<tr>
<td>$EA=a</td>
<td>Optional emitter area, coded as a comment. Overrides &lt;AREA=a&gt; or &lt;a&gt;. A practical use would be to use $EA in the source, calculate the A property in the layout, then use Trace Property Q A A 1 for LVS comparison.</td>
<td>a</td>
</tr>
<tr>
<td>$W=w</td>
<td>Optional width, coded as comment.</td>
<td>w</td>
</tr>
<tr>
<td>$L=l</td>
<td>Optional length, coded as comment.</td>
<td>1</td>
</tr>
<tr>
<td>$X=x</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.</td>
<td></td>
</tr>
<tr>
<td>$Y=y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D=d</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device. Parsed but not used in LVS.</td>
<td></td>
</tr>
</tbody>
</table>

**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 interprets it as a 3-pin device.
JFET Element

\[ J_{xxx} \text{ nd ng ns} \ <nb> \ mname \ <\text{AREA}=a> \ <W=w> \ <L=l> \ <M=m> \ <\text{OFF}> \]
\[ + \ <\text{IC}=vds, vgs> \ <\text{parnam}=pval> \ … \ <$\text{SUB}=nb> \ <$X=x> \ <$Y=y> \ <$D=d> \]

Table \ref{tab:jfet_element} shows the arguments. The following are some examples:

\begin{verbatim}
J1 10 24 13 JM1
Jabc netd netg nets jmod AREA=8P W=3U L=7U M=2 AAA=5 BBB="zz"
J234 netd netg nets jmod 8P 3U 7U M=2 $SUB=netb
\end{verbatim}

To enter JFET devices with more than one substrate pin, define a primitive subcircuit as described in the section “.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO.”

\begin{table}[h]
\centering
\begin{tabular}{|c|p{10cm}|}
\hline
Argument Name & Description & Trace Property Name \\
\hline
Jxxx & JFET element name. Must begin with a J followed by any number of alphanumeric characters. & \\
\hline
nd & Drain terminal node name. String of any number of alphanumeric characters. & \\
\hline
ng & Gate terminal node name. String of any number of alphanumeric characters. & \\
\hline
ns & Source terminal node name. String of any number of alphanumeric characters. & \\
\hline
<nb> & Optional bulk connection node name. String of any number of alphanumeric characters. & \\
\hline
mname & Model name. String of any number of alphanumeric characters. & \\
\hline
a & Optional area. & a \\
\hline
w & Optional gate width. & w \\
\hline
l & Optional gate length. & l \\
\hline
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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement. & m \\
\hline
OFF & Ignored. & \\
\hline
\end{tabular}
\caption{JFET Element}
\end{table}
### 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> <$Y=y> <$D=d>

*Mxxx nd ng ns* <nb> mname <l> <w> <ad> <as> <pd> <ps> + <nr> <nrs> <rdc> <rsc> <OFF> <IC=vds, vgs, vbs> <M=m> + <parnam=pval> … <$LDD<type>> <$X=x> <$Y=y> <$D=d>

Table 14-8 shows the arguments. 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="zz"
Ma2 netd netg nets netb pmos 3U 2U 4P 5P 6U 7U M=2
```
### Table 14-8. MOSFET Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mxxx</td>
<td>MOSFET element name. Must begin with an M followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nd</td>
<td>Drain terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>ng</td>
<td>Gate terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>ns</td>
<td>Source terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nb</td>
<td>Optional bulk terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>mname</td>
<td>Model name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>Optional channel length.</td>
<td>l</td>
</tr>
<tr>
<td>w</td>
<td>Optional channel width.</td>
<td>w</td>
</tr>
<tr>
<td>ad</td>
<td>Optional drain area.</td>
<td>ad</td>
</tr>
<tr>
<td>as</td>
<td>Optional source area.</td>
<td>as</td>
</tr>
<tr>
<td>pd</td>
<td>Optional drain perimeter.</td>
<td>pd</td>
</tr>
<tr>
<td>ps</td>
<td>Optional source perimeter.</td>
<td>ps</td>
</tr>
<tr>
<td>nrd</td>
<td>Optional number of squares of drain diffusion.</td>
<td>nrd</td>
</tr>
<tr>
<td>nrs</td>
<td>Optional number of squares of source diffusion.</td>
<td>nrs</td>
</tr>
<tr>
<td>rdc</td>
<td>Optional additional drain resistance due to contact resistance.</td>
<td>rdc</td>
</tr>
<tr>
<td>rsc</td>
<td>Optional additional source resistance due to contact resistance.</td>
<td>rsc</td>
</tr>
<tr>
<td>OFF</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>vds</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>vgs</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>vbs</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>Optional multiplier factor to simulate multiple MOSFETs. Normally, multiplies w, ad, as, and ps. If <strong>LVS Spice Replicate Devices</strong> 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. Alternative multiplier names can be specified with the <strong>LVS Spice Multiplier Name</strong> rule file statement.</td>
<td>m</td>
</tr>
</tbody>
</table>
LVS interprets 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 you do not use the prefixes L= and W=, then l must precede w.

To enter MOS devices with more than one substrate pin, define primitive subcircuits as defined under “Primitive Subcircuits” on page 14-35.

Other characters can follow the parameter names W and L. Any parameter name that starts with W is interpreted as width, and any parameter that starts with L is interpreted as length. This behavior can be altered by specifying LVS Spice Strict WL YES in your rule file.

**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

---

**Table 14-8. MOSFET Element (cont.)**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>parnam=pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>$LDD</td>
<td>Optional LDD device designator, coded as a comment.</td>
<td></td>
</tr>
<tr>
<td>(none)</td>
<td>Optional implied W and L values. Computed if LVS Spice Implied MOS Area is specified in the rule file.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional LDD device type. String of any number of alphanumeric characters. Allowed (but not required) only in conjunction with $LDD. If specified, replaces mname.</td>
<td></td>
</tr>
<tr>
<td>$X=x</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.</td>
<td></td>
</tr>
<tr>
<td>$Y=y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D=d</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device. Parsed but not used in LVS.</td>
<td></td>
</tr>
</tbody>
</table>
SPICE Format

Element Statements

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).

Voltage Source Element

Vxxx nplus nminus <<DC <=>> dc> <M=m> <parnam=pval> …
+ <$X=x> <$Y=y> <$D=d>

Table 14-9 shows the arguments. The following are some examples:

V1 n1 n2 5
V2 n1 n2 DC 5
V3 n1 n2 DC=5 AAA=5 BBB=”zz”

Table 14-9. Voltage Source Element

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
<th>Trace Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vxxx</td>
<td>Voltage source element name. Must begin with a V followed by any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nplus</td>
<td>Positive terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>nminus</td>
<td>Negative terminal node name. String of any number of alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>dc</td>
<td>Optional DC and transient analysis value of the source.</td>
<td>dc</td>
</tr>
<tr>
<td>parnam=pval</td>
<td>Optional parameter name set to a numeric or string value. Arbitrary parameter names are allowed. The parnam must begin with a letter followed by any number of alphanumeric characters or underscores (_). You can specify any number of parnam=pval pairs.</td>
<td>parnam</td>
</tr>
<tr>
<td>m=</td>
<td>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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
<td>m</td>
</tr>
<tr>
<td>$X=x</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units. Used in LVS-H only.</td>
<td></td>
</tr>
<tr>
<td>$Y=y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LVS component type: V
LVS component subtype: none
LVS pin names: pos (positive), neg (negative).

Specification Statements for SPICE M Elements

These SVRF specification statements apply to handling SPICE M elements. You may use them for related situations that apply to your design.

Table 14-10. Specification Statements for SPICE M Elements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVS Spice Implied MOS Area</td>
<td>Indicates whether LVS should compute implied area values for MOS elements (element M) in SPICE netlists.</td>
</tr>
<tr>
<td>LVS Spice Multiplier Name</td>
<td>Specifies parameter names that should serve as multiplier factors in SPICE netlists instead of the standard SPICE parameter name M.</td>
</tr>
<tr>
<td>LVS Spice Replicate Devices</td>
<td>Indicates whether the LVS SPICE parser should physically replicate device elements in a SPICE netlist that own the M parameter.</td>
</tr>
</tbody>
</table>

Subcircuits

.SUBCKT, .SUBCIRCUIT, .SUB, or .MACRO

The following syntactical elements are equivalent:

.SUBCKT subname < n1 n2 ... < / m1 m2 ... > > < parnam = pval > …

.SUBCIRCUIT subname < n1 n2 ... < / m1 m2 ... > > < parnam = pval > …

.SUB subname < n1 n2 ... < / m1 m2 ... > > < parnam = pval > …

.MACRO subname < n1 n2 ... < / m1 m2 ... > > < parnam = pval > …
These statements are all equivalent. Table 14-11 shows the arguments.

**Table 14-11. Subckt Statement**

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subname</td>
<td>Subcircuit name. String of any number of alphanumeric characters.</td>
</tr>
<tr>
<td>n1 n2 …</td>
<td>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.</td>
</tr>
<tr>
<td>/ m1 m2 …</td>
<td>Optional additional node names for external reference. The / characters in the node name list are treated as white space, unless you specified the LVS Spice Slash Is Space NO specification statement in your rule file.</td>
</tr>
<tr>
<td>parnam = pval</td>
<td>Optional parameter name set to a numeric or string 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. The parnam must begin with a letter followed by any number of alphanumeric characters.</td>
</tr>
</tbody>
</table>

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 pp="abc"
R1 in1 3 res aaa=pp
R2 3 in2 res aaa=pp
.ENDS res2
```

LVS Spice Option allows you to control the SPICE parser reporting of subcircuit calls with floating pins, subcircuits sharing the same name but with differing numbers of pins, and the modification of duplicate subcircuit names.
Primitive Subcircuits

A subcircuit that does not contain any element statements, subcircuit calls, *.CONNECT statements, *.J statements, *.SEEDPROM statements, and any subcircuit that appears in an LVS Box rule file statement, is considered a primitive component in LVS:

LVS component type: subname
LVS component subtype: none
LVS pin names: n1 n2 … (and optional m1 m2 …)

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 connects to C1 in bar through pin a of bar. Node 2 does not connect to any devices in bar. A warning is 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. (Nevertheless, you can tell the SPICE parser to report warnings for them by specifying LVS Spice Option S in the rule file.)

Subcircuit definitions with the same name and the same number of pins are considered duplicate. When duplicate subcircuit definitions are present in a netlist, one of those definitions is used and all others are discarded with warnings. You should not rely on any specific
subcircuit definition (first, last, or other) to be chosen. Duplicate subcircuit definitions are reported as warnings. For example:

    Warning: Duplicate subckt definition "AAA" at line 5 in file "foo"

**.ENDS or .EOM**

Syntax:

```plaintext
.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**

```plaintext
Xyyy < n1 n2 … > </ > <subname> < parnam = pval > … <M=m>
+ <$[mname] | $.MODEL=mname> <$T=tx ty r a>
+ <$X=x> <$Y=y> <$D=d> <$PINS <pin=node> … >
```

Table 14-12 shows the arguments. Here is an example:

```
X1 2 4 17 opamp wn=100 ln=5 pp="abc"
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xyyy</td>
<td>Subcircuit call name. Must begin with an X followed by any number of alphanumeric characters.</td>
</tr>
<tr>
<td>n1 n2 …</td>
<td>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.</td>
</tr>
<tr>
<td>/</td>
<td>Optional. The / characters in the node name list and in the subcircuit reference name are treated as white space, unless you specify the LVS Spice Slash Is Space NO specification statement in your rule file. See $PINS also.</td>
</tr>
<tr>
<td>subname</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 14-12. Subcircuit Calls (cont.)

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parnam = pval</td>
<td>Optional parameter name set to a numeric or string 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. (Within the referenced subcircuit, parameter values explicitly defined in a subcircuit call, or explicitly defined in the original subcircuit definition of a called subcircuit, are not overridden.) Overrides a parameter value assigned in the referenced subcircuit’s definition, but can be overridden by a value set in a .PARAM statement. The parameter does not have to be specified in the subcircuit definition in order to apply. The parnam must begin with a letter followed by any number of alphanumeric characters.</td>
</tr>
<tr>
<td>m</td>
<td>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. Alternative multiplier names can be specified with the LVS Spice Multiplier Name rule file statement.</td>
</tr>
<tr>
<td>mname</td>
<td>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.</td>
</tr>
<tr>
<td>$T=tx ty r a</td>
<td>Optional layout transform consisting of x translation, y translation, reflection and rotation-angle. Used in Calibre LVS-H only. The translation components tx and ty are integer numbers in database units. The reflection r is along the X axis and is either 0 or 1 which denote the absence or presence of reflection respectively. The rotation angle a is an integer number in degrees and the only valid values are 0, 90, 180, or 270; rotation is counter-clockwise.</td>
</tr>
<tr>
<td>$X=x</td>
<td>Optional X,Y coordinates coded as comments. Integer numbers in database units.</td>
</tr>
<tr>
<td>$Y=y</td>
<td></td>
</tr>
<tr>
<td>$D=d</td>
<td>Optional rule file Device operation identifier, coded as comment. Non-negative integer number. Designed for use in extracted layout netlists to identify the rule file Device operation that generated the device.Parsed but not used in LVS.</td>
</tr>
</tbody>
</table>
Table 14-12. Subcircuit Calls  (cont.)

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PINS &lt;pin=node&gt; …</td>
<td>Optional argument that specifies pin connections by name (as opposed to by order). Coded as a comment. Each pin is a pin name in the .SUBCKT definition, and each node is a node name in the subcircuit call. The specified node connects to the specified pin in the subcircuit. Pin and node names are strings of any number of alphanumeric characters. Embedded / characters are treated as part of the names. For example:</td>
</tr>
<tr>
<td></td>
<td>.SUBCKT FOO A B</td>
</tr>
<tr>
<td></td>
<td>$ subcircuit contents...</td>
</tr>
<tr>
<td></td>
<td>.ENDS</td>
</tr>
<tr>
<td></td>
<td>X1 FOO $PINS B=1 A=2</td>
</tr>
<tr>
<td></td>
<td>In subcircuit call X1, pin B connects to node 1 and pin A connects to node 2.</td>
</tr>
<tr>
<td></td>
<td>A subcircuit call may reference fewer pins than specified in the respective .SUBCKT definition; any pins that are not referenced in the subcircuit call are floating. For example:</td>
</tr>
<tr>
<td></td>
<td>.SUBCKT SSS A B C D</td>
</tr>
<tr>
<td></td>
<td>$ subcircuit contents...</td>
</tr>
<tr>
<td></td>
<td>.ENDS</td>
</tr>
<tr>
<td></td>
<td>X2 1 2 SSS</td>
</tr>
<tr>
<td></td>
<td>X3 SSS $PINS B=1 D=2</td>
</tr>
<tr>
<td></td>
<td>In subcircuit call X2, pins C and D are floating. In subcircuit call X3, pins A and C are floating. See following notes.</td>
</tr>
</tbody>
</table>

Notes:

a. 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.
b. The $PINS argument overrides the standard SPICE node specification <n1 n2 …> if one is present.
c. 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 later in this section.
d. 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 classifies the subcircuit call.

Here is an 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.
```
The following example concerns what happens in the x2 call:

```
subckt test1 plus minus pw=10 pl=10
  r1 plus minus pr w=pw l=pl
.ends

.subckt test2 in out pw=99 pl=99
  x1 in tmp test1 pw=20 pl=20
  r2 tmp out pr w=pw l=pl
.ends

  x2 a b test2 pw=30
```

In this example, pw=30 is substituted in the place of pw=99 for the subcircuit test2 in the subcircuit call x2. However, pw=30 does not apply to test1 in x1 for this series of calls because x1 explicitly defines pw=20. If x1 did not define pw at all, then the value pw=10 would apply to test1, because the original subcircuit definition of test1 has pw=10. If neither x1 nor the original definition of test1 specified parameter pw, then pw=30 would apply to test1 in x1.

**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]
```

You can enter the parameter names, parnam, as property names in the Trace Property specification statement. As always, they override parameter values assigned in the subcircuit definition. In the previous example, you could trace properties par1 and par2.

All primitive subcircuit calls have 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.

You can instruct LVS to ignore primitive subcircuit names that do not appear in the rule file (in Device or LVS Box statements) by using the LVS Spice Cull Primitive Subcircuits specification statement.

**Duplicate subcircuit calls** — Duplicate subcircuit calls are subcircuit calls having the same parent and identical names. Duplicate subcircuit calls are reported as warnings, and their names are modified so as to make them unique. The modification consists of appending a unique string to the end of the subcircuit call name. The string is of the form ===<number>, where <number> is a running count of duplicate subcircuit calls in each parent subcircuit, starting with 2. The count is re-initialized in each parent subcircuit. The SPICE parser makes sure that these
generated names are unique; if a generated name was already encountered in the subcircuit, then the running count is incremented and a new name is generated.

Example:

```plaintext
.SUBCKT AAA
C1 1 2
.ENDS
.SUBCKT BBB
X1 AAA
X1 AAA $ Duplicate; renamed X1===2.
X1 AAA $ Duplicate; renamed X1===3.
X2 AAA
X2 AAA $ Duplicate; renamed X2===4.
.ENDS
.SUBCKT CCC
X1 AAA
X1 AAA $ Duplicate; renamed X1===2.
X1 AAA $ Duplicate; renamed X1===3.
.ENDS
```

The following warnings are issued in the previous example:

```
Warning: Duplicate subckt call "X1" at line 7 in file "z.net" renamed "X1===2"
Warning: Duplicate subckt call "X1" at line 8 in file "z.net" renamed "X1===3"
Warning: Duplicate subckt call "X2" at line 10 in file "z.net" renamed "X2===4"
Warning: Duplicate subckt call "X1" at line 15 in file "z.net" renamed "X1===2"
Warning: Duplicate subckt call "X1" at line 16 in file "z.net" renamed "X1===3"
```

You can prevent the SPICE parser from modifying duplicate subcircuit call names by specifying LVS Spice Option X in the rule file. This option is provided for backward compatibility with older versions of Calibre LVS or ICtrace and should be used with caution because it may generate incorrect connectivity.

**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:

```plaintext
GLOBAL VCC VSS
.SUBCKT SSS A B C D
...
.ENDS
.SUBCKT XXX E VCC VSS
...
.ENDS
.SUBCKT ZZZ
```
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 previous example, 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 previous example, 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 a hierarchical pathname. If this hierarchical pathname also appears literally as a net name in the containing subcircuit then, in flat operation, they both refer to the same net. In hierarchical operation, they remain separate nets.

If this generated hierarchical pathname also appears literally as a net name in the containing subcircuit, then the generated net and the original net remain separate. This is consistent with the general treatment of slash-separated net names in hierarchical operation in the SPICE reader.

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. You can instruct the SPICE parser to report subcircuits with floating pins by specifying LVS Spice Option F.
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 above or below the parent in the nesting hierarchy.

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                $$   scope is ccc.
r2 b c
.ends bbb
$.subckt ddd a b c     $$ Call to bbb #2.
.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
```

Eldo Format Y Element

The Calibre LVS SPICE reader supports the Y element used in Eldo ®. The Y element is another form of the instantiation element X (see “Subcircuits” on page 14-33), but with a different syntax.

Syntax:

```
Yxxx <subname> [GENERIC: <parnam = pval > …] [PORT: <net> …] [$ comments]
```

The **subname** is a required subcircuit name. Specification parnam = pval may appear zero or more times to define all the desired parameters. At least one net parameter must appear to
connect the instantiation into the circuit. The actual number of nodes given must match the
nodes of the subcircuit being instantiated.

Comment extensions are supported exactly as in the X element.

Keywords PORT and GENERIC are case-insensitive, and can be followed by an optional
whitespace before the colon, that is, “PORT:” and “port :” are both allowed.

The Y device is treated by LVS exactly as an X element for all purposes, including device
reduction and filtering, gate recognition, and logic injection. However, if a Y element is written
by LVS, for example, in a discrepancy report, the instance name starts with Y instead of X.

**.PARAM**

Syntax:

```
.PARAM < parnam = pval > …
```

where “<parnam = pval>…” are parameter names assigned to numeric or string 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 pp="abc"
XI 9 10 mos2 width=5U length=6U pp="def"
*
.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 rr=pp
.ENDS mos2
```

In the previous 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)
- \( rr = "abc" \) (from the .PARAM statement)

See LVS Spice Redefine Param in the SVRF Manual as a related specification statement.
String Parameters

The LVS SPICE parser supports character-string parameters in addition to standard SPICE numerical parameters. The general form for parameter assignment is:

```
parnam=pval
```

where parnam is a parameter name and pval may be a literal value or expression of either numeric or string type. By default, literal string values must be enclosed in double quotes. For example:

```
Z1 = "FOO"
```

Unquoted values may also be used, but you must explicitly allow this by specifying LVS Splice Allow Unquoted Strings YES in the rule file.

For example:

```
Z1 = FOO
```

Generally, string values in parameter assignments are allowed anywhere numeric values are allowed. For example, parameters with string values can be specified in SPICE element statements, primitive or non-primitive subcircuit calls, subcircuit definitions, and in .PARAM statements. For details, refer to the discussion of the individual statements.

The semantics of parameter passing and of the .PARAM statement are identical for string and numeric parameter values. For example:

```
.PARAM PP="ABC"
.SUBCKT AAA 1 2 QQ="DEF"
M1 1 2 3 4 P W=1 L=0.2 Z1="GHI" Z2=QQ Z3=RR Z4=PP
.ENDS
X1 1 2 AAA RR="JKL"
```

In the example, the MOS device M1 in the X1 instance of AAA has the following parameter values:

```
Z1 = GHI (from the literal assignment)
Z2 = DEF (from the default value of QQ)
Z3 = JKL (from the RR value specified in the X1 subcircuit call)
Z4 = ABC (from the PP value specified in the .PARAM statement)
```

Only trivial string expressions are supported, specifically: literal string values, string parameter names, any of the above enclosed in arbitrary number of parentheses, and any of the above enclosed in single quotes. Quoted string values may not appear within parentheses or within single quotes. Parentheses and single quotes have no effect on the semantics of a string expression; they are allowed but provide no benefit. No other operators are supported.
String values may not be assigned to built-in SPICE parameters that are expected to have numeric values. Such assignments are reported as errors.

String values may appear only in explicit parameter assignments of the form param=param as shown previously; they may not appear in implicit assignments where the parameter name is not explicitly specified.

**.GLOBAL**

Syntax:

```
.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. It does not matter where this statement appears in a netlist, the behavior is always the same. The .GLOBAL statement provides a convenient means of communicating power supplies and clocks 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 suffixes after the colon (:) from node names with the Virtual Connect Colon specification statement. The Virtual Connect Semicolon As Colon statement provides similar control for net names with semicolons (;).

The Dracula CDL statement *.GLOBAL is supported and is equivalent to .GLOBAL as described previously.

**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.
This chapter describes the input requirements, invocation, and procedures for the following utilities:

- **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.
- **EDIF-to-LVS (E2LVS)** — a converter that translates an EDIF structural netlist into a SPICE-like netlist for use as input to Calibre LVS/LVS-H.
- **Compare Two GDSII Databases** — discusses the compare_gds utility.
- **Create a Rule File for Generating Unused Layers** — discusses the create_layer_rules utility.
- **Create a Rule File for Comparing Two Layout Databases** — discusses the create_compare_rules utility.

## Verilog-to-LVS

The V2LVS (Verilog-to-LVS) converter translates a Verilog structural netlist into an LVS SPICE netlist suitable for Calibre LVS/LVS-H comparison against a layout. The following sections describe its use.

### Description

V2LVS compiles a structural Verilog design into an equivalent netlist in extended SPICE form. The section “SPICE Format” on page 14-1 describes the netlist format and extensions used in a LVS SPICE netlist. The SPICE netlist can then be used as input to Calibre LVS by specifying this in the rule file:

```
SOURCE PATH <output SPICE file>
SOURCE PRIMARY <top cell name>
SOURCE SYSTEM SPICE
```

V2LVS 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 15-1 shows the 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. 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.

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**

Here is the complete set of options:

```
   v2lvs -v verilog_design_file -o output_spice_file
       [-l verilog_lib_file]
       [-lsp spice_library_file]
       [-lsr spice_library_file]
       [-s spice_library_file]
       [-s0 groundnet] [-s1 powernet] [-sk]
       [-p prefix]
       [-werror] [-w warning_level]
       [-a array_delimiters]
       [-c charl[char2]]
       [-u unnamed_pin_prefix]
       [-t svdb_dir]
       [-addpins pin_name]
       [-b] [-n] [-i] [-e] [-h]
       [-cb]
       [-i_trace]
       [-64]
```

V2LVS requires only the -v (Verilog design file) switch.

**Arguments**

- **-v verilog_design_file**
  Specifies the filename of the input Verilog structural netlist.

- **-o output_spice_file**
  Specifies where to place the output LVS SPICE netlist to be used for LVS. This is the translation of the Verilog netlist. Default is standard out.

- **-l verilog_lib_file**
  Specifies the location of the Verilog primitive library file. It is not translated. The Verilog library is parsed for interface pin configurations (see -s option). When calls are made to SPICE modules, a description of the SPICE interface can be provided in Verilog syntax using this file. Since V2LVS parses behavioral Verilog files for this interface information, it is often possible to use Verilog modules provided for use in simulation. It is essential, however, that the Verilog modules match the SPICE modules on details like pin ordering and so forth.

- **-lsp spice_library_file**
  Specifies SPICE library file name using pin mode. The SPICE file is parsed for interface configurations. Pins with pin select ([] ) annotation are kept as individual pins using escaped identifiers. Details are on page 15-28. See also -lsr and -l options.

  When calls are made to SPICE modules, the SPICE interface can be read from the SPICE file directly using this option, or the -lsr option. The -lsp option translates SPICE pins directly into Verilog ports, with no assembly into arrays or ranged ports.
Utilities

Verilog-to-LVS

• **-lsr spice_library_file**
  Specifies SPICE library file name using range mode. The SPICE file is parsed for interface configurations. Pins with pin select ([ ]) annotation are assembled into Verilog ranges. See page 15-26 for details.

  When calls are made to SPICE modules, the SPICE interface can be read from the SPICE file directly using this option, or the -lsp option. The -lsr option translates SPICE pins that have array delimiter characters into arrays or ranged ports. See also the -a and -l options, as well as the *.BUSDELIMITER description later in this section.

• **-s spice_library_file**
  Specifies that the -o option output file has a .INCLUDE statement that points to the SPICE library file. The -s option does not cause V2LVS to read the library file. See the -lsp and -lsr options for reading SPICE library files.

• **-s0 groundnet**
  Specifies that the default global ground is changed to `groundnet`. The `groundnet` is declared as a .GLOBAL net. The -s0, -s1, and -sk options can be useful in handling various power/ground conventions in different design flows.

  The supply net options are discussed further on page 15-11.

• **-s1 powernet**
  Specifies that the default global ground is changed to `powernet`. The `powernet` is declared as a .GLOBAL net. See also -s0 and -sk options.

• **-sk**
  Specifies that Verilog supply0 and supply1 nets are not connected to the global power and ground nets.

• **-sl**
  Specifies to create local supply0 and supply1 nets that are not connected to global power and ground.

• **-p prefix**
  Adds `prefix` to Verilog gate-level primitive cells. This allows construction of specific SPICE subcircuits that correspond to the Verilog gates used. It can help to resolve naming conflicts in certain circumstances. See page 15-16 for details.

• **-werror**
  Treats warnings as errors. Verbosity is controlled by -w option.

• **-w warning_level**
  Controls the amount of warning message output. Possible level choices are:
  
  0 Output no warning messages.
  
  1 Output warning messages for skipped blocks and modules only.
2 Output level 1 and calls to undeclared modules and pin arrays with widths wider than ports. This is the default.

3 Output level 2 and called port array mismatches, and unsupported compiler directives.

4 Output level 3 plus all ignored constructs (as in delays and charges).

- **-a delimiter1 delimiter2**

Changes the array delimiter characters to the ones you specify. The default delimiters are brackets `[ ]`. Use this option to change characters used to describe Verilog net selections as SPICE nets. Verilog supports bus type nets and ports with a bracket `[ ]` syntax. SPICE only represents simple nets. The default mapping in v2lvs is to use the Verilog bus name followed by a pin selection enclosed in `[ ]`.


- **-c char1[char2]**

Sets the substitution characters for characters illegal in SPICE. Parameter `char1` replaces $, comma (,), and =. Parameter `char2` replaces /. When characters normally illegal in SPICE are encountered in Verilog, substitutions take place. This switch controls which characters get substituted.

- **-u unnamed_pin_prefix**

Specifies a prefix to add to unnamed pin connections in module instantiations, and to unnamed primitive and gate instantiations.

- **-t svdb_dir**

Adds source template file information to the SVDB database. This is used in Calibre xRC. See page 15-32.

- **-addpin pin_name**

This option should be used with care because it adds pins that are not explicitly present in the original Verilog netlist. Use of this option is therefore discouraged. Generally, SPICE .GLOBAL declarations are better suited for this purpose. Additionally, the rule file specification statement `LVS Spice Override Globals` can provide additional flexibility for designs with multiple power/ground nets.

This switch adds `pin_name` as a pin to subcircuit definitions and subcircuit calls in the output SPICE netlist where such a pin did not exist in the input Verilog or in relevant SPICE libraries. Specifically:

1. The `pin_name` is added as pin to `.SUBCKT` statements in the output SPICE netlist if the original Verilog module definition did not already have a pin with that name.
2. The assignment \texttt{pin} \texttt{=pin} is added to subcircuit calls in the output SPICE netlist if the original Verilog instance did not already have a connection to pin \texttt{pin}.

Exception: If the subcircuit being called is defined in a SPICE netlist read with the \texttt{-lsr} or \texttt{-lsp} switches, and that subcircuit does not have a pin called \texttt{pin}, then \texttt{pin} is not added to the subcircuit call.

This option is incompatible with the \texttt{v2lvs -i} option.

Example

Consider the following Verilog and SPICE netlist:

---- file v.lib -----------
// Verilog Library Module
// -addpin pins are added to calls to this module
module B ( P1,P2 );
input P1;
output P2;
endmodule

---- file s.lib -----------
** Spice Library Module
**
** No VSS or VDD pins are defined in A0
** -addpin pins are not added to calls to this SUBCKT
.ENDS

** VSS and VDD pins are declared here
** -addpin pins are added to calls to this SUBCKT
.ENDS

---- file ex.v -------
module C ();
wire a, b, c, d, e, f, g;

// Call to Module from SPICE Library
A0 inst1 ( a, b );
A1 inst2 ( a, b );

// Call to Module from Verilog Library
B inst3( .P1(c), .P2(d) );

// Undeclared Module
// -addpin pins are added to calls to undeclared modules.
UNDECL inst4 ( .P(a), .Q(a) );

// gate
// -addpin pins are added to gate instantiations in the order
// specified on the \texttt{v2lvs} command line.
nand inst5( e, f, g );
Translate the file ex.v with libraries s.lib and v.lib using the -addpin option to add VSS and VDD pins with the following command line:

```
v2lvs -v ex.v -l v.lib -lsr s.lib -addpin VSS -addpin VDD -o ex.out
```

This results in the following SPICE output:

```
.SUBCKT C VSS VDD
 Xinst1 A0 $PINS P[0]=a Q=b
 Xinst2 A1 $PINS P[0]=a Q=b VSS=VSS VDD=VDD
 Xinst3 B $PINS P1=c P2=d VSS=VSS VDD=VDD
 Xinst4 UNDECL $PINS P=a Q=a VSS=VSS VDD=VDD
 Xinst5 e f g VSS VDD nand
 .ENDS
```

- **-b**
  Retains the leading backslash for escaped identifiers. Verilog allows special identifiers called *escaped identifiers* to contain any non-whitespace character. These escaped identifiers normally map to the same character string in SPICE (when characters used are legal in SPICE). The leading backslash (\) character that indicates an escaped identifier is normally stripped. This switch causes it to be retained. See page 15-22 for details.

- **-n**
  Specifies unconnected pins to receive numbered connections, starting with 1000. By default, unconnected pins are not specified in the SPICE netlist because LVS interprets missing pin connections as unconnected pins. This option causes V2LVS to generate explicit connections to unconnected nets. See page 15-20 for details.

- **-i**
  Specifies that calls to subcircuits with pins be done in order, according to traditional SPICE rather than with $PINS construct. Calibre LVS has a special SPICE extension for named pin connections in the $PINS construct. This option is for non-Calibre applications only. See page 15-31 for details.

- **-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” on page 15-30.

- **-h**
  Prints a help message.

- **-cb**
  Specifies to use a Calibre CB (caldrclvseve) license.
Utilities
Verilog-to-LVS

- `--ictrace`
  Specifies to use an ictrace license.

- `--64`
  Specifies to use the 64-bit algorithm for processors that have this capability.

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 in “Using V2LVS Without Verilog or SPICE Library Files” on page 15-29.

Supported Verilog Syntax


Modules

Section F.1 of the formal Bachus Naur Form 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.

V2LVS supports modules and macromodules. 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 — 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:

```verilog
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 — 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:

```verilog
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 — 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.

```verilog
module AA (TH[2:3], UH[4], SH) ;
input  [3:0] TH;
input  [0:7] UH;
endmodule

module BB ( );
wire[0:1] w1;
wire w2;
wire [1:2] w3;

AA inst1 ( .TH(w1), .UH(w2), .SH(w3));
endmodule
```

translates to the SPICE netlist:

```
.ENDS

.SUBCKT BB
.ENDS
```

User-Defined Primitives

See “UDP Declaration and Instantiation” on page 15-21.

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.

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. Any “vectored” or “scalared” connections are broken into individual pins in SPICE, so these keywords are ignored.

Handling Power and Ground Connections

Since Verilog is a simulation language, and is not generally used to specify power and ground routing, several options are provided that help to support different power and ground conventions used in different chips.

Interaction Between Net Types and –s0, –s1, –sk, and -sl Switches

By default, numeric values are translated as references to global VDD and VSS nets unless local supply0 or supply1 nets are present in the module. If such a translation is made, a .GLOBAL declaration for the VDD and VSS net is also created. The -s0 and -s1 command line switches alter this default translation so that all numeric values of 1 are translated to the -s1 argument, and all numeric values of 0 are translated to the -s0 argument.

By default, these switches also cause the local supply0 and supply1 nets to be connected to the .GLOBAL ground and power nets. The -sk switch causes supply0 and supply1 nets to remain distinct from -s0 and -s1 nets. The -s0 and -s1 arguments are also specified as a .GLOBAL nets in the output netlist.

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 you want to create one logical net from several names used in different places in the design. You can use the –s0 and –s1 switches to accomplish this.

For example, when you use the –s1 PWR switch, 1’b1 is always translated to PWR. Use these switches to translate numeric signals into the same net name throughout the design. The supply0 and supply1 declarations are connected to this global net using *.CONNECT. Also note that these switches 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 that 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 15-1. Power Signal Pin Translation

<table>
<thead>
<tr>
<th>Local power signal declaration</th>
<th>Default 1’b1 translation</th>
<th>1’b1 translation using “-s1 PWR” option</th>
<th>1’b1 translation using “-s1 PWR -sk” option</th>
</tr>
</thead>
<tbody>
<tr>
<td>no local supply1</td>
<td>VDD .GLOBAL VDD</td>
<td>PWR .GLOBAL PWR</td>
<td>PWR .GLOBAL PWR</td>
</tr>
<tr>
<td>supply1 VDD1</td>
<td>VDD1 .GLOBAL VDD1 .GLOBAL VDD</td>
<td>PWR .GLOBAL PWR</td>
<td>VDD1 .GLOBAL VDD1 .GLOBAL PWR</td>
</tr>
</tbody>
</table>

**Comment:**
- Keep nets separate. Use VDD naming convention.
- Connect all nets. Change default power naming from VDD to PWR.
- Keep nets separate. Change default power naming from VDD to PWR.

Ground signals, supply0 nets, and 1’b0 values interact similarly.

**Example — use of supply0 and supply1 net types**

The following example uses supply0 and supply1 net types within a module:

```verilog
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]
```

All vector and scalar connections are broken apart into individual pins for translation to SPICE, so these keywords are ignored.

**Keeping supply0 and supply1 nets local and separate from global power and ground** — Using the -sk option with the -s0 and -s1 options keeps supply0 and supply1 nets local and separate from global power and grounds nets.

**Example — using -sk with -s0 and -s1**

Consider the following Verilog modules:

```verilog
module B ( );
    supply0 VS1;
    supply1 VD1;
    A A1( 1'b1 );
    A A2( 1b0 );
    A A3( VD1 );
    A A4( VS1 );
endmodule

module C ( );
```
A A1( 1'b1 );
A A2( 1'b0 );
A A5( VSS );
A A6( VDD );
endmodule

If you specify -s0 VSS -s1 VDD, v2lvs connects all power and ground signals together to the global power and ground nets:

.SUBCKT B
  *.CONNECT VDD VD1
  *.CONNECT VSS VS1
XA1 A $PINS p=VDD
XA2 A $PINS p=VSS
XA3 A $PINS p=VD1
XA4 A $PINS p=VS1
.ENDS

.SUBCKT C
XA1 A $PINS p=VDD
XA2 A $PINS p=VSS
XA5 A $PINS p=VSS
XA6 A $PINS p=VDD
.ENDS
GLOBAL VDD
GLOBAL VSS

Using the -sl switch in addition to -s0 and -s1, it is now possible to keep the supply0 and supply1 nets local and separate from the global power and ground nets:

.SUBCKT B
XA1 A $PINS p=VD1
XA2 A $PINS p=VS1
XA3 A $PINS p=VD1
XA4 A $PINS p=VS1
.ENDS

.SUBCKT C
XA1 A $PINS p=VDD
XA2 A $PINS p=VSS
XA5 A $PINS p=VSS
XA6 A $PINS p=VDD
.ENDS
GLOBAL VDD
GLOBAL VSS

Adding Pins

See the -addpin option on page 15-5.

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 — use of continuous assignment with nets

The following Verilog code:

```verilog
module AA ( OUT1, OUT2 );
output OUT1, OUT2;
assign OUT1=OUT2;
endmodule
```

is converted to the following SPICE netlist:

```spice
.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 causes a syntax error:

```verilog
module and ( in, out, control ); // causes a syntax error.
endmodule
```

V2LVS emits 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 line in the following example). Warnings are always issued for instantiation of gate-level primitives since these are not usually part of a purely structural netlist.

Example — use of Verilog primitive gates

The following Verilog code:

```verilog
module A () ;
wire a, b, c, d, e, f;
and i1 ( a, b, c );
xor ( d, e, f );
endmodule
```
Verilog-to-LVS

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** — You can use the –p switch on the command line to add a prefix to all gate level primitive calls. For example, the previous Verilog module is converted into the following SPICE subckt if you use the command line switch “–p v2lvs_”:

```
.SUBCKT A
Xi1 a b c v2lvs_and
X0 d e f v2lvs_xor
.ENDS
```

Use this 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:

<table>
<thead>
<tr>
<th>and</th>
<th>nand</th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>nor</td>
<td>notif0</td>
</tr>
<tr>
<td>xor</td>
<td>xnor</td>
<td>notif1</td>
</tr>
<tr>
<td>buf</td>
<td>bufif0</td>
<td>bufif1</td>
</tr>
<tr>
<td>pulldown</td>
<td>pullup</td>
<td>nmors</td>
</tr>
<tr>
<td>nmos</td>
<td>pmos</td>
<td>rpmos</td>
</tr>
<tr>
<td>cmos</td>
<td>rcmos</td>
<td>tran</td>
</tr>
<tr>
<td>rtran</td>
<td>tranif0</td>
<td>tranif1</td>
</tr>
<tr>
<td>rtranif0</td>
<td>rtranif1</td>
<td></td>
</tr>
</tbody>
</table>

**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 the first example on page 15-9 for a simple case 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 the following:

- simple arrays or ranges (for example a[0:3])
- binary, hex, octal, or decimal bit expressions (for example 2’b01)
- concatenations of the previous — 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 } })

The port selection example on page 15-10 shows a simple case of instantiations using ranges. The supply example on page 15-12 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 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 one 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 — 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:

```verilog
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:

```spice
.ENDS

.SUBCKT B
.ENDS
```

Unnamed Concatenation Expressions in Declarations

For this case:

```verilog
module A ( {B, C} );
    input B, C;
endmodule

module B();
    wire [0:1] a;
    A inst1( a );
endmodule
```

The port is unnamed. This module can be instantiated with positional calling only. Attempts to call via named connections always leaves the unnamed port unconnected.

Example — unnamed concatenation in module declaration

In this example, module B creates an instance of module A called inst1 in which module A is declared with an unnamed concatenation.

```verilog
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]
.Q=X
.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 — 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:

```
.ENDS

.SUBCKT B
.ENDS
```

A warning is issued when calling pins are wider than the port to which they are passed.

When the calling pin expression is narrower than the port expression in the module, calling pins are mapped to called ports from the rightmost called port. Remaining ports are mapped to random undeclared nets.
Example — 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:

```verilog
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:

```spice
.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 do not appear in the output. Calibre LVS handles these unconnected pins appropriately. 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 — unconnected pins

The following Verilog circuit:

```verilog
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:

```spice
.SUBCKT A B C D[1] D[0]
.ENDS
```
UDP Declaration and Instantiation

Section F.5 of the IEEE standard describes User Defined Primitives. UDP instances are translated similarly to module instances. 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 so are ignored (after warning) during translation.

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). You can use the assign statement only 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 — assignment and net initialization

The module:

```verilog
code
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
code```

translates to the SPICE circuit:

```plaintext
.SUBCKT B a[0] a[1]
```
Utilities

Verilog-to-LVS

*.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 those in Table 15-3, cause the module they are in to be skipped during translation

<table>
<thead>
<tr>
<th>Table 15-3. Behavioral Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
</tr>
<tr>
<td>force</td>
</tr>
<tr>
<td>posedge</td>
</tr>
<tr>
<td>case-endcase</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>for</td>
</tr>
<tr>
<td>begin-end</td>
</tr>
</tbody>
</table>

Specify Section

Specify blocks are discussed in section F.7 of the IEEE standard. The Specify block is used to specify timing information for paths across a Verilog module. V2LVS ignores Specify blocks.

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, hexadecimal, octal, and decimal numbers, port ranges, port selections, bit selections concatenations, multiple concatenations, and so forth.

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, use the Source Case YES specification statement 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 do 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 causes the leading \ character to be retained on escaped identifiers that are not also legal Verilog identifiers. This has the effect of making sure that bit 3 of A is not the same as \A[3].

Example — escaped identifiers and use of the –b switch

```verilog
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
```

translates as follows when the –b switch is not used:

```verilog
*.CONNECT b[1] b[0]
*.CONNECT sum[0] b[1]
.ENDS

.SUBCKT A i zn ci
.ENDS
```

It translates as follows when the –b switch is used:

```verilog
```
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 $, comma, (,), =, and / are escaped identifier characters which, under certain circumstances, are illegal in SPICE.

Leading $, comma, (,), and = are mapped to #. For example, aa$ is mapped to aa#. If multiple identifiers would map to the same name, the second identifier encountered is mapped to the identifier with a # prefix. Multiple collisions result in additional # characters prefixed.

The / characters are left as-is in V2LVS. In certain circumstances, / characters 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 replaces the characters leading $, comma, (,), and =, while the second character (if present) replaces / characters. For example, if you desire to have leading $, comma (,), and = replaced with _, use –c _. If you desire to have / characters replaced with # characters, use –c ## on the command line. (The first # character keeps the substitution for leading $, comma (,), and = as #, while the second # causes / characters to be changed to # as well.)

**Integer numbers** — Decimal, hexadecimal, octal, and binary representations of numbers are supported. V2LVS does not support x or z in numbers. Translation terminates 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 — use of a parameter value in a structural netlist**

The following module:

```verilog
module A ( inA );
parameter width=8;
input [width-1:0] inA;
endmodule
```
is translated into the following subcircuit:

```
.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) …).

**Library Interface Files**

V2LVS can use a Verilog library file to specify declarations to represent leaf-level modules which are actually defined in SPICE. V2LVS reads the Verilog library file to gather the port interface names, which 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 with SPICE library files, or without any library file.

**Using V2LVS With SPICE Library Files**

V2LVS reads SPICE files in order to deduce Verilog module interfaces. This method can be used in conjunction with the Verilog library modules (-l option).

The SPICE library functionality can operate in two modes. Range mode causes V2LVS to interpret sequences of pins using the array delimiters (typically the [ ] characters) as parts of a Verilog port range. Pin mode causes V2LVS to interpret each spice pin as a separate Verilog port regardless of array bracket characters.
In order for SPICE library files to be used, the pin order in the SPICE library .SUBCKT must be the same as the pin order in the Verilog library module. This is important when positional Verilog calls are made, and when the -i switch is used, because V2LVS connects pins in order based on this information.

For example, consider the following SPICE .SUBCKT and corresponding Verilog module:

```
.SUBCKT EX1 A C B
.ENDS
module EX1 ( A, B, C );
endmodule
```

These descriptions have inconsistent pin order (pins B and C are swapped). In a case like this, the Verilog call:

```
EX1 e( w1, w2, w3 );
```

produces different results based on whether the Verilog library is used, or the SPICE library is used. The SPICE library generates the following SPICE call:

```
Xe EX1 $PINS A=w1 B=w2 C=w3
```

while the Verilog library generates the following SPICE call:

```
Xe EX1 $PINS A=w1 C=w2 B=w3
```

In either case, named Verilog connections are connected correctly as long as the -i switch is not used. This is because the $PINS connection creates a named connection in SPICE. (Recall that -i turns off the non-standard $PINS connection.)

**Using range mode (-lsr)** — The V2LVS -lsr switch causes the SPICE file to be read for interface information. SPICE pins of the form p[n] are interpreted as Verilog port ranges. Contents of the subcircuits are ignored. The .INCLUDE statements are processed. Multiple instances of this switch and the -lsp switch on the command line are allowed.

**Example — range mode**

The SPICE subcircuit

```
...
.ENDS
```

is interpreted as a Verilog module with the signature:

```
module BLK_A ( A, B );
inout [0:3]A;
inout B;
endmodule
Use `-lsr` if Verilog module calls treat the pins A[0:3] as a port range. An example of a call that treats them as a port range is:

```plaintext
module test;
wire [0:3] w1;
wire w2;
// note that wire bundle w1 is passed as a unit to port A.
BLK_A instA ( .A( w1 ), .B( w2 ) );
endmodule
```

**.*.BUSDELIMITER** — This SPICE statement specifies the bus delimiter characters used to identify the index portion of SPICE pin names. The indexed pins indicate the SPICE equivalent of a Verilog port specified with a range expression. The `.*.BUSDELIMITER` statement supports flows that have multiple SPICE subcircuits using different bus delimiter character conventions, which are being called from the same higher-level structural netlist.

The SPICE syntax for the `.*.BUSDELIMITER` statement is: `.*.BUSDELIMITER symbol` where the symbol parameter is one of [, {, <, or (. When a SPICE library is specified with the `-lsr` switch, the `.*.BUSDELIMITER` statement causes V2LVS to start searching for pins that use the bus delimiter characters specified when creating port interfaces for the subcircuit being read.

For example, consider the following SPICE library:

```plaintext
.*.BUSDELIMITER <
.SUBCKT A I1<3>I1<2>I1<1>I1<0>
...
.ENDS

.*.BUSDELIMITER {
.SUBCKT B I2{3}I2{2}I2{1}I2{0}
...
.ENDS
```

The previous library is called by the following Verilog netlist:

```plaintext
module TOP;
wire [3:0] w1;
wire [3:0] w2;
A A1 (.I1(w1));
B B1 (.I2(w2));
endmodule
```

The `-lsr` switch causes the following Verilog interface interpretation of the SPICE library:

```plaintext
module A (I1);
inout [3:0] I1;
endmodule module B (I2);
inout [3:0] I2;
endmodule
```
and generates the following netlist:

```
.SUBCKT TOP
XA1 A $PINS I1<3>=w1 [3] I1<<2>=w1 [2] I1<<1>=w1 [1] I1<<0>=w1 [0]
XB1 B $PINS I2{3}=w2 [3] I2{{2}=w2 [2] I2{{1}=w2 [1] I2{{0}=w2 [0]
.ENDS
```

A `*.BUSDELIMITER` statement goes into effect at the start of the next `.SUBCKT` statement. It remains in effect until another `*.BUSDELIMITER` statement changes the bus delimiter characters.

By default the bus delimiter characters are specified by the `-a` switch. If the `-a` switch is not specified, the `-a` characters are `[ ]`. The translated netlist uses the character specified by the `-a` switch for net names in the translated modules. Only port names for called SPICE subcircuits are affected by the `*.BUSDELIMITER` statement.

### Using pin mode (-lsp)

—the `-lsp` switch causes the SPICE file to be read for interface information. Pins of the form `p[n]` are interpreted as individual Verilog ports. Contents of the subcircuits are ignored. Multiple instances of this switch and the `-lsr` switch on the command line are allowed.

#### Example — pin mode

The SPICE subcircuit:

```
...
.ENDS
```

is interpreted as a Verilog module with the signature:

```
inout \A[0] ;
inout \A[1] ;
inout \A[2] ;
inout \A[3] ;
inout B;
endmodule
```

Use `-lsp` if Verilog module calls treat the pins `A[0:3]` as individual pins. An example of a call that treats them as individual pins is:

```
module test;
wire [0:3] w1;
wire w2;
// note that wire bundle w1 is passed using pin select notation to individual
// pins of A.
```
The -lsr and -lsp switches are both sensitive to the -a switch (array delimiter characters).

**Using V2LVS Without Verilog or SPICE Library Files**

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 — call to named port of undeclared primitive module**

```verilog
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 — call to pin array in undeclared primitive module**

The following Verilog code:

```verilog
wire [3:0] in1;
wire in2, in3;
spice_module instance_aa ( .A(in1), .B(in2), .C(.in3) );
```

maps to the SPICE instance:

```
```

**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 — 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 supports 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 cause 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, use the following steps:

1. Generate a structural netlist using V2LVS in the usual way:
   ```
   v2lvs -v model.v -l vlib.vlib -o model.spi
   ```

2. Generate a SPICE netlist from the Verilog library file or files using the –e switch:
   ```
   v2lvs -v vlib.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).
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 Calibre xRC 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 option) or SPICE library (-lsp/-lsr options) are necessary when the -i switch is used. In addition, care must be taken to insure that the pin order is consistent across Verilog and SPICE modules being 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. The -lsr and -lsp switches can be used to read SPICE library files.

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 — using the -i switch for standard SPICE output

File src.v:

```verilog
// Verilog source
module top ();
wire w1, w2, w3;
A inst1( .in1(w1), .in2(w2), .out3(w3) );
endmodule
```

File lib.v:

```verilog
// Verilog library
module A ( in1, in2, out3 );
input in1, in2;
output out3;
endmodule
```

File lib.spi:

```spice
// SPICE library with pin order that does not match the
// Verilog library:
.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:**

```plaintext
// 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 a Calibre xRC Source Template File**

The –t svdb switch may be used to generate a file which Calibre xRC can use to determine directions of ports declared in the Verilog netlist.

When the –t switch is used, V2LVS generates a file named `svdb/template/%source.stl`.

This file contains a single starting line which consists of:

```plaintext
%%SOURCE
```

followed by a .stl format template for each module in the Verilog netlist. The module template consists of a module name declaration line:

```plaintext
%  <module_name> <module_name>
```

followed by one line per interface pin:

```plaintext
<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

```plaintext
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:

```verbatim
%%SOURCE
% A A
X X 0 i
Y Y 0 o
Z[0] Z[0] 0 io
% B B
V V 0 o
W W 0 i
```

Calibre xRC can read the `%source.stl` file in the same way that it reads individual layout .stl files when they are available. See “Templates and Specifying Pin Direction” in the *Calibre xRC / ADvanceMS User’s Manual* for more information regarding templates.

## V2LVS Errors and Warnings

Table 15-4 describes errors reported by V2LVS. The arrangement is alphabetical, arranged by the first non-variable word that appears after the word ERROR: in the error message.

### Table 15-4. V2LVS Errors

<table>
<thead>
<tr>
<th>Error message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument for the -a switch was missing.</td>
<td>Specify the array delimiter character with -a option.</td>
</tr>
<tr>
<td>Argument for the -c switch was missing.</td>
<td>Specify the substitution characters for escaped identifier characters illegal in SPICE. These follow the -c option.</td>
</tr>
<tr>
<td>Argument for the -l switch is missing.</td>
<td>Specify the path to the Verilog library file with the -l option.</td>
</tr>
<tr>
<td>Argument for the -lsp switch is missing.</td>
<td>Specify a filename with the -lsp option.</td>
</tr>
<tr>
<td>Argument for the -lsr switch is missing.</td>
<td>Specify a filename with the -lsr option.</td>
</tr>
<tr>
<td>Argument for the -u switch was missing.</td>
<td>Specify a prefix for unnamed pin connections with the -u option.</td>
</tr>
<tr>
<td>Argument for the -v switch is missing.</td>
<td>Specify the Verilog netlist filename with the -v option.</td>
</tr>
<tr>
<td>Cannot create the MASK SVDB DIRECTORY <code>&lt;template_directory&gt;</code></td>
<td>Mask SVDB directory specified with -t option is not writable.</td>
</tr>
<tr>
<td>End of source encountered before closing all <code>endif</code> directives.</td>
<td>Ensure all <code>ifdefs</code> are balanced with <code>endifs</code>.</td>
</tr>
</tbody>
</table>
### Table 15-4. V2LVS Errors (cont.)

<table>
<thead>
<tr>
<th>Error message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error processing include statement at <code>&lt;filename&gt;</code>:&lt;line_number&gt;</td>
<td>Ensure the included file is present.</td>
</tr>
<tr>
<td>Failed to open include file <code>&lt;filename&gt;</code>.</td>
<td>Ensure the included file specified with the -s option can be opened.</td>
</tr>
<tr>
<td>File name with the -o switch was missing.</td>
<td>Specify the pathname of the desired output netlist with the -o option.</td>
</tr>
<tr>
<td>File name with the -t switch was missing.</td>
<td>Specify an existing SVDB directory with the -t option.</td>
</tr>
<tr>
<td>Identifier is too long.</td>
<td>Shorten identifier to one within the supported V2LVS limit (2048).</td>
</tr>
<tr>
<td>Identifiers in port expression must all be the same mode: <code>&lt;identifier&gt;</code></td>
<td>Port expression like A causes this error:</td>
</tr>
<tr>
<td></td>
<td>module( .A({p1,p2}) );</td>
</tr>
<tr>
<td></td>
<td>input p1;</td>
</tr>
<tr>
<td></td>
<td>output p2;</td>
</tr>
<tr>
<td></td>
<td>p1 and p2 must have the same mode (input or output).</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code> instantiates named port <code>&lt;port&gt;</code>.</td>
<td>Ensure that <code>&lt;port&gt;</code> is in the declaration for the module being called.</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code> instantiates new port <code>&lt;port&gt;</code>.</td>
<td>Ensure that <code>&lt;port&gt;</code> is in the declaration for the module being called.</td>
</tr>
<tr>
<td>Integer expression required in port or bit select: <code>&lt;string&gt;</code></td>
<td>Port select or bit select expression must contain an integer value.</td>
</tr>
<tr>
<td>Invalid Inout Declaration in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix Inout declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Invalid Input Declaration in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix Input declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Invalid Net Declaration in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Invalid or unsupported Parameter Declaration in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Invalid Output Declaration in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Invalid Port References in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Fix declaration to conform to Verilog standard.</td>
</tr>
<tr>
<td>Error message</td>
<td>Resolution</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Line continuation character at end of file.</td>
<td>A line continuation character inside a macro definition occurs at the end of a file. Fix the macro definition.</td>
</tr>
<tr>
<td>-lsl requires consistent use of array delimiters for pin indices: &lt;port_name&gt; at &lt;filename&gt;: &lt;line_number&gt;</td>
<td>SPICE file contains pin definitions that V2LVS cannot convert into a port range. Use a Verilog library, or fix SPICE pin definitions.</td>
</tr>
<tr>
<td>-lsl requires numeric use of bit-select values: &lt;port_name&gt; at &lt;filename&gt;: &lt;line_number&gt;</td>
<td>SPICE file contains pin definitions that V2LVS cannot convert into a port range. Use a Verilog library, or fix SPICE pin definitions.</td>
</tr>
<tr>
<td>Maximum string length exceeded in &lt;number_type&gt; number representation: &lt;number&gt;</td>
<td>Shorten number to one within the supported V2LVS limit (2048).</td>
</tr>
<tr>
<td>Module instantiation &lt;instance&gt; has too many pins &lt;pin_list&gt;.</td>
<td>Correct the pins in the instance.</td>
</tr>
<tr>
<td>Multiple -s0 switches supplied—only one is allowed.</td>
<td>Specify one -s0 option.</td>
</tr>
<tr>
<td>Multiple -s1 switches supplied—only one is allowed.</td>
<td>Specify one -s1 option.</td>
</tr>
<tr>
<td>Multiple declarations of net: &lt;net&gt;</td>
<td>Specify only one declaration of the net.</td>
</tr>
<tr>
<td>Multiple declarations of port: &lt;port&gt;</td>
<td>Specify only one declaration of the port.</td>
</tr>
<tr>
<td>Multiple references to the same named port: &lt;port name&gt;.</td>
<td>Specify only one reference to the port.</td>
</tr>
<tr>
<td>No module declaration for module &lt;module&gt; first encountered in module &lt;module&gt;. (-i switch requires Verilog declarations for all modules.)</td>
<td>Specify a module declaration in the Verilog module or do not use the -i switch.</td>
</tr>
<tr>
<td>Only one -o switch is allowed.</td>
<td>Specify one -o option.</td>
</tr>
<tr>
<td>Parameter &lt;parameter&gt; redefined in module &lt;module&gt;.</td>
<td>Remove one of the parameter definitions.</td>
</tr>
<tr>
<td>Port Declaration not present in module port declaration: &lt;string&gt;.</td>
<td>Specify a port declaration in the Verilog module.</td>
</tr>
<tr>
<td>Range Values for port declaration and net declaration must be the same: &lt;value&gt;</td>
<td>Make the range values consistent between the port and net.</td>
</tr>
<tr>
<td>Reference to undefined variable &lt;variable&gt;.</td>
<td>Fix the variable reference.</td>
</tr>
<tr>
<td>Spice Parse error: &lt;filename&gt;: &lt;line_number&gt;</td>
<td>Fix SPICE syntax in file.</td>
</tr>
</tbody>
</table>
Table 15-5. V2LVS Warnings

<table>
<thead>
<tr>
<th>Warning message</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Reference to unsupported expression <code>&lt;expression&gt;</code></td>
<td>V2LVS does not support conversion of the encountered expression.</td>
</tr>
<tr>
<td><code>-t</code> pathname too long.</td>
<td>Use a shorter pathname for output file.</td>
</tr>
<tr>
<td>Trouble writing to template file.</td>
<td>SVDB file specified with <code>-t</code> option has incorrect permissions or format. Fix the SVDB file.</td>
</tr>
<tr>
<td>Unexpected end of <code>file\n</code>.</td>
<td>Complete the file with valid Verilog syntax.</td>
</tr>
<tr>
<td>Unknown switch or invalid argument <code>&lt;argument&gt;</code>.</td>
<td>Specify a valid argument.</td>
</tr>
<tr>
<td>Unsupported expression in port connection <code>&lt;expression&gt;</code>.</td>
<td>Remove the unsupported expression.</td>
</tr>
<tr>
<td>Unsupported expression in port range declaration <code>&lt;expression&gt;</code>.</td>
<td>Remove the unsupported expression.</td>
</tr>
<tr>
<td>Untertminated comment block starting at <code>&lt;string&gt;</code>.</td>
<td>Properly terminate the comment block.</td>
</tr>
<tr>
<td>User Defined Primitive <code>&lt;primitive&gt;</code> not translated</td>
<td>UDPs are not translated by V2LVS. Remove the UDP.</td>
</tr>
<tr>
<td>Warning level following the <code>-w</code> switch was missing.</td>
<td>Specify a warning level with the <code>-w</code> option.</td>
</tr>
<tr>
<td>Warning level following the <code>-w</code> switch was incorrect.</td>
<td>Specify a warning level from integers 0 through 4.</td>
</tr>
<tr>
<td>Warnings being treated as errors.</td>
<td>Resolve all warnings or remove <code>-werror</code> switch.</td>
</tr>
</tbody>
</table>

Table Table 15-5 shows V2LVS warnings. The arrangement is alphabetical, arranged by the first non-variable word that appears after the word WARNING: in the warning message.

Table 15-5. V2LVS Warnings

<table>
<thead>
<tr>
<th>Warning message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Construct in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>Only purely structural modules are translated. (warning level 1)</td>
</tr>
<tr>
<td><code>&lt;module&gt; / &lt;instance&gt; calls array of unknown dimension </code>&lt;port&gt;<code>in undeclared module</code>&lt;called_module&gt;`</td>
<td>Provide port information for called module using Verilog library (<code>-l</code>) or SPICE library (<code>-lsp</code> or <code>-lsr</code>). (warning level 1)</td>
</tr>
<tr>
<td><code>&lt;module&gt; / &lt;instance&gt; calls port </code>&lt;port&gt;<code>in undeclared module</code>&lt;called_module&gt;`</td>
<td>Provide port information for called module using Verilog library (<code>-l</code>) or SPICE library (<code>-lsp</code> or <code>-lsr</code>). (warning level 1)</td>
</tr>
<tr>
<td>Warning message</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Identifier <code>&lt;identifier&gt;</code> renamed to <code>&lt;identifier&gt;</code> (use <code>-b</code> to avoid stripping <code>\</code>).</td>
<td>By default, V2LVS strips the <code>\</code> character from escaped identifiers. <code>-b</code> avoids this. (warning level 3)</td>
</tr>
<tr>
<td>Ignoring unsupported variable identifier in variable list at <code>&lt;string&gt;</code>.</td>
<td>Variable is a selected name (A.B.C) not supported by V2LVS. (warning level 1)</td>
</tr>
<tr>
<td><code>&lt;string&gt;</code> in <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code> ignored.</td>
<td><code>&lt;line_number&gt;</code> contains a Verilog construct that is not supported in a purely structural context. (warning level 4)</td>
</tr>
<tr>
<td><code>&lt;module&gt;</code> / <code>&lt;instance&gt;</code> instantiates new port <code>&lt;port&gt;</code> in undeclared module <code>&lt;called_module&gt;</code></td>
<td>Provide port information for called module using Verilog library (-l) or SPICE library (-lsp or -lsr). (warning level 2)</td>
</tr>
<tr>
<td>Instantiating call to gate level primitive: <code>&lt;primitive&gt;</code></td>
<td>Verilog calls a gate level primitive. (warning level 2)</td>
</tr>
<tr>
<td>Macros with arguments not supported by V2LVS: <code>&lt;macro&gt;</code> at <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>V2LVS does not support <code>define</code> macros with arguments. (warning level 1)</td>
</tr>
<tr>
<td>Module instantiation <code>&lt;string&gt;</code> / <code>&lt;string&gt;</code> has pin mismatches with module <code>&lt;string&gt;</code></td>
<td>Port ranges on the call site do not match ranges on the port of the called module. (warning level 3).</td>
</tr>
<tr>
<td>Multiple declarations of module <code>&lt;name&gt;</code> in <code>&lt;filename&gt;</code>.</td>
<td>Module is declared multiple times. (warning level 1)</td>
</tr>
<tr>
<td>No module declaration for module <code>&lt;name&gt;</code> first encountered in module <code>&lt;name&gt;</code></td>
<td>A call was made to an undeclared module. (warning level 2).</td>
</tr>
<tr>
<td>Positional call to undeclared module <code>&lt;module&gt;</code> pin order will match Verilog call.</td>
<td>A call was made to an undeclared module using a positional instance call. (warning level 1)</td>
</tr>
<tr>
<td>Redefinition of defined value: <code>&lt;value&gt;</code> at <code>&lt;filename&gt;</code> : <code>&lt;line_number&gt;</code></td>
<td>A <code>define</code> macro was redefined. (warning level 1)</td>
</tr>
<tr>
<td>The <code>-s</code> switch is missing the file name for the SPICE Include file.</td>
<td>An argument is missing.</td>
</tr>
<tr>
<td>The <code>-s0</code> switch is missing the name for SUPPLY0.</td>
<td>An argument is missing.</td>
</tr>
<tr>
<td>The <code>-s1</code> switch is missing the name for SUPPLY1.</td>
<td>An argument is missing.</td>
</tr>
</tbody>
</table>
### Table 15-5. V2LVS Warnings (cont.)

<table>
<thead>
<tr>
<th>Warning message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupported compiler directive <code>&lt;directive&gt;</code> ignored.</td>
<td>Compiler directive is not supported by V2LVS and is being ignored. (warning level 3).</td>
</tr>
<tr>
<td>Unknown compiler directive or undefined macro <code>&lt;string&gt;</code> ignored.</td>
<td>Compiler directive or macro is not supported by V2LVS and is being ignored. (warning level 3).</td>
</tr>
<tr>
<td>V2LVS does not support instance ranges <code>&lt;string&gt;</code>.</td>
<td>Unsupported Verilog syntax. (warning level 1)</td>
</tr>
</tbody>
</table>
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 issues. The last section provides a sample netlist translation.

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” on page 14-2 describes the netlist format and extensions used in an LVS SPICE netlist.

E2LVS 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 15-2 illustrates the E2LVS flow:

Figure 15-2. E2LVS Flow

E2LVS creates a file named e2lvs_names when cells with the same name in different libraries are found because these names would 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” on page 15-45 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, you must include certain specification statements in the rule file. These statements are identified in the section “Rule File” on page 3-2.

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.

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] 
   [-64] 
```

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. You must specify the files in the order cells are instantiated. Refer to section “Cell Statement” on page 15-50 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 you do not use -sb command line switch with the optional cell_file parameter, 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 you do not use -ss command line switch with the optional cell_file parameter, 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 underscore character _ is permitted.

-b char
Specifies a bundle delimiter to use when expanding portBundle and netBundle EDIF names to SPICE names. The default is underscore _.

-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. You can create a unique name by inserting the specified char in front of a cell name.

The default delimiter is #. It is substituted in place of the illegal SPICE characters comma, (,), =, 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 remains.
• `-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” on page 15-44 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” on page 15-46 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. Note that when you use this option in hierarchical LVS, 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. This is the default.

• `-cb`  
  Specifies to use a Calibre CB (caldrclvseve) license.

• `-ictrace`  
  Specifies to use an ictrace license.

• `-64`  
  Specifies to use the 64-bit algorithm for processors that have this capability.

### Examples

1. This example specifies an input EDIF file and an output SPICE file:

   ```
   e2lvs -e design.edif -o design.spi
   ```

2. This example expands upon Example 1 by specifying an optional SPICE input file:

   ```
   e2lvs -e design.edif -s cells.spi -o design.spi
   ```
3. This example specifies the filename that contains a list of EDIF files, the array delimiter, a SPICE input file, the substitution character for illegal SPICE names, and the output SPICE file:

```
e2lvs -l edif_files -a '()' -s prim.spi -c '@' -o spice_file.spi
```

4. This example specifies the input EDIF file; a SPICE input file; the bus delimiters for a renamed, expanded, string array; a file containing new cell names for duplicate cells, and an output SPICE file:

```
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, E2LVS ignores the following EDIF constructs 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 Versus SPICE Syntax Considerations

This section describes the following EDIF versus 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 of 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 $, =, comma, (,), and / characters.
The / restriction is relaxed for $PINS strings as described next. 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
- The / 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 appears 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 can have the same name. However, if two cells are in different scopes, they can have identical names.

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 global name scope. Refer to section “EDIF Cell Names Versus SPICE Subcircuit Names” on page 15-45 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. 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 collects 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:

```plaintext
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:

```plaintext
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]
```

You can change default array and bundle delimiters 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:

```plaintext
(port (array (rename A_8_TO_5 “A<8:5>”) 4))
```


If a name collision occurs during array or bundle expansion, E2LVS makes the name unique by inserting as many # characters as necessary before each duplicate name. You can change the default collision character, #, with the `-c` command line switch.

**EDIF-to-SPICE Translation Issues**

This section describes the following EDIF versus 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 uses 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.

You can resolve the first issue by mapping the EDIF cell name to a SPICE subcircuit name. In the next example, the SPICE subcircuit name becomes fcell:

```spice
(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” on page 15-50 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:

```edif
(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:

```plaintext
(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:

```plaintext
(port (rename bus22 “bus(22)")
```

generates a SPICE pin name bus22 when -i is specified. Otherwise, the renamed names (such as bus(22)) 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 connections are represented as follows in the SPICE netlist:

```plaintext
*.J <net> ==<port>
*.J ==<port1> ==<port2> ==<port3>
```

Ports are preceded by == to distinguish them from nets.

**EDIF-to-SPICE Translations**

This section describes how various EDIF statements are translated directly into SPICE. The following statements are covered:

<table>
<thead>
<tr>
<th>EDIF</th>
<th>SPICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>edif</td>
<td>cell</td>
</tr>
<tr>
<td>status</td>
<td>instance</td>
</tr>
<tr>
<td>port and portBundle</td>
<td>joined</td>
</tr>
<tr>
<td>net and netBundle</td>
<td>rename</td>
</tr>
</tbody>
</table>
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 to 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:

```plaintext
(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.

This example shows how a written statement is translated into SPICE:

```plaintext
(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:

```plaintext
(port portNameDef
   {<direction>|<unused>|<designator>|<dcFaninLoad> <dcFanoutLoad>|
    <dcMaxFanin> <dcMaxFanout>|portDelay|property|comment|userData})
```

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 follows:

```plaintext
(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. The portBundle statements are expanded into individual pins when translated to SPICE, as follows:

```
(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” on page 15-52 and “Instance Statement” on page 15-51 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.

E2LVS expands netBundles and translates individual net names to SPICE node names. Refer to the section “Arrays and Bundles” on page 15-44 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” on page 15-52 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. For cells that have the same name, E2LVS makes the names unique by inserting as many # characters as necessary before each duplicate name.

If the netlist view of an EDIF cell had only an interface with no contents, 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” on page 15-44. 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. Sometimes 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, LVS creates a unique name for the collided name by adding # characters as prefixes.

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” on page 15-44 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 ampersand (&) if the first character is not a letter.

This example 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 contains the original name, portA0.

```plaintext
(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:

```plaintext
(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
```
Utilities
EDIF-to-LVS

```
"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))
```

Note that CellLib is external, implying that 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:

```
* .INCLUDE cells.spi
* * edif DicTracy
* * edifVersion 2 0 0
* * edifLevel 0
* * keyWordLevel 0
* * status
* * written
* * timestamp 97 12 09 10 49 26
```
Compare Two GDSII Databases

The compare_gds utility compares two GDSII databases on files up to 2 GB. It does a flat layer-by-layer XOR (or NOT) operation and writes an ASCII DRC results database. It allows you to make comparisons such as layers in one database to different layers in the other database, sets of layers to different sets, particular datatypes, and sets of datatypes to different sets of datatypes. 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 ]
```

In performing comparisons, compare_gds queries each rule file for its Layer Map statements. 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 shapes in at least one of the input databases, the shapes 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 is an empty rule check unless the -NOKEEPEMPTY switch is specified; in that event, diff_L does 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.

Note that the data must have the same origin point or it is considered different data.

There is a 64-bit version of this executable called compare_gds64. It can take files over 2 GB in size. The syntax and usage are the same as for compare_gds. However, it is usually more efficient to use the dual-database capability within Calibre for this type of comparison. See “Dual Database Capability” on page 4-43.

Create a Rule File for Generating Unused Layers

The create_layer_rules utility scans an input rule file for the layers that are needed in the run, and determines which of the original layers in the input layout database will not be used during the run. The utility then generates an output rule file, which specifies all the unused layers in the layout, and contains rules for outputting the unused layers to separate GDS or OASIS files. The syntax is:

```
$MGC_HOME/bin/create_layer_rules input_rule_file output_rule_file
```

The Layout Path statement in the input_rule_file must be resolvable.

To show how this utility works, suppose you have a rule file, called my_rules, with the following form:

```
LAYOUT PATH "./my_design.gds"
LAYOUT SYSTEM GDSII

... LAYER poly 1
LAYER diff 2 //not needed by the run
LAYER big 100 //not in the gds database

my_rule {COPY poly}
```

The diff layer is not needed for this runset because it appears in no rule check or connectivity statement. Layer 100 is in the rule file, but not in my_design.gds. The create_layer_rules utility
scans my_design.gds for the layers and datatypes in this database. It then generates a rule file like this:

```
INCLUDE my_rules

// INPUT DATABASES PROCESSED:
// ./my_design.gds (more than one database can appear here)

// NON-EMPTY LAYER/DATATYPEs IN INPUT DATABASE(s):

// LAYER  DATATYPE
//     1      0     REQUIRED (in the layout, needed for the run)
//     2      0     NOT REQUIRED (in the layout, but not needed)
//     4      0     NOT REQUIRED

// OUTPUT RULES FOR NON-REQUIRED LAYER/DATATYPEs:

LAYER L2_D0 0
LAYER MAP 2 DATATYPE 0 0
L2_D0 { COPY L2_D0 } DRC CHECK MAP L2_D0 2 0

LAYER L4_D0 3
LAYER MAP 4 DATATYPE 0 3
L4_D0 { COPY L4_D0 } DRC CHECK MAP L4_D0 4 0
```

- The first line is an Include statement with the name of the `input_rule_file`. This statement includes the input rule file in any subsequent run.

- The INPUT DATABASES PROCESSED section shows the databases that are in the Layout Path statement. There can be more than one database specified.

- The NON-EMPTY LAYER/DATATYPEs IN INPUT DATABASE(s) section shows all the non-empty layers, with their datatypes, in the layout databases specified in the Layout Path statement. It also shows whether the layers are required in the input rule file runset or not.

  In the example, layer 1 is in the layout database and is required for the run (layer 1 is in a rule check in the input rule file). Layer 2 is in the layout, and in the input rule file, but is not needed for the run. Layer 4 is in the layout, but is not in the input rule file. Layer 100 appears in the input rule file, but is not in the layout database, so it is ignored.

- The OUTPUT RULES FOR NON-REQUIRED LAYER/DATATYPEs section contains SVRF rules in this format:

  LAYER L<n1>_D<d> <n2>
  LAYER MAP <n1> DATATYPE <d> <n2>
  L<n1>_D<d> { COPY L<n1>_D<d> } DRC CHECK MAP L<n1>_D<d> <n1> <d>

  Where the parameters are as follows:

  <n1> — The number of the unused layer in the layout database.

  <d> — The datatype of the unused layer in the layout database.
<n2> — A layer number not appearing in a Layer statement in the input rule file. These are ordered sequentially, starting from 0. This number is selected so that there will be no conflict with any Layer statement in the original rule file.

You can use this output rule file to run your original rule file, and to output all the unused layers (from the original rule file) in the layout.

Create a Rule File for Comparing Two Layout Databases

The create_compare_rules utility scans an input layout database (GDS or OASIS) and outputs a rule file that can be used to compare the original layout with another layout. The comparison is a layer-by-layer XOR by default, but the utility can also generate a layer-by-layer copy of the layout. This utility takes advantage of the features discussed under “Dual Database Capability” on page 4-43.

The syntax is as follows:

```
$MGC_HOME/bin/create_compare_rules \ 
[-COPY] output_rule_file layout_database1 | output_rule_file layout_database1 layout_database2
```

The parameters are as follows:

- **-COPY**
  An optional argument that causes the utility to use the Copy operation in the output_rule_file rather than the XOR operation. The rule checks in the output_rule_file generate copies of all the layers from the layout_database1.

- **output_rule_file**
  A required pathname of the comparison rule file generated by this utility.

- **layout_databaseN**
  A required pathname of a GDS or OASIS layout database. When one pathname is provided, a generic comparison rule file is output, which requires you to specify information about the second database to be used for comparison. When two pathnames are provided, the output_rule_file assumes the two databases are compared, and the appropriate statements appear in it.

Assume you execute the following command:

```
$MGC_HOME/bin/create_compare_rules rules.xor my.gds
```

The rules.xor file will contain a header that looks similar to this:

```
LAYOUT SYSTEM GDS
LAYOUT SYSTEM2 GDS
LAYOUT PATH 'my.gds'
```
Utilities

Other Utilities

```
LAYOUT PATH2 //!! FILL ME IN
LAYOUT PRIMARY `*'
LAYOUT PRIMARY2 `*'
LAYOUT ERROR ON INPUT NO
DRC RESULTS DATABASE //!! FILL ME IN
LAYOUT BUMP2 61
```

All of the required statements for dual-database comparison appear in the file rules.xor. You must fill in the Layout Path2 statement with the pathname of the database you intend to compare to my.gds. You must also fill in the pathname of the DRC Results Database.

The remainder of the output rule file (not shown) contains Layer Map statements and XOR rule checks for a layer-by-layer comparison. It is assumed the second database used in the comparison will have corresponding layers to my.gds.

If you specify two layout databases for this command, the Layout Path2 statement will be filled in.

Other Utilities

The GDSII-to-OASIS translator is documented under “Using the gds2oasis Translator,” in the Calibre for the Open Artwork System Interchange Standard.
Appendix A
Related Publications

Configuration and Licensing

*Configuring and Licensing Calibre Tools* contains information on system configuration and licensing information for the Calibre® verification, MDP, RET, and parasitic extraction toolsets.

*Calibre Configuration, Licensing, and MT/MTflex Release Notes* contains new and changed functionality for general Calibre® issues, configuration and licensing, and multi-threaded and Calibre® MTflex™ operations.

Calibre RET and MDP

*Calibre Mask Data Preparation User’s Manual* describes the input requirements, invocation usage, and concepts for the Calibre® FRACTURE tools.

*Calibre MDPview User’s Manual* describes the data viewing and inspection capabilities available through Calibre® MDPview™.

*Calibre MDP Release Notes* contains information on new or changed functionality specific to the Calibre® FRACTURE tools.

*Calibre RET Reference Manual* contains a dictionary-style reference providing syntax and descriptions of keywords, commands, and variables that you can use in any RET setup file to control model-based simulations. This information is relevant to these products: the Calibre® WORKbench™ application and the Calibre® OPCpro™, Calibre® ORC™, and Calibre® PRINTimage™ batch tools.

*Calibre RET Release Notes* contains information on new or changed functionality specific to the Calibre® RET toolset: Calibre® WORKbench™, Calibre® LITHOview™, Calibre® OPCpro™, Calibre® ORC™, Calibre® PRINTimage™, Calibre® PSMgate™, Calibre® TDopc™, and Calibre® OPCsbar™.

*Calibre Model-Based OPC User’s Manual* describes model-based resolution enhancement technologies available through the Calibre® LITHO operation. These are Calibre® OPCpro™, Calibre® ORC™, and Calibre® PRINTimage™.

*Calibre Rule-Based OPC User’s Manual* describes the rule-based OPC functionality available through Calibre® DRC. These are rule-based OPC using SVRF statements, OPCLineend and OPCbias.
Related Publications

**Calibre OPCsbar User’s and Reference Manual** describes the functionality available through the Calibre® OPCsbar™ tool for rule-based scattering bar creation.

**Calibre PSMgate User’s Manual** explains how to use Calibre® PSMgate™ for performing OPC on phase shifting masks.

**Calibre WORKbench User’s Manual** describes the layout viewing and editing capabilities available through both Calibre® WORKbench™ and Calibre® LITHOview™ plus the optical and process modeling capabilities available through Calibre WORKbench.

**Batch Commands User and Reference Manual** covers the Tcl/Tk batch commands available through the Calibre® WORKbench™, Calibre® DESIGNrev™, and Calibre® MDPview™ tools.

**Calibre Matrix OPC User’s Manual** describes using the Calibre® Matrix OPC tool.

### Calibre Verification

**Calibre DESIGNrev Release Notes** contains information on new or changed functionality specific to the Calibre® DESIGNrev™ tool.

**Calibre DESIGNrev User’s Manual** describes the layout viewing and editing capabilities available through Calibre® DESIGNrev™.

**Calibre Interactive User’s Manual** describes how to use the Calibre® Interactive™ GUI to perform interactive verification for DRC, LVS, and extraction runs.

**Calibre Query Server Manual** contains information about the commands and usage of the Calibre® Query Server and Calibre® Connectivity Interface, which are used to examine connectivity information contained in a Standard Verification Database (SVDB).

**Calibre Verification Release Notes** contains information on new or changed functionality specific to the Calibre® DRC, Calibre® DRC-H™, Calibre® LVS™, Calibre LVS-H™, Query Server, Calibre® RVE™, and Calibre® Interactive™ toolsets.

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

### Calibre YieldAnalyzer and Calibre YieldEnhancer

**Calibre Design For Manufacturability (DFM) Guide** contains key concepts related to using Calibre® YieldAnalyzer and Calibre YieldEnhancer applications, which are used to improve manufacturing yield.
Calibre xL

*Calibre xL Release Notes* contains information on new or changed functionality specific to the Calibre® xL tool.

*Calibre xL User’s Manual* contains concepts and usage procedures for using the Calibre® xL tool for parasitic inductance extraction.

Calibre xRC

*Calibre xRC/ADvanceMS User’s Manual* describes how to set up and run Calibre® xRC™ on an analog/mixed-signal (AMS) design as part of the Calibre xRC/ADvanceMS™ (ADMS) flow.

*Calibre xRC Release Notes* contains information on new or changed functionality specific to the Calibre® xRC™ tool.

*Performing Transistor-Level Parasitic Extraction* contains concepts and usage procedures for using the Calibre® xRC™ tool for transistor-level parasitic extraction.

*Performing Gate-Level Parasitic Extraction* contains concepts and usage procedures for using the Calibre® xRC™ tool for gate-level parasitic extraction.

*Performing Hierarchical Parasitic Extraction and Netlisting* explains using the Calibre® xRC™ tool for creating hierarchical parasitic netlists.

*SVRF Capacitance and Resistance Statements — Concepts and Usage* presents specific information for creating Standard Verification Rule Format (SVRF) Capacitance and Resistance statements for subsequent use with the Calibre® xRC™ tool.

*xCalibrate Rule File Generator Release Notes* contains information on new or changed functionality specific to the xCalibrate™ Rule File Generator tool.

*Using the xCalibrate Rule File Generator* contains key concepts, process and modeling descriptions, procedures, and reference information xCalibrate™ tool users can use when calibrating, validating, and generating SVRF rule file capacitance specification statements.

Common User's Manuals

*Using Mentor Graphics Documentation with Acrobat Reader* describes how to set up and use the Mentor Graphics-supplied Acrobat Reader with enhancement plugins for online viewing of Mentor Graphics PDF-based documentation and help. The manual also includes tips on using Reader.

*Using MTflex with the Calibre Toolset* describes the invocation procedures for the Calibre® MTflex functionality, which allows you to run supported Calibre tools in a multiple-processor environment.
Related Publications

*Calibre for the Open Artwork System Interchange Standard* describes the Calibre® application support for and implementation of the OASIS (Open Artwork System Interchange Format) layout format standard.
— Symbols —
*BUSDELMITER control statement, 14-11
*CAPA control statements, 14-11
*CONNECT control statements, 14-11
*DIODE control statements, 14-12
*EQUIV control statements, 14-13
*J control statements, 14-11
*LDD control statement, 14-14
*MEGA control statement, 14-14
*PININFO control statement, 14-16
*XPINS control statement, 14-16
.END control statement, 14-6
.ENDL control statement, 14-7
.ENS statement, 14-36
.EOM statement, 14-36
.GLOBAL statement, 8-14, 14-45
.INCLUDE control statement, 14-7
.LIB control statement, 14-8
.MACRO statement, 14-33
.OPTION SCALE control statement, 14-9
.PARAM statement, 14-43
.SUB statement, 14-33
.SUBCKT statement, 8-7, 14-33
duplicates, 14-35

— Numerics —
-64 switch, 3-12, 3-22

— A —
-a array_delimiters argument, 15-5
-a char1 argument, 15-41
Ambiguities
  defined, 11-16
  resolving, 11-16
AND operation, 4-7
  hierarchical, 6-1
Antenna checks, 4-49
Appropriate angle, 4-33
Appropriateness criteria, 4-31
Area-based filtering of layout database, 4-61
AREF output, 4-48
Arguments
  -64 (DRC), 3-12
  -64 (LVS), 3-22
  -a array_delimiters, 15-5
  -a char1, 15-41
  -automatch, 3-17
  -b, 15-7
  -b char, 15-41
  -bpf, 3-15
  -c char1, 15-41
  -c subs_char, 15-5
  -cb, 3-10, 3-17, 3-25
  -cb (V2LVS), 15-7
  -cell, 3-15
  -cl, 3-23
cnet_file_name, 3-15
  -cs, 3-22
dblayers, 3-15
drc, 3-10
drc -hier, 3-10
e, 15-7
e edif_input_file, 15-40
h (V2LVS), 15-7
hcell, 3-20
-hier, 3-17
-i (E2LVS), 15-42
-i (V2LVS), 15-7
-ictrace (V2LVS), 15-8
-ixf (LVS), 3-17
-l input_list_file, 15-40
-l verilog_lib_file, 15-3
-lsp (V2LVS), 15-3
-lsr (V2LVS), 15-4
-lvs, 3-14
-n, 15-7
-n cell_name_file, 15-42
-nl, 3-16
-nonames, 3-15
-nowait, 3-12, 3-22
Index-2

Calibre Verification User's Manual

- -nxf, 3-18
-o output_file, 15-40
-o output_spice_file, 15-3
-p prefix, 15-4
-r char1, 15-42
-rule_file_name (DRC), 3-12
-rule_file_name (LVS), 3-23
-s spice_input_file (E2LVS), 15-41
-s spice_input_file (V2LVS), 15-4
-s0 groundnet, 15-4
-s1 powernet, 15-4
-sb cell_file, 15-41
-sk, 15-4
-sl, 15-4
-spice, 3-18
-ss cell_file, 15-41
-t svdb_dir, 15-5
-tl, 3-14
-ts, 3-14
-turbo no_of_cpus (DRC), 3-10
-turbo no_of_cpus (LVS), 3-19
-turbo_all, 3-11, 3-19
-turbo_litho, 3-11
-u unnamed_pin_prefix, 15-5
-v verilog_design_file, 15-3
-w warning_level (E2LVS), 15-42
-w warning_level (V2LVS), 15-4
-wait n, 3-12, 3-22
Array reference output, 4-48
ASCII DRC results database format, 7-8
ASCII layout format, 3-3, 3-6
Attach operation, 8-8
  connectivity extraction, 8-8
  net names, 8-8
-automatch switch, 3-17

B —
-b argument, 15-7
-b char argument, 15-41
Bad device reporting, 10-6
Binary DRC results database format, 7-12
Binary layout format, 3-3, 3-7
Binary polygon file (BPF), 13-54
Binary polygon format, 3-7
Bit structure, 12-21
BJT element (Table), 14-26
BNF, 3-4
Boolean operations, 4-8
  one-layer, 4-3
  two-layer, 4-8
-bpf switch, 3-15
bpf, see Binary polygon format, 3-7
Built-in device types
  bipolar transistors, 11-12
  capacitors, 11-10
  defined, 11-7
  diodes, 11-11
  inductors, 11-13
  Jfet transistors, 11-12
  MOS transistors, 11-9
  resistors, 11-10
  voltage sources, 11-13
Built-in language
  examples, 10-14
Built-in property classification, 11-69
Built-in W/L partner properties, 11-69

C —
-c char1 argument, 15-41
-c subst_char argument, 15-5
Calibre CB
  described, 2-2, 3-25
Calibre command line, 3-8
Calibre DRC
  described, 2-1
  invocation, 3-9
Calibre DRC-H
  described, 2-1
Calibre invocation, 3-8
Calibre LVS
  described, 2-2
  example invocation, 3-23
Calibre LVS-H
  described, 2-2
  example invocation, 3-23
Calibre verification overview, 2-1
Calling conventions for Verilog modules, 15-17
Capacitor element (Table), 14-21
Case preservation for Device operations, 12-45
-cb argument (V2LVS), 15-7
-cb switch, 3-10, 3-17, 3-25
<table>
<thead>
<tr>
<th>Index-3</th>
</tr>
</thead>
</table>

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|Cell correspondence file example, 3-21| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Cell exclusion, 4-60| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Cell pushdown, 12-4| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Cell renaming, 4-60| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-cell switch, 3-15| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Check set, 5-2| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Check text defined, 4-12| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|CIF input control, 4-58| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|CIF layout format, 3-3| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Circuit matching signature based, 11-15| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|tracing, 11-15| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-cl switch, 3-23| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Clustered output, 4-25| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|CNET database format, 3-8 layout, 3-3 source, 3-8| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|cnet_file_name argument, 3-15| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Command line Compare GDSII, 15-56| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|DRC, 3-9| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|E2LVS, 15-40| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|LVS, 3-13| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|V2LVS, 15-3| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Compare GDSII command line, 15-56| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|compare_gds, 2-2, 15-56| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Comparing two layouts, 2-2| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Comparison layout versus layout, 4-43| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Component subtype defined, 11-3 extracted layout device, 11-4| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Component type defined, 11-3 extracted layout device, 11-3| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Concurrency, 5-3| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Concurrency checks, 5-5| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Conjunctive checks, 5-7| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Conjunctive design rule checking, 4-29 Connect By operation, 8-4| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Connect operation, 8-3 Connectivity establishing and verifying, 8-1| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Connectivity comparison initial correspondence points, 11-16 matching of circuit elements, 11-15 resolving ambiguities, 11-16 results, 11-16| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Connectivity extraction errors and warnings, 8-18 mask mode, 8-2 operations, 8-3 recognizing electrical nets, 8-3 Virtual Connect statements, 8-14| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Constraints, 4-16 for output suppression, 4-29 Contact checks, 5-8| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|Correspondence file hcell, 3-21 create_compare_rules utility, 2-2 create_layer_rules utility, 2-2.cs switch, 3-22| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|D Database pre-merging, 4-60 Databases combining, 4-43 DRC results, 7-6 GDSII, 3-4 GDSII comparison, 15-56| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
layout, 3-3
mask results, 13-48
OASIS, 3-6
source, 3-8
Datatypes, 4-42
-dblayers switch, 3-15
DEBUG statement, 10-28
Debugging rule checks using Copy, 4-11
Derived edge layers, 4-3
Derived error layers, 4-4
Derived layers, 4-3
Derived polygon layers, 4-3
Device
 defined, 10-2
 property specification error messages,
 10-36
Device operation, 11-3, 11-4, 11-64, 12-2
Device recognition
 aspects of a device, 10-2
defined, 10-1
example, 10-6
hierarchical, 10-9, 12-2
ill-formed devices, 10-5
layer relations, 10-3
pin fill-in algorithm, 10-5
property computation, see Property
 computation, 10-9
Device reduction
defined, 11-17
parallel bipolar transistors, 11-28
parallel capacitor, 11-30
parallel MOS transistors, 11-18
parallel resistor, 11-32
programs, 11-35
semi-series MOS transistors, 11-21
semi-split gate, 11-26
series capacitor, 11-29
series resistor, 11-31
tolerance, 11-36, 11-38
unequally reduced devices, 11-34
Devices
 built-in, 11-7
Dimensional check operations, 4-13
 appropriate angle, 4-33
 appropriateness criteria, 4-31
edge breaking, 4-33
edge measurement, 4-16
edge output clusters, 4-16
edge-directed, 4-25
error-directed, 4-13
intersection criteria, 4-33
key concepts, 4-16
metrics, 4-18
output suppression, 4-29
polygon suppression criteria, 4-36
polygon-directed, 4-14
region output, 4-37
trivial edges, 4-27
Discrepancies
 LVS report, 13-17
types, 13-32
Disk-based layers, 4-41
DRC
 arguments, 3-10
data flow, 5-1
execution characteristics, 5-3
hcells and, 6-6
invocation, 3-13
maximizing capacity, 5-7
minimizing time, 5-7
required rule file statements, 3-2
results limiting, 7-16
results limits, 5-2
specification statements, 5-2
DRC Check Map statement, 7-14, 7-16
DRC Check Text statement, 7-10
DRC Exclude False Notch statement, 6-5
DRC executive
 check set, 5-2
disk-based layers, 4-41
empty rule checks, 5-3
limiting the result count, 7-16
polygon segmentation, 5-11
transcript section, 7-3
transferring text, 4-12
DRC Keep Empty statement, 5-3, 7-17
DRC Map Text statement, 6-6
DRC Maximum Results statement, 7-16
DRC Maximum Vertex statement, 7-16
DRC Print Area statement, 6-5
DRC results database, 7-6
   empty rule checks, 5-3
   format, 7-8
   GDSII format, 4-46
   hierarchical DRC, 7-16
   limiting the result count, 7-16
   OASIS format, 7-14
   polygon segmentation, 5-11
   precision, 7-8
   transferring text, 4-12
DRC Results Database statement, 7-8, 7-13, 7-16
DRC summary report, 7-17
DRC Summary Report statement, 7-18
-drc switch, 3-10
DRC tolerances, 4-41
DRC-H
   arguments, 3-10
   usage, 3-9
Dual database capability, 4-43
   hierarchical semantics, 4-45
   layer bump process, 4-44
   rule file statements, 4-44
   supported formats, 4-44
Duplicate .SUBCKT definitions, 14-35
Duplicate cells, 4-59

— E —
-e argument, 15-7
-e edif_input_file argument, 15-40
E2LVS
   command line arguments, 15-40
   described, 15-39
   EDIF-to-SPICE translations, 15-47
   examples, 15-42
   invoking, 15-40
   netlist example, 15-53
   overview, 2-2
   translation issues, 15-45
   untranslated EDIF syntax, 15-43
   usage, 15-40
Edge breaking, 4-33
Edge measurement
   described, 4-16
   region construction, 4-16
Edge output clusters, 4-25

Edge-directed dimensional check operations, 4-14
Edge-directed output, 4-36
   negative, 4-36
   positive, 4-36
EDIF-to-LVS, see E2LVS
Efficiency
   concurrency checks, 5-5
   conjunctive checks, 5-7
   hierarchical operation, 6-4
   pad checks, 5-9
   property computation, 10-22
   rectangle checks, 5-8
   rule file, 5-7
Element statements
   BJT, 14-26
   capacitor, 14-20
   JFET, 14-28
   junction diode, 14-24
   MOSFET, 14-29
   resistor, 14-18
   voltage source, 14-32
   X, 14-36
   Y, 14-42
Empty rule checks, 5-3
Enclosure operation, 4-14, 4-31
ERC
   definition, 9-1
   DRC execution, 9-3
   DRC rule check selection, 9-4
   LVS execution, 9-2
   LVS rule check selection, 9-3
   operations, 9-1
   PRINT keyword, 9-2
   PRINT NETS operation, 9-5
   PRINT POLYGONS operation, 9-5
   results, 9-4
   statements, 9-1
   usage examples, 9-6
Error suppression, false notch, 6-5
Error tolerance, 4-41
Error-directed dimensional check operations, 4-13
   Escaped identifiers, 15-7
   Establishing connectivity, 8-1
Expand Cell statement, 6-6
Explicit layer definition, 4-5
External operation, 4-14, 4-31, 6-5
Extracted layout device pin names, 11-4

— F —
False enclosure reduction, 4-39
False measurement algorithm, 4-40
False notch error reduction hierarchical DRC, 6-5
False notch reduction, 4-39
Filtering unused devices bipolar transistors, 11-40
MOS transistors, 11-39
Flagging geometries, 4-61
Flat instantiations, hierarchical DRC, 6-4
Flatten Cell statement, 6-6
Flatten operation, 6-4
Free-standing layer operation, 4-3

— G —
Gate-level primitives in V2LVS, 15-16
GDSII datatypes, 4-42
DRC results, 4-46
DRC results database format, 7-13
input control, 4-58
layout database format, 3-4
texttypes, 4-42
GDSII database comparison, 2-2
Geometric DRC output, 4-46
Geometry flagging, 4-61
Geometry grid-snapping, 4-61
Geometry promotion in DRC-H, 6-2
Getting started, 3-1
Global layers, 4-5

— H —
-h argument (V2LVS), 15-7
Hcell, 12-5
  connectivity, 12-2
  DRC treatment, 6-6
  list, 3-21
  pins, 11-68, 12-8
  specification, 12-1
-hcell switch, 3-20
-hier switch, 3-10, 3-17
Hierarchical application semantics, 4-45
Hierarchical cell statistics, 12-44
Hierarchical cells
  forming a cycle, 13-31
Hierarchical device recognition, 10-9
Hierarchical DRC
described, 6-1
efficiency, 6-4
false notch error suppression, 6-5
flat instantiations, 6-4
layer area printing, 6-5
performance, 6-2
results database, 7-16
text mapping, 6-6
Hierarchical LVS, 12-1
circuit extraction, 12-1
comparison, 12-1
device recognition, 12-2
Hierarchy-specific statements, 6-6
High-short resolution, 6-6

— I —
-i argument (E2LVS), 15-42
-i argument (V2LVS), 15-7
-ictrace argument (V2LVS), 15-8
Ill-formed device defined, 10-5
Implicit layer definition, 4-6
Include statement, 4-6
Incremental connectivity, 4-49
Inductor element (Table), 14-23
Initial correspondence points defined, 11-16
Initialization section, 7-2
Input control layout, 4-58
Input errors bad instances, 13-30
conflicting instances, 13-30
defined, 13-30
missing component types, 13-30
missing pin names, 13-30
Instance defined, 10-2
Instance pins and pin names, 11-4
Internal operation, 4-14, 4-31
Intersection criteria, 4-33
Interval constraints, 4-29
Isolated net, 10-3
Isolated nets, 11-41
ixf file format, 13-52

— J —
JFET element (Table), 14-28
Junction diode element (Table), 14-25

— L —
-l input_list_file argument, 15-40
-l verilog_lib_file argument, 15-3
Labels, net names, 8-8
Layer area printing
   hierarchical DRC, 6-5
Layer constructors, 4-8
Layer definition
   explicit, 4-5
   implicit, 4-6
Layer Map statement, 6-6
Layer of origin, 4-9
Layer operations, 4-7
   executing concurrently, 5-3
   hierarchical, 12-3
   layer constructors, 4-8
   layer selectors, 4-8
   net preserving, 4-8
   redundancy elimination, 5-6
   scheduling, 5-6
Layer selecters, 4-8
Layer set, 4-2
Layer statistics, 7-3
   DRC, 7-3
   DRC-H, 7-4
   flat count, 7-2
   hierarchical count, 7-2
   LVHEAP, 7-6
Layers, 4-2
Layout data input section, 7-1
Layout database
   area-based filtering, 4-61
   formats
      ASCII, 3-3, 3-6
      binary, 3-3, 3-7
      CIF, 3-3
      CNET, 3-3, 3-8
      GDSII, 3-3, 3-4
      OASIS, 3-3
      SPICE, 3-3, 3-6
   Layout database magnification, 4-62
   Layout Depth statement, 3-5, 3-6
   Layout formats, 3-3
   Layout Path statement, 3-3, 3-4, 3-14
   Layout Primary statement, 3-3, 3-4, 7-9, 7-13
   wildcards, 4-59
   Layout Process Box Record statement, 3-5
   Layout System statement, 3-3, 3-4
   Layout Top Layer statement, 6-6, 12-1
   Layout translation, see tl / ts
   Layout versus layout
      comparison, 4-43
   Layout versus schematic, see also connectivity
      comparison, LVS
   Local layers, 4-5
   Logic gate recognition
      CMOS and-or-invert, 11-45
      CMOS gates, 11-44, 11-54
      CMOS inverter, 11-44
      CMOS NAND, 11-44
      CMOS NOR, 11-44
      CMOS or-and-invert, 11-46
      CMOS serial pulldown, 11-48
      CMOS serial pullup, 11-47
      CMOS serial-parallel pulldown, 11-49
      CMOS serial-parallel pullup, 11-49
      high level serial-parallel structures, 11-52
      NMOS gates, 11-54
      NMOS inverter, 11-54
      NMOS NAND, 11-54
      NMOS NOR, 11-55
      NMOS or-and-invert, 11-55
      NMOS serial pulldown, 11-56
      NMOS serial-parallel pulldown structure, 11-57
      restrictions, 11-27
      serial-parallel structure, 11-58
      serial-parallel structure of MP or MN devices, 11-51
series of MD or ME devices, 11-57
series of MP or MN devices, 11-50
Logic injection, 12-14
Logically equivalent pins
default pin swapping for devices, 11-64
defined, 11-64
lph file format, 13-53
-lsp argument (V2LVS), 15-3
-lsr argument (V2LVS), 15-4
LVHEAP statistics, 7-6
LV
built-in device types, 11-7
circuit comparison, 11-1
circuit extraction, 8-1
command line arguments, 3-14
command line examples, 3-23
component subtype, 11-3
component types, 11-3
data flow, 11-1
device reduction, 11-17
filtering unused devices, 11-39
instance pins and pin names, 11-4
logic injection, 12-14
logically equivalent pins, 11-64
MS and MF schematic devices, 11-14
net and instance names, 11-6
ports and port names, 11-6
power and ground nets, 11-41
recommended configuration, 11-3
required rule file statements, 3-3
sample rule file, 3-25
spice-like property syntax, 11-72
user given names, 11-5
LVS Device Type statement, 11-8
LVS Filter
statement, 11-72
LVS Filter Unused Bipolar statement, 11-40
LVS Filter Unused MOS Transistors statement,
11-40
LVS Ground Name statement, 11-5, 11-41,
12-9
LVS Power Name statement, 11-5, 11-41, 12-9
LVS Recognize Gates statement, 11-27, 11-42,
12-8
LVS Reduce Parallel Bipolar statement, 11-29
LVS Reduce Parallel Capacitors statement,
11-30
LVS Reduce Parallel Diodes statement, 11-33
LVS Reduce Parallel MOS statement, 11-20
LVS Reduce Parallel Resistors statement,
11-32
LVS Reduce Series Capacitors statement,
11-29
LVS Reduce Series Resistors statement, 11-31
LVS Reduce Split Gates statement, 11-24
LVS report
analysis, 13-14
connectivity extraction, 13-16
detailed instance connections, 13-15
discrepancies, 13-17
discrepancy types, 13-32
error and warning messages
  analysis, 13-14
  flat example, 13-2
  hierarchical example, 13-6
incorrect objects, 13-15
information and warnings, 13-15, 13-43
input errors, 13-30
instance identification, 13-17
instance pin identification, 13-19
layout input errors, 13-30
listing conventions, 13-17
logic gate identification, 13-18
logic gate pin identification, 13-19
net identification, 13-17
non-identical signal pins, 13-28
numbers of objects after transformation,
13-15
overall comparison results, 13-20
overall structure, flat, 13-1
overall structure, hierarchical, 13-6
port identification, 13-17
power and ground nets, 13-27
primary messages, 13-21
source input errors, 13-30
SPICE netlist, 13-16
stamp errors, 13-16
unconnected instance pin identification,
13-20
LVS Report statement, 3-18
LVS SPICE netlist notational conventions  
(Table), 14-2
-lvs switch, 3-14
LVS warnings  
defined, 13-43
LVS-H command line  
equations, 3-23

— M —
Many-many cell correspondence, 12-7
Mask connectivity extraction, 8-2
Mask SVDB Directory statement, 3-19, 3-21, 13-16
Measurement metrics, 4-18
Measurement regions, 4-16
Merged layers, transferring info to, 8-7
Metrics, 4-18  
  Euclidean, 4-18
  Opposite, 4-18
  Opposite (special considerations), 4-19
  Opposite extended, 4-18
  Opposite Extended Fsymmetric, 4-21
  Opposite Extended Symmetric, 4-21
  Opposite Fsymmetric, 4-21
  Opposite Symmetric, 4-21
  Opposite symmetric, 4-18
  Square, 4-18
  Square Orthogonal, 4-21
Missing and unknown property values, 11-34
Module instantiations in V2LVS, 15-16
MOSFET element (Table), 14-30
MS and MF schematic devices, 11-14

— N —
-n argument, 15-7
-n cell_name_file argument, 15-42
Negative edge-directed output, 4-36
Net and instance names  
defined, 11-6
  Mask LVS, 11-6
  SPICE netlist, 11-3, 11-4, 11-5, 11-6
Net names  
  attachment operations, 8-8
  Expand Cell Text statement, 8-10
  label attachment, 8-10
  labels, 8-8
layout database text objects  
in Calibre applications, 8-9
Layout Text applications, 8-9
specifying, 8-8
Text Depth statement, 8-10
Text Layer statement, 8-10
Text statements, 8-9
Net-preserving operations, 4-8
-nl switch, 3-16
-nonames switch, 3-15
Non-commutative measurements, 4-18
NOT operation, 4-7
-nowait switch, 3-12, 3-22
nxf file format, 13-52
-nxf switch, 3-18

— O —
-o output_file argument, 15-40
-o output_spice_file argument, 15-3
OASIS  
database format, 3-3
DRC results, 4-46
DRC results database format, 7-14
layout database, 3-6
layout input, 4-58
One-layer Boolean operations, 4-3
Operations  
  connectivity extraction, 8-3
described, 3-2
Device, 11-3, 11-4, 11-64
execution time, 5-9
layer, 4-7
net name attachment, 8-8, 8-10
pin related, 8-5
port related, 8-5
text attachment, 8-10
text related, 8-9
Opposite extended Fsymmetric metric, 4-22
Opposite extended symmetric metric, 4-22
Opposite Fsymmetric metric, 4-22
Opposite Symmetric metric, 4-22
Opposite symmetric, 4-18
Opposite extended Fsymmetric, 4-22
Opposite extended symmetric metric, 4-22
Opposite Fsymmetric metric, 4-22
Opposite Fsymmetric metric, 4-22
Opposite Symmetric metric, 4-22
Opposite symmetric, 4-18
OR operation, 4-7
Original (drawn) layers, 4-2
Overall comparison results  
correct, 13-21
defined, 13-20
error, 13-22
example, 13-20
incorrect, 13-21
not compared, 13-21
number of instances, 13-26
number of nets, 13-26
number of ports, 13-26

— P —

-p prefix argument, 15-4
Pad checks, 5-9
Parallel bipolar transistor reduction, 11-28
Parallel capacitor reduction, 11-30
Parallel MOS transistor reduction, 11-18
Parallel resistor reduction, 11-32
Parameterized cells, 12-13
Partner properties, 11-70
Pass-through nets, 11-41
Performance optimization, 5-7
Pin operations, 8-5
Pin swapping, 11-64
Point-to-point measurement output, 4-28
Polygon containment criteria, 4-35
   edge breaking, 4-33
   overview, 4-36
Polygon segmentation, 5-11
Polygon-directed dimensional check operations, 4-14
Polygon-directed output, 4-37
Port Layer Polygon statement, 11-6
Port Layer Text statement, 11-6
Ports
   depth, 8-7
   port names, 11-6
   text objects, 8-6
      hierarchical netlisting, 8-7
Positive edge-directed output, 4-36
Power and ground nets, 11-7, 11-41
Precision statement, 7-13
PRINT NETS operation (ERC), 9-5
PRINT POLYGONS operation (ERC), 9-5
Property computation
   built-in language examples, 10-14
default computations, 10-11

— R —

-r char1 argument, 15-42
Reading LVS report, 13-14
Recognizing electrical nets
   Attach operation, 8-8
   Connect By operation, 8-4
   Connect operation, 8-3
   hierarchical netlisting, 8-7
   hierarchical processing, 8-6
   port text objects
      hierarchical netlisting, 8-7
      hierarchical processing, 8-6
   ports and pins, 8-5
   shapes on a single layer, 8-3
   summary, 8-3
   transferring logical information to merged layers, 8-7
REGION keyword, 4-37
Register File Bit structure, 12-41
Required rule file statements, 3-2
Resistor element (Table), 14-19
Rule checks
   comments, 4-12
   concurrent, 4-12
   conjunctive, 5-7
   debugging, 4-11
   empty, 5-3
   exclusion, 5-2
   results limits, 5-2
   selection, 5-2
   statements, 4-10
Rule file
   basics, 3-2
   DRC required statements, 3-2
   layers, 4-7
LVS required statements, 3-3
optimized, 5-7
required statements, 3-2
rule_file_name (LVS) argument, 3-23
rule_file_name argument, 3-12

S
-s spice_input_file (V2LVS), 15-4
-s spice_input_file argument (E2LVS), 15-41
-s0 groundnet argument, 15-4
-s1 powernet argument, 15-4
-sb cell_file argument, 15-41
Sconnect
description, 8-4
Secondary keywords, 4-15
Semi-series MOS transistor reduction, 11-21
Semi-split gate reduction, 11-26
Series capacitor reduction, 11-29
Series MOS Transistor Reduction, 11-20
Series resistor reduction, 11-31
Short isolation, 8-15
Signature based circuit matching, 11-15
Single layer, shapes on, 8-3
Single net, 8-3
-sk argument, 15-4
-sl argument, 15-4
Snapping geometries to grid, 4-61
Soft connections
checking for, 4-55
Source database
accepted formats, 3-8
format
CNET, 3-8
SPICE, 3-8
Source Path statement, 3-15
Source System statement, 3-8
Source translation, see tl / ts
Special cases in layout
isolated nets, 11-41
pass-through nets, 11-41
Specification statement
described, 3-2
DRC related, 5-2
Include, 4-6
Layout Depth, 3-5, 3-6
Layout Path, 3-4, 3-14
Layout Primary, 3-4
Layout Process Box Record, 3-5
Layout System, 3-4
LVS Device Type, 11-5, 11-8
LVS Filter, 11-72
LVS Filter Unused Bipolar, 11-40
LVS Filter Unused MOS Transistors, 11-40
LVS Ground Name, 11-5, 11-41, 12-9
LVS Power Name, 11-5, 11-41, 12-9
LVS Property Map, 11-72
LVS Recognize Gates, 11-27, 11-42, 12-8
LVS Reduce Parallel Bipolar, 11-29
LVS Reduce Parallel Capacitors, 11-30
LVS Reduce Parallel Diodes, 11-33
LVS Reduce Parallel MOS, 11-20
LVS Reduce Parallel Resistors, 11-32
LVS Reduce Series Capacitors, 11-29
LVS Reduce Series Resistors, 11-31
LVS Reduce Split Gates, 11-24
LVS Report, 3-18
Mask SVDB Directory, 3-19, 3-21
Port Layer Polygon, 11-6
Port Layer Text, 11-6
Source Path, 3-15
Source System, 3-8
Text Print Maximum, 7-2
sph file format, 13-53
SPICE
capacitors, 14-4
diodes, 14-4
format, 3-6
hierarchical, 12-42
layout format, 3-3
MOS transistors, 14-3
netlist
subcircuit statements, 14-33
syntax summary, 14-2
resistors, 14-4
source database format, 3-8
subcircuits, 14-33
summary, 14-2
syntax check report, 13-13
syntax defined, 11-72
SPICE string parameters, 14-44
-spice switch, 3-18
SPICE syntax check results, 13-26
SRAM Bit-Cell, 12-9
-sx cell_file argument, 15-41
Stamp operation, 8-5
Statistics
  counts, flat, 7-2
  counts, hierarchical, 7-2
DRC layers, 7-3
DRC-H layers, 7-4
layer, 7-3
LVHEAP, 7-6
String parameters, 14-44
Subcircuit call (Table), 14-36
Subcircuit statement, 14-33
  .ENDS, 14-36
  .EOM, 14-36
  .GLOBAL, 14-45
  .MACRO, 14-33
  .PARAM, 14-43
  .SUB, 14-33
  .SUBCKT, 14-33
calls, 14-36
Subckt statement (Table), 14-34
SVDB
  header, 13-53
SVRF described, 2-1
— T —
-t svdb_dir argument, 15-5
Text attachment operations, 8-10
Text mapping
  hierarchical DRC, 6-6
Text operations, 8-9
Text Print Maximum statement, 7-2
Texttypes, 4-42
-tl switch, 3-14
Tolerance
  device reduction, 11-36
Trace Property statement, 10-9
Tracing circuit matching, 11-15
Tracing properties, 11-69
  SPICE syntax, 11-72
Transcript, 7-1
  DRC executive section, 7-3
  initialization section, 7-2
  layout data input section, 7-1
  rule file compilation section, 7-1
Transmission Gate Multiplexer (TGM)
  structure, 12-39
Trivial edges, 4-27
-ts switch, 3-14
turbo no_of_cpus argument (LVS), 3-19
turbo number_of_processors argument (DRC),
  3-10
-turbo switch, 3-10, 3-19
-turbo_all switch, 3-11, 3-19
-turbo_litho switch, 3-11
Two-layer Boolean operations, 4-8
— U —
-u unnamed_pin_prefix argument, 15-5
Unattached ports, 11-42
Unit Capacitance statement, 10-12
Unit Length statement, 7-13, 10-12
Unit Resistance statement, 10-12
Unused layers, 2-2
Usage
  DRC, 3-9
  DRC-H, 3-9
  E2LVS, 15-40
  V2LVS, 15-3
User given names, 11-5
User-defined device property calculations, 10-14
Utilities
  compare_gds, 2-2
  create_compare_rules, 2-2
  create_layer_rules, 2-2
  E2LVS, 2-2, 15-39
  GDSII comparison, 15-56
  overview, 2-2
  V2LVS, 2-2, 15-1
  Verilog-to-LVS, 2-2
— V —
-v verilog_design_file argument, 15-3
V2LVS
  command line arguments, 15-3
  described, 15-1
  Errors and warnings, 15-33
gate-level primitives, 15-16
generating Calibre xRC source template, 15-32
invoking, 15-3
library files, 15-8
LVS Box subcircuits, 15-30
module instantiations, 15-16
overview, 2-2
usage, 15-3
using without Verilog library, 15-29
Verifying connectivity, 8-1
Verilog
  behavioral statements, 15-21
  bit expressions, 15-17
  calling conventions, 15-17
  expressions, 15-22
  gate-level primitives, 15-16
  module instantiations, 15-16
  modules, 15-8
  primitive instances, 15-15
  specify blocks, 15-22
  syntax in V2LVS, 15-8
  user-defined primitives, 15-10
Verilog to SPICE translation, 15-1
Verilog-to-LVS, see V2LVS
Virtual Connect statements, 8-14
  LVS Box and, 8-14
  Virtual Connect Box Colon, 8-14
  Virtual Connect Box Name, 8-15
  Virtual Connect Colon, 8-14
  Virtual Connect Name, 8-15
Virtual nets, 8-13
Voltage source element (Table), 14-32

— W —
  -w warning_level argument (E2LVS), 15-42
  -w warning_level argument (V2LVS), 15-4
W/L partner properties, 11-69
  -wait n switch, 3-12, 3-22

— X —
X+ devices, 11-14, 11-27
Open Source and Third-Party Information

This section provides information on open source and third-party software that may be included in the Calibre family of products.

- This product may use TclPro extension to Tcl/Tk open source software.
  © The Regents of the University of California. All rights reserved.
  © Lucent Technologies. All rights reserved.
  © Sun Microsystems, Inc. All rights reserved.
  © Scriptics Corporation. All rights reserved.
  © XXXX. All rights reserved.
  © Ajuba Solutions. All rights reserved.
  © Karl Lehenbauer and Mark Diekhans. All rights reserved. Karl Lehenbauer and Mark Diekhans make no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.
  © Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts. All rights reserved. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted.
  DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- This product may use ITcl/Itk extension to Tcl/Tk open source software.
  © Lucent Technologies, Inc. All rights reserved.
  © The Regents of the University of California. All rights reserved.
  The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.
  © Sun Microsystems, Inc. All rights reserved.
  © Cadence Design Systems, Inc. All rights reserved.
  © XXXX. All rights reserved.
  © Lockheed Missle & Space Company. All rights reserved.
  © XXX. All rights reserved.
  © Ajuba Solutions. All rights reserved.
  © Scriptics Corporation. All rights reserved.
  © AT&T Bell Laboratories. All rights reserved.
  AT&T disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall AT&T be liable for any special, in direct or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

- This product may use Tcl open source software.
  © Sun Microsystems, Inc.
  © Jens-Uwe Mager, Helios Software GmbH
  © ActiveState Corporation.
  © Henry Spencer. All rights reserved.
  © Scriptics Corporation.
  © Lucent Technologies.
  © Kevin B. Kenny. All rights reserved.
THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This product may use TkTable extension to Tcl/Tk open source software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This product may use BLT extension to Tcl/Tk open source software.

Lucent Technologies disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall Lucent Technologies be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortuous action, arising out of or in connection with the use or performance of this software.
AT&T disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall AT&T be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

© Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts. All rights reserved. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

© The Regents of the University of California. All rights reserved.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

© Silicon Metrics Corporation. All rights reserved.

Silicon Metrics disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall Lucent Technologies be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

• This product may use Tix 8.4.0 open source software.

© ActiveState. All rights reserved.
© Ioi Kim Lam. All rights reserved.
© Tix Project Group. All rights reserved.
© Sun Microsystems, Inc. All rights reserved.
© Expert Interface Technologies. All rights reserved.
© Steen Lumholt. All rights reserved.
© The Regents of the University of California. All rights reserved. Code derived from software contributed by the Regents of the University of California carry this disclaimer.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS AND CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES.

• This product may use Milkyway open source software.

© Avant! Corp., All Rights Reserved.
© Synopsys Inc., All Rights Reserved.

• This product may use Tablelist open source software

© Csaba Nemethi All rights reserved. Code derived from software contributed by Csaba Nemethi are required to contain the following.

This library is free software; you can use, modify, and redistribute it for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. This software is
distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.
END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software between you, the end user, as an authorized representative of the company acquiring the license, and Mentor Graphics Corporation and Mentor Graphics (Ireland) Limited acting directly or through their subsidiaries or authorized distributors (collectively “Mentor Graphics”). Except for license agreements related to the subject matter of this license agreement which are physically signed by you and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties’ entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. 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 five days after receipt of Software and receive a full refund of any license fee paid.

1. GRANT OF LICENSE. The software programs you are installing, downloading, or have acquired with this Agreement, including any updates, modifications, revisions, copies, documentation and design data (“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 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; (c) for the term; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics’ standard policies and programs, which vary depending on Software, license fees paid or service purchased, apply to the following: (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 technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions.

2. EMBEDDED SOFTWARE. If you purchased a license to use embedded software development (“ESD”) Software, if applicable, Mentor Graphics 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, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into your products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

3. BETA CODE. 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. 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. You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made 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 section 3 shall survive the 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 employees and on-site contractors, excluding Mentor Graphics' competitors, whose job performance requires access and who are under obligations of confidentiality. 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 applicable and mandatory 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, whether by operation of law or otherwise (“attempted transfer”), without Mentor Graphics’ prior written consent and payment of Mentor Graphics’ then-current applicable transfer charges. Any attempted transfer without Mentor Graphics’ prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics’ option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The terms of this Agreement, including without limitation, the licensing and assignment provisions shall be binding upon your successors in interest and assigns. The provisions of this section 4 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 improper 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 LICENSED TO YOU FOR A LIMITED TERM OR LICENSED 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, IMPLIED OR STATUTORY, 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 AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

6. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, 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.

8. **INDEMNIFICATION.** 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 YOUR USE OF SOFTWARE AS DESCRIBED IN SECTION 7.
9. **INFRINGEMENT.**

9.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright or misappropriates a trade secret in the United States, Canada, Japan, or member state of the European Patent Office. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the infringement action. You understand and agree that as conditions to Mentor Graphics' obligations under this section you must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to defend or settle the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

9.2. If an infringement claim is made, Mentor Graphics may, at its option and expense: (a) replace or modify Software so that it becomes noninfringing; (b) procure for you the right to continue using Software; or (c) require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

9.3. Mentor Graphics has no liability to you if 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 make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by you that is deemed willful. In the case of (h) you shall reimburse Mentor Graphics for its attorney fees and other costs related to the action upon a final judgment.

9.4. **THIS SECTION IS SUBJECT TO SECTION 6 ABOVE AND 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 OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.**

10. **TERM.** This Agreement remains effective until expiration or termination. All obligations and liabilities stated to or which by their nature are intended to survive the termination of this Agreement will be effective after termination. This Agreement will immediately terminate upon notice if you exceed the scope of license granted or otherwise fail to comply with the provisions of Sections 1, 2, or 4. For any other material breach under this Agreement, Mentor Graphics may terminate this Agreement upon 30 days written notice if you are in material breach and fail to cure such breach within the 30 day notice period. 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.

11. **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 any Software or direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

12. **RESTRICTED RIGHTS NOTICE.** Software was 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 SW Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

13. **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 herein.

14. **AUDIT RIGHTS.** You will monitor access to, location and use of Software. With reasonable prior notice and during your normal business hours, Mentor Graphics shall have the right to review your software monitoring system and reasonably relevant records to confirm your compliance with the terms of this Agreement, an addendum to this Agreement or U.S. or other local export laws. Such review may include FLEXlm report log files that you shall capture and provide at Mentor Graphics’ request. Mentor Graphics shall treat as confidential information all of your information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement or addendum to this Agreement. The provisions of this section 14 shall survive the expiration or termination of this Agreement.
15. **CONTROLLING LAW AND JURISDICTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF OREGON, USA, IF YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH OR SOUTH AMERICA. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. This section shall not restrict Mentor Graphics’ right to bring an action against you in the jurisdiction where your place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

16. **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.

17. **PAYMENT TERMS AND MISCELLANEOUS.** You will pay amounts invoiced, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any unpaid invoices will be subject to the imposition of interest charges in the amount of 1% per month or the applicable legal rate currently in effect, whichever is lower. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 050602, Part No. 225434